LA-UR-00-3809

Attribute Measurement System Computational Block Subsystem

Design Documentation Manual Volume 1

1 Computational Block Overview

2 CPU Card

3 Video Card

4 Quad COM Port and Digital-I/O Card

5 Additional Hardware Support Documentation

5.1 Ampro Common Utilities (Card Specific Software)
5.2 Ampro Application Handbook
5.3 PC-104 Specifications
5.4 Parts List for the Overall Computational Block
5.5 Supporting Hardware
  5.5.1 Photographs
  5.5.2 X-Rays
  5.5.3 Specifications for the Corcom 6EC1 RFI Power-Line Filter
  5.5.4 Specifications for the Condor GSC20-5 Switching Power Supply

6 Commercial System Software

6.1 Datalight, ROM-DOS Version 6.22 – Operating System
  6.1.1 Description
  6.1.2 Software Development Kit
  6.1.3 List of Removed Extraneous Functions
  6.1.4 Narrative Regarding Use of All Retained Components
  6.1.5 List of ROM-DOS Setup Parameters
  6.1.6 Listing of CONFIG.SYS and AUTOEXEC.BAT Files
    6.1.6.1 CONFIG.SYS Listing
    6.1.6.2 AUTOEXEC.BAT Listing
  6.1.7 Memory Map Showing Configuration in All Memory ICs
  6.1.8 Interrupt Vector Locations and Values
  6.1.9 I/O Information
    6.1.9.1 IRQ Data
    6.1.9.2 DMA Data
    6.1.9.3 Ports Used
    6.1.9.4 Other I/O Parameters
  6.1.10 ROM-DOS User's Guide Manual
  6.1.11 ROM-DOS Developer's Guide Manual
  6.1.12 Commented Source-Code for ROM-DOS
  6.1.13 Third-Party ROM-DOS References
6.2 Phoenix, Award BIOS – System BIOS
  6.2.1 Narrative Description of Award ROM BIOS
  6.2.2 BIOS Function Information and Calling Protocols
  6.2.3 Ampro Information on Award ROM BIOS
  6.2.4 CMOS Setup Utility User's Guide for ALI M6117 Chipset
  6.2.5 Third-Party BIOS References

7 Computational Block Applications Software

8 PROM ICs

9 Internet References

1.1 High-Level Overview of the Computational Block

The computational block computer serves to combine the classified data from the four independent attribute measurement computers (NMC, PU600/PU300, PU900, and Symmetry) into a single set of unclassified pass/fail indications. The computational block is the last subsystem in the attribute measurement system (AMS) to contain classified information. The computational block computer is the only computer in the AMS to contain the classified results of all the measurement subsystems.

The computational block calculates attribute values, which are then compared to attribute threshold values contained within its software. Some attribute calculations combine data received from two of the independent measurement subsystems. Other attributes only use data from one of the independent measurement subsystems.

1.2 Computational Block Threshold Parameters

The computational block software contains attribute threshold values as defined parameters in an included header file, DATA_ATT.H. This approach allows the threshold values to be readily changed at one location in all the AMS software in response to future negotiations. The following table summarizes these attribute threshold values.

Table of Attribute Threshold Values for the Attribute Measurements

Threshold Value	**Subsystem**	**Explanation**
500 grams	NMC	Minimum total weight of plutonium in canister
0.50	NMC	Maximum ratio of alpha-n neutrons to fission neutrons to avoid being considered plutonium-oxide
1.0 cts/second	Pu900	Maximum 871-keV peak amplitude to avoid being considered plutonium-oxide
0.10	Pu600	Maximum ^{240}Pu:^{239}Pu ratio to be accepted as weapons-grade plutonium
3 years	Pu300	Minimum age of plutonium to be accepted as from a weapon
0.15	Symmetry	Maximum fractional deviation from the mean at any angle accepted as from a axially symmetric object
3-sigma	Symmetry	Minimum statistical significance of fractional deviation from the mean required to reject as axially symmetric object

The LANL report "Application Guide to Neutron Multiplicity Counting" LA-13422-M defines alpha, α, as the ratio of neutrons produced by any (α,n) neutron reaction to spontaneous fission neutrons (eqn 5-2). Induced fissions due to neutron multiplication are different from spontaneous fission and how they affect this definition should be understood from this report. Several formulas for extracting alpha are provided, which depend on various experimental parameters and measurement values. Alpha values for oxides between 1 and 10.4 are mentioned.

1.3 Listing of All Major Computational Block Components and Functions

The computational block contains commercial and custom hardware and software components, which are listed in the following tables.

Table of Computational Block Hardware Components

Vendor	**Part Number**	**Description**
Ampro	CM2-SXI-Q-73	3SXi 386 Processor Card
Ampro	MM2-SVG-Q-71	SVG-II VGA Video Card
Diamond Systems	Emerald-MM-DIO	Extra 4 Serial Ports & 48-bits of Digital I/O Card
Corcom	6EC1	RFI Power-Line Filter with AC connector
Condor	GSC20-5	5V 3.8A Switching Power Supply
		Push-button On/Off Switch
		Fuse
		PC104 card standoffs
		External Case
Amp		9-pin D-Sub Male Connector for Serial Port
Amp		15-pin D-Sub Female Connector for Parallel Port
Amp		25-pin D-Sub Female Connector for Serial Port
Amp		15-pin D-Sub Female Connector for Video Output

Table of Computational Block Software Components

Vendor	**Part Number**	**Description**
Datalight	ROM-DOS version 6.22	Operating System
Phoenix	Award BIOS version 3.10	System BIOS
LANL	DATA_ATT.C	Application Software
WCSC	COMM-DRV/LIB version 17	Communication Library
Microsoft	Visual C version 1.52	C-Compiler & Standard Library

1.5 Pictures of the Computational Block Subsystem

The following pictures are included to illustrate the design and layout of the computational block hardware.

- Front-view with the lid on showing the power on switch
- Inside-view with the lid off showing the internal components and the back panel connectors
- Back-view with the lid on showing the various back panel connectors
- Side-view with the lid on showing the side panel connector to the data barrier.

1.4 Block Diagram of Computational Block Subsystem Functions

Figure 1 shows the location of the computational block subsystem in the larger attribute measurement system.

Figure 1. Schematic Layout of the Five CPUs in the Attribute Measurement System

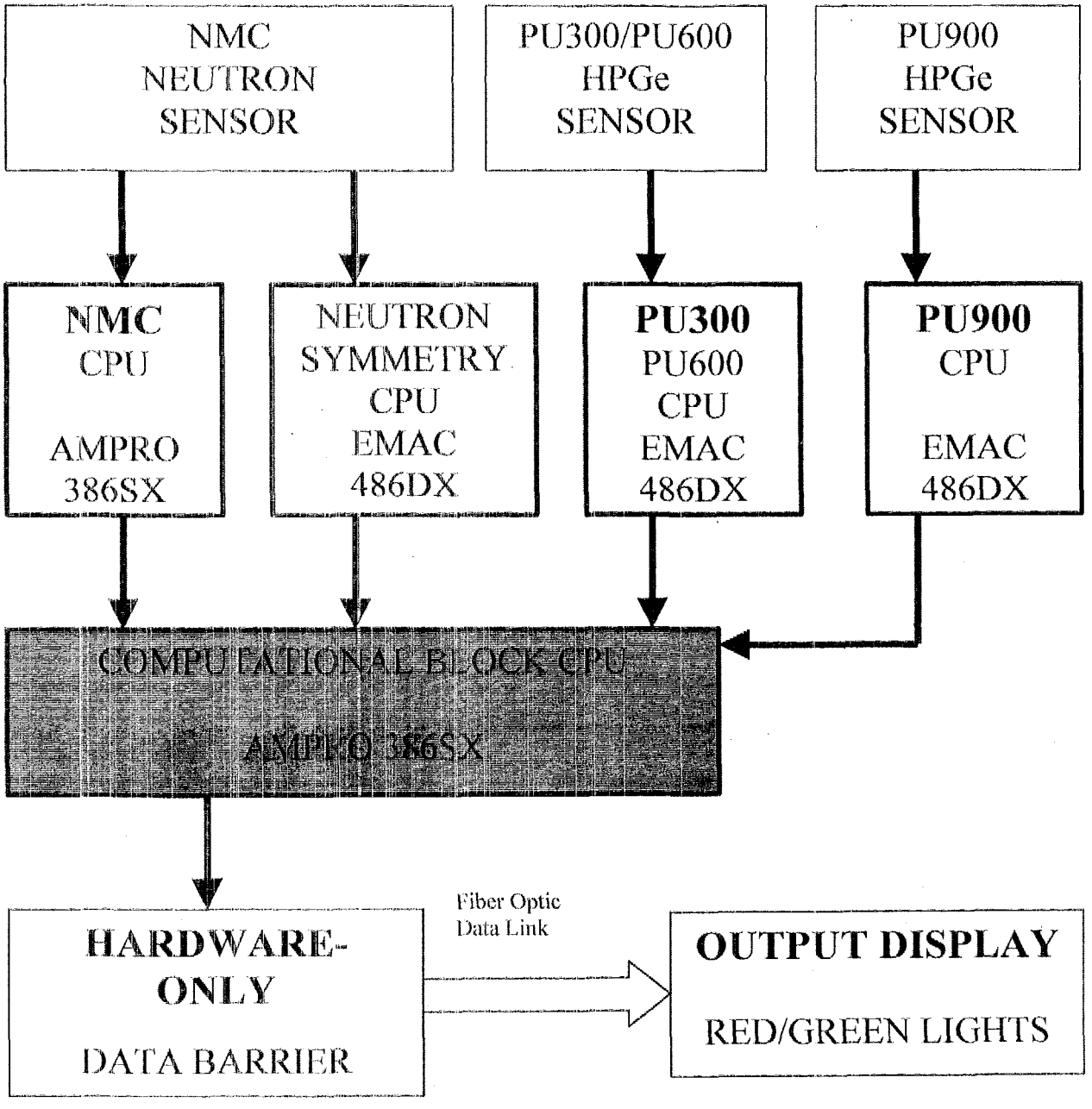
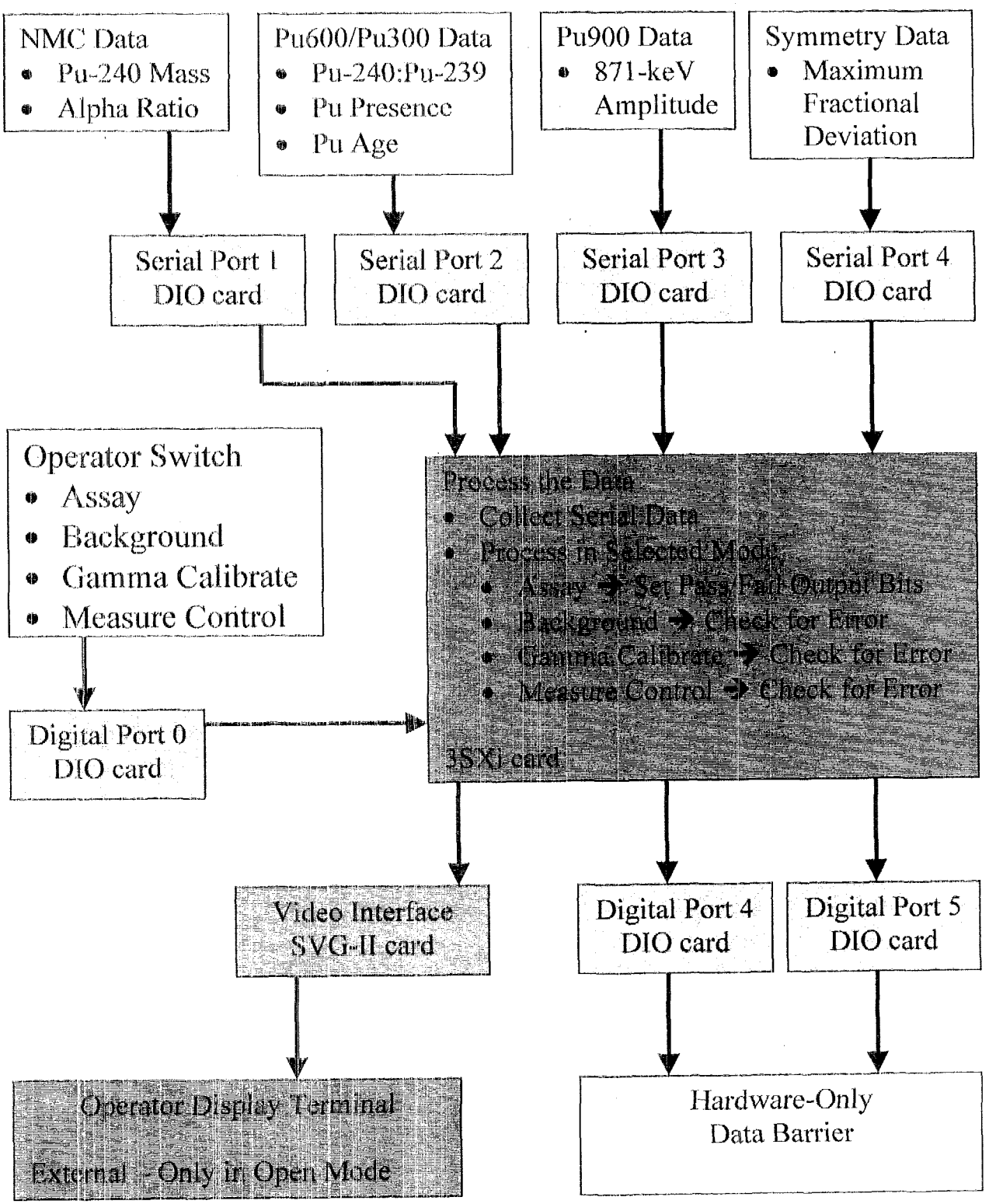


Figure 2 shows the functional block diagram of the computational block.

Figure 2. Block Diagram of the Computational Block Subsystem

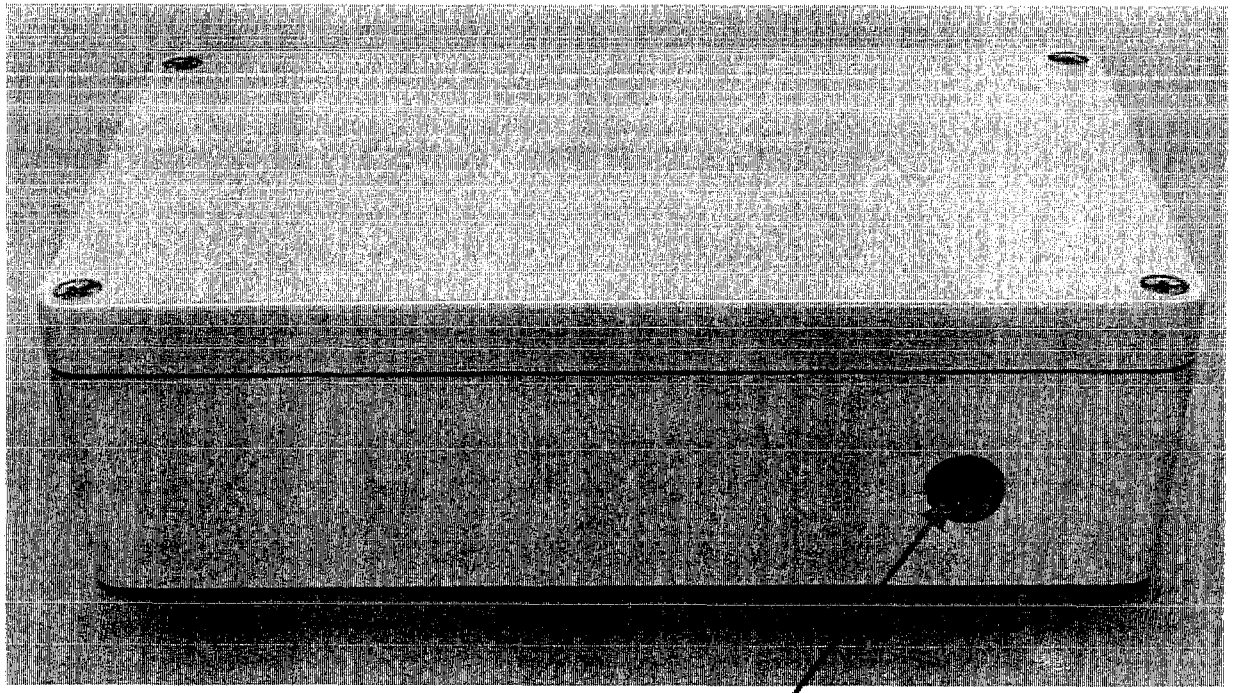


1.5 Pictures of the Computational Block Subsystem

The following pictures are included to illustrate the design and layout of the computational block hardware.

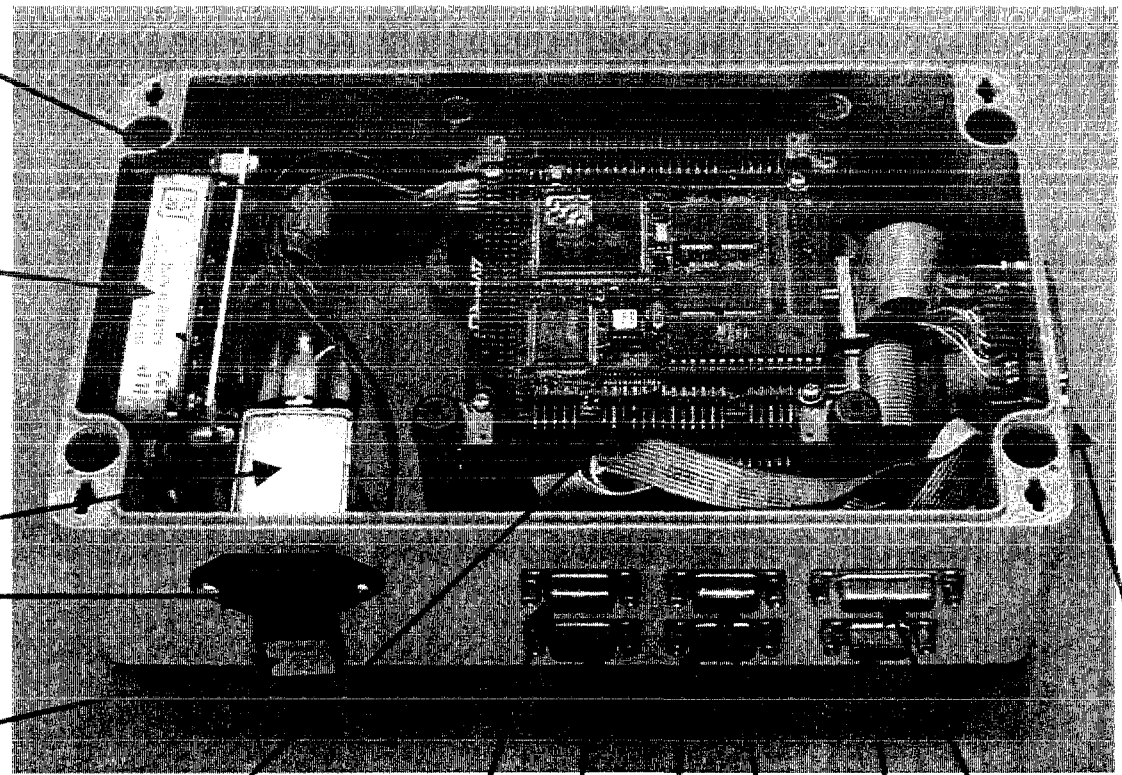
- Front-view with the lid on showing the power on switch
- Inside-view with the lid off showing the internal components and the back panel connectors
- Back-view with the lid on showing the various back panel connectors
- Side-view with the lid on showing the side panel connector to the data barrier.

Computational Block
Front-View Photograph



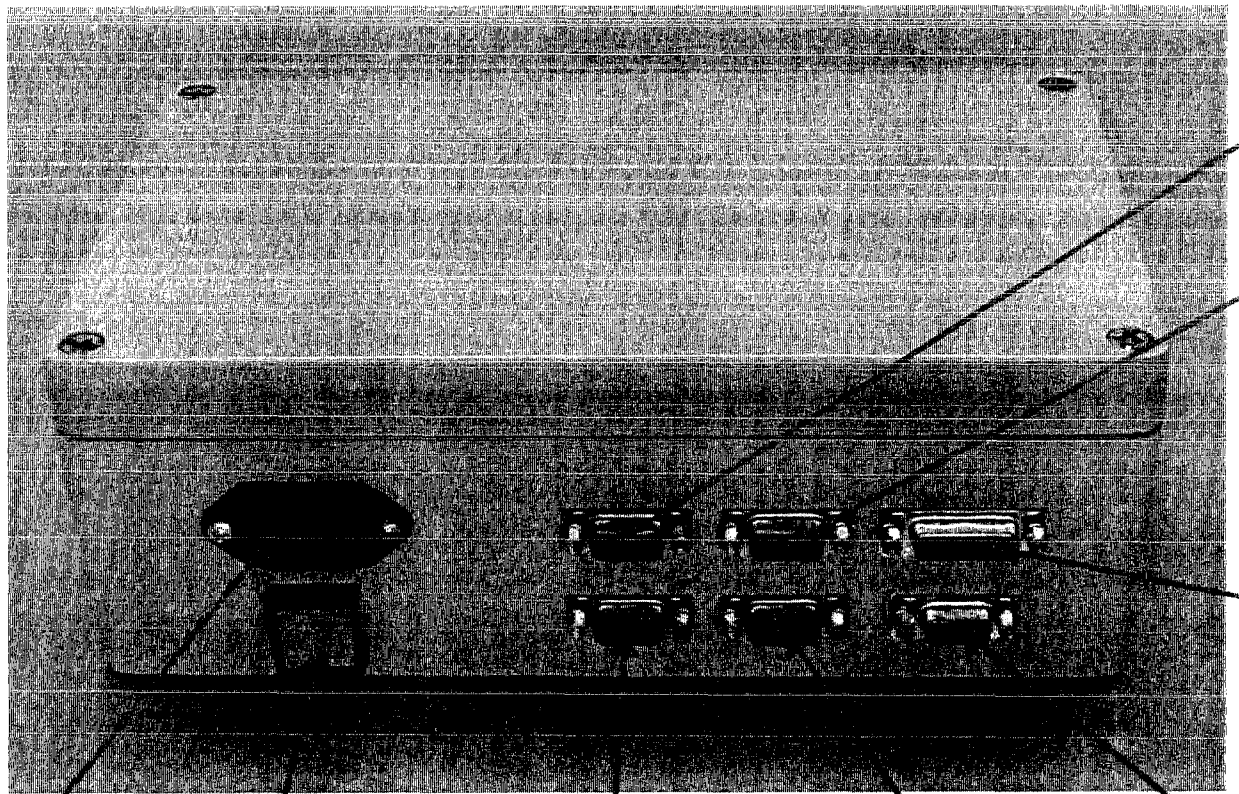
Push-button Power Switch

Computational Block Inside-View Photograph

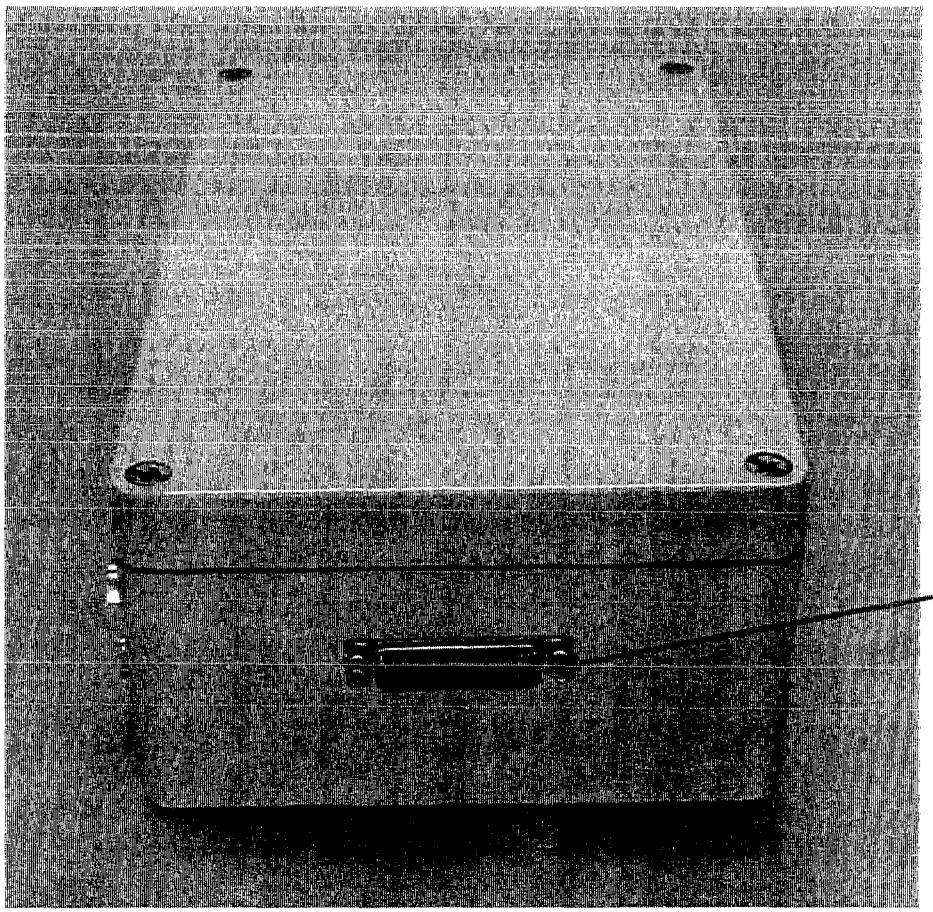


- Ampro: 3SXi processor card
- Ampro: SVG-II video card
- Diamond Systems: Emerald-MM-DIO card

Computational Block Connector-Panel Photograph



Computational Block
Connector-to-Data-Barrier Photograph



1.6 Explanation of Classified Data Strings Received with an Example Data Set of Each

The various measurement computers provide textual data to the computational block computer. These limited text strings are in principle human readable and provide all the potentially classified communication between the computers. The "x.xxx" and "y.yyy" indicate numeric data that is passed by the analysis programs to the computational block. When the data is passed as a pair of numeric values, the first is the value and the second is error in the value.

Table of Textual Input Strings to the Computational Block from the Analysis Computers

Mode	**NMC**	**Symmetry**
Assay **(Verify)**	Alpha, x.xxx, y.yyy, Pu240e, x.xxx, y.yyy (in one line)	Symmetry: x.xxx, x.xxx Symmetry: Physics Error Symmetry: Acquisition Error
Gamma Calibrate		
Measure Control	Bias PASS Bias FAIL	
Background	Background PASS Background FAIL	Symmetry: Background Complete Symmetry: Background Error

Table of Textual Input Strings to the Computational Block from the Analysis Computers

Mode	**Pu300/600**	**Pu900**
Assay **(Verify)**	PU300: x.xxx, x.xxx PU600: x.xxx, x.xxx Presence: Yes Presence: No PU300:Physics Error PU600:Physics Error Presence: Physics Error PU300: Acquisition Error PU600: Acquisition Error	PU900: x.xxx,x.xxx PU900: Physics Error PU900: Acquisition Error
Gamma Calibrate	PU300: Calibrate Complete PU300: Calibrate Error PU600: Calibrate Complete PU600: Calibrate Error	PU900: Calibrate Complete PU900: Calibrate Error
Measure Control		
Background	PU300: Background Complete PU300: Background Error PU600: Background Complete PU600: Background Error	PU900: Background Complete PU900: Background Error

In addition to the textual strings from the measurement computers, the computational block receives 6-bits of data from the operator control buttons. The operator button data determines the mode or type of calculations expected of the computer. Each of the measurement computers also receives this same operator-input data. The following table indicates the path of this mode data

- from the input connector on the real panel
- to the DIO card connector pins
- to the DIO ports
- to the Start_Data values.

The Start_Data variable is used by the DATA_ATT program to determine which calculations are appropriate for the operator-requested mode.

Table of Input Control Lines to the Computational Block from Operator Switches

Mode	**Input Pin**	**DIO Card Pin**	**DIO Port**	**DIO Bit**	**Start_Data Value**	**Explanation**
	1					NC
Assay (Verify)	2	J6-47	0	0	1	
Background	3	J6-45	0	1	2	
Gamma Calibrate	4	J6-43	0	2	3	
Measure Control	5	J6-41	0	3	4	
Cancel	6	J6-39	0	4	5	
	7					NC
	8					NC
	9					+5 volts from Power Supply
	10					NC
	11					NC
	12					NC
	13					NC
	14	J6-50				Ground from DIO card
	15					Ground from Power Supply
		J6-49				+5 volts from DIO card cut & unused
		J6-48				Signal grounds cut & unused

1.7 Explanation of Threshold Calculations Required

The computational block DATA_ATT.C program makes simple comparison calculations between the values of the measurements received from the measurement computers and the threshold values in the DATA_ATT.H header file. A few attributes require combining results from two measurement computers.

Table Summarizing the Computational Block Calculations

Attribute	**Computer**	**Calculation**
Isotopic Ratio	**Pu600**/Pu300	Compare to threshold only
Plutonium Age	Pu600/**Pu300**	Compare to threshold only
Pu Presence	Pu600/Pu300	Check for a "YES" only
Oxide	Pu900 NMC	Logical AND of • 871-keV peak area over threshold • NMC alpha over threshold
Total Pu Mass	**Pu600**/Pu300 NMC	• Calculate total mass =NMC_mass * (1 + 1/Pu_ratio) • Compare to mass threshold
Symmetry	Symmetry	Logical AND of • Maximum fractional deviation over threshold • Measured sigma over sigma threshold

1.8 Summary of Unclassified Output (Pass/Fail)

The following table indicates the path of the unclassified output data bits determined and set by the DATA_ATT program

- from the DIO ports and bits
- to the DIO card connector pins
- to the output connector on the side panel.

There is one additional bit (DIO port3, bit0), which is used to set the data into the data barrier flip-flops.

Table of Digital Output Values from Emerald-MM-DIO Card

DIO Port	**DIO Bit**	**DIO Card Pin**	**Output Pin**	**Explanation**
3	0	J5-47	10	Initializing the Data Barrier Flip-Flops
4	2	J5-27	22	Age Red
4	3	J5-25	21	Age Green
4	4	J5-23	20	Symmetry Red
4	5	J5-21	19	Symmetry Green
4	6	J5-19	18	Pu Presence Red
4	7	J5-17	17	Pu Presence Green
5	0	J5-15	9	Pu Oxide Red
5	1	J5-13	8	Pu Oxide Green
5	2	J5-11	7	Pu Mass Red
5	3	J5-9	6	Pu Mass Green
5	4	J5-7	5	Pu Isotope Red
5	5	J5-5	4	Pu Isotope Green
5	6	J5-3	3	System Error
5	7	J5-1	2	Measurement Complete
		J5-49	14	+5V
		J5-50	25	Ground
		J5-others		Cut & Unused
			1	NC
			11	NC
			12	NC
			13	NC
			15	NC
			16	NC
			23	NC
			24	NC

2.1 High-Level Description of CPU Card Operation

The CPU card is a PC104 processor card, which the computational block uses to communicate with the four measurement computers, to perform minor calculations and comparisons, and to set unclassified pass/fail bits for eventual output to the operators.

2.1.1 Brief Overview Narrative for the CPU Card

The processing power that is required of the CPU card is minimal. Commercially available PC104 systems generally use 80386, 80486, or Pentium processors and this selection was for minimal excess functionality. The 80386SX processor selected is more than adequate to meet the processing requirements.

The Ampro 3SXi CPU card does not use an Intel 80386SX processor, but rather uses an Acer Laboratories ALi M6117C embedded microcontroller IC. In addition to the work-alike 386SX processor, the M6117C contains interface capabilities often contained in a separate I/O support IC.

2.1.2 Critical Specifications for the CPU Card

The critical specifications for the CPU card are minimal. The PC104 architecture satisfies the requirement of an inspectable, trusted-processor architecture.

2.2 Ampro, CoreModule 3SXi – 386SX Processor Card

The following sections provide information regarding the Ampro CoreModule 3SXi card.

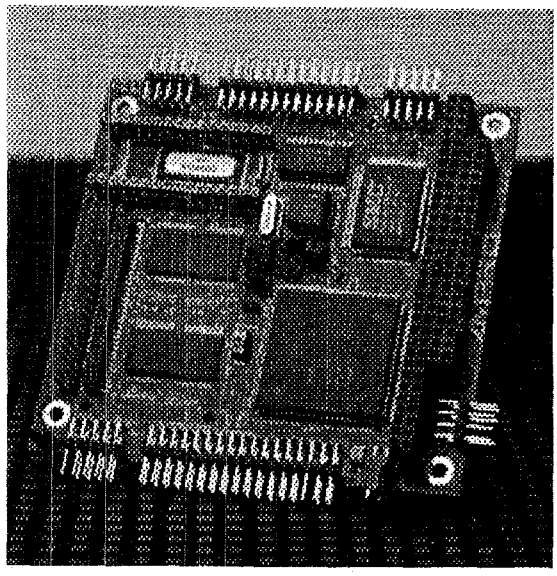
2.2.1 Specifications for the Ampro CoreModule 3SXi Card

The following specification sheet was downloaded from the Ampro Internet site at:

http://www.ampro.com/products/coremod/cm3sxi2.pdf

CoreModule/3SXi

PC/AT motherboard functions in a PC/104 compliant form factor



Features

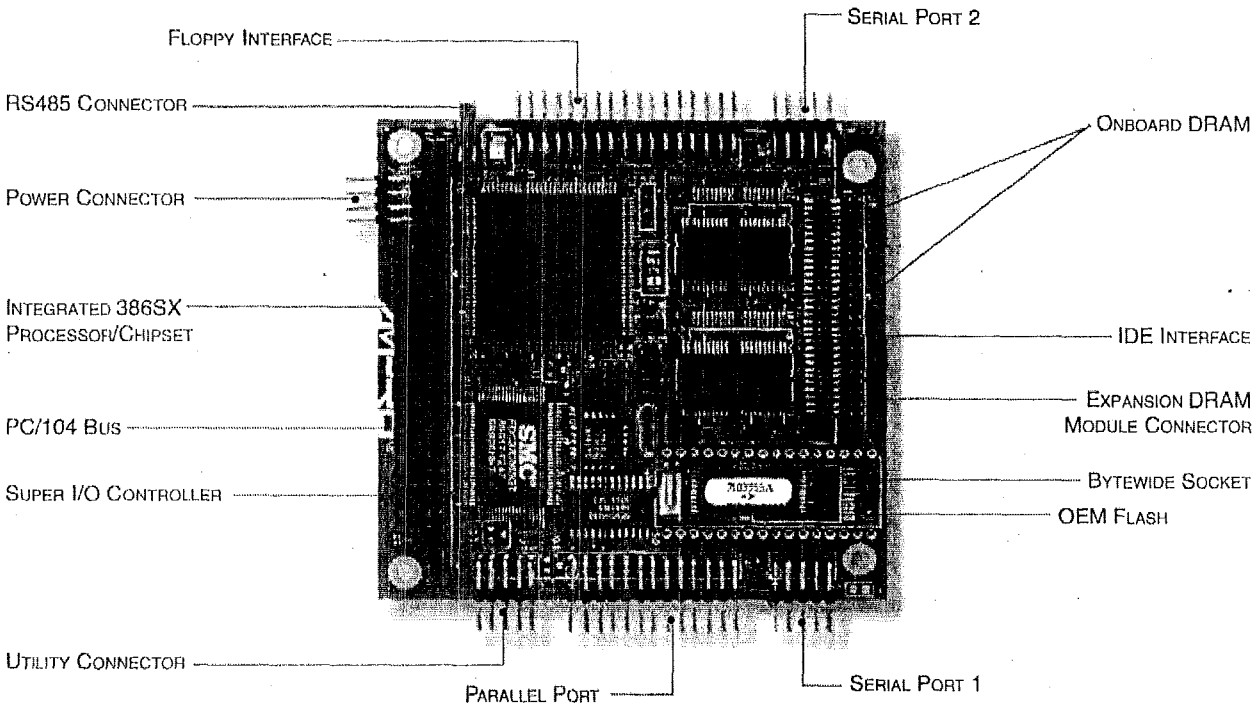
- 25MHz 386SX-compatible processor
- UP to 8M bytes onboard DRAM
- Floppy and IDE disk controllers
- Dual FIFO-buffered serial controllers
- One RS232 and one RS232/RS485
- Bi-directional (EPP/ECP) parallel port
- Optimized for embedded applications
- Wide operating temperature
- Enhanced embedded-PC BIOS
- Bootable "Solid State Disk"
- Watchdog timer
- Advanced Power Management
- Low power consumption
- Batteryless boot

More Information

- Download Product Datasheet (140K PDF)
- Download Technical Manual (846K PDF)
- How to Download...
- How to Purchase

CM2-SXi

CoreModule™/3SXi • *Low Cost PC/AT-compatible PC/104™ compliant CPU module*



The CoreModule/3SX*i* provides cost-effective 386SX processing power and PC/AT compatibility in a compact, preconfigured subsystem module. Within just 14 square inches of space, the CoreModule/3SX*i* includes the equivalent functions of a PC/AT motherboard plus several additional expansion cards. Cost-sensitive embedded applications that formerly required chip-based custom designs can now benefit from an off-the-shelf module powered by a 25 MHz 386SX-compatible CPU, along with hardware and software standards like PC/AT and MS-DOS compatibility.

The CoreModule/3SX*i* is designed to meet the demands of embedded systems through its extremely compact design, low power consumption, +5V-only operation, wide operating temperature range, and high reliability.

CONFIGURATION FLEXIBILITY

The CoreModule/3SX*i* can be used as a macrocomponent, plugged into a proprietary application board, or it can be combined with PC/104-compatible expansion products to form compact, highly integrated control subsystems. Multiple modules can be stacked together without the cost and space penalties of additional mounting hardware.

Ampro Computers, Inc. ▪ 4757 Hellyer Ave., San Jose, CA 95138 ▪ Tel. (408)360-0200 ▪ Fax (408)360-0222 ▪ sales@ampro.com

CM2-5Xi

SPECIFICATIONS

CPU
- 386SX-compatible, 25 MHz internal clock rate

MEMORY
- 2 or 4 Mbytes directly surface-mounted. Additional 4 Mbytes via plug-in DRAM module

SYSTEM CONTROLLERS
- 7 DMA channels (8237 equivalent)
- 14 interrupt channels (8259 equivalent)
- 3 programmable counter/timers (8254 equivalent)

KEYBOARD
- PC/AT-compatible keyboard port
- Speaker port with 0.1 watt drive

REAL TIME CLOCK
- CMOS RAM (MC146818 equivalent); requires external 3.0 - 3.6V battery (Tadiran TL-5242/W or equivalent)

BIOS
- Award BIOS with Ampro embedded-PC enhancements

SERIAL
- Two RS232 serial ports with full handshaking
- One port is jumper-configurable for RS232 or RS485
- Both ports implemented using 16C550 equivalent with 16 byte data FIFOs

PARALLEL
- EPP/ECP compatible bidirectional parallel printer port

IDE
- Support for 1 or 2 IDE hard disk drives
- Low profile 44 pin compact 2mm connector

FLOPPY
- Supports 1 or 2 drives

BYTEWIDE SOCKET
- Usable with 32K - 1 Mbyte EPROMs, 32K - 512K Flash EPROMs, 32K - 512K SRAMs, or 32K - 512K NOVRA
- SRAM backup using off-board battery
- Configurable as 64K, or 128 Kbyte window, addressed in the range D0000-EFFFFh
- Usable with DiskOnChip2000™ read/write Flash SSD device

OEM FLASH OPTION
- Onboard 1 Mbyte OEM Flash
- 960 Kbytes available for OEM use (balance used by system BIOS)
- Configurable as 64 Kbyte window, addressed in the range D0000-DFFFFh or E0000-EFFFFh
- SSD 5.31 Support Software converts into an in-system programmable, read-only SSD devices
- OEM Flash TFFS software converts into a full read/write SSD drive (not usable simultaneously with DiskOnChip2000™)

CONFIG EEPROM
- 2K bit configuration EEPROM, with 512 bits for OEM use
- Supports battery-free boot capability

WATCHDOG TIMER
- Utilizes real-time clock alarm function
- Timeout triggers hardware reset or non-maskable interrupt

SIZE
- 3.6 x 3.8 x 0.9 in. (90 x 96 x 23mm); PC/104 compliant form factor
  (Includes stackthrough pins. Please refer to PC/104 specification for stacking and other dimensions.)

BUS
- 16-bit PC/104 bus

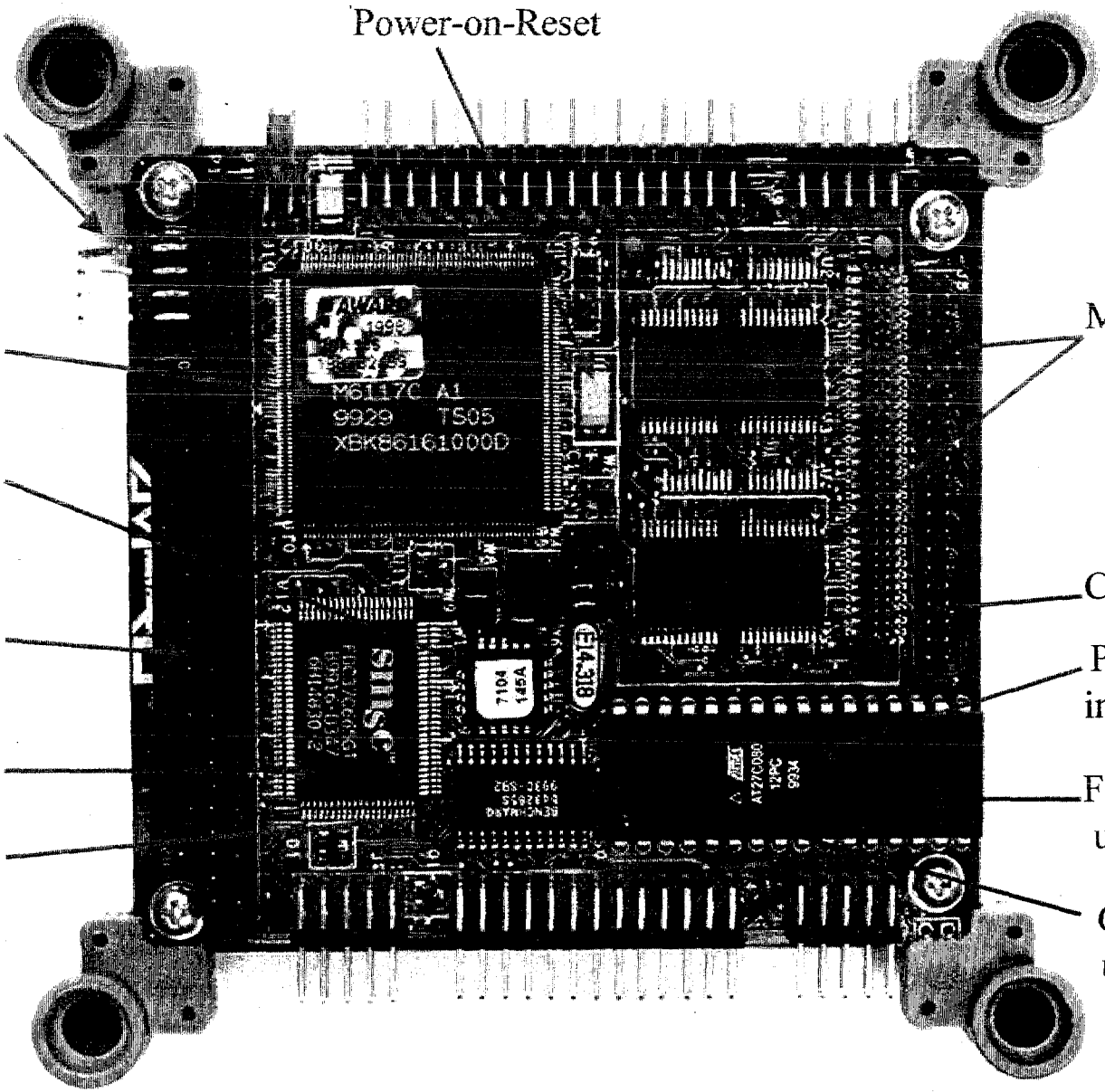
POWER
- Requirements (typical 25 MHz with 4 Mbytes RAM): +5V ±5%
  — 500 mA active / 160 mA sleep

ENVIRONMENTAL
- Operating temperature: 0° to 70° C standard; –40° to +85° C extended (special order)
- 5% to 95% relative humidity, non-condensing
- Storage temperature: –55° to +85° C
- Weight: 3.0 oz. (85 gm)

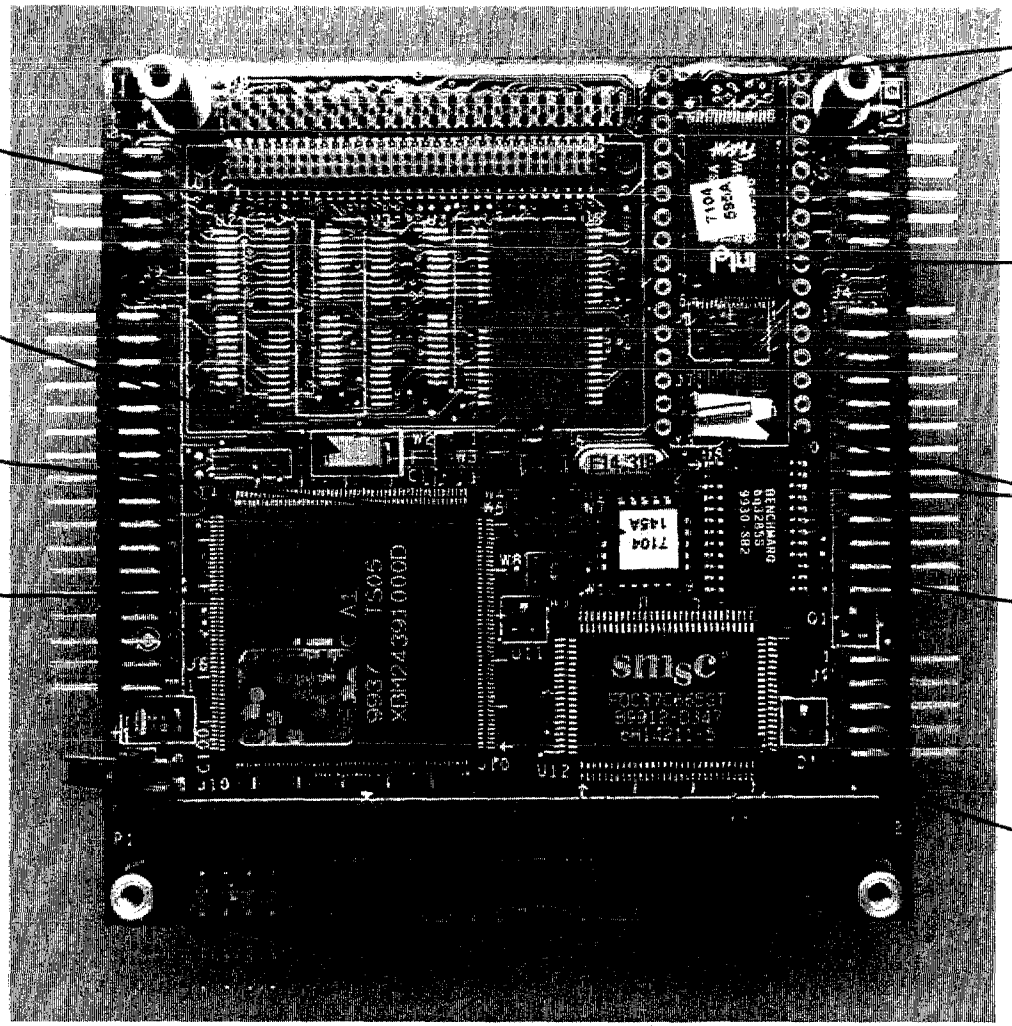
NOTE: Contact Ampro regarding custom configurations and special order options.

THE EMBEDDED SOLUTIONS PROVIDER™

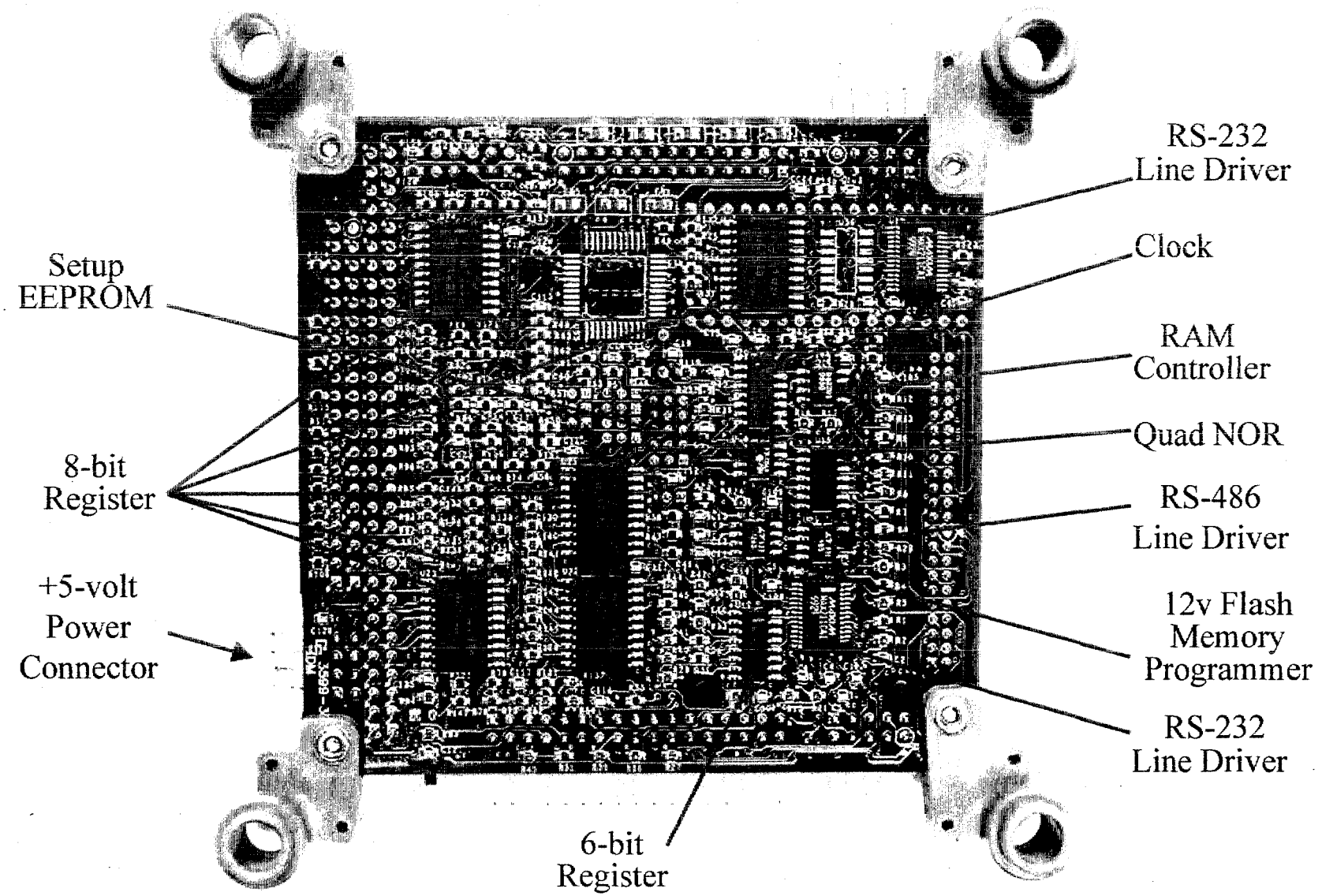
AMPRO 3SXi – CPU Card – Top-View Photograph



AMPRO 3SXi TOP-VIEW PHOTOGRAPH



AMPRO 3SXi – CPU Card – Bottom-View Photograph



2.2.3 X-Ray of the Ampro CoreModule 3SXi Card

The section contains:

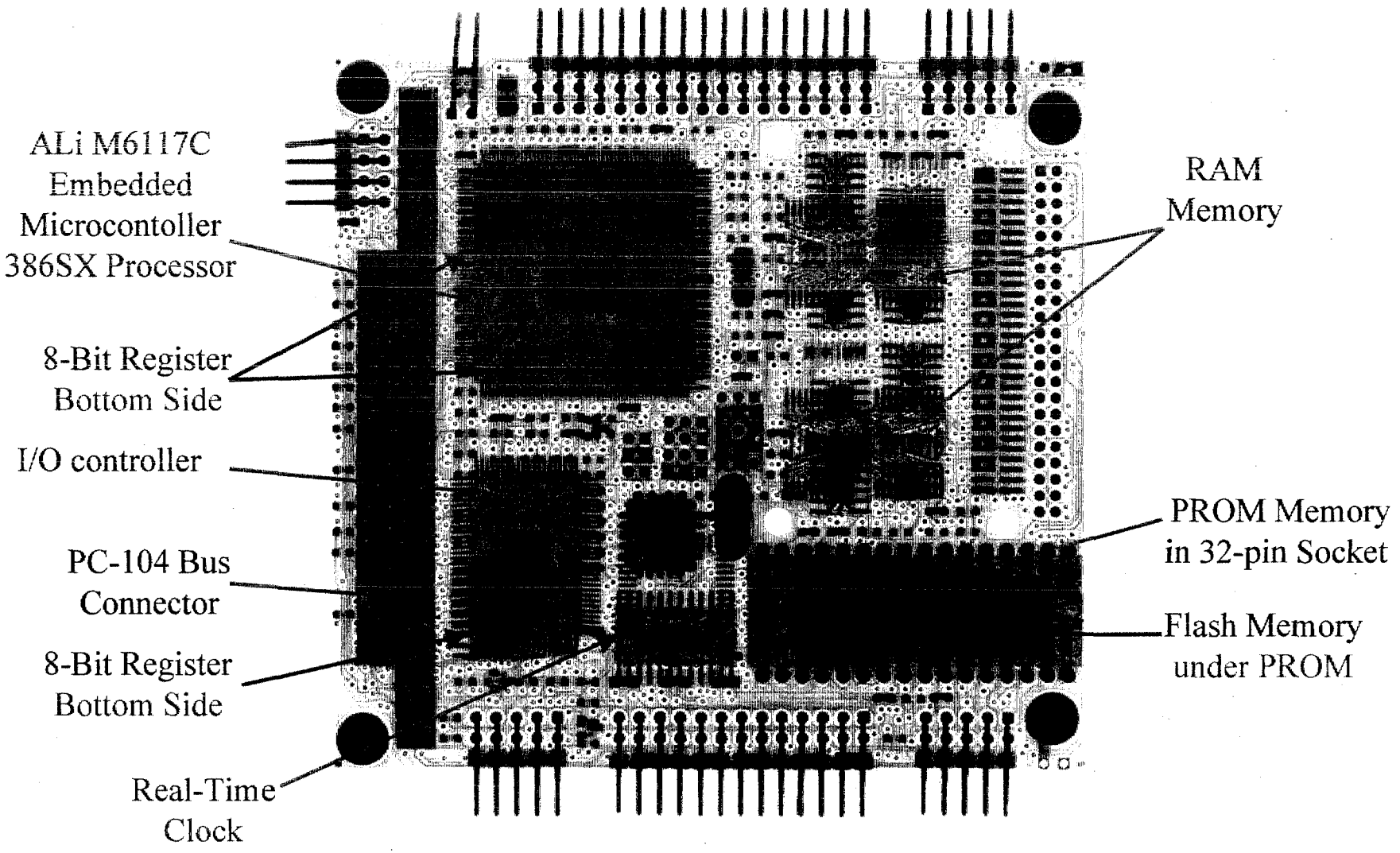
- Labeled top-view x-ray of the Ampro CoreModule 3SXi Card.

The x-ray can be readily compared to the top-view photograph in section 2.2.2. Note that ICs are mounted on both sides of this printed circuit (PC) board. Thus, the large die in the center of the ALi M6117C, which is surface mounted on the topside, can be seen surrounded by three ICs, which are surface mounted on the bottom-side of the PC board.

The x-ray of the region of the socketed PROM IC is more complex. The PROM, which is inserted in the 32-pin socket, is above the Flash memory, which is surface mounted to the topside of the PC board, and above a line driver, which is surface mounted to the bottom side of the PC board. In addition, the PC board has several layers of traces and ground planes.

AMPRO 3SXi – CPU Card – Top-View X-ray

Note: ICs are mounted on both sides of the PC board



2.2.4 Jumper Settings for the Ampro CoreModule 3SXi Card

Note locations of the jumpers is shown in the 3SXi technical manual on page 2-3 in Figure 2-1, which is reproduced below. The W3 jumpers are illustrated on manual pages 23-25.

Table with Jumper Values Used

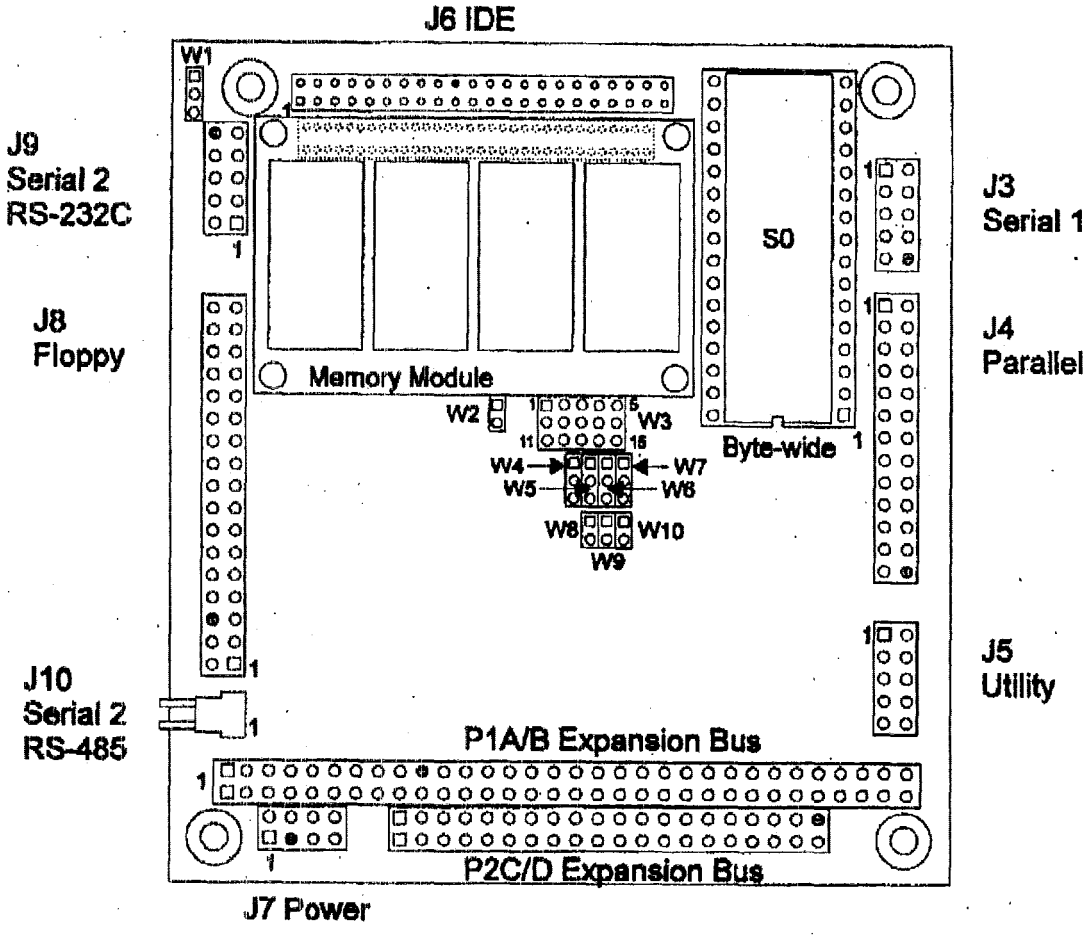
Jumper	Function	Default	Used	Description
W1	RS-232C/RS-485 Select	1=2	1=2	1=2 COM2 uses RS-232C 2=3 COM2 uses RS-485
W2	BIOS/OEM Flash programming power enable	Off	Off	On = connects V_{pp} for Flash EPROM programming
W3	Byte-Wide Socket Configuration	3=4 6=7 8=13 9=14 10=15	2=7 3=4 9=14 10=15 12=13	15-pins see page 2-23 for EPROM See diagram below 27C080 = 1-Mbyte EPROM used
W4	Watchdog Timer Output Selection [if enabled in SETUP]	Off	Off	1=2 = Hardware Reset 2=3 = I/O Channel Check (NMI) open = IRQ8 turns off interrupt
W5	DMA ACK1/ACK3 For Parallel Port = LPT	Off	Off	1=2 selects DMA channel 1 2=3 selects DMA channel 3
W6	DMA REQ1/REQ3 For Parallel Port = LPT	Off	Off	1=2 selects DMA channel 1 2=3 selects DMA channel 3
W7	Byte-Wide Backup Power Select	1=2	2=3	1=2 enable external battery for SO 2=3 connects device directly to V_{cc}
W8	Byte-Wide Battery Backup Power	Off	Off	On = enables battery backup for memory in SO
W9	BIOS/Byte-Wide Swap	On	On	Off = enables access of a system BIOS from SO
W10	RS-485 Termination	Off	Off	On = 100 ohm terminator

W3 **W7** **W8** **Default = 256-kbyte Flash EPROM**

```
01  02  03--04  05        1    1          = 28F020

06--07  08  09  10        2    2
            |   |         |
11  12  13  14  15        3
```

W3 **W7** **W8** **Used = 1-Mbyte EPROM**

```
01  02  03--04  05        1    1          = 27C080
        |
06  07  08  09  10        2    2
                |   |     |
11  12--13  14  15        3
```



(Key pins on connectors are shaded.)

Figure 2–1. Connector and Jumper Locations

2.2.5 Parameter Settings Set by Software for the Ampro CoreModule 3SXi Card

The setup program can be accessed via a CTRL-ALT-ESC key combination when the computer is first powered up. Actual settings were determined with the sample of the configured system.

Setup Page 1 – Standard CMOS Setup

Item	Default	Setting
1st Floppy:	1.4M	1.4M
2nd Floppy:	1/2M	None
ATA/IDE Disk 1:	17	None
ATA/IDE Disk 2:	None	None
Video:	EGA/VGA	EGA/VGA
Base Memory:	640	640
Extended Memory:	1024	3072
Error Halt:	No Halt on Any Error	No Halt on Any Error
Video Shadow RAM:	Enabled	Enabled
System POST: (Power on Self Test)	Normal	Normal

Setup Page 2 – Options/Peripheral Configuration

Item	Default	Setting
CoreModule Extended BIOS:	Enabled	Enabled
Advanced Power Mgmt BIOS:	Enabled	Enabled
Serial Port 1:	Enabled	Disabled
Serial Port 2:	Enabled	Disabled
Parallel Port:	Enabled	Enabled
Mode:	SPP	SPP
Floppy Interface:	Enabled	Enabled
IDE Interface:	Enabled	Enabled
Mono/Color Jumper: ←	Color	
PC/104 Bus Speed	Normal	Normal
Socket SO:	64K @D0000 hex	64K @D0000 hex
OEM Flash S1:	64K @E0000 hex	Disabled
Default Socket:	SO	SO
Video State:	Enabled	Enabled
Blank POST Test:	Enabled	Disabled
Serial boot loader:	Disabled	Disabled
Watchdog Timer:	Disabled	Disabled
Hot key setup:	Enabled	Enabled

← Referred to in documentation but was not displayed by the software

Setup Page 3 – SCSI Hard Disk

Item	Default	Setting
SCSI/BIOS Hard Disk Service	Enabled	Disabled
SCSI Initiator ID	7	0
SCSI Disk I/O Retries	10	0
SCSI DISK MAP		
SCSI Disk 1	Not Active	Not Active
SCSI Disk 2	Not Active	Not Active
SCSI Disk 3	Not Active	Not Active
SCSI Disk 4	Not Active	Not Active
SCSI Disk 5	Not Active	Not Active
SCSI Disk 6	Not Active	Not Active
SCSI Disk 7	Not Active	Not Active
DOS DISK MAP		
Default Boot Device	Floppy Drive	Floppy Drive
1st Hard Disk	IDE Disk 1	AT Bus HDC, Disk 1
2nd Hard Disk	Not Active	Not Active
3rd Hard Disk	Not Active	Not Active
4th Hard Disk	Not Active	Not Active
5th Hard Disk	Not Active	Not Active
6th Hard Disk	Not Active	Not Active
7th Hard Disk	Not Active	Not Active
8th Hard Disk	Not Active	Not Active

Setup Page 4 – Serial Console

Item	Default	Setting
Console Output Device:	Video	Video
Console Input Device:	Keyboard	Keyboard
Serial Console OUTput Setup	No entry	No entry
Data Length	No entry	No entry
Stop Bits	No entry	No entry
Parity	No entry	No entry
Baud	No entry	No entry
Delete from Com Port Table	No entry	No entry
Console Output Handshake	No entry	No entry
Serial Console INput Setup	No entry	No entry
Data Length	No entry	No entry
Stop Bits	No entry	No entry
Parity	No entry	No entry
Baud	No entry	No entry
Delete from Com Port Table	No entry	No entry

2.2.6 Schematics for the Ampro CoreModule 3SXi Card

The schematics for the Ampro 3SXi card require a non-disclosure agreement with Ampro.

2.2.7 Parts List for the Ampro CoreModule 3SXi Card

The Ampro 3SXi card is a 6-layer PCB using the latest surface mount technology. The following table lists the integrated circuits (ICs) mounted on the card. The "Location" column is a reference used in the schematics and on the printed circuit card.

Table of the ICs Mounted on the 3SXi Card

Location	Vendor	Part Number	Description
SO	Atmel	27C080-12-PC or 27C080-12-DC	1Mx8 (1-Mbyte) PROM Memory inserted in 32-pin Socket for PROM
U3 + U5	Toshiba	TC5118160CFT-60	1Mx16 (2-Mbyte) DRAM Memory
U6	Intel	28F008SA	1Mx8 (1-Mbyte) Flash Memory
U8	Benchmarq	BQ3285S	Real-Time Clock (RTC) with 114 Bytes of Setup CMOS Memory
U9		16V8	Programmable Electrically Erasable Logic (PEEL) IC
U10	Acer Labs	ALi M6117C	Embedded Microcontroller (386SX)
U11	MAXIM	MAX809	Power-on-Reset IC
U12	SMSC	FDC37C666GT	I/O Controller
U13 -U14	SIPEX	SP211CA	RS-232 Line Driver
U15	Fairchild	74ACT02	Quad 2-Input NOR Gate
U16	MAXIM	MXD1210	Non-Volatile RAM Controller
U17	Harris	74HCT174	Hex D-Type Flip-Flop
U18	Linear Technologies	LTC1262	Flash Memory Programming Supply
U19	Motorola	93LC56X	2-kbit Setup EEPROM
U20	ICS	AV9154-16	Clock IC
U21-U23	Fairchild	74HCT245	8-bit Register
U24	Motorola	74HCT244A	8-bit Register
U25	Linear Technologies	LTC485	RS486 Line Driver
U35	Fairchild	74HCT245	8-bit Register
Y1	Missing ? KDS	32768 or ? DT-26	Clock crystal for internal RTC – Missing because internal RTC is not used
Y2	Unknown	E14.318	14.318-MHz Clock Crystal for Benchmarq RTC
Y3	Unknown ? KDS	No label 32768 or ? DT-26	32.768-kHz Quartz Clock Crystal for Clock IC at U20

Section 9 at the end of the computational block documentation provides a set of Internet addresses, at which specifications and data sheets are publicly available. Data sheets for the

principle components (i.e., ALi M6117C=processor, AT27C080=PROM, and 28F008SA=Flash memory) are included in sections 2.3 through 2.5 at the end of this CPU section for clarity.

Part numbers were determined from a combination of the schematics and viewing an actual 3SXi board. Vendor information was not included on the schematics and was taken from a combination of symbols on the actual parts and Internet search results.

2.2.8 Memory Details for the Ampro CoreModule 3SXi Card

The following table shows details of the memory usage as gleaned from the 3SXi manual and other Internet sources. Additional memory information (e.g., a memory map output from MEM commands) is found in section 6.1.7. The 3SXi in the computational block has two 2-Mbyte DRAMs, a large flash memory, and a large PROM installed. These memory options are denoted by arrows, ←, in the table.

Table Showing Available Memory within 3SXi and Potential Memory Map Locations

Address Range Hex	**Size** bytes	**Function or Contents**
DRAM	1Mx16	2-Mbyte main memory = Toshiba, TC5118160CFT-60
200000 – 3FFFFF	2048k	Extended memory – 2-Mbyte of DRAM in second DRAM ←
100000 – 1FFFFF	1024k	Extended memory – 1-Mbyte of DRAM
0F0000 – 0FFFFF	64k	Ampro ROM-BIOS – possible shadow copy in RAM
0E0000 – 0EFFFF	64k	Possible memory window into extended memory
0D0000 – 0DFFFF	64k	Flash or Socket memory window into 1 Mbyte of memory
0C0000 – 0CFFFF	64k	Video BIOS – possible shadow copy in RAM for speed
0A0000 – 0BFFFF	128k	Video Screen RAM window into 512k bytes of video DRAM
000000 – 09FFFF	640k	Possible 640-kbytes of DRAM available for programs
		See MEM memory map in section 6.1.7 for details
FLASH – A ←	1Mx8	1-Mbyte BIOS & OEM flash memory = Intel, 28F008SA
010000 – 0FFFFF	960k	Unused
000000 – 00FFFF	64k	Ampro BIOS
FLASH – B	1Mx8	128-kbyte BIOS & OEM flash memory = Intel, 28F010
010000 – 01FFFF	64k	Unused
000000 – 00FFFF	64k	Ampro BIOS
PROM – A ←	1Mx8	Byte-Wide Socket =1-Mbyte memory = Atmel, AT27C080
000000 – 0FFFFF	1024k	User files
PROM – B	512kx8	Byte-Wide Socket = 512-kbyte memory = 27F040
000000 – 07FFFF	512k	User files
Setup EEPROM	256	2-kbit EEPROM = Motorola, 93LC56X
080 – 0FF	128	**Documentation is unclear**
040 – 07F	64	512 bits for OEM use
000 – 040	64	Setup info

Address Range Hex	**Size** bytes	**Function or Contents**
Setup NVSRAM	128	Real-time clock w/ 114x8 NVSRAM = Benchmarq BQ3285S
00E -- 07F	114	Setup storage resisters
000 -- 00D	14	Clock & control status resisters
		Note: Requires battery backup
Setup internal RAM	128	Internal real-time clock in M6117 w/ 114x8 CMOS RAM
00E -- 07F	114	Setup storage resisters
000 -- 00D	14	Clock & control status resisters
		Note: Requires battery backup & documentation unclear

Additional memory information (e.g., a memory map output from MEM commands) is found in section 6.1.7.

2.2.9 Voltage and Current Requirements for the Ampro CoreModule 3SXi Card

The following table provides input power connection information for the 3SXi card. J7 is an 8-pin input power connector to the card. P1 is the AT expansion bus connector that provides power to additional PC104 cards in the stack.

Table Providing Power Connector (J7) Information

J7 Pin	**Connection**	**PC-104 Standard**	**P1-B Pin**	**Comments**
J7-1	Ground	Ground	B1	
J7-2	+5 VDC	+5 VDC	B3	450 MA at +5V ±5% for 25 MHz operation 240 MA at +5V ±5% for standby power mode
J7-3	Missing as key	Key		
J7-4	NC	+12 VDC	B9	
J7-5	NC	-5 VDC	B5	
J7-6	NC	-12 VDC	B7	
J7-7	NC	Ground	B31&32	
J7-8	NC	+5 VDC	B29	
				±9 volts for RS-232 ports generated onboard
				+12 volts for flash memory programming generated onboard

2.2.10 Supplementary Narrative Describing Functional Blocks and Implementation for the Ampro CoreModule 3SXi Card

The Ampro 3SXi card uses the ALi M6117C embedded microcontroller as the basic component. The processor has internal configuration memory that is explained in the data sheet provided in section 2.3.3.

The processor can access several banks of memory with the bank switching handled by the ALi M6117C processor, the MXD1210 Non-Volatile RAM controller, and the 16V8 programmable electrically erasable logic (PEEL) IC . The processor can address a flash memory IC, which contains the system BIOS and has 960-kbytes available for other use by the original equipment manufacturer (EOM) or a sophisticated user. The processor can address a once programmable read-only memory (PROM) or erasable programmable read-only memory (EPROM) mounted in a 32-pin byte-wide socket (SO). In this case that socket contains a 1-Mbyte PROM (AT27C080) configured as a disk file containing the operating system and application software files. The processor can also address the main memory, which consists of 2-Mbyte dynamic random-access memory (DRAM) ICs each configured as 1Mx16 (1M of 16-bit words). The 3SXi card may contain either one or two 2-Mbyte DRAM ICs (TC5118160CFT-60). In this case, two DRAM-ICs are installed for 4-Mbytes of main memory.

An external real-time clock (RTC), Benchmarq BQ328S, is used rather than the RTC internal to the M6117C. A 14.318-MHz clock crystal of unknown manufacture (no label is visible on the component) drives the RTC. The setup memory in the RTC can be battery backed, but the battery is removed for this application. A 2-kbit setup electronically erasable programmable read-only memory (EEPROM) supplies the setup information at power on. The EOM or a sophisticated user can use at least 512 bits (possibly 1536 bits) of this EEPROM for other parameters.

A SMSC FDC37C666GT I/O controller handles the I/O. This Controller handles I/O for and IDE bus, a Floppy interface, two serial ports, and a parallel port. The serial ports each use a SIPEX SP211CA RS-232 line driver/receiver, which internally generates ±9 volts for the RS-232 line.

A Power-on-Reset IC senses the power condition. This provides a power_good signal to the processor, power_ok to the real-time clock and reset signal to the utility connector. The processor handles the reset for the I/O components and the PC-104 bus.

Two IC pads are empty on the board. They were intended for a keyboard controller and a hex inverter for the keyboard signals.

2.2.11 Test Point Information for the Ampro CoreModule 3SXi Card

The following table contains illustrative examples of test points determined from the schematics and the individual IC data sheets. More test points can be readily added for a more thorough joint inspection.

Table of Recommended Test Points for 3SXi card

Location IC-Pin	**Test Object**	**Expected Result**	**Signal Name**	**Comments**
J7-1	Input Power	Ground		Input Power Connector
J7-2	Input Power	+5VDC ±5%		Input Power Connector
U10-39	CPU M6117	25 MHz	CPU BCLK2	Input clock for the CPU
U10-160	CPU M6117	25 MHz	CPU ATCLK	Output CPU clock for ISA bus
U10-41	CPU M6117	25 MHz	CPU BCLK1	1X CPU bus clock = BCLK2
U10-117	CPU M6117	14.318 MHz	CPU OSC14M	12X Clock for 8254 timer
U10-120	CPU M6117	7.159 MHz	CPU CK7M	14.318/2 MHz for keyboard
U10-196	CPU M6117	L	I/O CHKJ	H=ISA parity error
U10-163	CPU M6117		RefreshJ*	DRAM refresh rate
U10-113	CPU M6117	H	PWG	H=Good system power
U10-1	CPU M6117	+5VDC ±5%	CPU VCC	+5 VDC for CPU
U10-8	CPU M6117	Ground	CPU VSS	Ground for CPU
U8-20	RTC	3V	RTC BC	3V battery backup
U8-2	RTC	32.768 MHz	RTC X1	Crystal input to RTC
U18-6	Flash Supply	+12V	Vpp	+12V supply for write-to-Flash
U6-11	Flash Mem	+5V	Flash Vpp	+12V → can write to Flash Mem
U6-9	Flash Mem	+5V after start	CE*	L=Chip enable (only copy BIOS once during startup)
U6-37	Flash Mem	+5V after start	OE*	L=Output enable
U6-38	Flash Mem	+5V	WE*	L=Write enable (never write)
SO-22	PROM Mem	0 & 5 volts	CE*	L=Chip enable
SO-24	PROM Mem	0 & 5 volts	OE*/VPP	L=Output enable 13v=Write Should never be writing to SO Should be reading from SO

2.2.12 Technical Manual for the Ampro CoreModule 3SXi Card

The following technical manual was downloaded from the Ampro Internet site at:

http://www.ampro.com/techman/coremodule/5001131e.pdf.

CoreModule™/3SX*i*

Technical Manual

P/N: 5001131

Revision: D

Ampro Computers, Incorporated
4757 Hellyer Avenue ■ San Jose, CA 95138
Tel (408) 360-0200 ■ FAX (408) 360-0220
WEBSITE: www.ampro.com

NOTICE

DISCLAIMER

Ampro Computers, Incorporated makes no representations or warranties with respect to the contents of this manual or of the associated Ampro software products, and specifically disclaims any implied warranties of merchantability or fitness for any particular purpose. Ampro shall under no circumstances be liable for incidental or consequential damages or related expenses resulting from the use of this product, even if it has been notified of the possibility of such damages. Ampro reserves the right to revise this publication from time to time without obligation to notify any person of such revisions. If errors are found, please contact Ampro at the address listed on the title page of this document.

TRADEMARKS

The Ampro logo is a registered trademark, and Ampro, Little Board, StackPlane, MiniModule, MiniBackplane, and CoreModule are trademarks of Ampro Computers, Inc. All other marks are the property of their respective companies.

TECHNICAL SUPPORT

- **Telephone technical support is available from 8:00 AM to 5:00 PM, Pacific time. The telephone number is 800 966-5200. (Please have the product you wish to discuss at hand when you call.)**
- **E-mail address: techsupport@ampro.com**
- **Web site: http://www.ampro.com**
- **Ampro Technical Support Bulletin Board (BBS): 408 720-1332**

REVISION HISTORY

REVISION	REASON FOR CHANGE	DATE
A	Initial Release	1/97
B	Production Release	7/97
C	Update Mech & Environ Specs	7/97
D	Correct W7 Information	9/98

PREFACE

This manual is for designers of systems based on the Ampro CoreModule/3SX*i* CPU, a PC/AT compatible modular computing engine. This manual contains information on hardware requirements and connections, and details about how to program the device and integrate it with other devices to create an embedded system customized to your requirements.

There are three chapters, organized as follows:

- **Chapter 1—Introduction.** General information pertaining to the CoreModule/3SX*i* CPU, its features, and technical specifications.
- **Chapter 2—Configuration and Installation.** A description of the jumper options, connector pinouts, and hardware-related technical information needed to configure and install the module.
- **Chapter 3—Operation.** A description of software-related system features. Includes instructions on how to use the BIOS SETUP feature to configure your system. Includes descriptions of specialized utilities provided with the Development Kit.

TABLE OF CONTENTS

CHAPTER 1—INTRODUCTION

CHAPTER 2—CONFIGURATION AND INSTALLATION

CHAPTER 3—OPERATION

FIGURES

TABLES

CHAPTER 1

INTRODUCTION

1.1 GENERAL DESCRIPTION

The CoreModule/3SX*i* CPU is an exceptionally high integration, high performance, 386SX-based PC/AT compatible system in the PC/104 form factor. This rugged and high quality single-board system contains all the component subsystems of a PC/AT motherboard plus the equivalent of several PC/AT expansion boards.

Key functions included on the CoreModule/3SX*i* module are CPU, RAM, embedded-PC BIOS, keyboard and speaker interfaces, two serial ports, a multimode IEEE-1284 enhanced parallel port, floppy drive controller and IDE hard disk controller. In addition, the CoreModule/3SX*i* CPU includes a comprehensive set of system extensions and enhancements that are specifically designed for embedded systems. It is designed to meet the size, power consumption, temperature range, quality, and reliability demands of embedded applications.

Among the many embedded-PC enhancements that ensure fail-safe embedded system operation are a watchdog timer and an onboard bootable "solid state disk" (SSD) capability. The unit requires a single +5 Volt power source and offers "green PC" power-saving modes under support of Advanced Power Management (APM) BIOS functions (APM Release 1.1-compliant).

The CoreModule/3SX*i* CPU is particularly well suited to demanding environments such as embedded or portable applications. The flexibility of the CoreModule/3SX*i* CPU makes system design quick and easy. Stack it with Ampro MiniModules™ or other PC/104-compliant expansion modules, use the Ampro MiniBackplane and ordinary plug-in cards, or use it as the computing engine in a fully customized application.

1.2 FEATURES

CPU/Motherboard

The CoreModule/3SX*i* CPU implements a fully PC-compatible motherboard architecture, with an 80386SX CPU running at 25 MHz.

The standard DRAM compliment of the CoreModule/3SX*i* CPU is 2M bytes, soldered on the board. A model with 4M bytes is also available. For DRAM expansion, you can order an Ampro custom memory module which allows you to add an additional 4M bytes.

Serial Ports

The board provides two PC-compatible RS-232C serial ports, implemented using 16C550-type UARTs. These UARTs are equipped with 16-byte FIFO buffers to improve throughput. Baud rates up to 115K baud are supported. Onboard voltage converters provide the RS-232C voltage levels from the +5 volt supply.

The second serial port can be configured for either RS-232C or RS-485. RS-485 uses a bi-directional differential-pair signaling scheme. RS-485 is generally used for a serial bus. Up to 32 nodes can be bussed together, sharing a single twisted-pair cable.

Parallel Port

An enhanced bi-directional parallel port interface conforms to the IEEE-1284 standard. It provides new features attractive to embedded system designers, including increased speed, an internal FIFO buffer, and DMA transfer capability.

Floppy Interface

An onboard floppy disk interface provides access to standard floppy drives. The interface supports up to two floppy drives, 5.25 inch or 3.5 inch, in any combination.

IDE Interface

An onboard IDE interface provides hard disk and CD-ROM drive access. The interface supports up to two drives. The BIOS supports hard drives greater than 528 M bytes through Logical Block Addressing (LBA).

Enhanced Embedded-PC BIOS

One of the most valuable features of the CoreModule/3SX*i* CPU is its enhanced embedded-PC BIOS, which includes an extensive set of functions that meet the unique requirements of embedded system applications. These enhancements include:

- Solid State Disk (SSD) support (see below)
- SCSI services—full SCSI BIOS services are integrated with the module's hard disk support
- Watchdog timer—monitors the boot process and provides a watchdog function call for applications
- Fast boot operation—normal or accelerated POST, selectable by SETUP options
- Configurable POST display—select what will be displayed at boot time
- Fail-safe boot support—intelligently retries boot devices until successful
- Battery-free boot support—saves system SETUP information in non-volatile EEPROM
- Serial console option—lets you use a serial device as a console
- Serial loader option—supports loading boot code from an external serial source
- EEPROM access function—512 bits of EEPROM storage available to user; useful for serialization, copy protection, security, etc.
- OEM customization hooks—can execute custom code prior to system boot via ROM extensions; allows sophisticated system customization without BIOS modification

Modular PC/104 Expansion Bus

The CoreModule/3SX*i* CPU provides a PC/104-compatible expansion bus for additional system functions. This bus, a compact version of the standard PC ISA bus, offers compact, self-stacking, modular expandability. The growing list of PC/104 modules available from Ampro and hundreds of other PC/104 vendors includes such functions as communications interfaces, LAN interfaces, video framegrabbers, digital signal processors (DSPs), data acquisition and control functions, and many specialized interfaces and controllers.

In addition, you can mount the CoreModule/3SX*i* CPU on your own custom application-specific base board using its PC/104 expansion bus interface as a rugged and reliable interconnect. This eliminates the need for you to design a PC engine for your product and facilitates easy upgrades and troubleshooting.

Byte-Wide Socket and Solid State Disk (SSD)

An important feature of the CoreModule/3SX*i* CPU is its byte-wide memory socket, in which you can install a bootable "solid state disk" (SSD) or other embedded application software.

An SSD substitutes EPROMs, Flash EPROMs, battery-backed SRAMs, or Non-Volatile RAM (NOVRAM) modules for conventional rotating-media drives. Using Ampro's SSD/DOS Support Software, any DOS-based application, including the operating system, utilities, drivers, and application programs, can be run from SSD without modification. SSD operation is also supported by a growing number of real-time operating systems.

The module's 32-pin byte-wide socket is configurable for nearly every available 28-pin and 32-pin byte-wide memory device. The socket supports all varieties of devices, CMOS SRAM, SRAM non-volatile modules, EPROM, and Flash EPROM. It accommodates devices from 32K bytes to 1M byte and larger using a simple memory-paging scheme implemented with custom BIOS calls.

To support the use of 12 volt Flash memory devices in the byte-wide socket, the board is equipped with an onboard 5 volt to 12 volt converter.

OEM Flash Memory

The system BIOS is stored in a portion of an onboard Flash memory device. The remaining part of the Flash memory device can be used by OEMs for embedded software. Ampro provides a utility for programming this memory. The onboard Flash memory device is accessed as a second byte-wide memory device, using the same custom BIOS calls provided in Ampro's extended BIOS that are used to access the byte-wide memory socket.

Two models of the CoreModule/3SX*i* are available. One has a 128K byte onboard Flash memory device, 64K bytes of which are used for the ROM BIOS, and the remaining 64K bytes available for OEM Flash memory. Another version comes equipped with a 1M byte Flash memory device, with all but 64K bytes available for OEM use. This larger version permits using the OEM Flash memory with Ampro's SSD/DOS to create a read-only solid state disk for the operating system and application programs. Or, using OEM Flash True Flash File system (TFFS), you can create a solid state disk with full read/write capability.

1.3 ENHANCED RELIABILITY

Reliability is especially important in embedded computer systems. Ampro, specializing in embedded system computers and peripherals, knows that embedded systems must be able to run reliably in rugged, hostile, and mission-critical environments without operator intervention. Over the years, Ampro has evolved system designs and a comprehensive testing program to ensure a reliable and stable system for harsh and demanding applications. These include:

ISO 9001 Manufacturing. Ampro is a certified ISO 9001 vendor.

Regulatory testing. Knowing that many embedded systems must qualify under ESD, EMC emissions, and susceptibility testing, Ampro designs boards with careful attention to EMI issues. Boards are tested in standard enclosures to ensure that they can pass such tests. Tests include CE MARK directives EMC EN55022 and EN55011, ESD EN 61000-4-4, RF susceptibility ENV 50140, EFT EN 61000-4-5, and conducted emissions at US voltages per FCC Subpart 15.

Wide-range temperature testing. Ampro Engineering qualifies all of its designs by extensive thermal and voltage margin testing.

Shock and Vibration Testing. Boards intended for use in harsh environments are tested for shock and vibration durability to MIL-STD 202F, Method 214A, Table 214I, Condition D at 5 minutes per axis for random vibration, and to MIL-STD 202F, Method 213B, Table 213-1, Condition A for resistance to mechanical shock. (Contact your Ampro sales representative to obtain *Shock and Random Vibration Test Report for the* CoreModule/3SX*i* CPU for details.)

1.4 SOFTWARE

The vast array of commercial and public-domain software for the IBM PC and PC/AT is usable in CoreModule/3SX*i* CPU based systems. You can use the most popular software development tools (editors, compilers, debuggers, etc.) for developing code for your application. With this software and the standard Ampro-supplied utilities and drivers, you can quickly tailor a system to your needs.

Use the board's SETUP function for all system configuration. SETUP can be invoked using a "hot-key" combination (CTRL-ALT-ESC) or from the DOS command line using a utility program, SETUP.COM, available on the Common Utilities diskette. Table 1–1 summarizes the configuration parameters you can modify using SETUP.

Table 1–1. Summary of SETUP Options

- Date and time in the battery-backed real-time clock
- Floppy drive quantity and type
- IDE Hard disk drive quantity and type
- Video controller type (for an external video controller)
- Serial port enable/disable
- Parallel port enable/disable/mode
- Byte-wide socket address and size
- OEM Flash memory address and size
- Serial console option
- Video BIOS Shadow RAM enable
- SCSI disk drive parameters (using Ampro SCSI adapter)
- DOS hard disk map
- Choice of default boot drive (hard disk or floppy)
- Enable/Disable *hot-key* access to SETUP
- Watchdog timer startup time-out
- Serial loader enable/disable/port selection
- POST speed options
- POST screen display and blanking options

SETUP information is stored in both the battery-backed CMOS RAM-portion of the real-time clock, and in a configuration EEPROM. For a complete discussion of SETUP, see Chapter 3.

1.5 DESIGNING COREMODULE SYSTEMS

The CoreModule/3SX*i* CPU affords a great deal of flexibility in system design. You can build a system using only the CoreModule, serial or parallel devices for input/output, and a Solid State Disk drive in the byte-wide socket or OEM Flash device.

Self-stacking Modules—The simplest way to expand a CoreModule system is with self-stacking Ampro MiniModules. MiniModules are available for a wide variety of functions There are MiniModules that provide video interfaces, from monochrome through Super VGA, including flat panel displays. Other MiniModules provide additional serial and parallel ports, Ethernet LAN adapter, PCMCIA interface, sound card, and other functions. You can stack the MiniModules with the CoreModule and avoid the need for bus cables, card cages, and backplanes.

MiniModules mount directly on the PC/104 bus connector of the CoreModule. PC/104-compliant modules can be stacked with an inter-board spacing of ~0.66 inches. Thus, a 3-module system fits in a 3.6 inch by 3.8 inch by 2.4 inch space. A complete description of self-stacking options with various Ampro MiniModules and other PC/104-compatible modules can be found in Ampro Application Note AAN-9402, available from Ampro.

MiniBackplane Systems—You can also use a CoreModule/3SX*i* system with an Ampro MiniBackplane and standard PC/AT plug-in cards. Using the MiniBackplane, two standard cards can be added.

OEM Motherboard—You can add the CoreModule/3SX*i* CPU as the computing engine to a dedicated OEM logic or interface board. Compatible connectors can be arranged on the OEM "motherboard", and the CoreModule/3SX*i* CPU can be mounted directly on these connectors. Not only does this eliminate the need for OEMs to design their own CPU subsystem, but it also allows for substitution of new models as technology changes, without requiring an expensive redesign of the motherboard.

1.5.1 CoreModule Development Chassis

Whatever your CoreModule application, there will always be a need for an engineering development cycle. To help developers quickly assemble an embedded system, Ampro offers the CoreModule Development Chassis. It includes a power supply, floppy disk drive, hard disk, a StackPlane/AT (for mounting the CoreModule and additional MiniModules or other PC/104-compliant modules), speaker, I/O connectors, and a two-slot PC backplane.

The Development chassis provides a "known good" environment for your development work. You can install the CoreModule/3SX*i* CPU, MiniModules or conventional expansion boards, keyboards, monitors, and I/O devices to quickly create a platform for your hardware and software engineering needs. Often, development chassis are used in repair and support facilities as well, and on the production floor for system test. Contact your Ampro sales representative for information.

1.6 COREMODULE/3SX*i* CPU SPECIFICATIONS

The following section provides technical specifications for the CoreModule/3SX*i* CPU.

1.6.1 CPU/Motherboard

- CPU: 25 MHz 386SX

- System RAM: 2M or 4M bytes DRAM (soldered on the board)
  - Provision for an Ampro custom 4M byte memory module
  - Supports up to 8M bytes of DRAM, 4M bytes onboard plus one 4M byte memory module.
- Shadow RAM support provides fast system and video BIOS execution
- 14 interrupt channels (8259-equivalent) (IRQ12 is not supported.)
- 7 DMA channels (8237-equivalent)
- 3 programmable counter/timers (8254-equivalent)
- Standard PC/AT keyboard port
- Standard PC speaker port with .1 watt output drive
- Battery-backed real-time clock and CMOS RAM, with support for battery-free operation
- Award ROM BIOS with Ampro embedded-system extensions

1.6.2 Onboard Peripherals

This section describes standard peripherals found on every CoreModule/3SX*i* CPU.

- Two buffered serial ports with full handshaking
  - Implemented with 16550-equivalent controllers with built-in 16-byte FIFO buffers
  - Onboard generation of RS-232C signal levels
  - The second port supports RS-485
  - Logged as COM1 and COM2 by DOS. May be disabled using SETUP.
- Multimode Parallel Port
  - Superset of standard LPT printer port
  - Bi-directional data lines
  - IEEE-1284 (EPP/ECP) compliant
  - Standard hardware supports all four IEEE-1284 protocol modes
  - Internal 16-byte FIFO buffer
  - DMA option for data transfers

- Floppy Disk Controller
  - Supports one or two drives
  - Reliable digital phase-locked loop circuit
  - Supports all standard PC/AT formats: 360K, 1.2M, 720K, 1.44M
- IDE Disk Controller
  - Standard PC-compatible IDE hard disk controller
  - Supports up to two devices, generally hard disk drives or CD-ROM drives. (CD-ROM drives require a driver.)
  - BIOS supports drives larger than 528 M bytes through Logical Block Addressing (LBA)

1.6.3 Embedded-PC System Enhancements

- 32-pin byte-wide memory socket:
  - Usable with 32K to 1M byte byte-wide memory devices, including EPROMs, Flash EPROMs, SRAMs, and NOVRAMs (Non-volatile RAMs)
  - Backup battery automatically converts SRAM to NOVRAM
  - Onboard programming of 5 V and 12 V Flash EPROMs
  - Configurable as 64K or 128K byte window, addressed in the range of D0000h to EFFFFh
  - Usable with DiskOnChip read/write Flash memory device
  - Supports a PCMCIA memory card connection via an Ampro Memory Card Adapter
  - Supported by Ampro SSD Support Software and many third-party operating systems
  - OEM Flash Memory—an additional 64K (or 960K by special order) of onboard Flash memory for OEM use. Operates like a second byte-wide socket.
- 2K-bit configuration EEPROM:
  - Stores system SETUP parameters
  - Supports battery-free boot capability
  - 512 bits are available for OEM use
- Watchdog Timer
  - Utilizes the onboard real-time clock alarm function
  - Timeout triggers a hardware reset or non-maskable interrupt

1.6.4 Support Software

- Enhanced Embedded-PC BIOS Features:
  - Solid State Disk (SSD) support
  - SCSI services (supports SCSI interfaces found on Ampro MiniModule boards.)
  - Watchdog timer (WDT) support

- Fast boot and blank POST options
- Fail-safe boot logic
- Battery-free boot
- Serial console option
- Serial loader option
- EEPROM access function
- BIOS OEM customization hooks

See the Ampro Embedded-PC BIOS data sheet for additional details about these features.

- Software Utilities Included
  - SETUP utility
  - Watchdog timer support
  - Serial access and development support

1.6.5 Mechanical and Environmental Specifications

- Dimensions:
  - Board Envelope: 3.6 x 3.8 x 0.92 inches (90.2 x 95.9 x 23.4 mm.). Refer to Figure 1–1 for mounting dimensions.
  - Board-to-board spacing: 0.6 inches (15.2 mm.)
- Provision for system expansion with one or more Ampro MiniModule products or other PC/104 expansion modules.
- Power requirements (typical, with 4M bytes DRAM installed):
  - 25 MHz configuration: *450MA* at +5V ±5%
  - Standby power mode: *240MA* at +5V ±5%
- Operating environment:
  - Standard: 0° to 70° C (with adequate airflow)
  - Extended temperature range can be tested by special order. Contact Ampro for details.
  - 5% to 95% relative humidity (non-condensing)
- Storage temperature: -55° to +85° C
- Weight:
  - 3.4 Oz. (95 gm)
- PC/104 expansion bus
  - Stackthrough 16-bit bus connectors, for expansion via PC/104 Version 2 "double-stackthrough" (DST) modules
  - Four mounting holes
- 6-layer PCB using latest surface mount technology

Note

Contact Ampro regarding custom configurations and special order options.

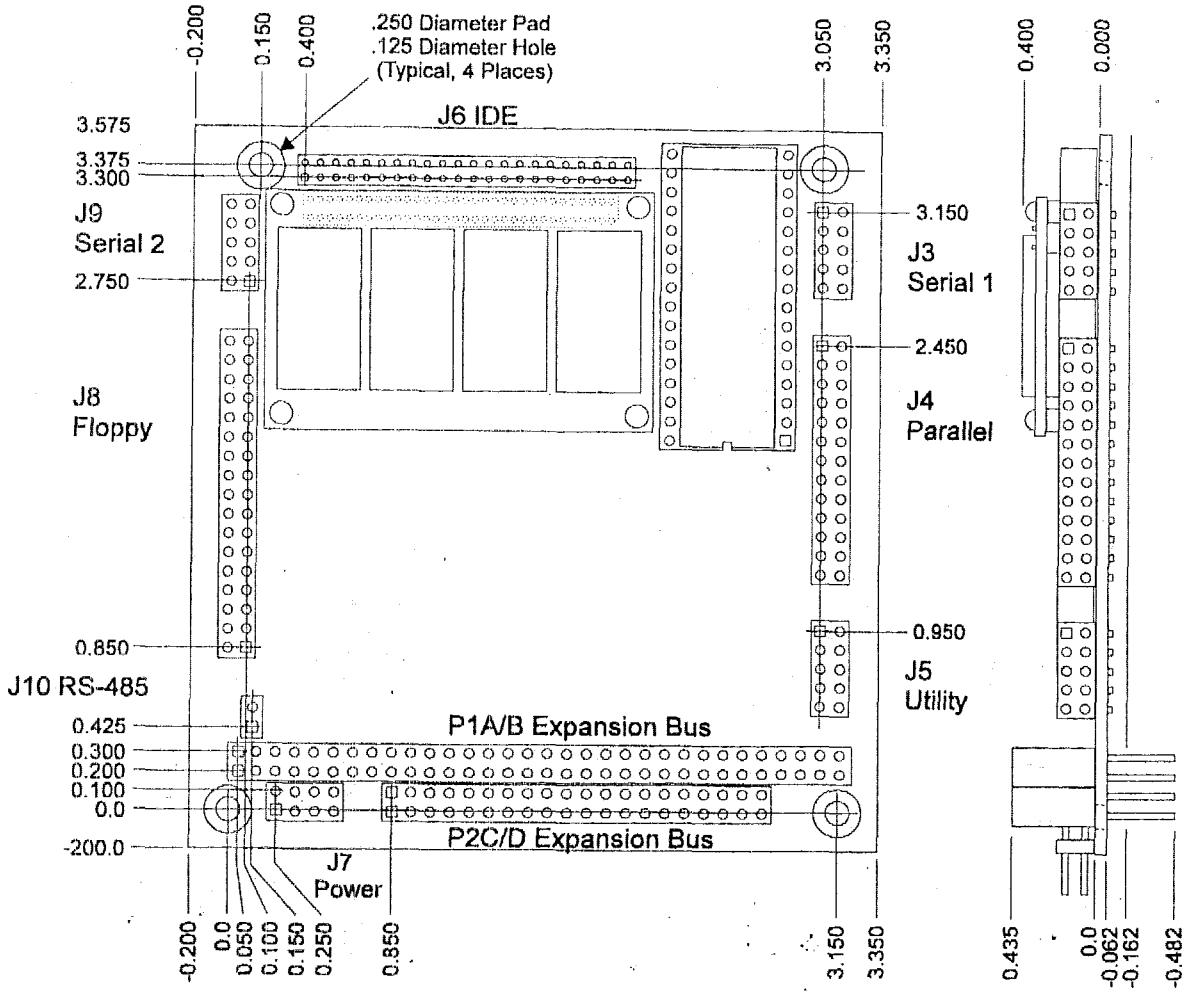


Figure 1–1. Mechanical Dimensions

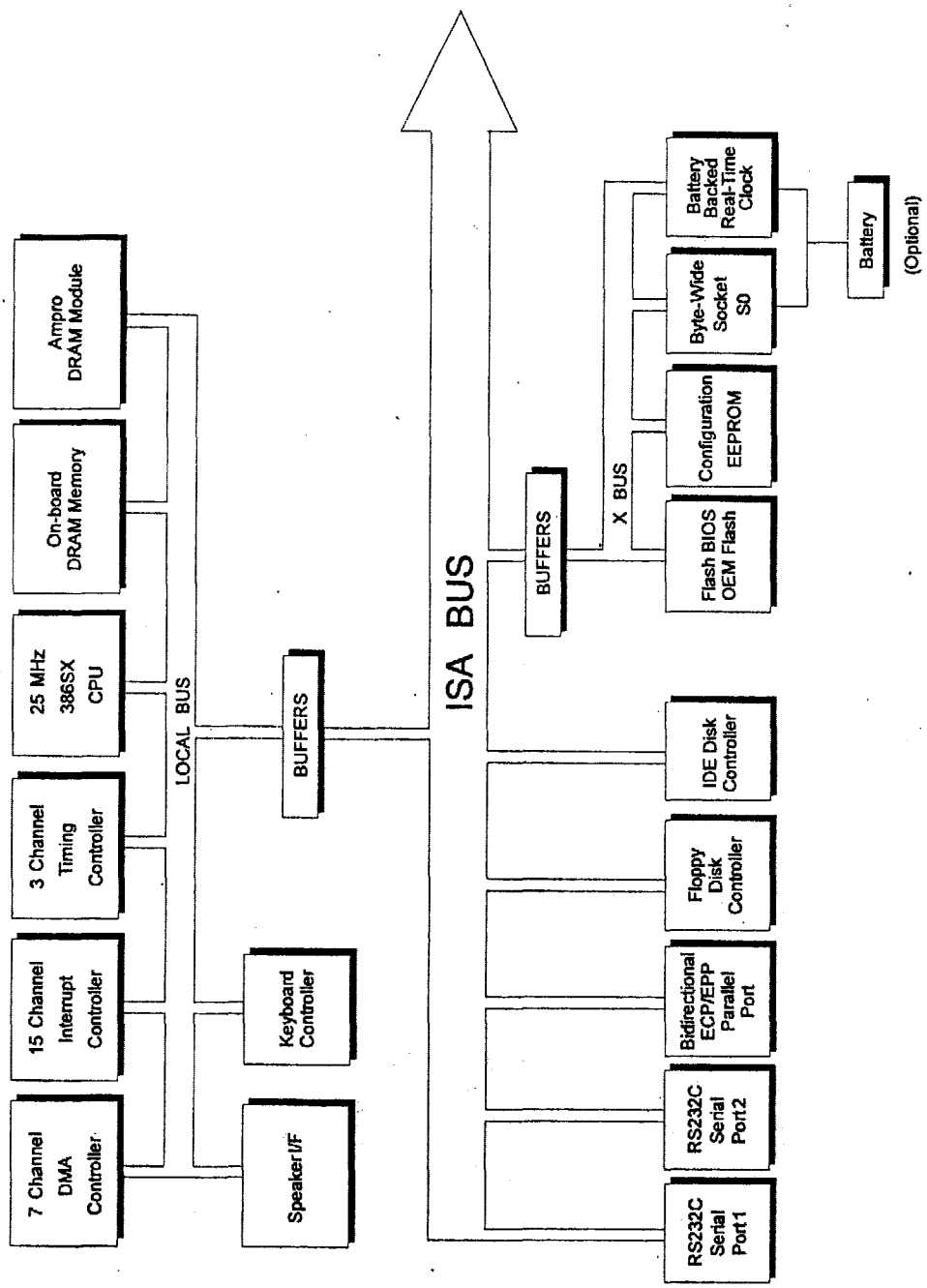


Figure 1–2. Block Diagram

CHAPTER 2

CONFIGURATION AND INSTALLATION

2.1 INTRODUCTION

This chapter covers configuration and installation of the CoreModule/3SX*i* CPU. It includes the board's connector signals and pinouts, external device requirements, interconnection cable wiring, and jumper configuration options.

The topics covered in this chapter are:

- Power Connector
- DRAM memory
- RS-232C/RS-485 serial ports
- Enhanced parallel port
- Floppy disk interface
- IDE hard disk interface
- 32-pin byte-wide socket
- OEM Flash memory
- Utility connector (Keyboard, PC speaker, reset button, external battery)
- Watchdog timer
- Battery-backed clock
- PC/104-compatible expansion bus

2.1.1 Interface Connector Summary

Refer to Figure 2–1 for the locations of the connectors (P1, P2, J3 - J10) and configuration jumpers (W1 - W9).

Table 2–1 summarizes the use of the I/O connectors and Table 2–2 summarizes use of the configuration jumpers.

Each interface is described in its own section, showing connector pinouts, signal definitions, required mating connectors, and configuration jumper options.

Many of the connectors have a *key pin* removed. This allows you to block the corresponding cable connector socket to help prevent improper assembly. Table 2–1 indicates which pins are key pins, and Figure 2–1 shows their locations.

Table 2–1. Connector Usage Summary

Connector	Function	Size	Key Pin
P1A/B	PC/104 Expansion Bus	64-Pin	B10
P2C/D	PC/104 Expansion Bus	40-pin	C19
J3	Serial 1	10-pin	10
J4	Parallel Port	26-pin	26
J5	Utility/Keyboard	10-pin	None
J6	IDE Hard Disk Interface	44-pin 2 mm	20
J7	Power, +5V; +12V, -12V, and -5V to PC/104 Bus	8-pin	None
J8	Floppy Disk Interface	34-pin	6
J9	Serial 2, RS-232C	10-pin	10
J10	Serial 2, RS-485	2-pin	None

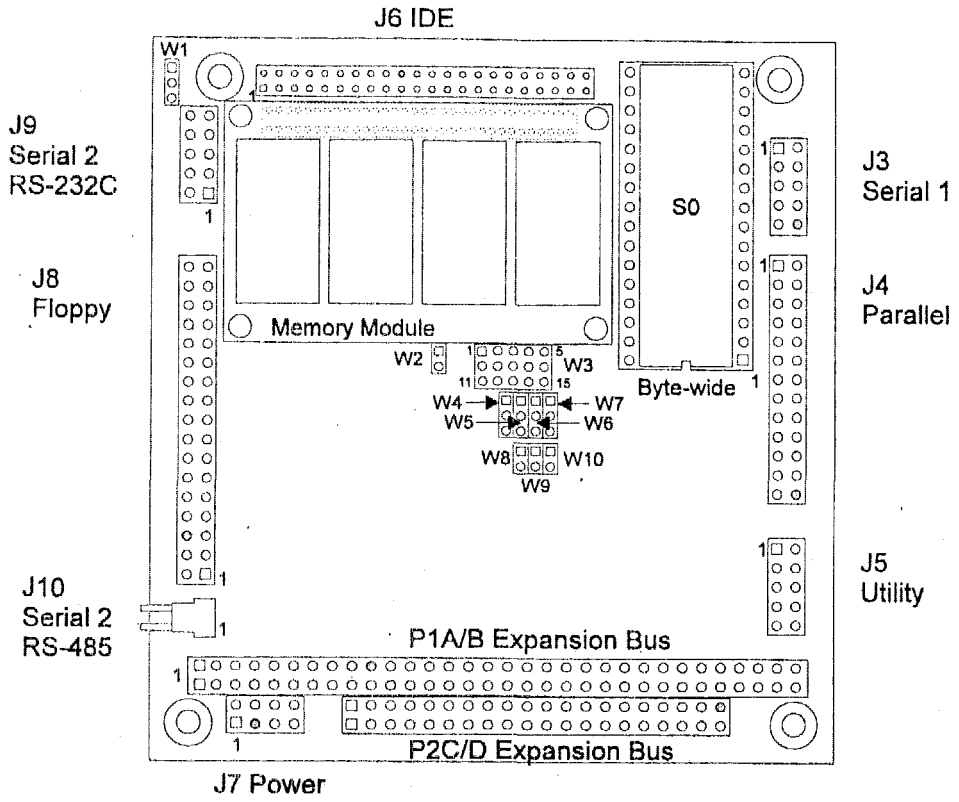
Connectors

Most of the I/O connectors are dual-row headers for use with insulation displacement connectors (IDC) and flat ribbon cable. J5 is usually implemented with discrete wires rather than flat ribbon cable. J10 is a 2-pin connector for an RS-485 twisted-pair cable.

A number of the connectors have "key pins". Install a blocking key in the corresponding connector socket on the mating ribbon cable to prevent misalignment.

You can design a PC board assembly, made with female connectors in the same relative positions as the CoreModule's connectors, to eliminate cables, meet packaging requirements, add EMI filtering, or customize your installation in other ways.

The PC/104-compatible expansion bus appears on two connectors (P1 and P2). You can expand the system with Ampro MiniModule products or other PC/104-compliant expansion modules. These modules stack directly on the P1 and P2 connectors, or you can use conventional or custom expansion hardware, including solutions available from Ampro. Contact your Ampro sales representative for information about alternatives offered by Ampro.



(Key pins on connectors are shaded.)

Figure 2–1. Connector and Jumper Locations

2.1.2 Jumper Configuration Options

Ampro installs option jumpers in default positions so that in most cases the CoreModule/3SX*i* CPU requires no special jumpering for standard operation. You can connect the power and peripherals and operate it immediately. The only jumpers of concern are those that configure the byte-wide socket for the device you install.

Jumper-pin arrays are designated W1, W2 and so forth. Jumper pins are spaced 2 mm apart. A square solder pad identifies pin 1 of each jumper array. Table 2–2 is a summary of jumper use. In the Default column, two numbers separated by a slash (for example, 1/2) means that pins 1 and 2 are shorted with a 2 mm jumper block.

Table 2–2. Configuration Jumper Summary

Jumper Group	Function	Default	Description
W1	RS-232C/RS-485 Select	1/2	1/2=RS-232C; 2/3=RS-485
W2	BIOS/OEM Flash programming power enable	Off	Vpp for Flash EPROM programming
W3	Byte-Wide Socket Configuration		See "Jumpering the Byte-Wide Socket" in Chapter 2
W4	Watchdog Timer Output Selection	Off	See "Watchdog Timer" on Page 2–27.
W5	DMA ACK1/ACK3	Off	Parallel port DMA ACK select
W6	DMA REQ1/REQ3	Off	Parallel port DMA REQ select
W7	Byte-Wide Backup Power Select	1/2	(1/2) enables external battery backup for S0
W8	Byte-Wide Battery Backup Power	Off	On enables backup for S0
W9	BIOS/Byte-Wide Swap	On	Off enables access of a system BIOS from S0.
W10	RS-485 Termination	Off	On for 100 ohm terminator

2.2 DC POWER

To power the module and to supply power to the PC/104 expansion bus, connect the voltages you need for your system to J7. Refer to Table 2–3 for power connections and Table 2–4 for mating connector information.

Table 2–3. J7 Power Connector

Pin	Connection
1, 7	Ground
2, 8	+5VDC
4	+12VDC
5	-5VDC
6	-12VDC

Table 2–4. J7 Mating Connectors

Connector Type	Mating Connector
Discrete Wire, 8-pin	MOLEX Housing 22-55-2081 Pin 16-02-0103

2.2.1 Power Requirements

The CoreModule/3SX*i* CPU requires only +5VDC (±5%) for operation. The ±9 volts for the RS-232 ports is generated onboard from the +5VDC supply.

The module is equipped with a +5 volt to +12 volt voltage converter circuit for programming 12 volt Flash EPROMs. The supply is switched electronically, controlled by the FLASHWRI program (supplied with the CoreModule/3SX*i* CPU Development Kit). Note that this supply is not intended to supply 12 volts to peripherals, and is not connected to the 12 volt input pin on the power connector or on the PC/104 connector. There may be a requirement for an external +12 volt supply, depending on what peripherals you connect to the CoreModule system.

The exact power requirement of the CoreModule/3SX*i* CPU system depends on several factors, including the quantity of DRAM, installed byte-wide memory device, the peripheral connections, and which, if any, MiniModule products or other expansion boards are attached to the PC/104 bus. For example, AT keyboards draw their power from the board, and there can be some loading from the serial and parallel ports. Consult the specifications in Chapter 1 for the basic power requirements of your model.

If you use a switching power supply, be sure it regulates properly with the load your system draws. Some switching power supplies do not regulate properly unless they are loaded to some minimum value. If this is the case with your supply, consult the manufacturer about additional loading, or use another supply or another type of power source (such as a linear supply, batteries, etc.).

2.2.2 CPU Speed

The CPU speed is fixed at 25MHz. There is no user adjustment.

2.2.3 Backup Battery

You can add an external 3.6 volt lithium battery to the Utility Connector, J5, to power the onboard real-time clock and to back up an SRAM installed in the byte-wide socket. Connect the positive terminal to J5-9 and the negative terminal to J5-1.

Here is the formula for calculating battery life (in hours):

Battery life = (battery mA-hour specification ÷ (1 uA + SRAM backup current)) × Duty Cycle

The real-time clock battery drain is approximately 1 uA. To calculate battery life, divide the battery rating by the sum of the clock current and the SRAM current. Then, multiply that result by the duty cycle of the battery. That is, estimate the percentage of time the battery supplies power (while the system is off).

2.3 DRAM

Standard boards have 2M or 4M bytes of DRAM installed on the board. There is also a position for an Ampro custom memory module which currently allows adding 4M bytes of additional DRAM. Contact your Ampro sales representative for the latest information about Ampro's custom memory modules.

When the system boots, the BIOS measures the amount of memory installed and configures the internal memory controller for that amount. (No jumpering or manual configuration is required.) The amount of memory the BIOS measured can be displayed by running SETUP. Saving SETUP automatically stores this figure in the Configuration Memory.

Note

If you change the amount of memory installed, you must run SETUP again to save the new value in the Configuration Memory.

Onboard memory is allocated as follows (standard for the PC architecture):

- The first 640K bytes of DRAM are assigned to the DOS region 00000h to 9FFFFh.
- DRAM in the top 384K bytes of the first 1M byte is not available for user programs. DRAM is mapped into the top 64K to shadow the ROM BIOS. DRAM can also be mapped into a portion of this region to shadow a video BIOS (a SETUP option). (Shadowing is described in the following section.)
- The remaining memory is mapped to extended memory starting at the 1M byte boundary.

2.3.1 Shadowing

One way to improve system performance is to "shadow" the ROM BIOS and video BIOS. When the system operates directly from ROM code, it accesses an 8-bit memory device. When the ROM contents are shadowed, the contents are copied into system DRAM where they are accessed as 16-bit wide data. Shadowing a BIOS ROM substantially enhances system performance, especially when an application or operating system repeatedly accesses the ROM. ROM BIOS shadowing is built into the Ampro Extended BIOS. There is no user setting. Shadowing the video BIOS is a SETUP option. For information about how to set the video BIOS shadowing option, refer to the SETUP section in Chapter 3.

2.3.2 Expanded Memory and Extended Memory

Memory above the 1 megabyte boundary is called "extended" memory. It is a contiguous linear block of memory. Some programs require that memory be available as "expanded" (or "EMS") memory, which makes memory available as pages rather than as a contiguous block. The exact manner for accessing expanded memory is defined in the EMS LIM 4.0 specification.

You can convert the board's extended memory into expanded memory using DOS EMS emulation utilities. Current versions of DOS provide EMS emulation utilities (such as EMM386) that conform to the LIM 4.0 specification. Refer to your DOS technical documentation for instructions for using their EMS emulation utility.

2.4 MATH COPROCESSOR

The 386SX CPU does not contain a floating point math coprocessor. There are no configuration jumpers or options for a math coprocessor.

2.5 SERIAL PORTS

The CoreModule/3SX*i* module provides two standard RS-232C serial ports at J3 and J9. At your option, the second serial port can be configured as an RS-485 port.

You can use the serial ports for printers, modems, terminals, remote hosts, or other RS-232C serial devices. Many devices, such as printers and modems, require handshaking in one or both directions. Consult the documentation for the device(s) you use for information about handshaking, cabling, and other interface considerations.

Use of the RS-485 option offers a low cost, easy-to-use communications and networking multidrop interface that is ideally suited to a wide variety of embedded applications requiring low-to-medium-speed data transfer between two or more systems.

The serial ports are based on a 16550 UART-compatible controller. This is an advanced UART that has a 16-byte FIFO buffer to improve throughput.

Both serial ports support software selectable standard baud rates up to 115.2K baud, 5-8 data bits, and 1, 1.5, or 2 stop bits. Note that the IEEE RS-232C specification limits the serial port to 19.2K baud on cables up to 50 feet in length.

2.5.1 I/O Addresses

The serial ports appear at the standard port addresses as shown in Table 2–5. These are fixed assignments and cannot be changed. Each serial port, however, can be independently disabled using the SETUP function, freeing its I/O addresses for use by other devices installed on the PC/104 expansion bus. For information about serial port configuration using SETUP, see Chapter 3.

Table 2–5. Serial Port I/O Addresses and Interrupts

Port	I/O Address	Interrupt
Serial 1	3F8h - 3FFh	4
Serial 2	2F8h - 2FFh	3

2.5.2 Interrupt Assignments

As shown in Table 2–5, Interrupt 4 (IRQ4) is assigned to Serial 1 and Interrupt 3 (IRQ3) to Serial 2. These assignments can be disabled, but they cannot be changed. When a serial port is disabled, its IRQ is available to other peripherals installed on the PC/104 expansion bus. For information about disabling the serial ports using SETUP, see Chapter 3.

2.5.3 ROM-BIOS Installation of the Serial Ports

Normally, the ROM BIOS supports Serial 1 as the DOS COM1 device, Serial 2 as the DOS COM2 device, and so on. If you disable a serial port, and there is no substitute serial port in the system, then the ROM-BIOS assigns the COM designations as it finds the serial ports, starting from the primary serial port and searching to the last one. Thus, for example, if Serial 1 is disabled, the ROM-BIOS assigns COM1 to Serial 2 (unless another Serial 1 is discovered). The ROM BIOS scans I/O addresses for serial ports in the following order: 3F8h, 2F8h, 3E8h, 2E8h.

2.5.4 Serial Port Connectors (J3, J9)

Serial 1 appears on connector J3 and Serial 2 appears on connector J9. Table 2–6 gives the connector pinout and signal definitions for both ports. Both connectors are wired the same.

In addition, the table indicates the pins to which each signal must be wired for compatibility with standard DB25 and DB9 connectors. The serial port pinout is arranged so that you can use a flat ribbon cable between the header and a standard DB9 connector. Normally PC serial ports use male "DB" connectors. Table 2–7 shows the manufacturer's part numbers for ribbon cable mating connectors to J3 and J9.

Table 2–6. Serial Port Connectors (J3, J9)

Pin	Signal Name	Function	In/Out	DB25 Pin	DB9 Pin
1	DCD	Data Carrier Detect	In	8	1
2	DSR	Data Set Ready	In	6	6
3	RXD	Receive Data	In	3	2
4	RTS	Request To Send	Out	4	7
5	TXD	Transmit Data	Out	2	3
6	CTS	Clear to Send	In	5	8
7	DTR	Data Terminal Ready	Out	20	4
8	RI	Ring Indicator	In	22	9
9	GND	Signal Ground	-	7	5
10	N/A	Key pin	-	-	-

Table 2–7. J3 and J9 Mating Connector

Connector Type	Mating Connector
Ribbon	3M 3473-7010
Discrete Wire	MOLEX Housing 22-55-2101 Pin 16-02-0103

2.5.5 Configuring Serial 2 for RS-485 (J10, W1, W10)

Serial 2 provides circuitry for both an RS-232C and RS-485 interface. Using jumpers, you can configure the port to support either interface (but not both at the same time).

The RS-232C interface appears on J9. Table 2–6 shows the pinout for J9. The RS-485 interface appears on the two-pin connector, J10. Table 2–9 shows the pinout for J10.

Figure 2–2 shows how to set W1 to select the output interface for Serial 2.

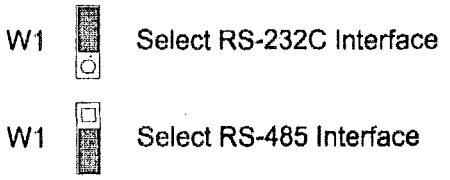


Figure 2–2. Serial 2 Interface Selection

Note

The RS-485 and RS-232C interfaces share some circuitry. If you configure Serial 2 for RS-485, do not connect a serial device to J9. Similarly, if you configure Serial 2 for RS-232C, do not connect anything to J10.

The RS-485 interface specification requires that both ends of the twisted-pair cable be terminated with 100 ohm resistors. You can terminate the RS-485 interface on J10 with a resistor provided on the CoreModule/3SX*i*. To terminate the line, install a jumper on W10.

Table 2–8. RS-485 Termination using W10

W10	Result
On	Connects a 100 ohm termination resistor between J10-1 (+I/O) and ground.
Off	No termination

Table 2–9. RS-485 Serial Port 2 Connector (J10)

Pin	Signal Name
1	+I/O
2	-I/O

Table 2–10. J10 Mating Connector

Connector Type	Mating Connector
Discrete Wire (Locking Connector)	MOLEX Housing 22-01-2027 Pin 08-55-0102

For further information about the RS-485 interface, see Chapter 3, Section 3.9 Serial Ports.

2.5.6 RS-485 Twisted-Pair Cabling Using RJ11 Connectors

Connector J10 is used for an RS-485 twisted-pair connection. In RS-485 multidrop installations, standard RJ11 modular telephone connector jacks are often used to attach standard twisted-pair cables between systems.

RJ11 modular connectors have 6 available contact positions, but only 4 are populated. The 4 center conductors are wired so that the two outside and the two inside conductors are connected together. This eliminates any confusion about pin numbering conventions, as a reversal of connections has no effect. In addition, the lines have been chosen to minimize the possibility of circuit damage should the unit be accidentally plugged into a standard telephone outlet. (It sets the phone line to its "offhook" state to prevent the phone from ringing.)

The recommended wiring for a J10-to-RJ11 cable is shown in Table 2–11.

Table 2–11. J10/RJ11 Cable Wiring

J10 Pin	RJ11 Pin	Signal	Standard Wire Color
	1	N/C	
2	2	- I/O Signal	Black
1	3	+ I/O Signal	Red
1	4	+ I/O Signal	Green
2	5	- I/O Signal	Yellow
	6	N/C	

When connecting the RS-485 port into a multidrop network, the devices at the ends of the network should be terminated with a 100 ohm resistor. Installing a jumper on W10 connects a termination resistor across the RS-485 line on the CoreModule/3SX*i*.

2.5.7 Serial Console

Unique to Ampro is ROM BIOS support for using a serial console (keyboard and display) in place of the conventional video controller, monitor, and keyboard. See Chapter 3 for an explanation of the serial console option.

2.5.8 Serial Downloader

Also unique to Ampro is ROM BIOS support for downloading a program from a host computer via a serial port. The downloaded program is then run as if it had been loaded from disk. See Chapter 3 for an explanation of the serial download option.

2.6 MULTIMODE PARALLEL PORT

The CoreModule/3SX*i* incorporates a multimode parallel port. This port supports four modes of operation:

- Standard PC/AT printer port (output only)
- PS/2-compatible bi-directional parallel port (SPP)
- Enhanced Parallel Port (EPP)
- Extended Capabilities Port (ECP)

See "Multimode Parallel Port" in Chapter 3 for a description of the parallel port's modes.

This section lists the pinout of the parallel port connector and describes how to configure it for its I/O port and interrupt assignments, and how to assign a DMA channel to the port when operating in ECP mode. Refer to Chapter 3 for programming information, including how to use the port for bi-directional I/O.

2.6.1 I/O Addresses

The parallel port functions are controlled by eight I/O ports and their associated register and control functionality. By enabling the parallel port in SETUP, you configure the parallel port as the primary port (typically LPT1). You may disable the port to free the hardware resources for other peripherals.

Table 2–12 lists the parallel port addresses.

Table 2–12 Parallel Port Address Configuration

Selection	I/O Address
Primary	378h - 37Fh
Disable	None

For details about the parallel port I/O addresses and the data, status, control, EPP, and ECP port bit definitions, refer to the Parallel Port section in Chapter 3.

2.6.2 ROM-BIOS Installation of Parallel Ports

Normally, the BIOS assigns the name LPT1 to the primary parallel port, and LPT2 to the secondary parallel port (if present in the system), and so on. However, the BIOS scans the standard addresses for parallel ports and if it only finds a secondary port, it assigns LPT1 to that one. The BIOS scans for parallel ports in the following address order: 3BCh, 378h, 278h.

2.6.3 Interrupts

The parallel port can be configured to generate an interrupt request upon a variety of conditions, depending on the mode the port is in. (These are described in Chapter 3.) In most applications, the interrupt is not used. The standard parallel port interrupts are:

- Primary port IRQ7
- Secondary port IRQ5

The parallel port on the CoreModule/3SX*i* is assigned IRQ7 when enabled in SETUP. It cannot be changed.

2.6.4 DMA Channels

In ECP enhancement mode, the parallel port can send and receive data under control of an on-board DMA controller. DMA channels operate with a request/acknowledge handshake protocol between an internal DMA controller and the parallel port logic. You can select DMA request (DRQ) and DMA Acknowledge (DACK) assignments using the jumpers at W5 and W6. The parallel port may use either DMA channel 1 or DMA channel 3. To select DMA channel 1, shunt jumper W5 (1/2) and W6 (1/2). To select DMA channel 3, shunt jumper W5 (2/3) and W6 (2/3). See Figure 2–3.

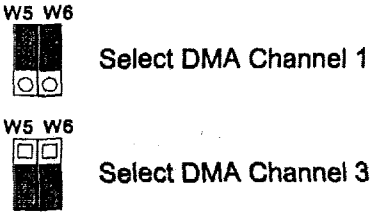


Figure 2–3. DMA Channel Selection (W5, W6)

If you will not be using DMA with the parallel port, leave the jumpers off. This makes the DMA controls available to other peripherals installed on the expansion bus.

2.6.5 Parallel Port Connector (J4)

Connection to the parallel port is through connector J4. Table 2–13 gives this connector's pinout and signal definitions. You can use a flat ribbon cable between J4 and a female DB25 connector. The table also gives the connections from the header pins to the DB25 connector. Table 2–14 gives manufacturer's part numbers for mating connectors.

Table 2–13. Parallel Port Connector (J4)

J4 Pin	Signal Name	Function	In/Out	DB25 Pin
1	STROBE*	Output data strobe	OUT	1
3	Data 0	LSB of printer data	I/O	2
5	Data 1		I/O	3
7	Data 2		I/O	4
9	Data 3		I/O	5
11	Data 4		I/O	6
13	Data 5		I/O	7
15	Data 6		I/O	8
17	Data 7	MSB of printer data	I/O	9
19	ACK*	Character accepted	IN	10
21	BUSY	Cannot receive data	IN	11
23	PAPER OUT	Out of paper	IN	12
25	SEL OUT	Printer selected	IN	13
2	AUTOFD*	Autofeed	OUT	14
4	ERROR	Printer error	IN	15
6	INIT*	Initialize printer	OUT	16
8	SEL IN	Selects printer	OUT	17
26	N/A	Key pin		
10,12, 14,16 18,20 22,24	GROUND	Signal ground	N/A	18-25
Data lines: 24 mA sink (.4 V max.), 12 mA source (2.4 V min.). Control lines: 24 mA sink (.4 V max.), open collector with 4.7K pull-ups.				

Table 2–14. J4 Mating Connector

Connector Type	Mating Connector
RIBBON	3M 3399-7600
DISCRETE WIRE	MOLEX HOUSING 22-55-2262 PIN 16-02-0103

Note

For maximum reliability, keep the cable between the board and the device it drives to 10 feet or less in length.

2.6.6 IEEE-1284-Compliant Cables

Using the parallel port for high-speed data transfer in ECP/EPP modes requires special cabling for maximum reliability.

Some of the parameters for a compliant IEEE-1284 cable assembly include:

- All signals are twisted pair with a signal and ground return
- Each signal and ground return should have a characteristic unbalanced impedance of 62 +/- 6 ohms within a frequency band of 4 to 16 MHz
- The wire-to-wire crosstalk should be no greater than 10%

Please refer to the IEEE-1284 standard for the complete list of requirements for a compliant cable assembly, including recommended connectors. For information about the IEEE-1284 standard, see Enhanced Parallel Port in Chapter 3.

Latch Up Protection

The parallel port incorporates chip protection circuitry on some inputs, designed to minimize the possibility of CMOS "latch up" due to a printer or other peripheral being powered up while the CoreModule/3SX*i* is turned off.

2.7 FLOPPY DISK INTERFACE

The onboard floppy disk controller and ROM BIOS support one or two floppy disk drives in any of the standard DOS formats shown in Table 2–15

Table 2–15. Supported Floppy Formats

Capacity	Drive Size	Tracks	Data Rate
360K	5-1/4 inch	40	250 KHz
1.2M	5-1/4 inch	80	500 KHz
720K	3-1/2 inch	80	250 KHz
1.44M	3-1/2 inch	80	500 KHz

2.7.1 Floppy Drive Considerations

Nearly any type of soft-sectored, single or double-sided, 40 or 80 track, 5-1/4 inch or 3-1/2 inch floppy disk drive is usable with this interface. Using higher quality drives improves system reliability. Here are some considerations about the selection, configuration, and connection of floppy drives to the CoreModule/3SX*i* CPU .

- **Drive Interface**—The drives must be compatible with the board's floppy disk connector signal interface, as described below. Ampro recommends any standard PC-or AT-compatible 5-1/4 inch or 3-1/2 inch floppy drive.
- **Drive Quality**—Use high quality, DC servo, direct drive motor floppy disk drives.
- **Drive Select Jumpering**—Jumper both drives for the second drive select (standard on PC drives).
- **Floppy Cable**—For systems with two drives, use a floppy cable with conductors 10-16 twisted between the two drives. This is standard practice for PC-compatible systems.
- **Drive Termination**—Resistive terminations should be installed only on the drive connected to the last interface cable connector (farthest from the board). Near-end cable termination is provided on the CoreModule/3SX*i* CPU.
- **Head Load Jumpering**—When using drives with a Head Load option, jumper the drive for head load with motor on rather than head load with drive select. This is the default for PC-compatible drives.
- **Drive Mounting**—If you mount a floppy drive very close to the Little Board or another source of EMI, you may need to place a thin metal shield between the disk drive and the device to reduce the possibility of electromagnetic interference.

2.7.2 Floppy Interface Configuration

The floppy interface is configured using SETUP to set the number and type of floppy drives connected to the system. Refer to the SETUP section in Chapter 3 for details.

If you don't use the floppy interface, disable it in SETUP. This frees its I/O addresses (3F0h - 3F7h), DMA2, and IRQ6 for use by other peripherals installed on the PC/104 bus.

2.7.3 Floppy Interface Connector (J8)

Table 2–16 shows the pinout and signal definitions of the floppy disk interface connector, J8. The pinout of J8 meets the AT standard for floppy drive cables. Table 2–17 shows the manufacturer's part numbers for mating connectors.

Table 2–16. Floppy Disk Interface Connector (J8)

Pin	Signal Name	Function	In/Out
2	RPM/-RWC	Speed/Precomp	OUT
4	N/A	(Not used)	N/A
6	N/A	Key pin	N/A
8	-IDX	Index Pulse	IN
10	-MO1	Motor On 1	OUT
12	-DS2	Drive Select 2	OUT
14	-DS1	Drive Select 1	OUT
16	-MO2	Motor On 2	OUT
18	-DIRC	Direction Select	OUT
20	-STEP	Step	OUT
22	-WD	Write Data	OUT
24	-WE	Write Enable	OUT
26	-TRKO	Track 0	IN
28	-WP	Write Protect	IN
30	-RDD	Read Data	IN
32	-HS	Head Select	OUT
34	-DCHG	Disk Change	IN
1-33	(all odd)	Signal grounds	N/A

Table 2–17. J8 Mating Connector

Connector Type	Mating Connector
Ribbon	3M 3414-7600
Discrete Wire	MOLEX Housing 22-55-2342 Pin 16-02-0103

2.8 IDE HARD DISK INTERFACE

The CoreModule/3SX*i* CPU provides an interface for one or two Integrated Device Electronics (IDE) hard disk drives. IDE drives, the most popular and cost-effective type of hard drive currently available, have an internal hard disk controller. There are also many CD-ROM drives designed to use the IDE interface. If you attach a CD-ROM drive to the IDE port, you will need a driver (supplied by the CD-ROM drive manufacturer or your operating system) to access the device .

2.8.1 IDE Connector (J6)

The IDE interface appears at connector J6, a 44-pin, dual-row 2 mm. right-angle connector. Table 2–18 shows the interface signals and pin outs for the IDE interface connector. Table 2–19 shows manufacturer's part numbers for mating connectors to J6.

Note

For maximum reliability, keep IDE drive cables less than 18 inches long.

Table 2–18. IDE Drive Interface Connector (J6)

Pin	Signal Name	Function	In/Out
1	-HOST RESET	Reset signal from host	OUT
3	HOST D7	Data bit 7	I/O
4	HOST D8	Data bit 8	I/O
5	HOST D6	Data bit 6	I/O
6	HOST D9	Data bit 9	I/O
7	HOST D5	Data bit 5	I/O
8	HOST D10	Data bit 10	I/O
9	HOST D4	Data bit 4	I/O
10	HOST D11	Data bit 11	I/O
11	HOST D3	Data bit 3	I/O
12	HOST D12	Data bit 12	I/O
13	HOST D2	Data bit 2	I/O
14	HOST D13	Data bit 13	I/O
15	HOST D1	Data bit 1	I/O
16	HOST D14	Data bit 14	I/O
17	HOST D0	Data bit 0	I/O
18	HOST D15	Data bit 15	I/O
20	KEY	Keyed pin	N/C
21	RSVD	Reserved	N/C
23	-HOST IOW	Write strobe	OUT
25	-HOST IOR	Read strobe	OUT
27	RSVD	Reserved	N/C
28	RSVD	Reserved	N/C
29	RSVD	Reserved	N/C
31	HOST IRQ14	Drive interrupt request	IN

Table 2–15. IDE Drive Interface Connector (J6) (cont.)

Pin	Signal Name	Function	In/Out
32	RSVD	Reserved	N/C
33	HOST A1	Drive address 1	OUT
34	RSVD	Reserved	N/C
35	HOST AD0	Drive address 0	OUT
36	HOST AD2	Drive address 2	OUT
37	-HOST CS0	Chip select	OUT
38	-HOST CS1	Chip select	OUT
39	-HOST SLV/ACT	Drive active/drive slave	10K Pull-up
41, 42	+5V	Power	OUT
2, 19, 22, 24, 26, 30, 40, 43	GND	Ground	OUT
21, 28, 29	N/C	No Connection	-

Table 2–19. J6 Mating Connector

Connector Type	Mating Connector
Ribbon Cable, 1 mm. 44 cond.	3M 3625/44
Ribbon Cable Connector	ASTRON AT-IDCSK-44-11-GF

2.8.2 IDE Cable Adapter

Many IDE hard drives and CD-ROM drives have 40-pin connectors with .1 inch pin spacing. Ampro makes an adapter board that you can install on your drive to convert it from the larger 40-pin format to the 44-pin 2 mm. format to make it compatible with the 2 mm. cable defined by the components listed in Table 2–19. For details, ask your Ampro sales representative about the *IDE Cable Adapter*.

2.8.3 IDE Interface Configuration

Use SETUP to specify your IDE hard disk drive type. Refer to the SETUP section in Chapter 3 for details.

If you do not find a drive type whose displayed parameters match the drive you are using, use drive type 48 or 49. These allow you to manually enter the drive's parameters. The drive manufacturer provides the drive parameters—check the drive's documentation for the proper values to enter.

If you are using a newer IDE drive, use drive type AUTO. It automatically configures the drive type parameters from information provided by the drive itself. The Autoconfigure function is described in Chapter 3.

If you use an IDE drive in your system, you can still add SCSI drives or other SCSI peripherals. The Ampro ROM BIOS provides a means for allowing both IDE and SCSI drives on the same system. See the SETUP description in Chapter 3 for details.

2.9 BYTE-WIDE SOCKET

The CoreModule/3SX*i* CPU has a 32-pin onboard byte-wide memory socket, designated **S0**. This socket can accept a wide variety of EPROM, Flash EPROM, SRAM, and nonvolatile RAM (NOVRAM) devices. Battery backup power can be connected to S0 using a jumper option to make a standard SRAM "non-volatile" (retains data while system power is off).

You can use a memory device installed in the byte-wide socket for a variety of purposes:

- Simple program storage
- BIOS extension
- Solid State Disk (SSD) drive

Table 2–20 shows representative byte-wide memory devices that can be installed in the byte-wide socket. The table gives examples of generic part numbers, the size of the device (K bytes), and the DIP package pin count. It also lists the SSD device type, used by the Ampro Solid State Disk (SSD) Support Software to identify memory devices.

Table 2–20. Typical Byte-wide Devices

SSD Device Type	Size	Package Pins	Generic Part Number
EPROMs			
EPROM32	32K byte	28	27C256
EPROM64	64K byte	28	27C512
EPROM128	128K byte	32	27C010
EPROM256	256K byte	32	27C020
EPROM512	512K byte	32	27C040
EPROM1024	1024K byte	32	27C080
Flash EPROMs			
EPROM128	128K bytes	32	28F010
EPROM256	256K bytes	32	28F020
EPROM512	512K bytes	32	29F040

Table 2–21. Typical Byte-wide Devices (Cont.)

SRAMs			
SRAM32	32K bytes	28	43256
SRAM128	128K bytes	32	62204
SRAM512	512K bytes	32	434000

The pinout of the 32-pin socket can be configured to comply with both the 28-pin and 32-pin JEDEC standards. You can install a 28-pin device in the 32-pin socket. Install the 28-pin device with pin 1 oriented to the socket's pin 3, as indicated in Figure 2–4.

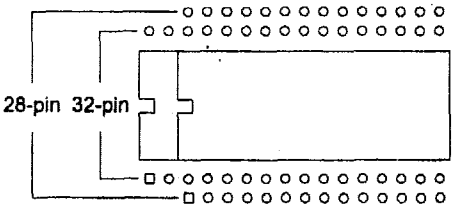


Figure 2–4. Using 28- and 32- pin Devices in 32-pin Sockets

2.9.1 Addressing the Byte-wide Socket

Use SETUP to specify the size and starting address of the byte-wide socket, and whether the BIOS enables the socket upon system initialization.

Table 2–22 lists the possible settings for sizes and address ranges of the byte-wide socket.

Note

When the byte-wide socket is enabled, the memory address space it uses is unavailable for other devices, even if no memory device is installed in the socket. You must disable the byte-wide socket in SETUP before you can use the memory space for other purposes.

Table 2–22. Window Size and Address Selection

Window	Address
DISABLE	N/A
64K	D0000-DFFFFh
64K	E0000-EFFFFh
128K	D0000-EFFFFh

The size of the device installed in the byte-wide socket is not limited to 128K bytes. Using a page addressing scheme, devices (or modules) up to 1M bytes can be used. Higher address lines (A16-A19) are synthesized and can be set by software using Ampro extended BIOS function calls. A description and examples of byte-wide page control are provided in Chapter 3.

If devices larger than 64K bytes are installed, you must select which page is visible in the address window. A page is 64K bytes. (If the size is set to 128K bytes, a single 128K byte window is established. Paging is only available with the 64K window setting.) The Ampro Extended BIOS provides convenient software calls to manage enabling/disabling the socket and selecting pages. Refer to Chapter 3 for details about the byte-wide extended ROM-BIOS calls.

If you install a device that is smaller than the selected window size, the contents of the device are duplicated in the byte-wide socket's memory space. For example, the software will see two copies of a 32K device in a 64K window, and 4 copies in a 128K window.

ROM-BIOS Extensions

The system can be configured to run its application from the byte-wide socket instead of loading it into DRAM from a disk drive. This technique, known as a ROM BIOS extension, directly executes the application during the Power On Self Test (POST) instead of booting from floppy or hard disk. The ROM-BIOS extension concept, and its practical implementation, is discussed in Ampro Application Notes AAN-8702 and AAN-9003.

Performance Issues

Note that executing programs directly from the byte-wide socket can adversely affect system performance. There are a number of factors that can contribute to the performance impact:

- The byte-wide device is substantially slower than DRAM, as it is an 8-bit device instead of 16-bit.
- The device is accessed from the PC expansion bus which is much slower than the high-speed processor memory bus.

You can improve performance substantially by copying the contents of the byte-wide device into RAM and executing the RAM copy.

2.9.2 OEM Flash Memory

Access to the byte-wide socket is integrated with access to an onboard Flash memory device, designated the **OEM Flash Memory** in SETUP. The OEM Flash memory acts as a second byte-wide device, in that you access it through the same code mechanisms as the byte-wide socket. These mechanisms are described in Chapter 3. (There are no jumpers to configure the OEM Flash memory.)

Only one of the two devices can be enabled at a time. When you enable the OEM Flash memory (using an extended BIOS call), the byte-wide is automatically disabled, and vice-versa.

The first 64K bytes of the Flash memory device hold the ROM BIOS. The remaining portion can be used by OEMs or end-users in a manner similar to the byte-wide socket. In the standard version of the CoreModule/3SX*i*, the remaining portion is 64K bytes (the second 64K portion of a 128K device). By special order, you can substitute a 1M byte Flash device. As with the 128K device, the first 64K is used for the ROM BIOS. The remaining area is available for the embedded system. (Contact your Ampro Sales Representative for details about ordering the larger memory size.)

Set the address parameters for both the byte-wide socket and the OEM Flash device with SETUP. Refer to Chapter 3 for details.

2.9.3 Solid State Disk (SSD) Drives

Using the Ampro Solid State Disk (SSD) Support Software, you can configure an EPROM, Flash EPROM, or SRAM solid-state device, installed in the byte-wide socket, or the OEM Flash device, to act as a solid-state floppy disk drive.

No custom programming is required. Regular DOS-compliant programs, including standard DOS utilities, can be used without modification. Ampro's SSD support software creates data image files, based on your application programs and operating system, which are programmed into the device you install in the byte-wide socket. The Ampro ROM-BIOS treats the device like one or more disk drives, loading the programs into DRAM for execution. You can use SSD drives in addition to, or instead of, normal floppy and hard disk drives. You can increase the system SSD capacity by adding one or more of Ampro's SSD expansion modules.

If your board is equipped with the optional 1M byte OEM Flash device, you can install the TrueFFS Flash file system in it. This creates a fully read/write-capable solid state disk without adding any additional components to your system. Instructions on how to configure the OEM Flash device with TrueFFS Flash file system is in the TrueFFS manual that comes with the software.

2.9.4 Jumpering the Byte-Wide Socket

You must jumper the byte-wide socket for the device you install. Jumper array W3 configures S0 for a particular device type.

Table 2–23 shows how to install jumpers for a variety of supported EPROM memory devices.

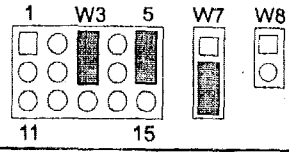
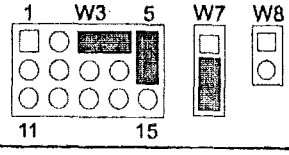
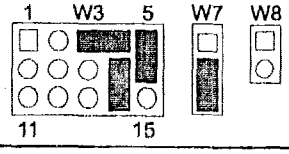
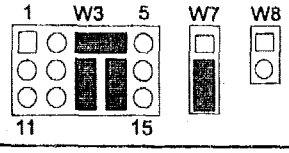
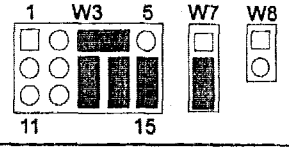
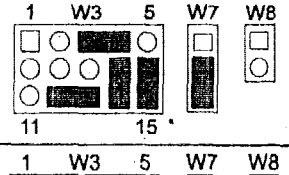
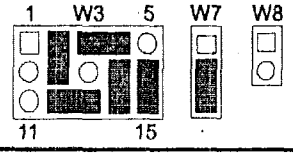
Table 2–24 shows how to install jumpers for a variety of supported Flash EPROM memory devices.

Table 2–25 shows how to install jumpers for a variety of supported SRAM and NOVRAM memory devices.

2.9.5 Using EPROMs

Table 2–23 shows the jumpering for supported EPROM devices. If you install an EPROM, make sure the jumper on W8 is removed and the jumper on W7 is on pins 2/3 to prevent premature discharge of the onboard backup battery. Some EPROMs draw current through their chip select lines (or other pins) when powered down.

Table 2–23. EPROM Jumpering for S0

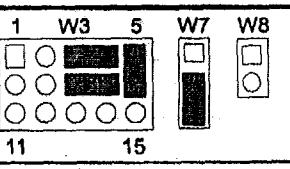
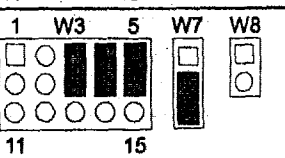
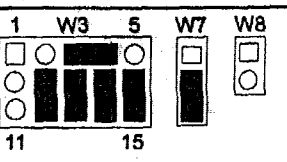
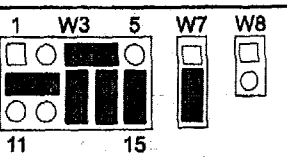
EPROM	Pins	Jumper Diagram
8K EPROM 27C64 16K EPROM 27C128 8K EEPROM 28C64	28	
32K EPROM 27C256	28	
64K EPROM 27C512	28	
128K EPROM 27C010	32	
256K EPROM 27C020	32	
512K EPROM 27C040	32	
1M EPROM 27C080	32	

2.9.6 Using Flash EPROMs

Flash programming power for +12V Flash devices is provided by an onboard power supply. You do not need to connect an external +12V power supply to program Flash devices. Programming power is switched under software control so that it is applied only during the actual programming process (to prevent accidental corruption of the data). A utility for programming supported Flash devices is included on the utility disk that is provided with the CoreModule/3SX*i* CPU Development Kit.

If you install a Flash EPROM, make sure the jumper on W8 is removed and the jumper on W7 is on pins 1/2 to prevent premature discharge of the onboard backup battery. Some Flash EPROMs draw current through their chip select lines (or other pins) when powered down.

Table 2–24. Flash EPROM Jumpering for S0

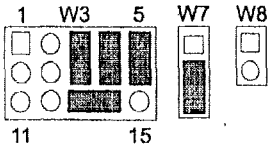
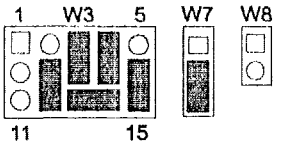
Flash EPROM Typical Devices		Pins	Jumper Diagram
32K 5V Flash EPROM	29C256	28	
32K 5V Flash EPROM	28C256	28	
64K 5V Flash EPROM 128K 5V Flash EPROM 256K 5V Flash EPROM 512K 5V Flash EPROM	29F512 29F010 29F020 29F040	32	
32K 12V Flash EPROM 64K 12V Flash EPROM 128K 12V Flash EPROM 256K 12V Flash EPROM	28F256 28F512 28F010 28F020	32	

2.9.7 Using SRAMs

If you install an SRAM, you can provide backup power from the battery when power is off by shorting W8 and W7-1/2. If you use the SRAM for "scratchpad" storage and do not want to retain data when power is off, remove the jumper from W8 and install a jumper on W7-2/3.

A typical 165 milliamp-hour external battery provides sufficient current for the onboard real-time clock for a 10 year life, but if you are going to battery-back-up a device in S0, Ampro recommends a larger battery. For calculating battery life, see page 2–5, Backup Battery.

Table 2–25. SRAM and NOVRAM Jumpering for S0

SRAM Typical Devices			Pins	Jumper Diagram
32K SRAM		43256	28	
32K NOVRAM	Dallas	DS1230Y		
	Benchmarq	BQ4011Y		
128K SRAM		628128	32	
128K NOVRAM	Dallas	DS1245Y		
	Benchmarq	BQ4013Y		
512K SRAM		628512		
512K NOVRAM	Dallas	DS1650Y		
	Benchmarq	BQ4015Y		

NOTE: W7 and W8 are show configured for (self-powered) NOVRAMs. To configure W7 and W8 for SRAM battery backup, install a jumper on W8 and move the jumper on W7 to 1/2.

2.9.8 Byte-Wide Socket Signals

W3 is used to configure the byte-wide socket for specific memory devices. In addition, jumpers W7 and W8 control the backup battery to S0 for use with SRAMs.

Table 2–26 lists the signals that appear on the pins of W3.

Table 2–26. Byte-Wide Jumper Pin Signals (W3)

W3 Pin	Signal Name	Description
1	N/C	No connection
2	A19	Address A19
3	Pin 29	Connection to pin 29 of the byte-wide socket
4	A14	Address SA14 from the expansion bus
5	Vcc or backup battery	Connected to the center pin of W7. W7-3 connects to +5V. W7-1 connects to the backup battery.
6	Vpp	Programming power for Flash devices
7	Pin 1	Connection to pin 1 of the byte-wide socket
8	-SMEMW	Write strobe
9	Pin 3	Connection to pin 3 of the byte-wide socket
10	Pin 30	Connected to pin 30 of the byte-wide socket
11	N/C	No Connection
12	A18	Address A18
13	Pin 31	Connection to pin 31 of the byte-wide socket
14	A15	Address SA15 from the expansion bus
15	A17	Address A17

W7 and W8 Options

Some EPROMs draw power through their chip select lines when Vcc is off. This could drain the real-time clock battery if it were connected to such a device. Removing the jumper from **W8** disconnects the battery from the byte-wide circuit (leaving it connected to the real-time clock) and prevents an EPROM from draining the battery prematurely.

Some byte-wide devices require more current than can be handled by the power switch that controls Vcc to the byte-wide socket. The power switch is designed to switch between battery-power and Vcc for an SRAM which has very low current-drain. If you are using a Flash memory or EPROM in the byte-wide socket, set W7-2/3 to connect the memory device directly to Vcc rather than through the power switch (W7-1/2).

2.10 BATTERY-BACKED CLOCK

An AT-compatible battery-backed real-time clock (with CMOS RAM) is standard on the CoreModule/3SX*i* CPU. The clock can be powered by a 3.6 volt Lithium battery connected to the Utility Connector, J5. Battery drain for the clock is less than 1 uA.

Use the Ampro SETUP utility to set the current time and date in the real-time clock, as well as SETUP information in the CMOS RAM portion of the clock chip (configuration memory).

The contents of the configuration memory are also stored in an onboard EEPROM. The ROM BIOS reads the EEPROM to get configuration information if the CMOS RAM data is lost. This means that the board will function without the battery. Note that without a battery, the real-time clock date and time will not be correct.

2.11 WATCHDOG TIMER

A unique feature of the onboard clock circuitry is a watchdog timer. You can program this timer to generate an interrupt or reset signal if the programmed time interval expires before the timer is reinitialized. Use SETUP to select the time interval. The options are: Disable, 30 seconds, 60 seconds, and 90 seconds.

The watchdog timer uses the standard alarm feature of the real-time clock. In a standard AT, the alarm output is connected to IRQ8. On the CoreModule/3SX*i* CPU you can also jumper the alarm output to I/O Channel Check (-IOCHCK) or RESET with W4. I/O Channel Check is the bus signal that triggers a non-maskable interrupt (NMI). RESET is a hard reset signal, the same as pressing the Reset button. Watchdog timer responses are summarized in Table 2–27.

Table 2–27. Watchdog Timer Setup

Jumper W4	SETUP	WDT Response
W4-1/2 Shorted	Enabled	Hardware Reset
W4-2/3 Shorted	Enabled	I/O Channel Check (NMI)
W4 Open	Enabled	IRQ8 turns off interrupt. System continues unaffected.
W4 Open	Disabled	No action.

Note

If you use the MS-DOS operating system, you cannot use the watchdog timer to monitor the boot process. MS-DOS resets the alarm clock in the real-time clock at boot time.

2.12 UTILITY CONNECTOR (J5)

Six functions appear on the 10-pin connector at J5. These are:

- PC speaker
- Push-button reset switch
- Standard PC keyboard interface
- External back-up battery for the real-time clock and byte-wide S0

Table 2–28 shows the pinout and signal definitions of the Utility Connector. Since there are connections for diverse features on this single connector, you would usually choose a discrete-wire connector rather than a ribbon cable connector, though this is not a requirement. Table 2–29 shows manufacturer's part numbers for both types of mating connectors.

Table 2–28. Utility Connector (J5)

Pin	Signal Name	Function
1	Speaker +	PC audio signal output
2	BATV-	Negative terminal of external backup battery
3	Reset	Manual reset button.
4	N/C	No connection
5	Keyboard Data	Keyboard serial data
6	Keyboard Clock	Keyboard clock
7	Ground	Keyboard ground
8	Keyboard Power	Keyboard +5V power
9	BATV+	Positive terminal of external backup battery
10	N/C	No connection

Table 2–29. J5 Mating Connector

Connector Type	Mating Connector
Ribbon	3M 3473-7010
Discrete Wire	MOLEX Housing 22-55-2101 Pin 16-02-0103

2.12.1 Speaker Connections

The board supplies about 100 mW for a speaker on J5-1. Connect the other side of the speaker to ground (J5-2). A transistor amplifier buffers the speaker signal. Use a small general purpose 2 or 3 inch permanent magnet speaker with an 8 ohm voice coil. Refer to Chapter 3 for an explanation of the PC speaker circuit architecture.

2.12.2 Push-button Reset Connection

J5-3 provides a connection for an external normally-open momentary switch to manually reset the system. Connect the other side of the switch to ground. The reset signal is "de-bounced" on the board.

2.12.3 Keyboard Connections

You can connect an AT (not PC) keyboard to the keyboard port. J5-5 through J5-8 provide this function. Normally, AT keyboards include a cable that terminates in a male 5-pin DIN plug for connection to an AT. Table 2–30 gives the keyboard connector pinout and signal definitions, and includes corresponding pin numbers of a normal AT DIN keyboard connector.

Table 2–30. Keyboard Connector (J5)

J5 Pin	Signal Name	DIN Pin
5	Keyboard Clock	1
6	Keyboard Data	2
N/C	No connection	3
7	Ground	4
8	Keyboard power	5

2.12.4 External Battery Connections

To connect an external battery, connect its positive terminal to J5-9 and its negative terminal to J5-2. Use a 3.6 volt lithium cell.

The battery is connected by a low-drop Schottky diode. Two blocking devices are in series with the battery, complying with UL recommendations for lithium batteries.

2.13 AT EXPANSION BUS

The PC/AT expansion bus appears on a pair of header connectors at P1 and P2. P1 is a 64-pin female dual-row header. P2 is a 40-pin female dual-row header. Pins from both headers extend through the board, providing male connections for PC/104-compliant peripherals or other devices.

The PC-bus subset of the expansion bus connects to the first 62 positions of P1; the two additional positions of P1 (A32 and B32) are added grounds to enhance system reliability. Connector P2 replaces the 36-pin edge card connector of a conventional ISA expansion bus. It has extra ground positions at each end of the connector (C0, D0, D19). (C19 is a key pin.) The extra grounds C0 and D0 are numbered "0" to keep the pin numbers of the remaining signals on the connector the same as those on the standard ISA bus. The layout of signals on P1 and P2 is compliant with the PC/104 bus specification (IEEE P996.1 (proposed)). PC/104-compatible expansion modules can be installed on the CoreModule/3SX*i* CPU expansion bus.

The buffered output signals to the expansion bus are standard TTL level signals. All inputs to the CoreModule/3SX*i* CPU operate at TTL levels and present a typical CMOS load to the expansion bus. The current ratings for most output signals driving the AT expansion bus are shown in Table 2–31 through Table 2–34 , and indicate how the signals are terminated on the CoreModule/3SX*i* CPU.

2.13.1 Onboard MiniModule Expansion

You can install one or more Ampro MiniModule products or other PC/104 modules on the CoreModule/3SX*i* CPU expansion connectors. When installed on P1 and P2, the expansion modules fit

within the CoreModule/3SX*i* CPU's outline dimensions. Most Ampro MiniModule products have stackthrough connectors compatible with the PC/104 Version 2.1 specification. You can stack several modules on the CoreModule/3SX*i* CPU headers. Each additional module increases the thickness of the package by 0.66 inches (17 mm). See Figure 2–5.

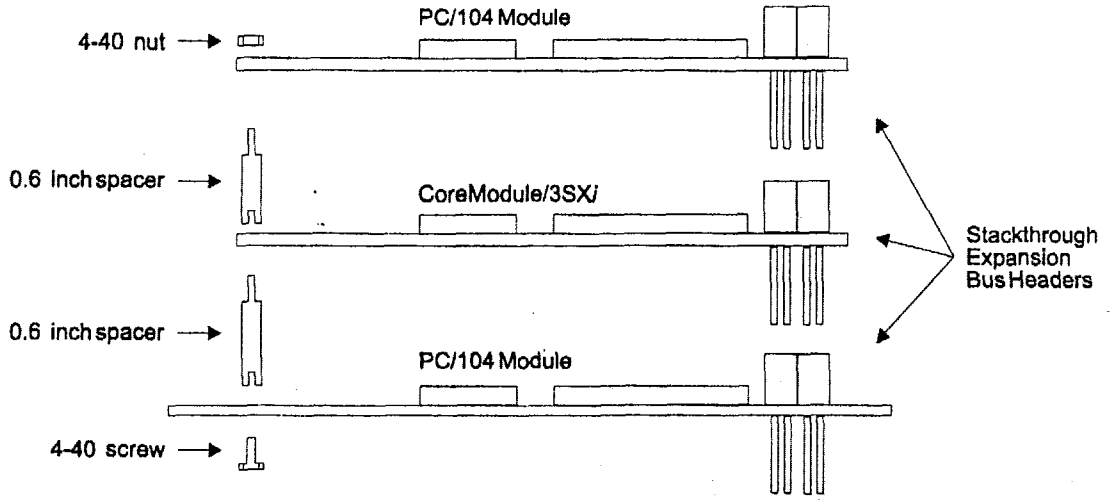


Figure 2–5. Stacking PC/104 Modules with the CoreModule/3SX*i* CPU

2.13.2 Using Standard PC and AT Bus Cards

Ampro offers several options that allow you to add conventional 8-bit and 16-bit ISA expansion cards to the CoreModule/3SX*i* CPU system. Contact Ampro for further information about optional bus expansion products.

2.13.3 Bus Expansion Guidelines

Note

Ampro does not recommend the use of ribbon cables for bus expansion in production configurations. If cables are unavoidable, the following guidelines apply.

There are restrictions when attaching peripherals to the expansion bus with ribbon cables. If cables are too long or improperly terminated, noise and cross-talk introduced by the ribbon cables can cause errors. Ampro strongly recommends that you conform to the following guidelines:

Cable Length and Quality—In general, keep the bus expansion cable as short as possible. Long cables reduce system reliability.

- Do not use cables longer than 6 inches.
- Carefully measure signal quality on each bus line. You may need to add termination to correct signal degradation.

Backplane Quality—If you connect a backplane to the CoreModule/3SX*i* CPU, be sure to use a high quality backplane that minimizes signal crosstalk. Use a backplane that has power and ground planes between trace layers, and run guard traces between sensitive bus signals.

Eliminating Reset and TC Noise—Many cards have asynchronous TTL logic inputs that are susceptible to noise and crosstalk. The active high RESET and TC bus lines are especially vulnerable. You can make these signals more reliable by adding a 200 pF to 500 pF capacitor between the signal and ground to prevent false triggering by filtering noise on the signals. These RESET and TC filters are included on most Ampro backplane expansion products.

Bus Termination

Some backplanes include bus termination to improve system reliability by matching backplane impedance to the rest of the system. The IEEE-P996 draft specification for the AT expansion bus recommends the use of AC termination (sometimes called "snubbers") rather than resistive termination. The recommended AC termination is a 50 to 100 pF capacitor, in series with a 50 to 100 ohm resistor, from each signal to ground. Ampro provides positions for OEM addition of AC termination on most bus expansion products. These positions are designed to accommodate 9-pin 8-terminator Single Inline Package (SIP) terminators.

Here are some manufacturer part numbers for 9-pin, eight-terminator devices with 100 pF capacitors in series with 100 ohm resistors:

- Dale CSRC-09C30-101J-101M
- Bourns 4609H-701-101/101

Caution

Do not use resistive bus termination! If the signal requires termination, use AC termination only.

The actual requirements for signal termination depend on system configuration, interconnecting bus cable, and on the number and type of expansion modules used. It is the system engineer's responsibility to determine the need for termination.

For engineering development purposes, you can expand a CoreModule/3SX*i* system by connecting short ribbon cables to the header connectors. Ampro makes a small ribbon cable connector assembly, the *Double Stackthrough (DST) Cable Adapter*, that you can use to connect standard ribbon cables to the female expansion bus connectors on the CoreModule/3SX*i* CPU. Contact your Ampro sales representative for more information about the DST Cable Adapter.

2.13.4 Expansion Bus Connector Pinouts

Table 2–31 through Table 2–34 show the pinout and signal functions on the PC/104-compliant expansion bus connectors.

The CoreModule/3SX*i* CPU does not generate ±12VDC or -5VDC for the expansion bus. If devices on the bus require these voltages, they can be supplied to the bus connector from the Power Connector (J7).

You do not need to add a +12V supply to program Flash EPROMs installed in the byte-wide socket, or for the onboard Flash device. An onboard supply provides the programming voltage. However, this supply does not provide power to the expansion bus. Most Ampro expansion products provide onboard DC-to-DC converters to convert the +5V supply to other voltages they require.

The expansion bus pin numbers shown in the following tables correspond to the scheme normally used on ISA expansion bus card sockets. Rather than numerical designations (1, 2, 3) they have alpha-numeric designations (A1, A2..., B1, B2..., etc.)

Table 2–31. AT Expansion Bus Connector, A1-A32 (P1)

Pin	Signal Name	Function	In/Out	Drive Level	PU/PD/S *
A1	IOCHCK*	bus NMI input	IN	N/A	
A2	SD7	Data bit 7	I/O	6 mA	4.7K PU
A3	SD6	Data bit 6	I/O	6 mA	4.7K PU
A4	SD5	Data bit 5	I/O	6 mA	4.7K PU
A5	SD4	Data bit 4	I/O	6 mA	4.7K PU
A6	SD3	Data bit 3	I/O	6 mA	4.7K PU
A7	SD2	Data bit 2	I/O	6 mA	4.7K PU
A8	SD1	Data bit 1	I/O	6 mA	4.7K PU
A9	SD0	Data bit 0	I/O	6 mA	4.7K PU
A10	IOCHRDY	Processor Ready Ctrl	IN	N/A	1K PU
A11	AEN	Address Enable	I/O	12 mA	
A12	SA19	Address bit 19	I/O	6 mA	
A13	SA18	Address bit 18	I/O	6 mA	
A14	SA17	Address bit 17	I/O	6 mA	
A15	SA16	Address bit 16	I/O	6 mA	
A16	SA15	Address bit 15	I/O	6 mA	
A17	SA14	Address bit 14	I/O	6 mA	
A18	SA13	Address bit 13	I/O	6 mA	
A19	SA12	Address bit 12	I/O	6 mA	
A20	SA11	Address bit 11	I/O	6 mA	
A21	SA10	Address bit 10	I/O	6 mA	
A22	SA9	Address bit 9	I/O	6 mA	
A23	SA8	Address bit 8	I/O	6 mA	
A24	SA7	Address bit 7	I/O	6 mA	
A25	SA6	Address bit 6	I/O	6 mA	
A26	SA5	Address bit 5	I/O	6 mA	
A27	SA4	Address bit 4	I/O	6 mA	
A28	SA3	Address bit 3	I/O	6 mA	
A29	SA2	Address bit 2	I/O	6 mA	
A30	SA1	Address bit 1	I/O	6 mA	
A31	SA0	Address bit 0	I/O	6 mA	
A32	GND	Ground	N/A	N/A	

* PU = pull up; PD = pull down; S = resistance in series. All values in ohms.

Table 2–32. AT Expansion Bus Connector, B1-B32 (P1)

Pin	Signal Name	Function	In/Out	Drive Level	PU/PD/S *
B1	GND	Ground	N/A	N/A	
B2	RESETDRV	System reset signal	OUT	12 mA	
B3	+5V	+5 Volt power	N/A	N/A	
B4	IRQ9	Interrupt request 9	IN	N/A	27K PU
B5	-5V	To J16-3	N/A	N/A	
B6	DRQ2	DMA request 2	IN	N/A	
B7	-12V	To J16-1	N/A	N/A	
B8	ENDXFR*	Zero wait state	IN	N/A	
B9	+12V	To J10-1	N/A	N/A	
B10	Key	Key pin	N/A	N/A	
B11	SMEMW*	Mem Write(lwr 1MB)	I/O	12 mA	33 S
B12	SMEMR*	Mem Read(lwr 1MB)	I/O	12 mA	33 S
B13	IOW	I/O Write	I/O	8 mA	33 S, 10K PU
B14	IOR	I/O Read	I/O	8 mA	33 S, 10K PU
B15	DACK3*	DMA Acknowledge 3	OUT	6 mA	
B16	DRQ3	DMA Request 3	IN	N/A	
B17	DACK1*	DMA Acknowledge 1	OUT	6 mA	
B18	DRQ1	DMA Request 1	IN	N/A	
B19	REFRESH*	Memory Refresh	I/O	24 mA	33 S,4.7K PU
B20	SYSCLK	Sys Clock	OUT	12 mA	
B21	IRQ7	Interrupt Request 7	IN	N/A	27K PU
B22	IRQ6	Interrupt Request 6	IN	N/A	27K PU
B23	IRQ5	Interrupt Request 5	IN	N/A	27K PU
B24	IRQ4	Interrupt Request 4	IN	N/A	27K PU
B25	IRQ3	Interrupt Request 3	IN	N/A	27K PU
B26	DACK2*	DMA Acknowledge 2	OUT	6 mA	
B27	TC	DMA Terminal Count	OUT	12 mA	
B28	BALE	Address latch enable	OUT	12 mA	
B29	+5V	+5V power	N/A	N/A	
B30	OSC	14.3 Mhz clock	OUT	6 mA	33 S
B31	GND	Ground	N/A	N/A	
B32	GND	Ground	N/A	N/A	

* PU = pull up; PD = pull down; S = resistance in series. All values in ohms.

Table 2–33. AT Expansion Bus Connector, C0-C19 (P2)

Pin	Signal Name	Function	In/Out	Drive Level	PU/PD/S *
C0	GND	Ground	N/A	N/A	
C1	SBHE	Bus High Enable	I/O	12 mA	
C2	LA23	Address bit 23	I/O	24 mA	
C3	LA22	Address bit 22	I/O	24 mA	
C4	LA21	Address bit 21	I/O	24 mA	
C5	LA20	Address bit 20	I/O	24 mA	
C6	LA19	Address bit 19	I/O	24 mA	
C7	LA18	Address bit 18	I/O	24 mA	
C8	LA17	Address bit 17	I/O	24 mA	
C9	MEMR*	Memory Read	I/O	12 mA	33 S, 10K PU
C10	MEMW*	Memory Write	I/O	12 mA	33 S, 10K PU
C11	SD8	Data Bit 8	I/O	12 mA	4.7K PU
C12	SD9	Data Bit 9	I/O	12 mA	4.7K PU
C13	SD10	Data Bit 10	I/O	12 mA	4.7K PU
C14	SD11	Data Bit 11	I/O	12 mA	4.7K PU
C15	SD12	Data Bit 12	I/O	12 mA	4.7K PU
C16	SD13	Data Bit 13	I/O	12 mA	4.7K PU
C17	SD14	Data Bit 14	I/O	12 mA	4.7K PU
C18	SD15	Data Bit 15	I/O	12 mA	4.7K PU
C19	Key	Key Pin	N/A	N/A	

* PU = pull up; PD = pull down; S = resistance in series. All values in ohms.

Table 2–34. AT Expansion Bus Connector, D0-D19 (P2)

Pin	Signal Name	Function	In/Out	Drive Level	PU/PD/S *
D0	GND	Ground	N/A	N/A	
D1	MEMCS16*	16-bit Mem Access	IN	N/A	330 PU
D2	IOCS16*	16-bit I/O Access	IN	N/A	330 PU
D3	IRQ10	Interrupt Request 10	IN	N/A	27K PU
D4	IRQ11	Interrupt Request 11	IN	N/A	27K PU
D5	IRQ12	Interrupt Request 12	IN	N/A	27K PU
D6	IRQ15	Interrupt Request 15	IN	N/A	27K PU
D7	IRQ14	Interrupt Request 14	IN	N/A	27K PU
D8	DACK0*	DMA Acknowledge 0	OUT	6mA	
D9	DRQ0	DMA Request 0	IN	N/A	
D10	DACK5*	DMA Acknowledge 5	OUT	6mA	
D11	DRQ5	DMA Request 5	IN	N/A	
D12	DACK6*	DMA Acknowledge 6	OUT	6mA	
D13	DRQ6	DMA Request 6	IN	N/A	
D14	DACK7*	DMA Acknowledge 7	OUT	6mA	
D15	DRQ7	DMA Request 7	IN	N/A	
D16	+5V	+5 Volt Power	N/A	N/A	
D17	MASTER*	Bus Master Assert	IN	N/A	330 PU
D18	GND	Ground	N/A	N/A	
D19	GND	Ground	N/A	N/A	

* PU = pull up; PD = pull down; SER = resistance in series. All values in ohms.

2.13.5 Interrupt and DMA Channel Usage

The AT bus provides several interrupt and DMA control signals. When you expand the system with MiniModule products or plug-in cards that require either interrupt or DMA support, you must select which interrupt or DMA channel to use. Typically this involves switches or jumpers on the module. In most cases, these are not shared resources. It is important that you configure the new module to use an interrupt or DMA channel not already in use. For your convenience, Table 2–35 and Table 2–36 provide a summary of the normal interrupt and DMA channel assignments on the CoreModule/3SX*i* CPU.

Table 2–35. Interrupt Channel Assignments

Interrupt	Function
IRQ0	ROM BIOS clock tick function, from Timer 0 *
IRQ1	Keyboard interrupt *
IRQ2	Cascade input for IRQ8-15 *
IRQ3	Serial 2
IRQ4	Serial 1
IRQ5	Available
IRQ6	Floppy controller
IRQ7	Parallel port
IRQ8	Reserved for battery-backed clock alarm *
IRQ9	Available
IRQ10	Available
IRQ11	Available
IRQ12	Available
IRQ13	Available
IRQ14	IDE hard disk controller
IRQ15	Available
* Unavailable on the PC/104 bus.	

Table 2–36. DMA Channel Assignments

Channel	Function
0	Available for 8-bit transfers
1	Available for 8-bit transfers
2	Floppy controller
3	Available for 8-bit transfers
4	Cascade for channels 0-3
5	Available for 16-bit transfers
6	Available for 16-bit transfers
7	Available for 16-bit transfers

CHAPTER 3

OPERATION

3.1 INTRODUCTION

This chapter provides the information you need to software configure your CoreModule/3SX*i* CPU. The first section describes the SETUP function. It describes each option that can be set using SETUP. Additional sections describe important options you can set for each major functional block of the board.

Note

The SETUP descriptions in the following section also contain much useful information about each SETUP topic. Review these sections even if you already know how to set the SETUP parameters.

This chapter presumes you have some familiarity with DOS (PC-DOS, MS-DOS, or DR DOS). It does not attempt to describe the standard DOS and ROM BIOS functions. Refer to the appropriate DOS and PC reference manuals for information about DOS, its drivers and utilities, and about the software interface of the onboard ROM-BIOS. Where Ampro has added to or modified standard functions, these will be described.

The Ampro Common Utilities manual contains detailed descriptions of the Ampro utility programs supplied on the Utility diskette that is included with the CoreModule/3SX*i* CPU Development Kit.

3.2 SETUP OVERVIEW

Many options provided on the CoreModule/3SX*i* CPU are controlled by the SETUP function. You have access to these options when you activate the SETUP function. The parameters are displayed on four screens. To configure the board, you modify the fields on these screens and save the results in the onboard *configuration memory*. The configuration memory consists of portions of the CMOS RAM in the battery-backed real-time clock chip and an Ampro-unique configuration EEPROM. To enhance embedded-system reliability, the contents of the EEPROM mirror the contents of the CMOS memory. The EEPROM retains your configuration information even if the clock's backup battery should fail. If you choose to use the CoreModule/3SX*i* CPU without a battery, the system takes its SETUP parameters from the EEPROM, providing battery-free operation.

The SETUP information is retrieved from configuration memory when the board is powered up or when it is rebooted with a CTL-ALT-DEL key pattern. Changes made to the SETUP parameters (with the exception of the real-time clock time and date settings) do not take effect until the board is rebooted.

The SETUP function is located in the ROM BIOS. It can be accessed using CTRL-ALT-ESC while the computer is in the Power On Self Test (POST), just prior to booting up. This is called *hot key* access. The screen will display a message indicating when you can enter CTRL-ALT-ESC. You may also enter the SETUP function from the DOS command line using the SETUP.COM program provided on the Ampro Common Utilities diskette .

Table 3–1 summarizes the choices found on each SETUP page.

Table 3–1. Functions on Each SETUP Page

Page	Menu Name	Functions
1	Standard (CMOS/EEPROM) Configuration	Set date and time Define floppy drives Define IDE hard disks Select video type Display DRAM quantity Set error halt conditions Enable/disable video shadow RAM Set POST display option
2	Options/Peripheral Configuration	Enable/disable extended BIOS functions Enable/disable APM BIOS functions Enable/disable serial ports Enable/disable/configure parallel port Enable/disable floppy interface Enable/disable IDE interface Configure Mono/Color Enable/disable hot key access to SETUP Set video display state Select POST display option Configure byte-wide and OEM Flash memories Enable/disable serial boot loader Enable/disable watchdog timer
3	Extended SCSI and Hard Disk configuration	Set SCSI controller parameters Configure SCSI disk map Select floppy or hard disk boot Configure DOS disk map
4	Extended Serial Console Configuration	Configure serial port parameters for serial console output Configure serial port output handshake option Configure serial port parameters for serial console input Delete/include console port from DOS COM table
* SETUP pages 3 and 4 are available when you enable Extended BIOS from SETUP		

Note

Some SETUP options can put your system into an unrecoverable state. For instance, you might set a display option that prevents you from seeing the SETUP screens. Installing a jumper between J3-7 and J3-8 (Serial 1 DTR and RI) temporarily sets all SETUP functions to their default state, bypassing the SETUP parameters stored in the configuration memory so that you can reenter SETUP and correct the problem.

3.3 SETUP PAGE 1—STANDARD (CMOS) SETUP

The first SETUP page contains the parameters normally saved in CMOS RAM plus some additional parameters unique to the CoreModule/3SX*i* CPU. The only parameters not also saved in the EEPROM memory are the real-time clock date and time. If no battery is used or if the battery fails, the date and time will not be accurate. All other parameters are saved in the EEPROM.

Figure 3–1 shows what can be configured using SETUP page 1. Sections following the figure describe each option.

```
                    Standard  (CMOS/EEPROM)  Setup

Date  (mm/dd/yy)    9/20/96              Time  (hh:mm:ss)    10:08:00

1st Floppy          1.4M
2nd Floppy          1/2M
                         Cyls   Heads   Sectors   Precomp   Landzone
ATA/IDE Disk 1      17   655    14      17        0         0
ATA/IDE Disk 2      None

Video               EGA/VGA

Base Memory         640
Extended Memory     1024

Error Halt          NO HALT ON ANY ERROR
Video Shadow RAM    Enabled
System POST         Normal

          PgDn or (D)own for Extended Setup
   ↑ ↓ [Enter] Moves Between Items, ←→ + - Selects Values
(E)xit to quit without change, or (S)ave to record changes
```

Figure 3–1. SETUP Page 1

3.3.1 Date and Time

The time shown on the first SETUP screen is continuously updated and reflects the current state of the hardware real-time clock. The new time and date that you enter is immediately written to the device. Enter the date in the form *mm/dd/yy*. Enter the time in 24-hour format, in the form *hh:mm:ss*.

The ROM BIOS maintains the *system* real-time clock. It is incremented approximately 18.2 times per second by an interrupt from timer/counter 0. The ROM BIOS automatically initializes the *system* real-time clock from the *hardware* real-time clock upon system reset or power up. The accuracy of the hardware real-time clock depends, of course, on your connecting a battery to the appropriate terminals on J5, the Utility connector. If no battery is attached, the system time information will not remain accurate after a power cycle.

3.3.2 Floppy Drives

The ROM BIOS supports all of the popular DOS-compatible floppy disk formats. This includes all the 5-1/4 inch and 3-1/2 inch floppy formats—360K, 720K, 1.2M, and 1.44M. (Note: some formats are not supported by early versions of DOS.) In addition, the ROM BIOS supports dual-capacity use of high density floppy drives. That is, you can read and boot from 360K floppies in a 1.2M 5-1/4 inch drive, and from 720K floppies in a 1.44M 3-1/2 inch drive.

Drive Parameter Setup

Enter the number and type of floppy drives in the system. If the drives connected to the system do not match the parameters in the configuration memory, POST displays an error message. To eliminate the error message, set the drive parameters to match your floppy drives.

3.3.3 IDE Hard Disk Drives

The ROM BIOS supports one or two hard disk drives connected to the IDE interface. The BIOS allows you to mix IDE drives in combination with SCSI hard disk drives. (Use the IDE SETUP parameters for IDE drives only. SCSI hard drives are configured on SETUP screen 3.)

The IDE SETUP parameters are used for setting the physical parameters of the drives you install in your system. Physical drives can have one or more logical partitions. You can install up to eight *logical* drives or drive partitions, but only two physical drives. (Older versions of DOS may limit the number of logical drives you can install.)

To configure the system for one or two IDE drives, set the drive parameters with SETUP, as outlined here:

- **Drive Types**—The configuration memory contains a default list of parameters that specify the physical format of each drive. Each *type* specifies the total number of cylinders, the number of sectors per cylinder, number of heads, cylinder to begin precompensation, and landing zone cylinder number. The drive manufacturer supplies these parameters. The list contains "legacy values", standard for PCs—a number of older (smaller) drives are defined.

  Two special drive types, 48 and 49, let you enter drive parameters manually. If no built-in drive type matches your drive, select drive type 48 or 49 and enter the drive parameters in the fields provided.

  Drive type **AUTO** selects **Autoconfigure**. Autoconfigure queries the drive for its parameters. Most modern drives will respond to the query, allowing the BIOS to set the drive parameter values automatically. This option also provides Logical Block Addressing (LBA) capability, which is used to support drives larger than 512M bytes.

Note

LBA uses a translation scheme to convert physical heads, sectors and cylinders to logical block numbers. Due to differences in the translation schemes used by different system BIOSes, LBA-compatible drives that have been formatted on Ampro systems may not function properly in other systems that support LBA mode. However, due to the intelligent translation algorithm in the Ampro BIOS, drives formatted in other systems are likely to be usable on the CoreModule/3SX*i* CPU. Note that this only applies to IDE drives that support LBA mode. Consult the technical literature for your specific drive to find out if it supports LBA mode.

- **Drive Selection**—Besides specifying the physical characteristics of each IDE drive, you also must specify how they are to be used by the ROM BIOS. Two factors control how they are used, drive number jumper(s) and the DOS disk map.

  1. An IDE drive can be jumpered as a **master** or **slave**. Each manufacturer's drive is different, so you must refer to the drive's technical literature to find out how to jumper the drives you install. Drives default to **master** from the factory, so if you only have one IDE drive in a system it is generally already set up properly.
  2. Use the SETUP Extended SCSI and Hard Disk Configuration menu (SETUP page 3) to enter your IDE drive(s) in the DOS disk map. Disk 1 in the map will be logged by DOS as drive C, Disk 2 as drive D, and so on. See the description of SETUP page 3 for details.

Once you have set the system's configuration memory, the IDE drive(s) can be formatted and otherwise prepared normally. Refer to your operating system and disk drive documentation for specific procedures and requirements.

3.3.4 Video

Specify the initial video mode. Select **Mono**, **Color40**, **Color80**, or **EGA/VGA**. If your video display card is VGA, super VGA, or any other high resolution standard, specify **EGA/VGA** no matter how it is configured to come up.

3.3.5 DRAM Memory

The ROM BIOS automatically sets the amount of memory it discovers during Power-On Self-Test (POST) and stores the result when you save the configuration values when exiting SETUP. If you change the amount of memory installed on the board, however, you must run SETUP and do a **save** when you exit. This updates the configuration memory to reflect the new memory size. Until you do this, an error message will appear during POST.

Note that if an error message appears during POST when you have not changed the amount of memory installed, it indicates that at least part of the memory is not functioning properly.

3.3.6 Error Halt

Select which kinds of errors will halt the POST. If you plan to use the module without a keyboard, be sure to set this option to *not* halt on keyboard error.

3.3.7 Video Shadow RAM

This option, when enabled, allows the ROM BIOS to copy the contents of a video BIOS into DRAM. The actual video BIOS ROM on the video controller is disabled, and DRAM is mapped into the address space it occupied. This speeds up video BIOS accesses. Ampro video controllers are designed to allow video BIOS shadowing. If you are using a video controller from another manufacturer, it may not support shadowing. In that case, set video BIOS shadowing to "Disabled."

3.3.8 System POST

At boot time, the BIOS runs a series of tests called the "Power On Self Test", or POST. There are options in the Ampro BIOS to customize the POST to control how fast the computer powers up and to control what the user sees at power up time. The choices are:

- **Normal**—Displays the results of all tests
- **Fast**—Faster than Normal POST because it uses a shorter memory test
- **Express**—Skips most tests and does not display POST test results on the screen

3.4 SETUP PAGE 2—OPTIONS/PERIPHERAL CONFIGURATION

Use SETUP page 2 to enable or disable many of the functions and peripherals provided on the CoreModule/3SX*i*CPU. Figure 3–2 shows what can be configured on SETUP page 2, and the sections that follow describe each parameter.

```
          CM/3SXi Options/Peripheral Configuration

CoreModule Extended BIOS ..........Enabled
Advanced Power Mgmt BIOS ..........Enabled
Serial Port 1 .....................Enabled
Serial Port 2 .....................Enabled
Parallel Port .....................Enabled  Mode SPP
Floppy Interface ..................Enabled
IDE Interface .....................Enabled
Mono/Color Jumper .................Color

Socket S0 .........................64K @ D0000h
OEM Flash .........................64K @ E0000h
Default Socket ....................S0

Video State .......................Enabled
Blank POST Test ...................Enabled
Serial Boot Loader ................Disabled
Watchdog Timer ....................Disabled
Hot Key Setup .....................Enabled

            (S)ave to Record Extended Setup
↑ ↓ [Enter] Moves Between Items, ←→ + - Selects Values
(E)xit to quit without change, or (S)ave to record changes
```

Figure 3–2. SETUP Page 2

3.4.1 Extended BIOS

Normally, the Ampro Extended BIOS is enabled. This allows access to SETUP pages three and four and the features they define. If you do not want to use the BIOS extensions, you can disable them using this parameter. (Some UNIX implementations or other operating systems may require disabling the extended portion of the BIOS.) Ampro Application Note AAN-9210 documents the features in the extended BIOS, including the application program interface specifications.

3.4.2 Advanced Power Management BIOS

The CoreModule/3SX*i* CPU BIOS incorporates an Advanced Power Management BIOS (APM) compliant with Advanced Power Management (APM) BIOS Interface Specification Revision 1.1, created by Intel and Microsoft. This SETUP option allows you to enable or disable access to the APM BIOS functions.

Note that this option does not enable or disable power management on the CoreModule, it enables or disables access to the APM BIOS that drivers or applications use to control power management features.

3.4.3 Serial Ports

Use SETUP to independently enable or disable either of the two onboard serial ports. (When you use SETUP to enable or disable a port, the change does not take effect until you reboot the system.)

The I/O addresses and interrupt assignments (IRQs) for the serial ports cannot be changed. The following table lists the I/O addresses and IRQs of each port. These resources are freed for use by other peripherals installed on the PC/104 bus when their respective ports are disabled.

Table 3–2. Serial Port Resources

Port	Address	Interrupt
Serial 1	3F8h – 3FFh	IRQ4
Serial 2	2F8h – 2FFh	IRQ3

Normally, the BIOS logs Serial 1 and Serial 2 as COM1 and COM2. Note, however, that COM1 and COM2 are logical designations, not physical values. When the system boots, the BIOS scans the standard serial port addresses and installs the first port it finds as COM1. If it finds a second port, it installs that one as COM2, and so on. If you disable a serial port, the designations of all higher-numbered COM ports will change.

For more information about the serial ports, see Serial Ports, page 3—19.

3.4.4 Parallel Port

You enable or disable the CoreModule/3SXi parallel port using the **Parallel Port** option on this SETUP page. You set the parallel port mode (SSP, EPP, or ECP) by setting the **Mode** option.

Table 3–3 summarizes the resources that are used when the parallel port is enabled.

Table 3–3. Parallel Port Resources

Selection	I/O Address	Interrupt
Primary	0378h - 037Fh	IRQ7
Disable	None	None

The I/O ports and interrupt request channel are freed for use by other peripherals installed on the PC/104 bus when the parallel port is disabled.

Normally, the BIOS logs in the primary and secondary parallel ports as LPT1 and LPT2. Note, however, that LPT1 and LPT2 are logical designations, not physical values. When the system boots, the BIOS scans the standard parallel port addresses and installs the first port it finds as LPT1. If it finds a second port, it installs that one as LPT2, and so on. If you disable a parallel port, the designations of all higher-numbered LPT ports will change.

Setting the Parallel Port Mode

Set the parallel port mode to either **SPP**, **EPP**, or **ECP**.

Table 3–4 Parallel Port Modes

Mode	Description
SPP	Standard Parallel Port (default)—Bi-directional, compatible with standard and PS/2 ports.
EPP	Enhanced Parallel Port—Bi-directional, compatible with standard and PS/2 ports, but adding automatic read- and write- cycle modes.
ECP	Extended Capabilities Port—IEEE-1284-compliant port. Provides interlocking handshaking, 16-byte FIFO buffer, optional DMA transfer capability, and optional RLE data compression.

For more information about the parallel port, see Enhanced Parallel Port on page 3–23.

3.4.5 Floppy Interface Enable

Enable or disable the onboard floppy interface. When disabled, the I/O ports assigned to the floppy controller become available, allowing them to be used by other devices installed on the expansion bus. Table 3–5 lists the resources used by the floppy controller.

Table 3–5. Floppy Controller Resources

Selection	I/O Address	IRQ	DMA
Enabled	03F2h Digital Output Register 03F4h Main Status Register 03F5h Data Register 03F7h Control Register	IRQ6	DMA 2
Disable	None	None	None

3.4.6 IDE Interface Enable

Enable or disable the onboard IDE hard disk interface. When disabled, the I/O ports and IRQ assigned to the IDE controller become available, allowing them to be used by other devices installed on the expansion bus. Table 3–6 lists the resources used by the IDE interface.

Table 3–6. IDE Controller Resources

Selection	I/O Address	Interrupt
Enabled	01F0h - 01F7 Control and Data Registers 03F7h Shared with FDC	IRQ14
Disable	None	None

If you have an IDE drive attached to J6, just disabling the IDE interface will not free the interrupt, IRQ14, since it is connected directly to the drive. You must disconnect the cable.

3.4.7 Mono/Color Selection

Set the Mono/Color selection to Mono only if you have a monochrome monitor connected to a monochrome (MDA) video adapter. In all other cases, set this option to Color. Set it to Color even if you have a VGA monochrome monitor attached to a VGA or SuperVGA adapter.

3.4.8 Hot Key Setup Enable

In some embedded systems, you do not want an end-user to use the *hot-key* sequence (CTRL-ALT-ESC) to enter SETUP. You can enable or disable hot-key access to SETUP with this parameter. (This also prevents "+++" from entering SETUP when using the serial console feature.)

3.4.9 Video State

You can set this option to Enabled or Inhibited. Inhibited ***blanks*** the display until your program makes a call to the Video Restore State function in the video BIOS (via INT10h). This provides a means of controlling what appears on the screen when the system starts up. This option can be used to inhibit the POST test display and everything else that DOS or an application would display, until a call is made to the video BIOS.

The following is an example of code that reenables the display inhibited by this option:

```
;=============================================================================

init:   mov ah,1ch
        mov al,-1
        mov bx,414Dh
        mov cx,5052h
        int 10h

;=============================================================================
```

3.4.10 Blank Post Test

Enable or disable POST display. If set to **Disabled**, the messages from the POST will not be sent to the console. To inhibit display of a broader range of system and application messages, see Video State, above.

3.4.11 Byte-Wide Socket and OEM Flash Memory Configuration

The byte-wide socket, S0, and the user portion of the OEM Flash memory device can be independently configured for its *starting address* and the *size* of the memory block in which it appears to the processor, or it can be disabled. You can also specify which device is enabled at boot time. (This is the "Default Socket" SETUP option.) Note that only one can be enabled at boot time.

Table 3–7 lists the socket address configuration options that are available.

Table 3–7. Byte-Wide Memory and Onboard Flash Configuration

Size	Address
Disabled	None
64K bytes	D0000h – DFFFFh
64K bytes	E0000h – EFFFFh
128K bytes (Available only for S0)	D0000h – EFFFFh

If you configure both devices to occupy the same address space (or overlap), only one device will be visible to the CPU. This will be either the default socket (enabled during POST) or the socket that was last enabled. The Ampro BIOS provides a call for enabling and disabling each device. A code example is shown on page 3–28. Refer to Ampro Application Note AAN-9210 for a complete description of the BIOS functions that control the byte-wide sockets.

Devices larger than 64K can be installed in the byte-wide socket, independent of the memory block size setting. The memory block size setting specifies a "window" in which the memory device is visible. You can use an extended BIOS call to select which 64K page of the byte-wide device is visible to the processor. A code example is shown on page 3–29.

You must also set hardware jumpers to configure the byte-wide socket for the device you install in S0. Refer to Chapter 2 for jumper positions. If you are using the byte-wide socket for Solid State Disk (SSD), using Ampro's Solid State Disk software, follow the directions for setting the byte-wide socket that are in the SSD Technical Manual.

3.4.12 Serial Boot Loader Enable

This parameter enables or disables the Serial Boot Loader option in the Ampro ROM BIOS. The serial boot loader allows you to boot from either of the onboard serial ports, much in the same way you would boot from a local hard disk or from a LAN. A description of the Serial Boot Loader is provided in the Ampro Common Utilities manual (see SERLOAD and SERPROG), and in Ampro Application Note AAN-9403. If you are not using the Serial Boot Loader, set this parameter to "Disabled."

3.4.13 Watchdog Timer Configuration

This parameter allows you to set the time duration of the watchdog timer for monitoring the boot process. You can set it to 30, 60, or 90 seconds, or you can disable it.

Further information about the watchdog timer can be found later in this chapter under "Watchdog Timer." A description of the WATCHDOG utility program can be found in the Ampro Common Utilities manual.

3.5 SETUP PAGE 3—SCSI HARD DISK

A unique feature of the CoreModule/3SX*i* CPU is that its ROM BIOS contains hard disk support functions that allow easy integration of SCSI and IDE drives. Use this SETUP screen to configure for your hard disk drives and other SCSI peripherals. Figure 3–3 shows SETUP screen 3, and descriptions of each field are provided in sections below.

```
          Extended SCSI and Hard Disk Configuration

SCSI /BIOS Services ..... Enabled
SCSI Initiator ID ..... 7
SCSI Disk I/O Retries ..... 10
                SCSI Disk Map
SCSI 1 ..... Id 0, Lun 0
SCSI 2 ..... Not Active
SCSI 3 ..... Not Active
SCSI 4 ..... Not Active
SCSI 5 ..... Not Active
SCSI 6 ..... Not Active
SCSI 7 ..... Not Active
                DOS Disk Map
Default Boot Device ..... Floppy
1st Hard Disk ..... SCSI Disk 1
2nd Hard Disk ..... IDE Disk 1
3rd Hard Disk ..... Not Active
4th Hard Disk ..... Not Active
5th Hard Disk ..... Not Active
6th Hard Disk ..... Not Active
7th Hard Disk ..... Not Active
8th Hard Disk ..... Not Active

↑ ↓ [Enter] Moves Between Items, ←→ + - Selects Values
(E)xit to quit without change, or (S)ave to record changes
```

Figure 3–3. SETUP Page 3

With the Ampro Extended BIOS, SCSI hard disks are available to DOS through standard ROM BIOS functions (INT 13). (SCSI interface hardware is available on a variety of Ampro add-on products.) SCSI functions are in the SCSI BIOS portion of the ROM BIOS. The ROM BIOS hard disk support allows direct system booting from SCSI Common Command Set direct access devices. Other types of SCSI direct access devices can be used to provide a compatible hard disk function. These include CD ROM drives, tape drives, SCSI RAM disks, and other peripherals.

Most DOS or Windows applications run normally in this SCSI-based hard disk environment. Programs nearly always use either DOS or ROM BIOS functions for disk drive access. It is rare for software to attempt to access hard disk controller hardware directly.

Utilities for SCSI drive formatting, and other SCSI functions are included on the Ampro Common Utilities diskette. These are described in the Ampro Common Utilities manual.

3.5.1 SCSI Drive Parameter Setup

If you add a SCSI device to your CoreModule/3SX*i*-based embedded system, you must set several SCSI drive parameters in the configuration memory using SETUP. This section describes the SETUP parameters found on the SCSI Disk Configuration screen.

SCSI Controller Parameters

- **SCSI/BIOS Services**—To use the SCSI BIOS for hard disks, it must be enabled. When disabled, the system will not boot from an attached SCSI drive, nor will standard disk-related BIOS calls (INT13) be able to see a drive. SCSI services are still available from BIOS calls in your program, even when SCSI/BIOS Services is disabled. Disabling the SCSI BIOS services will speed up system booting when you don't use the SCSI port.

- **SCSI Initiator ID**—The Ampro CoreModule/3SX*i* CPU is the *SCSI Initiator* in its transactions with SCSI target devices such as hard disk drives. Every SCSI device (target or initiator) must have a unique ID between 0 and 7. The default ID for the SCSI controller on the CoreModule/3SX*i* CPU is 7. It is the highest priority ID, and this ID tells the SCSI BIOS to reset the SCSI bus on system power up or reset. In most cases you will not change the default SCSI initiator ID.

- **SCSI Disk I/O Retries**—You can specify the number of read/write retries when using SCSI drives as DOS drives. The default is 10 retries.

SCSI Disk Map

- **Target Device IDs and LUNs**—The specification of SCSI target device IDs and Logical Unit Numbers (LUNs) are stored in the configuration memory. Enter the IDs and LUNs of the SCSI drives you have installed in your system. Assign each drive to a SCSI Disk position in the SCSI Disk Map. Normally, all SCSI LUNs default to 0.

  The SCSI ID for target devices can be 0 to 6 (since the CPU is set for ID 7). A device's ID is usually set by jumpers or switches on the device. If you have multiple SCSI drives, assign each one a unique device ID.

DOS Disk Map

- **BOOT Device Specification**—You can choose to boot the system from a hard or floppy drive using the Default Boot Device parameter. You can specify Floppy for floppy A: or Hard Disk for drive C:. (When you select Hard Disk, the drive shown as 1st Hard Disk on the DOS Disk Map becomes the boot drive.)

- **DOS Disk Map**—Assign your disk drives, both IDE drives and SCSI drives, to positions on the DOS Disk Map. You can assign them in any order and in any mix. The 1st Hard Disk becomes drive C:, 2nd Hard Disk becomes drive D:, and so on. Any non-SCSI devices that will appear to the system as a drive should be configured in the DOS Disk Map as an AT Bus Drive. This includes any device that is installed with its own driver.

Note

SETUP screen 1, "Standard (CMOS) SETUP" is used for defining hard drives connected to the IDE interface. Do not attempt to use the IDE configuration menu to define disk drive parameters for SCSI-connected drives.

3.6 SETUP PAGE 4—SERIAL CONSOLE

The ROM BIOS includes a unique set of features which allow full access to the system at any time over standard RS232 serial ports. An embedded system may take advantage of these remote access capabilities using the serial console functions in the following ways:

Serial console—Use Serial 1 or Serial 2 as a console. Use a serial terminal to replace the standard video monitor and keyboard.

Serial boot loader—Boot from a serial port much like you would boot from a local hard disk or from a network. (This feature is enabled or disabled with the **Serial Boot Loader** option on Page 2 of SETUP.)

Serial programming—Automatically update system software, such as an SSD, through a serial port. This feature allows you to replace code in a Flash device installed in the byte-wide socket.

For more information about these serial console functions, see "Serial Console Features," under "Serial Ports", later in this chapter. For a thorough explanation of the remote host features, refer to Ampro Application Note AAN-9403.

Figure 3–4 shows the options you can set for the serial console. Since the DOS normally initializes the serial ports during boot, you have the option to remove the serial console port from DOS's COM port table. By doing this, the values you set on SETUP screen 4 will remain after you boot DOS.

```
                Extended Serial Console Configuration

                Console Output Device..... Video
                Console Input Device..... Keyboard

                Serial Console Output Setup
                  Data Length.....
                  Stop Bits.....
                  Parity.....
                  Baud.....
                  Delete from Com Port Table..
                Console Output Handshake.....

                Serial Console Input Setup....
                  Data Length.....
                  Stop Bits.....
                  Parity.....
                  Baud.....
                  Delete from Com Port Table..

          NOTE: When the Console Output Device is serial, the
                MONO/COLOR jumper must be removed and the
                Video in Standard Setup set to EGA/VGA.

      ↑ ↓ [Enter] Moves Between Items, ←→ + - Selects Values
                  PgUp or (U)p for previous page
```

Figure 3–4. SETUP Page 4

Console Output Device—Select the console output device, either Video, Serial 1, Serial 2, or None.

Console Input Device—Select the console input device, either the PC Keyboard, Serial 1, Serial 2, or None.

Serial Console Output Setup—Enter the communication parameters for your console *output* serial port. Set the data length, stop bits, parity, and baud rate to match your serial output device.

Console Output Handshake—Enable or disable hardware handshaking. If enabled, the DSR and CTS signals control the data flow. Be sure to connect the DSR and CTS signals on the serial port's connector to the appropriate handshake signals on the external serial device's interface connector.

Serial Console Input Setup—Enter the communication parameters for your console *input* serial port. Set the data length, stop bits, parity, and baud rate to match your serial input device.

Delete from COM Port Table—When DOS boots, it initializes the system serial ports. (Different versions of DOS may set the ports to different default settings.) By enabling this option, the BIOS does not include your console serial device(s) in the COM port table. This prevents DOS from changing the values you assign to the port in this SETUP screen.

Caution

Be careful when changing the console configuration. If you specify "None" for console input and output, there will be no console access to the system. (You can recover from this state by removing the serial console plug from the primary serial port connector and shorting pins J3-7/8.)

3.7 THE SETUP.COM PROGRAM

You can use the SETUP.COM utility from the command line to access the same SETUP functions as the "hot key" code, CTRL-ALT-ESC. SETUP.COM also adds additional functionality, such as the ability to load and store configuration settings to a disk file. This same feature is used to store up to 512 bits of OEM information in the configuration memory EEPROM. SETUP.COM is on the Ampro Common Utilities diskette, included with the CoreModule/3SX*i* CPU Development Kit.

3.7.1 Creating Configuration Files with SETUP.COM

The Ampro SETUP utility, SETUP.COM, offers the following options for command line entry:

```
SETUP [-switches]  [@file.ext | Wfile.ext]
```

The supported switches and their meaning are as follows:

Table 3–8. SETUP.COM Command Switches

Switch	Function
?	Display a usage help screen
T	Set the (hardware) real-time clock time and date from the current DOS time and date.
@file.ext	Writes the specified file to the board's CMOS RAM and configuration EEPROM. Drive and path are optional in the file name.
W*file.ext*	Write CMOS RAM and EEPROM contents to the file specified. The file name may contain an optional drive and path.

You can save a copy of the current contents of the board's configuration memory to a disk file by using the **W** switch. The data saved includes the entire contents of the nonvolatile configuration EEPROM. The first 512 bits are the SETUP information (excluding time and date). The next 512 bits are available for OEM storage. See Ampro Application Note AAN-8805 for a description of how to use the OEM storage portion of the EEPROM.

Note

If the SETUP is changed, the system must be rebooted before writing a configuration file using the SETUP W option. Otherwise, the changes will not appear in the setup file.

The file you create with this menu option can be used as a source for programming the configuration memory of a CoreModule/3SX*i* CPU at a later time.

For example, the following command initializes the EEPROM values with a previously saved configuration:

```
C>SETUP @SYSTEM.A
```

Assuming you created the file SYSTEM.A with SETUP's write option, SETUP will initialize the EEPROM configuration memory and CMOS RAM using the contents of SYSTEM.A.

Note

The system must be rebooted before new configuration information will take effect.

Using SETUP to save and load configuration memory parameters can be useful when many boards must be initialized automatically, for instance, during production, or when you want to change between several predefined system configurations.

3.8 OPERATION WITH DOS

The CoreModule/3SX*i* CPU supports IBM's PC-DOS or Microsoft's MS-DOS, Version 3.3 or later, or any version of Digital Research's DR DOS as the disk operating system. Any differences between these similar operating systems are noted in the text where applicable.

Caution

Sometimes MS-DOS is customized by a manufacturer for a specific system and may not work on the CoreModule/3SX*i* CPU. Use DR DOS (supplied by Ampro), IBM PC-DOS (supplied by IBM), or the generic version of MS-DOS (supplied by Microsoft on an OEM basis).

EMS Option—The CoreModule/3SX*i* CPU can emulate the Lotus-Intel-Microsoft Expanded Memory Specification Version 4.0 (LIM EMS 4.0), with the memory management capability of the 80386SX CPU, under control of a device driver. Such drivers are available with the newer versions of DOS. With Microsoft MS-DOS, the driver is called EMM386.EXE.

Serial Ports—DOS normally supports the board's two serial ports as COM1 and COM2.

At boot time, DOS initializes the serial ports, assigning them their COM port designations and their communication parameter settings. Although this might vary with different types and versions of DOS, typical communication parameter settings are 9600 baud, even parity, 7 bits, and 1 stop bit.

Usually an application program that uses a serial port will access the port's hardware and reinitialize the communication parameters to other values, based on settings that the user has entered when configuring the application program.

Parallel Port—The Parallel Printer port is normally the DOS LPT1 device. Most application software uses LPT1 as the default printer port. If you enable the port, printing to it is automatic.

The following DOS commands can be used to test printing with a parallel printer:

A>COPY *filename.ext* LPT1	Prints contents of *filename.ext*
A>DIR >LPT1	Prints the directory

In addition, the <PrtSc> (Print Screen) key will print the contents of the video screen to the LPT1 device. Also, you can use the Printer Echo function to print all characters typed on the keyboard. The command <Ctrl-P> enables the Printer Echo function. Entering <Ctrl-P> again disables Printer Echo.

Disk Drives—Older versions of DOS require you to divide disk drives larger than 32M bytes into more than one partition. More recent versions permit drives to be up to 2G bytes, though IDE drives are BIOS limited to 512M bytes. Drives larger than 512M bytes must use the **Auto** configuration type in SETUP or use a vendor supplied driver to access the entire drive.

3.9 SERIAL PORTS

This section describes uses for the serial ports on the CoreModule/3SX*i* CPU, including:

- Using the RS-485 interface
- The serial console feature

- Serial booting
- Serial downloading and programming
- Using a serial modem

3.9.1 Using the RS-485 Interface

This section describes the RS-485 interface circuit and discusses some RS-485 concepts to aid in using the interface in an embedded system.

RS-485 provides for half-duplex operation. It is a 5 volt differential interface, which has greater immunity against noise and interference than single-wire interfaces. This interface will drive cable lengths up to 4000 feet reliably at 57.6K bps. All communication, both transmission and reception, occurs via a single pair of wires. There are no handshaking lines.

RS-485 supports multidrop operation. That is, more than two devices can be connected to the same RS-485 balanced line. To prevent signal contention, only one transmitter is enabled at a time. The CoreModule/3SX*i* RS-485 transmitter is controlled by Serial 2's RTS signal. At power up, RTS is in its inactive state, ready to receive. When it is time to transmit, the RTS signal is made active, enabling the transmitter. It is the responsibility of the user's software to prevent two transmitters from being enabled at the same time.

Figure 3–5 illustrates the CoreModule/3SX*i* RS-485 interface wiring.

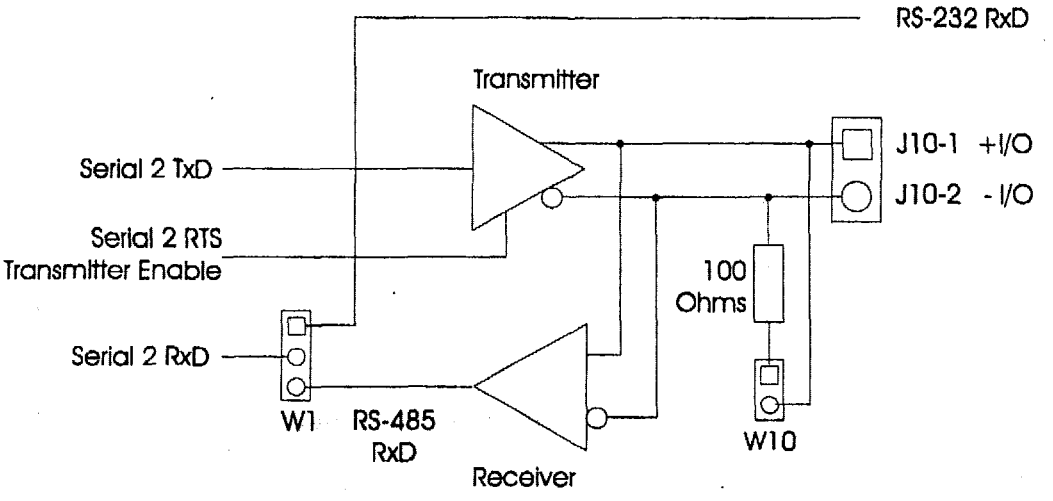


Figure 3–5. RS-485 Interface Wiring

The following are some examples of interconnection schemes that can be used to take advantage of the RS-485 serial connection:

- **One-way Broadcast**—A single device uses an RS-485 signal pair to transmit data to many receiving devices. To enable the broadcaster's transmitter, the RTS signal is turned on (True, High, Active) and left on. If the device is to be a receiver, RTS must be turned off and left off.
- **Simple Bi-Directional Communication**—Two devices use a single RS-485 bi-directional pair for half-duplex, two-way transmission of data. The Adapter's transceiver is placed in the send or receive

mode under control of the network software. This can be done using a simple alternation scheme or by messages contained within data packets.

- **Multidrop Network**—More than two devices share an RS-485 signal pair, for both transmission and reception of data. Only one device is permitted to talk at any one time. As with simple bi-directional communication, the board's RS-485 transceiver is placed in receive mode unless it is the one permitted to transmit. One popular way of managing who is the transmitter is by a "token" passing scheme. Each node is assigned an ID number. Whoever transmits also sends the ID of the next node allowed to transmit. If a node does not need to transmit, it just immediately sends the "token" to its next node. This simple scheme is easy to implement and trouble free. Time-outs can be implemented in software to prevent a lockup should a node fail to pass the token properly.

3.9.2 Serial Console Features

To use the serial console features, connect the serial console device(s) to Serial 1 or Serial 2. Use SETUP to configure the CoreModule/3SX*i* CPU to use its serial console support feature. The configuration memory stores serial console configuration parameters.

Caution

Be careful when changing the console configuration using SETUP. If you specify "None" for console input and output, there will be no console access to the system. (You can recover from this state by removing the serial console plug from the primary serial port connector and shorting pins J3-7/8.)

SETUP provides separate configurations for serial console input and output. Thus, you can use a serial port (and attached serial device) for either or both input and output. For instance, you can use a modem or other serial device for input, and a standard video display for output. Or you can use a standard keyboard for input and an ASCII terminal for serial display, or use an ASCII terminal for both input and output.

To use an ASCII terminal as the console device for your system, set both the input and output parameters to Serial Port 1 (or 2), and set the serial baud rate, data length, and stop bits to match the setting of your terminal. For proper display of SETUP and POST messages from the BIOS, you must use an IEEE-compatible terminal that implements the standard ASCII cursor commands. The required commands and their hexadecimal codes are listed in Table 3–9.

Table 3–9. Required Serial Console Commands

Hex	Command
08	Backspace
0A	Line Feed
0B	Vertical Tab
0C	Non-destructive Space
0D	Carriage Return

Note

Some programs that emulate an ASCII terminal do not properly support the basic ASCII command functions shown in Table 3–9. Ampro provides a suitable PC terminal emulator program, TVTERM, on the Common Utilities diskette.

After booting this system, the keyboard and screen of the terminal become the system console. The programs you use this way must use ROM BIOS video functions (rather than direct screen addressing) for their display I/O. You can enter keyboard data from both the external serial device and the standard AT keyboard.

Note

DOS programs that write directly to video RAM will not display properly on a serial console device.

COM Port Table

When the system boots, DOS initializes the serial ports to 9600 baud (typical). To preserve the selected console port parameters stored in SETUP, the ROM BIOS can be instructed to delete the selected console port from the internal COM port table, normally used by DOS to locate the serial ports. With the port deleted from the COM port table, DOS cannot change its parameters. If you use a serial console, be sure to select the option that deletes the console port from the COM port table.

Serial Handshake

The serial console device data format and the CoreModule/3SX*i* CPU serial port data format must match for the devices to properly communicate. In addition, the hardware handshake behavior must be compatible. Normally, a serial port's Data Set Ready (DSR) and Clear To Send (CTS) input handshake signals must be true (active) for the ROM BIOS to send data out. On the CoreModule/3SX*i* CPU, the hardware handshake can be enabled or disabled with SETUP. When hardware handshaking is enabled, be sure to connect the DSR and CTS signals to appropriate handshake signals on the external serial device's interface connector. As an alternative, loop the CoreModule/3SX*i*'s serial output handshake signals to its input signals as follows:

- DTR (out) to DSR (in)
- RTS (out) to CTS (in)

3.9.3 Serial Booting and Serial Programming

Serial console functionality has been expanded to incorporate two additional features useful in embedded applications.

- The *serial boot* facility enables the CoreModule/3SX*i* CPU to boot from code downloaded through a serial port in a manner similar to booting from a local hard disk or from a network.
- The *serial programming* facility permits updating Flash memory devices installed in the byte-wide socket over the serial port. It can also be used to program the OEM Flash device.

Refer to Ampro Application Note AAN-9403 for a complete description of these features. Refer to the Ampro Common Utilities manual for descriptions of SERLOAD and SERPROG, utility programs used to support serial booting and serial programming.

3.9.4 Using a Serial Modem

You can use any of the RS232C ports as a modem interface. You will not need to concern yourself with serial port initialization since most PC communications programs control the serial port hardware directly. If your program does not do this, use the DOS MODE command to initialize the port.

When installing a modem, be sure to connect appropriate input and output handshake signals, depending on what your communications software requires. Standard PC-compatible serial modem cables that connect all of the proper signals correctly are commonly available. The signal arrangement on the serial port connectors is described in Chapter 2.

Many powerful communications programs are available to control modem communications. Some of these programs offer powerful "script" languages that allow you to generate complex automatically functioning applications with little effort.

3.10 ENHANCED PARALLEL PORT

The enhanced parallel printer port is a superset of the standard PC-compatible printer port. It supports three modes of operation:

- **Standard PC/AT printer port (SPP)**—Centronics-type output only printer port, compatible with the original IBM PC printer port. Sometimes it is called a PS/2-compatible parallel port. It behaves the same as the standard PC/AT port on outputs, and provides an input mode as well.
- **Enhanced Parallel Port (EPP)**—Bi-directional parallel port, compatible with the Standard and PS/2 ports, and adding automatic read- and write-cycle modes. Automatically generates input and output handshaking signals for increased throughput. Data flow is monitored by a watchdog timer (separate from the board's watchdog timer) to ensure reliable transfers.
- **Extended Capabilities Parallel Port (ECP)**—Compliant with the IEEE-1284 Extended Capabilities Port Protocol and ISA Standard (Rev 1.09, January 7, 1993), developed by Microsoft. The ECP mode provides the highest level throughput for the parallel port. It provides interlocking handshaking, a 16-byte FIFO buffer, DMA transfers (optional), hardware RLE data compression (optional), and well-defined software protocols.

The low-level software interface to the parallel port consists of eight addressable registers. The address map of these registers is shown in Table 3–10.

Table 3–10. Parallel Port Register Map

Register Name	Address
Data Port	Base address
Status Port	Base address + 1
Control Port	Base address + 2
EPP Address Port	Base address + 3
EPP Data Port 0	Base address + 4
EPP Data Port 1	Base address + 5
EPP Data Port 2	Base address + 6
EPP Data Port 3	Base address + 7
Note: EPP registers are only accessible when in EPP mode	

3.10.1 Standard and Bi-Directional Operation (SPP)

You can use the parallel port as a standard output-only printer port or as a bi-directional data port with up to 12 output lines and 17 input lines. The bi-directional mode can be very valuable in custom applications. For example, you might use it to control parallel-connected external peripherals, an LCD display, scan keyboards, sense switches, or interface with optically isolated I/O modules. All data and interface control signals are TTL-compatible.

To use the parallel port in standard or bi-directional modes, set the parallel port **Mode** option on page 2 of SETUP to **SPP**.

Note

Note that the term "mode" in this section is used for both the SPP/EPP/ECP **Mode** as set with SETUP, and for input and output modes that port can be in when in the **SPP Mode**. For clarity, the port's SETUP **Mode** setting will be in bold type and capitalized to distinguish it from the various modes that the port can be in when in the **SPP Mode**.

The default mode of the port in **SPP Mode** is output only, to make the port compatible with the original IBM PC parallel port. To use the port as a bi-directional data port, put it in bi-directional mode with a Ampro extended BIOS call, as shown in the following code example.

```
;-------------------------------------------------------------
; Code to set the parallel port mode to "bi-directional"
;-------------------------------------------------------------
MOV   AH,0CDh      ; AMPRO command
MOV   AL,0Ch       ; AMPRO function
MOV   BX,01h       ; Bi-directional mode (00 for output-only)
INT   13h
```

Once the port is in bi-directional mode, you can dynamically change the port between input and output states by directly accessing the control register at I/O address 37Ah.. The initial state of the port after the BIOS call is input. A "1" written to 37Ah-bit 5 sets the port to input; a "0" sets it to output.

The following example is code for dynamically changing the primary parallel port's direction (the code assumes that the port is in **SPP Mode**).

```
;-------------------------------------------------------------
; Code to change the parallel port direction to input
;-------------------------------------------------------------
MOV   DX,37Ah
IN    AL,DX
OR    AL,20h       ;set bit 5
OUT   DX,AL
;
;-------------------------------------------------------------
; Code to change the parallel port direction to output
;-------------------------------------------------------------
MOV   DX,37Ah
IN    AL,DX
AND   AL,0DFh      ;clear bit 5
OUT   DX,AL
```

Using control lines for Input/Output

Besides the eight data lines, you can use the four control lines (-STROBE, -AUTOFD, -INIT, and -SEL IN) as general purpose output lines when the port is set to **SPP Mode**. Similarly, you can use the five status lines (-ERROR, SEL OUT, PAPER EMPTY, -ACK, and BUSY) as general purpose input lines.

You can also read the four control lines and use them as input lines. These lines have open collector drivers with 4.7K ohm pull-ups. To use a control line as an input line, you must first write to its corresponding bit in the control register. Refer to Table 3-12 for the parallel port control register bit definitions. If the line is inverting, write a "0", otherwise write a "1". This will cause the line to float (pulled up by the 4.7K ohm resistors). When they float, you can use them as inputs. Table 3-11 is a summary of the uses of the parallel port lines.

Table 3–11. Parallel Port Use

Signal Type	Number of Lines	Function	Output Drive
Data	8 lines	Read/Write	24 mA @ .5V 12 mA @ 2.4V
Control	4 lines	Read/Write*	12 mA @.5V 4.7K PU
Status	5 lines	Read Only	--
* Open collector control lines convert to TTL outputs in EPP and ECP modes. Output under those conditions is 4 mA @ 2.4V			

Parallel Port Interrupt Enable

Bit 4 in the control register, IRQEN, (see Table 3–12) enables the parallel port interrupt. If this bit is high, then a rising edge on the -ACK (IRQ) line will produce an interrupt on IRQ7.

Table 3–12. Parallel Port Register Bits

Register	Bit	Signal Name or Function	In/Out	Active High/Low	J15 Pin	DB25F Pin
DATA (378h)	0	Data 0	I/O	High	3	2
	1	Data 1	I/O	High	5	3
	2	Data 2	I/O	High	7	4
	3	Data 3	I/O	High	9	5
	4	Data 4	I/O	High	11	6
	5	Data 5	I/O	High	13	7
	6	Data 6	I/O	High	15	8
	7	Data 7	I/O	High	17	9
STATUS (379h)	0	0	In	---	---	---
	1	0	---	---	---	---
	2	0	---	---	---	---
	3	ERROR*	In	Low	4	15
	4	SLCT	In	High	25	13
	5	PE	In	High	23	12
	6	ACK* (IRQ)	In	Low	19	10
	7	BUSY	In	High	21	11
CONTROL (37Ah)	0	STROBE*	Out*	Low	1	1
	1	AUTOFD*	Out*	Low	2	14
	2	INIT*	Out*	High	6	16
	3	SLC	Out*	High	8	17
	4	IRQEN	---	High	---	---
	5	PCD	---	High	---	---
	6	1	---	---	---	---
	7	1	---	---	---	---
* Can also be used as input (see text).						

Register Bit Definitions

Table 3–13 defines the register bits shown in the "Signal Name or Function" column in Table 3–12.

Table 3–13. Standard and PS/2 Mode Register Bit Definitions

Signal Name	Full Name	Description
ERR*	Error	Reflects the status of the ERROR* input. 0 means an error has occurred.
SLCT	Printer selected status	Reflects the status of the SLCT input. 1 means a printer is on-line.
PE	Paper end	Reflects the status of the PE input. 1 indicates paper end.
ACK*	Acknowledge	Reflects the status of the ACK* input. 0 indicates a printer received a character..
BUSY*	Busy	Reflects the complement of the BUSY input. 0 indicates a printer is busy.
STROBE	Strobe	This bit is inverted and output to the STROBE* pin.
AUTOFD	Auto feed	This bit is inverted and output to the AUTOFD* pin.
INIT*	Initiate output	This bit is output to the INIT* pin.
SLC	Printer select input	This bit is inverted and output to the pin. It selects a printer.
IRQEN	Interrupt request enable	When set to 1, interrupts are enabled. An interrupt is generated by the positive-going ACK* input.
PCD	Parallel control direction	When set to 1, port is in input mode. In printer mode, the printer is always in output mode regardless of the state of this bit.
PD0-PD7	Parallel Data Bits	

3.10.2 EPP and ECP Operation

When set to either EPP or ECP Mode, the board's parallel port is compliant with the IEEE-1284 Extended Capabilities Port Protocol and ISA Standard (Rev 1.09, January 7, 1993), developed by Microsoft. The IEEE-1284 specification is complex and is beyond the scope of this manual. Contact IEEE Customer Service and request IEEE Std 1284 for information about EPP and ECP operation.

IEEE Customer Service
445 Hoes Lane
PO Box 1331
Piscataway, NJ 08855-1331 USA

Phone: (800) 678-IEEE (in the US and Canada)
(908) 981-0060 (outside the US and Canada)
FAX: (908) 981-9667
Telex: 833233

3.11 BYTE-WIDE SOCKET

The 32-pin byte-wide memory socket S0 supports a variety of 28- and 32-pin JEDEC pinout memory devices, including EPROM, Flash EPROM, NOVRAM, and SRAM. If you have a backup battery attached to the Utility connector, you can configure the socket to supply backup battery power to convert an SRAM into a Non-Volatile RAM (NOVRAM). Chapter 2 gives examples of the memory devices the socket will support.

Ampro's solid state disk (SSD) drive support in the ROM BIOS and optional SSD Support Software treat the byte-wide socket as one or more DOS disk devices, containing up to 1M byte of storage. The socket is highly configurable with jumpers to accept nearly any common JEDEC byte-wide device. Instructions on how to configure the byte-wide socket for common devices are in Chapter 2.

Access Time

A device used in the byte-wide socket must have access times of 250 nS or less.

Content Mirroring

If you install a device smaller than the memory window specified in SETUP, (for example, a 32K byte component in a 64K window) the contents will appear as multiple copies in the socket's address window.

OEM Flash Memory Device

The CoreModule/3SX*i* CPU has an onboard Flash memory, 64K of which is used to store the ROM BIOS. The remainder is available for semi-permanent storage of programs or data. The amount of available OEM Flash memory on the CoreModule varies between 64K and 960K, depending on the model. Contact your Ampro sales representative for details about CoreModule/3SX*i* models.)

The onboard Flash memory is architecturally equivalent to a second byte-wide socket. It uses the same software mechanisms in the BIOS to control access. It is also configured with SETUP in the same way as the byte-wide socket. It is designated **OEM Flash** in SETUP.

The BIOS accesses the byte-wide S0 and the OEM Flash memory, S1, as 8-bit devices on the PC expansion bus. If you are using both devices, your application program must manually enable and disable them, as only one can be enabled at a time.

3.11.1 Accessing the Byte-Wide Socket and OEM Flash Device

To access the byte-wide socket or the OEM Flash device, it must be enabled. Using SETUP, you can cause either device to be enabled at boot time. This places the contents of the enabled device at the address you specified in SETUP and the processor can access this memory in a normal fashion. If you want to use both the byte-wide socket and the OEM Flash device, you will need to enable each device as it is needed, as only one can be enabled at a time.

Here is a simple assembly language routine showing how to use an Ampro extended-BIOS call to enable or disable the byte-wide memory socket, S0, or the OEM Flash memory. (This code selects the first 64K page on large devices.) Note that when you enable a device, the BIOS call automatically disables the opposite device.

```
;-------------------------------------------------------------

; Access control code for a byte-wide socket (S0 or the
; OEM Flash memory)

;-------------------------------------------------------------
MOV   AH,0CDH          ; AMPRO function call
MOV   AL,nn            ; Use 03 for S0; 04 for the OEM Flash memory
MOV   BL,nn            ; Use 01 to turn ON or 00 to turn OFF
MOV   BH,00            ; Selects page 0 of the device
MOV   CX,414DH         ; Ampro identifier('AM')
INT   13H
```

3.11.2 Accessing Large Devices

For byte-wide devices over 64K bytes, select the 64K byte window size in SETUP. You then use software to select which segment of the device you want to appear in that window, using code equivalent to that illustrated below and using the values shown in Table 3–14. Table 3–14 gives the byte (in hex) to write to the BH register to select each 64K segment of a large device.

This assembly language routine can be used to select pages when accessing large memory devices:

```
;-------------------------------------------------------------

; Page select code for a byte-wide socket (S0 or the
; OEM Flash memory)

;-------------------------------------------------------------
MOV   AH,0CDH          ; AMPRO function call
MOV   AL,nn            ; Use 03 for S0; 04 for the OEM Flash memory
MOV   BL,nn            ; Use 01 to turn ON or 00 to turn OFF
MOV   BH,x0h           ; The upper nibble of BH contains the page
                       ; number for devices larger than 64 K.
MOV   CX,414Dh         ; Ampro identifier ('AM')
INT   13H
```

Table 3–14. Segment Addressing in Large Memory Devices

Device Size	64KB Segments	Segment Address (BH Value)	
128K	2	First	BH=00h
		Second	BH=10h
256K	4	First	BH=00h
		Second	BH=10h
		Third	BH=20h
		Fourth	BH=30h
512K	8	First	BH=00h
		Second	BH=10h
		Third	BH=20h
		Fourth	BH=30h
		Fifth	BH=40h
		Sixth	BH=50h
		Seventh	BH=60h
		Eighth	BH=70h
1M	16	First	BH=00h
		Second	BH=10h
		Third	BH=20h
		Fourth	BH=30h
		Fifth	BH=40h
		Sixth	BH=50h
		Seventh	BH=60h
		Eighth	BH=70h
		Ninth	BH=80h
		Tenth	BH=90h
		Eleventh	BH=A0h
		Twelfth	BH=B0h
		Thirteenth	BH=C0h
		Fourteenth	BH=D0h
		Fifteenth	BH=E0h
		Sixteenth	BH=F0h

128K Special Case

If you install a 128K byte device in the byte-wide socket, you can set the starting address to D0000h and the window size to 128K. It will occupy the entire D0000h - EFFFFh address region. This allows you to access the entire device without switching between windows. (The 128K byte window size is not available for the OEM Flash device, nor can it be used with Ampro's SSD/DOS Support Software.)

3.11.3 Flash EPROM Programming

To program a Flash device in byte-wide socket S0 or the OEM Flash memory, use the FLASHWRI.EXE utility supplied on the Common Utilities diskette. The Common Utilities manual describes its operation.

Programming power is handled automatically for both 5V and 12V Flash devices. The board provides 12V power for programming 12V Flash EPROMs. There are no jumpers to set (other than the Vpp jumper, W2-6), as the onboard 12V Flash programming supply is controlled by software.

You can also develop your own Flash programming routines using extended BIOS calls in the ROM BIOS. Refer to Ampro Application Note AAN-9210 for information about the extended BIOS call provided for Flash programming power. (Note that there is a 5 mS delay for the 12V Flash programming supply to come up to its full voltage after being switched on by software.)

3.12 SCSI CONTROLLER

A SCSI controller can serve many purposes, including controlling hard disk drives, tape drives, text scanners, and printer and communications servers. The ROM BIOS supports booting DOS from a SCSI device such as a hard disk. With Ampro's ROM BIOS support, you can use any device compatible with the SCSI Common Command Set (CCS) for "direct access devices." Ampro has several MiniModule products that can be used to provide a SCSI interface for a CoreModule/3SX*i* CPU system.

The CoreModule/3SX*i* Development Kit comes with a diskette containing an assortment of SCSI utilities for use with DOS. It includes a SCSI hard disk formatting utility that allows low-level formatting and changing the disk interleaving. Refer to the Ampro Utilities manual for details about using the SCSI utilities.

Besides direct access, SCSI devices include sequential access devices (tape), printer devices, read-only devices (CD-ROM), and processor devices (CPUs). These device types require special application programs, utilities, or driver software not included on the Ampro Utility diskette. Contact Ampro Technical Support for information about connecting these devices to an Ampro SCSI interface.

Hard disk support for operating systems other than DOS may or may not be available through the ROM BIOS hard disk driver. This depends on two things: whether the operating system in question uses ROM-BIOS calls exclusively for the hard disk function; and whether the operating system has any special ROM BIOS constraints, such as reentrancy. Some operating systems—multitasking ones in particular such as UNIX—bypass the BIOS and attempt to program the hard disk controller directly. With such systems, you must modify the operating system to add an appropriate SCSI hard disk driver that can take advantage of the SCSI interface. An alternative is to use the IDE interface instead of SCSI, as the IDE drive standard is more widely supported on PC platforms.

3.12.1 The Ampro SCSI BIOS

You can use a variety of mass storage devices with the SCSI universal bus interface and command protocols. Ampro has added a further layer of universality, the SCSI BIOS.

The SCSI BIOS, a set of low level functions in the ROM BIOS, is a hardware-independent interface between system software and SCSI peripherals. Using SCSI BIOS calls, programmers can write software for SCSI devices without concern for the operational details of the SCSI interface. Also, the SCSI BIOS enables you to import software from other environments more safely, quickly, and easily.

Application Note AAN-8804, available from Ampro, provides details of the SCSI BIOS functions.

3.13 PC SPEAKER

The CoreModule's motherboard logic includes a standard AT-compatible speaker port. The speaker logic signal is buffered by a transistor amplifier, and provides about 100 mW to an external 8 ohm speaker.

The audio output is based on two signals: the output of Timer 2; and the programming of two bits, 0 and 1, at I/O port 61h. Bit 1 of I/O port 61h is one term of a 2-input AND gate. The other term is the output from Timer 2. Thus, setting bit 1 to a logic 1 enables the output of Timer 2 to the speaker, and a logic 0

disables it. Disabling Timer 2 by setting bit 0 of port 61h to a 0 causes its output to go high. Then you can use bit 1 of port 61h to control the speaker directly.

3.14 WATCHDOG TIMER

The purpose of a watchdog timer function is to restart the system should some mishap occur. Possible problems include: a failure to boot properly; the application software losing control; temporary power supply problems including spikes, surges, or interference; the failure of an interface device; unexpected conditions on the bus; or other hardware or software malfunctions. The watchdog timer helps assure proper start-up after an interruption.

The CoreModule/3SX*i* CPU ROM BIOS supports the board's watchdog timer function in two ways:

- There is an initial watchdog timer setting, specified using SETUP, which determines whether the watchdog timer will be used to monitor the system boot, and if so, how long the time-out is (30, 60, or 90 seconds).

- There is a special ROM-BIOS function which may be used by application software to start, stop, and retrigger the watchdog timer function.

The initial time-out should be set (using SETUP) to be long enough to guarantee that the system can boot and pass control to the application. Once the system is booted and the application is running, the application must periodically retrigger the timer so that a watchdog timer time-out does not occur. If the time-out does occur, the system will respond in a manner determined by how the watchdog timer jumper, W3, is set (see Chapter 2).

The following assembly language routine illustrates how to reset the watchdog timer using an Ampro extended BIOS function call:

```
;-------------------------------------------------------
; Watchdog timer control program
;-------------------------------------------------------
MOV   AH,0C3h          ; Watchdog Timer BIOS function
MOV   AL,nn            ; Use "00" to disable; "01" to enable
                       ; timer.
MOV   BX,mm            ; Selects time, in seconds
                       ;(00-FFh; 1-255 seconds)
INT   15h
```

Ampro provides a simple DOS program that can be used from the command line or in a batch program to manage the watchdog timer. It is called WATCHDOG, and is described in the Ampro Common Utilities manual.

Note

Some operating systems, including some versions of DOS, turn off the real-time clock alarm at boot time. If your OS does this, make sure that your application program enables the alarm function using this BIOS call.

If you jumper the output of the Watchdog Timer to trigger a non-maskable interrupt (NMI), an NMI IO Channel Check will be asserted by the real-time clock alarm circuit when it times out. For the system to respond to the NMI, the NMI circuit must be enabled. (In the PC architecture, the non-maskable interrupt can be masked.) To enable (unmask) the NMI, execute the following code.

```
;-------------------------------------------------
; To enable NMI (IO channel check)
;-------------------------------------------------

IN    AL,61H
AND   AL,NOT 08H
OUT   61H,AL

;-------------------------------------------------
```

To use the NMI I/O Channel Check in a custom Watchdog Timer handler routine, the standard NMI handler would have to be replaced with your custom code. If you install your own NMI interrupt service routine, it can test to see if the I/O Channel Check NMI occurred by reading I/O port 61h, bit 6. Bit 6 is true (1) if the NMI occurred.

Note

Following the occurrence of an I/O Channel Check NMI, the function must be disabled and then re-enabled before the next one can occur.

3.15 POWERFAIL MONITOR

In embedded systems, it is important for the computer to execute a clean reset if its power supply fluctuates. In general, you would want to avoid erratic behavior that could result if the system voltage were to dip to marginal levels.

The CoreModule/386SX*i* has a built-in powerfail circuit that will generate a clean reset signal if power falls below 4.65V. It guarantees a minimum 140 mS reset signal, independent of how long the power falls below the 4.65V threshold.

3.16 SYSTEM MEMORY MAP

The CoreModule/3SX*i* CPU architecture allows it to address up to 64M bytes of memory. Table 3–15 shows how this memory is used.

The DRAM, the byte-wide socket, ROM BIOS, and OEM Flash memory occupy the first megabyte (starting at 00000h). You can install up to 8 megabytes of DRAM onboard with 4M bytes of base DRAM and a 4M byte custom add on memory module.

Table 3–15. CoreModule/3SX*i* Memory Map

Memory Address	Function
FF0000h - FFFFFFh	Duplicates BIOS at 0F0000-0FFFFFh.
100000h - FDFFFFh	Extended memory
0F0000h - 0FFFFFh	64K ROM BIOS.
0D0000h - 0EFFFFh	Byte-wide socket S0 or OEM Flash, if enabled. Otherwise, free.
0C0000h - 0CBFFFh	VGA Video BIOS.
0A0000h - 0BFFFFh	Normally contains video RAM, as follows: CGA Video: B8000-BFFFFh Monochrome: B0000-B7FFFh EGA and VGA video: A0000-AFFFFh
000000h - 09FFFFh	Onboard DRAM

3.17 SYSTEM I/O MAP

Table 3-15 is a list of the I/O port assignments used on the CoreModule/3SX*i* CPU. The I/O port functions and addresses (except for a few "Ampro reserved" addresses) shown in Table 3-15 are all standard for PC compatibles from both a hardware and software perspective.

Typically, the ROM BIOS provides all the services needed to use the onboard devices and devices connected to I/O ports. If you need to directly program the standard functions, refer to a programming reference for the PC/AT.

Table 3–16. CoreModule/3SX*i* I/O Map

I/O Address	Function
03F8h - 03FFh	Primary serial port
03F2h - 03F7h	Floppy disk controller ports 3F2: FDC Digital output register 3F4: FDC Main status register 3F5: FDC Data register 3F7: FDC Control register 3F0, 3F1 Ampro reserved
0378h - 037Fh	Parallel port (configured as Primary)
02F8h - 02FFh	Secondary serial port
0278h - 027Fh	Parallel port (configured as Secondary)
0202h	Ampro reserved
01F0h - 01F7h	IDE hard disk interface
00F0h - 00FFh	Reserved
00C0h - 00DFh	DMA controller 2 (8237 equivalent)
00A0h - 00A1h	Interrupt controller 2 (8359 equivalent)
0092h	Fast A20 gate and CPU reset
0080h - 009Fh	DMA page registers (74LS61 equivalent)
0070h - 0071h	Real-time clock and NMI mask
0060h, 0064h 0061h	Keyboard controller (8042 equivalent) Port B
0040h - 0043h	Programmable timer (8254 equivalent)
0022h, 0023h	Ampro reserved
0020h - 0021h	Interrupt controller 1 (8359 equivalent)
0000h - 000Fh	DMA controller 1 (8237 equivalent)

Note

All I/O ports below 100h are reserved for internal system functions and should not be accessed.

INDEX

2.3 Acer Laboratories, ALi M6117C – 386SX Embedded Microcontroller IC

The ALi M6117C is the primary component of the Ampro CoreModule 3SXi card and controls most of the CPU operations. The specifications and data sheet for this IC are included for extra clarity.

2.3.1 Specifications for the ALi M6117C

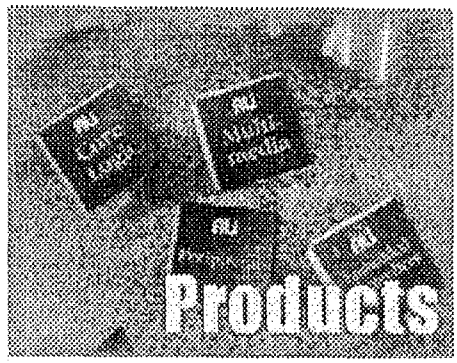
The following is the vendor's specification sheet for the ALi M6117C, which was downloaded from the Acer Labs Internet site at:

http://www.ali.com.tw/eng/product/embed/m6117c.htm.

Acer Laboratories Inc.
揚智科技股份有限公司
"Innovation is Integration"

About ALi
Investor Relations
News
Products
Support
Drivers
Career
中文

Acer Laboratories Inc. (ALi) is one of the world's leading manufactu integrated circuits for the personal computer and embedded PC ma



- ALi Home
- Product Overview
- Product Features
- FAQs
- Glossary
- Press Release

Product Overview

M6117C - 386SX Embedded Microcontroller

The M6117C is a highly integrated, low voltage, single-chip imple Intel™ 386SX compatible microprocessor plus ALi M1217B chips M6117C provides the following functions : 1) Intel™ 386SX core EDO DRAM controller including FP mode 3) Coprocessor Interface interface 5) Peripheral Interface (includes two cascaded 8237 DM a 74612 memory mapper, 2 cascaded 8259 interrupt controller, programmer counter 6) Built-in RTC 7) Built-in PS2 Keyboard Co Mouse 8)Built-in WATCHDOG timer 9)16-bit GPI/O10) IDE interf

Product Features

M6117C - 386SX Embedded Microcontroller

Static Intel 386SX compatible Core

- Operating Power Supply 5.0V
- Operating frequency 25Mhz to 40Mhz

Coprocessor Interface

- Supports 80387SX coprocessor interface

Memory Controller

- Supports EDO DRAM
- Supports on board memory size up to 16M bytes for 386SX upgrade system using 256K, 512K, 1M, 4M or 16M SIMMs
- Supports up to 4-bank DRAM interface
- Page interleave DRAM access for FP mode
- Programmable shadow RAM from A to B segment in 128K b segment in 32K byte unit
- Provides "RAS only" refresh or "CAS before RAS" refresh by
- Parity generation and checking

Peripheral Interface

- Includes 2 cascaded 8237 DMA controllers
- Includes 1 74612 memory mapper
- Includes 2 cascaded 8259 interrupt controllers
- Includes 1 8254 programming counter

ISA Interface

- Executes cycles for requests from CPU, DMA and ISA bus m
- Assembles or de-assembles data for multiple bus cycle or u data width
- Generates refresh signals to ISA slots during DRAM refresh

Built-in RTC

- Internal Real Time Clock that provides 128 byte CMOS RA

Built-In PS2/AT Keyboard Controller

- Internal PS2/AT keyboard controller and mouse

PMU interface

- Supports CPU SMM mode, SMI feature
- Supports APM control
- Provides External Suspend mode switch
- Provides four (4) system states for power saving (On, Doze Suspend)
- Supports RTC alarm wake up control

Expandable GPI/O signals

- Provides sixteen External power control input and output si

Watchdog timer

- When timer times out , a system reset or NMI or IRQ happ

IDE interface

- Provides a decoder for external IDE connection

Packaging

- 208-pin PQFP package

All specifications are subject to change without prior notice.

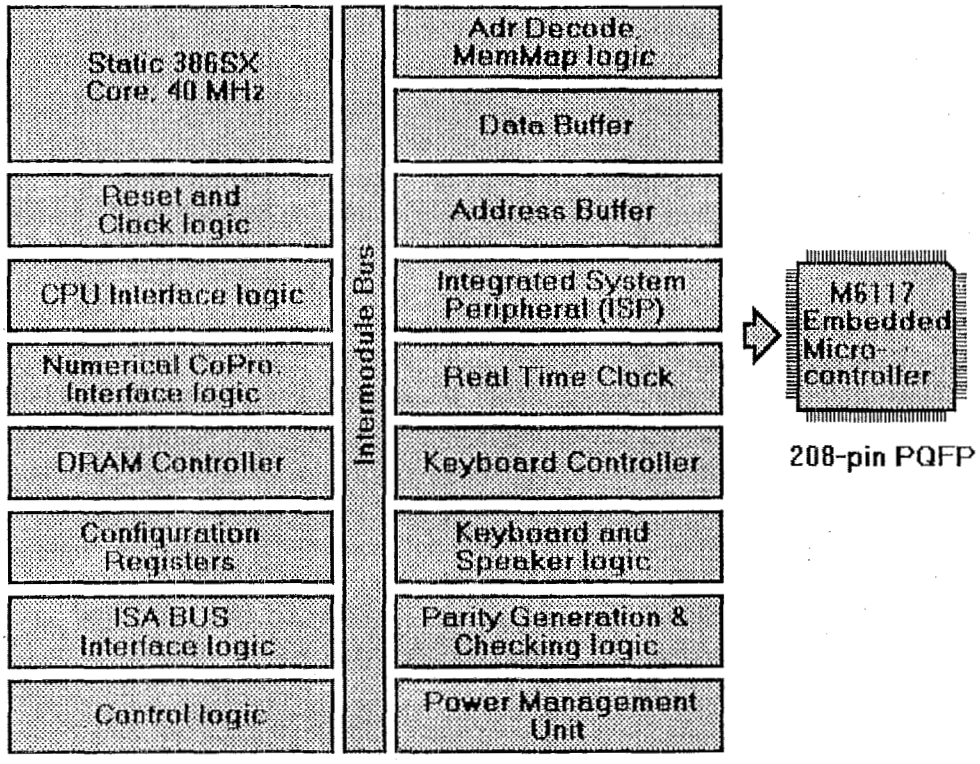
2.3.2 Functional Description of the ALi M6117C

The following is an alternate vendor's (SSV) functional description of the ALi M6117C, which was downloaded from the SVV Internet site at:

http://www.ssv-embedded.de/ssv/pc104/p24.htm.

ALI M6117 FUNCTIONAL DESCRIPTION

The M6117 is designed to perform like Intel's 386SX system with deep green features. Aside from the 386SX core, it contains (1) Keyboard Controller for any IBM PC/AT compatible keyboard, (2) Real Time Clock to store system boot data, (3) Integrated System Peripheral to serve the peripheral requests, (4) Power Management Unit to reduce the chip's power consumption efficiently, (5) LS245: TTL data buffer between ISA data bus SD[7:0] and ROM data bus XD[7:0], (6) DRAM Controller for four banks memory module with page interleave and up to 64 Mbytes space. The M6117 offers the following blocks:



Static 386SX Core: The 386SX core is the same as M1386SX of Acer Labs. Inc. (ALI) and 100% object code compatible with the Intel 386SX microprocessor. System manufacturers can provide 386 CPU based systems optimized for both cost and size. Instruction pipelining and high bus bandwidth ensure short average instruction execution times and high system throughput. Furthermore it can keep the state internally from charge leakage while external clock to the core is stopped without storing the data in registers. The power consumption here is almost zero when clock stops. The internal structure of this core is 32-bit data and address bus with very low supply current, 116 mA in the conditions of 5.0V, 20MHz, room temperature. Real mode as well as Protected mode are available and can run MS-DOS, MS-Windows, OS/2 and UNIX.

Reset and Clock Logic: The switching power supply sends a PWG (power good signal) to M6117 to generate system reset signals, like RSTDRV, RSTNP, and resets the chip to initial state. Also the reset signal can be generated by internal emulation RC reset and shutdown cycle.

There are two clock inputs: BCLK2 and OSC are 2 * system clock and 14.318 MHz respectively, and three clock outputs: BCLK1, ATCLK1 and CK7M which provide frequency operation for the system board and devices depending on 1 * clock used. The BLCK1 is a half frequency of BCLK2. The CK7M derived from the OSC input (divided by 2) is available as the key controller clock when power is on. To increase system performance, the M6117 supports variable AT clocks for faster ISA add-on cards. When the CPU accesses the register programmed special address range, the AT clock changes to a faster speed. The non-programmed address regions keep the normal speed. There are eight programmable frequencies of the ATCLK1 which can change on fly by different specific addresses and determined by D[2:0] of local port 1EH in both high and normal speed. This optional AT clock can achieve a higher performance when a faster add-on card is used.

CPU Interface Logic: The CPU interface logic decodes the status MIO, DC, WR with different equivalents. It will handle all the CPU-side instruct events and requests the bus ownership from the CPU.

Numerial Coprocessor Interface Logic: For 386SX systems, M6117 monitors ERRORJ, BUSYJ and PEREQ to support the 80387SX coprocessor. When ERRORJ is active, this indicates that an unmaskable coprocessor exception has occurred and the coprocessor interface will generate an IRQ13 to the CPU core. (If it immediatly follows a RESET signal, this indicates that a coprocessor is present in the system.) The coprocessor asserts BUSYJ signal while executing and asserts NPRDYJ to M6117 when it is finished. The coprocessor interface then passes the NPRDYJ signal to RDYOJ signal to the CPU core. This ready signal has to meet the CPU requirement.

DRAM Controller: The DRAM controller is capable of accessing up to 64 MBytes of local memory, and supporting four banks page interleave of DRAM using 256K, 512K, 1M, 2M, 4M, 16M single sided SIMMs, or 256K, 1M, 4M, 16M double sided SIMMs. Page interleave mechanism is able to shorten the memory read/write cycle and raise the data access speed between host and RAM, and works on any two banks with the same DRAM type. Each bank can be disabled through software. Programmable DRAM timing is provided for RAS precharge time and RAS-to-CAS delay to achieve highest performance and reliability. And they also explain how to use the 256/384 Kbyte memory remapping feature in unshadowed RAM region from A0000H to FFFFFH. Programmable shadowing features are supported on 32K boundaries between C0000H and FFFFFH regions (768KB...1MB). It also supports "RAS only", "CAS before RAS" refresh and self refresh cycle type of DRAMs.

Configuration Registers: The configuration register controls the whole system of environment under different frequencies. It enables the system to set these configuration registers to meet the compatible, reliable performance and functional requirements.

ISA BUS Interface Logic: This block includes the ISA bus state machine, 16- bit or 8- bit commands justified, command wait states and control logic. These signals were compatible with PC/AT standards.

Control Logic: The control logic controls the internal data bus and address bus flow. It also generates proper read-select to internal device and uses multiplexer to choose the correct data output. It selects the correct address bus for DMA and refresh cycles to send to system.

Address Decode and Memory Mapping Logic: The 16-bit address decode-circuit fully de- codes the BIOS ROM, keyboard controller, internal ISP devices, real time clock, port 61H, and configuration registers. When remap is enabled, it decodes the remap memory to the end of DRAM.

Data Buffer: This block generates signals which control data transfer between the CPU core data bus, memory data bus and ISA data bus during CPU cycles, ISA bus cycles, DMA cycles and master cycles.

Moreover, we added LS245, TTL data buffer between ISA data bus SD[7:0] and ROM data bus XD[7:0], on the ASIC. So that users could save some external TTL logic.

Address Buffer: The address buffer generated at address SA1, SA0 and BHEJ for ISA bus, initiates the byte- enable signal at DMA and master cycles.

ISP Devices (2 * 82C37, 2 * 82C59, 82C54, 74LS612): The integrated system peripheral (ISP) devices are built-in, thus no external 82C206 is required. There are two 82C37s, two 82C59s, one 82C54 and one 74LS612 built-in devices.

Real Time Clock: The real time clock (RTC) device is built-in, thus no external RTC is required. If the user does not use the internal RTC for something else, then it can be disabled by hardware setting.

Keyboard Controller: The keyboard controller (KBC) device is built-in, thus no exterternal KBC is required. If the user does not use the internal KBC for something else, then it can be disabled by hardware setting.

Keyboard and Speaker Logic: This block emulates the keyboard controller fast-RC and fast gate-A-20 functions for maximun performance. It combines with port 61H at this block to generate speaker signal.

Parity Generating and Checking Logic: During a local memory read cycle, M6117 not only monitors bus steering, but also checks the parity bit for each data byte from DRAM to ensure the correct data is read. If a parity error occurs, parity checking logic drives PCHERRJ active to peripheral decoder logic.Then peripheral decoder pro- duces a NMI to CPU core for the parity error. The parity checking can be disabled by hardware setting (pull XDACK1J low). During a local memory write circle, M6117 monitors the bus steering and uses the accepted data to generate the parity bit sent to DRAM for each data byte.

Power managemant Unit: The M6117 Power management unit includes SMM, I/O trap, APM, external SMI switch control and programmable clock timeout unit for I/O device. The PMU strictly controls and dramatically reduces overall system power consumption.This is accomplished via the activity monitors which detect the system inactivity timer timeout, and signals the power saving device to remove the power sources from the various peripherals. The M6117 provides one timer from one-second to 300 minutes to monitor the system states (ON/DOZE/STANDBY/SUSPEND modes). The M6117 provides a LED flash control to indicate the system state status.The M6117 also provides 8 programm- able output control signals for peripheral devices to control power-sourcing, or the external clock generator to change frequencey setting. The M6117 supports external SMI switch into suspend mode, SMI Setup, and wakeup events(RTC alarm).The M6117 also provides the interaction control for SMIJ and CPURST.

2.3.3 Data Sheet for the ALi M6117C

The following is the vendor's data sheet for the ALi M6117C.

M6117C : 386SX Embedded Microcontroller

Section 1 : Introduction

The M6117C is a highly integrated, low voltage, single-chip implementation of Intel• ™ 386SX compatible microprocessor plus ALi• M1217B chipset. The M6117C provides the following functions : 1) Intel™ 386SX core 2)Supports EDO DRAM controller including FP mode 3) Coprocessor Interface 4) ISA interface 5) Peripheral Interface (includes two cascaded 8237 DMA controllers, a 74612 memory mapper, 2 cascaded 8259 interrupt controller, and an 8254 programmer counter 6) Built-in RTC 7) Built-in PS2 Keyboard Controller and Mouse 8)Built-in WATCHDOG timer 9)16-bit GPI/O 10) IDE interface.

The following sections highlight the main features and functions of the M6117C chip. For additional information, see Section 3 of this data sheet.

1.1 Features and Functions

Static Intel 386SX compatible Core
- Operating Power Supply 5.0V
- Operating frequency 25Mhz to 40Mhz

Coprocessor Interface
- Supports 80387SX coprocessor interface

Memory Controller
- Supports EDO DRAM
- Supports on board memory size up to 16M bytes for 386SX or 64M bytes upgrade system using 256K, 512K, 1M, 4M or 16M SIMMs
- Supports up to 4-bank DRAM interface
- Page interleave DRAM access for FP mode
- Programmable shadow RAM from A to B segment in 128K byte and C to F segment in 32K byte unit
- Provides "RAS only" refresh or "CAS before RAS" refresh types
- Parity generation and checking

Peripheral Interface
- Includes 2 cascaded 8237 DMA controllers
- Includes 1 74612 memory mapper
- Includes 2 cascaded 8259 interrupt controllers
- Includes 1 8254 programming counter

ISA Interface
- Executes cycles for requests from CPU, DMA and ISA bus master
- Assembles or de-assembles data for multiple bus cycle or unmatched data width
- Generates refresh signals to ISA slots during DRAM refresh cycles

Built-in RTC
- Internal Real Time Clock that provides 128 byte CMOS RAM

Built-In PS2/AT Keyboard Controller
- Internal PS2/AT keyboard controller and mouse

PMU interface
- Supports CPU SMM mode, SMI feature
- Supports APM control
- Provides External Suspend mode switch
- Provides four (4) system states for power saving (On, Doze, Standby, Suspend)
- Supports RTC alarm wake up control

Expandable GPI/O signals
- Provides sixteen External power control input and output signals

Watchdog timer
- When timer times out , a system reset or NMI or IRQ happens

IDE interface
- Provides a decoder for external IDE connection

Packaging
- 208-pin PQFP package

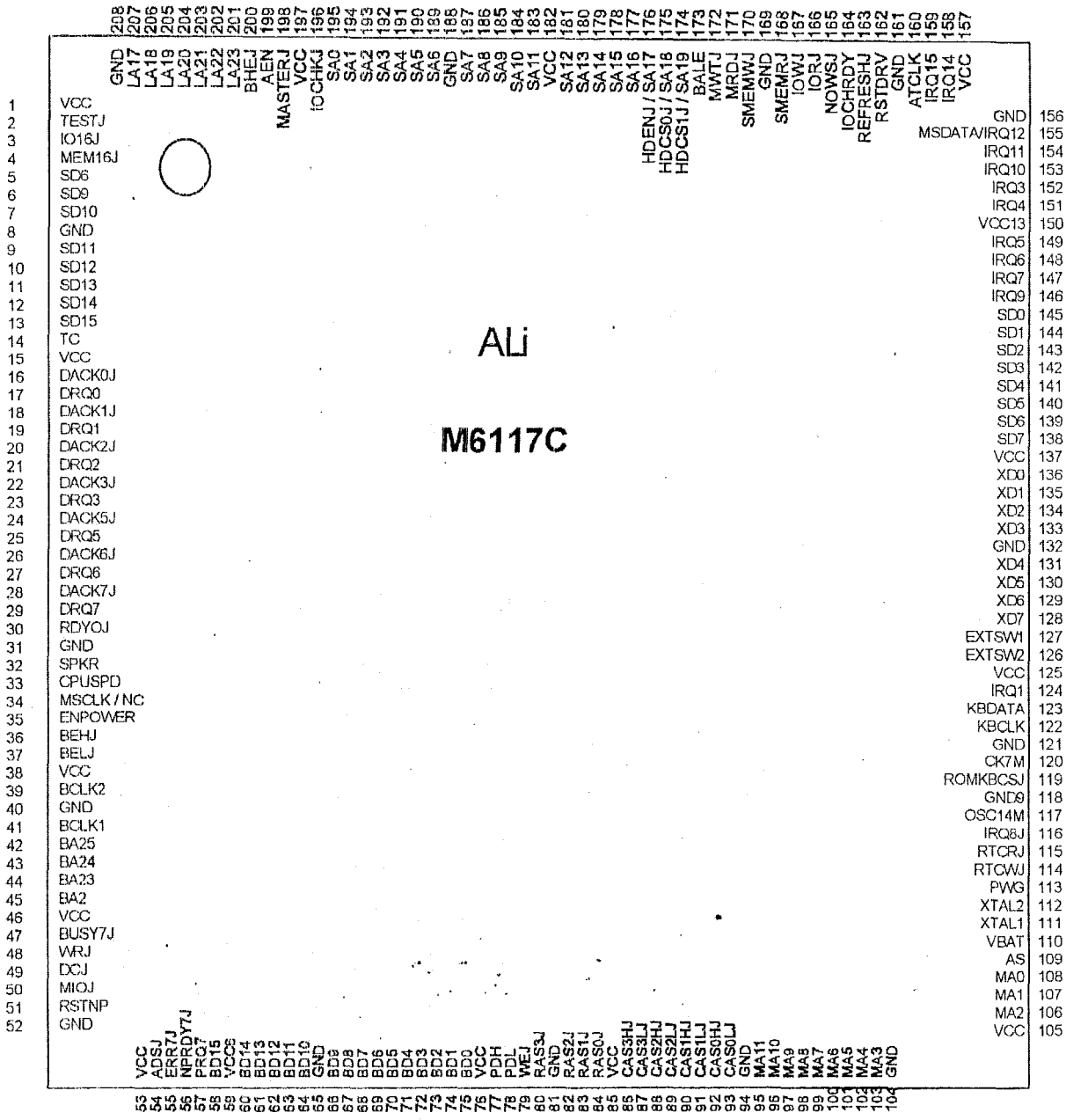
Acer Laboratories, Inc., USA, 1830B Bering Drive, San Jose, CA 95112; Tel: 408-467-7456; Fax: 408-467-7474
www.acerlabs.com

Table of Contents

Acer Laboratories, Inc., USA, 1830B Bering Drive, San Jose, CA 95112; Tel: 408-467-7456; Fax: 408-467-7474
www.acerlabs.com

Section 2 : Pin Description

2.1 Pin Diagram (M6117C) :



Acer Laboratories, Inc., USA, 1830B Bering Drive, San Jose, CA 95112; Tel: 408-467-7456; Fax: 408-467-7474
www.acerlabs.com

2.2 Pin Description Table : (M6117C)
All pins are 5V TTL compatible

Pin name	Type	Pin no.	Description
Clock & Reset interface :			
PWG	I	113	**Power Good**. This indicates that the system power is enough to maintain system integrity. It resets the system when the power is low.
BCLK2	I	39	**CPU-Bus Clock Input**. This is the clock input source for the internal circuit. The clock source should be the same as the CPU clock.
OSC14M	I	117	**14.318 MHz Oscillator**. This frequency is 12 times the frequency used to clock the 8254 timer counter.
ATCLK	O	160	**System Clock Output**. This signal clocks the ISA bus.
RSTDRV	O	162	**Driver Reset**. This output signal is driven active during system power up.
BCLK1	O	41	**1X CPU-bus Clock**. This is the clock divided by BCLK2.
RSTNP	O	51	**Coprocessor Reset**. This drives the reset pin of the coprocessor.
CK7M	O	120	**Keyboard Clock Output**. This signal clocks the keyboard controller that has a frequency of 14.318/2 MHz
ISA Bus interface :			
SD[15-0]	I/O	13-9, 7-5, 138-145	**ISA high and low byte Slot Data bus**. These are the system data lines. These signals read data and vectors into CPU during memory or I/O read cycles or interrupt acknowledge cycles and outputs data from CPU during memory or I/O write cycles.
XD[7-0]	I/O	136-133, 131-128	**External Data bus** for Keyboard Controller, BIOS ROM, RTC.
SA[16-0]	I/O	177-181 183-187 189-195	**ISA Slot Address bus.** These signals are high impedance during hold acknowledge.
HDENJ/SA[17]	I/O	176	**HD Enable control signal. /** ISA slot address bus for 62-pin slot.
HDCS1J/SA[18]	I/O	175	**HD Chip Select 1. /** ISA slot address bus for 62-pin slot.
HDCS0J/SA[19]	I/O	174	**HD Chip Select 0. /** ISA slot address bus for 62-pin slot.
LA[23-17]	I/O	201-207	**ISA Latched Address bus**. These are input signal during ISA master cycle.
IO16J	I	3	ISA 16-bit I/O device select indicator signal.
MEM16J	I	4	ISA 16-bit memory device select indicator signal.
MASTERJ	I	198	**ISA Master device active signal**. ISA master access indicator signal.
MRDJ	I/O	171	**ISA Memory Read**. This signal is an input during ISA master cycle.
MWTJ	I/O	172	**ISA Memory Write**. This signal is an input during ISA master cycle.
AEN	I/O	199	**ISA I/O Address Enable**. This active high output indicates that the system address is enabled during the DMA refresh cycles.
IOCHRDY	I	164	**ISA system ready**. This input signal is used to extend the ISA command width for the CPU and DMA cycles.
BALE	O	173	**Bus Address Latch Enable**. BALE indicates the presence of a valid address at I/O slots.
NOWSJ	I	165	**ISA zero Wait State**. This is the ISA device zero-wait state indicator signal. This signal terminates the CPU ISA command immediately.
IOCHKJ	I	196	**ISA parity error**. M6117C will generate NMI interrupt when this signal is asserted.
BHEJ	I/O	200	**ISA Byte High Enable.** In master cycle, it is an input polarity signal and is driven by the master device.
IORJ	I/O	166	**ISA I/O Read**. This signal is an input during ISA master cycle.
IOWJ	I/O	167	**ISA I/O Write**. This signal is an input during ISA master cycle.
SMEMRJ	O	168	**ISA System Memory Read**. This signal indicates that the memory read cycle is for an address below 1M byte address.
SMEMWJ	O	170	**ISA System Memory Write**. This signal indicates that the memory write cycle is for an address below 1M byte address.

Acer Laboratories, Inc., USA, 1830B Bering Drive, San Jose, CA 95112; Tel: 408-467-7456; Fax: 408-467-7474
www.acerlabs.com

Pin Description Table - M6117C (continued) :

Pin name	Type	Pin no.	Description
REFRESHJ	I/O	163	**Refresh cycle indicator**. ISA master uses this signal to notify DRAM needs refresh. During the memory controller's self-acting refresh cycle, M6117C drives this signal to the I/O channels.
Interrupt Unit :			
IRQ[7-3], IRQ[10-9], IRQ[11], IRQ[15-14]	I	147-149 151-153 146, 154, 159-158	**Interrupt Request signals**. These are interrupt request input signals.
DMA Unit :			
DRQ[7-5], DRQ[3-0]	I	29, 27,25, 23, 21,19, 17	**DMA Device Request**. These are DMA request input signals.
DACK[7-5]J, DACK[3-0]J	I/O	28, 26,24, 22, 20,18, 16	**DMA Device Acknowledge signals**. These are DMA acknowledge demultiplex select signals. Input function is for hardware setting.
TC	O	14	**DMA end of process**. This is the DMA channel terminal count indicating signal.
Power Management Unit :			
EXTSW1, EXTSW2	I	127, 126	**External SMI Switch.** This switch is caused to assert SMIJ for power saving or to wake up the system through the SMI routine.
ENPOWER	O	35	**Enable Power pin**. This active high signal updates the status of external switch.
Timer Unit :			
SPKR	O	32	**Speaker output**. Speaker data.
Coprocessor Interface :			
MIOJ	O	50	**Memory I/O select**. This is the 16-bit device select indicating signal.
WRJ	O	48	**Write/Read select**. This is CPU bus cycle definition pins.
ADSJ	O	54	**CPU Address Strobe**. This active low signal indicates that valid CPU bus cycle
BUSY7J	I	47	**Coprocessor Busy**. This is the busy condition signal from an Intel compatible coprocessor.
ERR7J	I	55	**Coprocessor Error**. This may be pulsed for reporting error of the coprocessor.
NPRDY7J	I	56	**Coprocessor Ready**. This signal indicates the coprocessor bus cycle is terminated.
PRQ7	I	57	**Coprocessor Extension Request**. This is the date transferred request to an Intel compatible processor.
BA2, BA23	O	45, 44	Local Bus Address A2 and A23.
DRAM Interface :			
BD[15-0]	I/O	58, 60-64 66-75	**Local data bus**. These signals are used for data transfer between local bus and DRAM interface.
RAS[3-0]J	O	80, 82-84	**RAS[3-0]J** allows eight on board DRAM SIMM slots. The detail memory configuration refers to the memory configuration table.
CAS[3-0][HL]J	O	86-93	**CAS[n]HJ** is the CAS signal to memory bank n for high byte, **CAS[n]LJ** is the CAS signal to memory bank n for low byte. Total memory bus width is 16 bits.
WEJ	O	79	WE is the **Write Enable** signal to the DRAM.
PDL, PDH	I/O	78, 77	**Memory data parity bit**. Parity bit is written to DRAM in write cycle, and is read from DRAM in read cycle.
MA[11:0]	O	95-103 106-108	**Memory Address bus**. DRAM multiplex ROW/COLUMN address.

Pin Description Table - M6117C (continued)

Pin name	Type	Pin no.	Description
X-bus Interface :			
XD[7-0]	I/O	128-131, 133-136	**X-bus Data**. X-bus data lines.
AS	O	109	**RTC Address Strobe**. This signal is used to demultiplex the address/data bus of the RTC
RTCWJ	O	114	External real time clock write strobe, active low signal.
RTCRJ	O	115	External real time clock read strobe, active low signal.
IRQ8J	I	116	**Interrupt Request**. This signal is from external RTC
VBAT	I	110	**Internal RTC Battery**. This must be connected to RTC battery supply when internal RTC is enabled.
XTAL1, XTAL2	I	111, 112	These pins are connected to the 32.768K crystal when internal RTC is used.
KBCLK	I/O	122	Keyboard interface CLK.
KBDATA	I/O	123	Keyboard interface DATA.
IRQ1/KBINH	I	124	**KB Inhibit input** (when enable internal keyboard)/ IRQ1 input (when enable external keyboard)
ROMKBCSJ	I/O	119	**KB or ROM Chip Select**. Must be pulled low for normal operation.
MSCLK / NC	I/O	34	**Mouse interface CLK . / (In M6117B, this is not connected).**
MSDATA/ IRQ12	I/O	155	**Mouse interface DATA / IRQ12 input .**
Miscellaneous :			
TESTJ	I	2	**Test**. This signal is used for ASIC testing, and must be pulled high during normal operation.
CPUSPD	O	33	**Speed LED output**. This is speed LED indicating signal.
RDYOJ	O	30	**CPU Ready**. This signal indicates the completion of the current bus cycle of processor.
BEHJ, BELJ	O	36, 37	**CPU cycle byte high, low enable.**
BA24, BA25	O	43, 42	**Local Bus Address** A24, A25.
DCJ	O	49	**CPU cycle Data/Code select**. This is a CPU bus cycle definition pin.
Power pins :			
VCC	1, 15, 38, 46, 53, 59, 76, 85, 105, 125, 137, 150, 157, 182, 197		**VCC**. 5.0V supply.
VSS	8, 31, 40, 52, 65, 81, 94, 104, 118, 121, 132, 156, 161, 169, 188, 208		**VSS**. Ground.

Acer Laboratories, Inc., USA, 1830B Bering Drive, San Jose, CA 95112; Tel: 408-467-7456; Fax: 408-467-7474
www.acerlabs.com

2.3 Numerical Pin List : M6117C(M6117B)

Pin No.	Pin Name	Type		Ioh/Iol	internal pull
1	VCC1	P	IO VDD		Double bound
2	TESTJ	I	I_TTL		pull_high
3	IO16J	I	I_TTL		pull_high
4	MEM16J	I	I_TTL		pull_high
5	SD8	B	I_TTL/O_CMOS	8/16 mA	pull_high
6	SD9	B	I_TTL/O_CMOS	8/16 mA	pull_high
7	SD10	B	I_TTL/O_CMOS	8/16 mA	pull_high
8	GND1	P	IO GND		Double bound
9	SD11	B	I_TTL/O_CMOS	8/16 mA	pull_high
10	SD12	B	I_TTL/O_CMOS	8/16 mA	pull_high
11	SD13	B	I_TTL/O_CMOS	8/16 mA	pull_high
12	SD14	B	I_TTL/O_CMOS	8/16 mA	pull_high
13	SD15	B	I_TTL/O_CMOS	8/16 mA	pull_high
14	TC	O	O_CMOS	6/8 mA	pull_high
15	VCC2	P	M1386 CORE		Double bound
16	DACK0J	B	I_TTL/O_CMOS	6/8 mA	pull_high
17	DRQ0	I	I_TTL		pull_low
18	DACK1J	B	I_TTL/O_CMOS	6/8 mA	pull_high
19	DRQ1	I	I_TTL		pull_low
20	DACK2J	B	I_TTL/O_CMOS	6/8 mA	pull_high
21	DRQ2	I	I_TTL		pull_low
22	DACK3J	B	I_TTL/O_CMOS	6/8 mA	pull_high
23	DRQ3	I	I_TTL		pull_low
24	DACK5J	B	I_TTL/O_CMOS	6/8 mA	pull_high
25	DRQ5	I	I_TTL		pull_low
26	DACK6J	B	I_TTL/O_CMOS	6/8 mA	pull_high
27	DRQ6	I	I_TTL		pull_low
28	DACK7J	B	I_TTL/O_CMOS	6/8 mA	pull_high
29	DRQ7	I	I_TTL		pull_low
30	RDYOJ	O	O_CMOS	6/8 mA	
31	GND2	P	M1386 CORE		Double bound
32	SPKR	O	O_CMOS	4/4 mA	
33	CPUSPD	O	O_CMOS	4/4 mA	pull_high
34	MSCLK (NC)	B	Mouse CLOCK	12/24 mA	pull_high
35	ENPOWER	O	O_CMOS	4/4 mA	
36	BEHJ	O	O_CMOS	8/16 mA	pull_high
37	BELJ	O	O_CMOS	8/16 mA	pull_high
38	VCC3	P	IO VDD		Double bound
39	BCLK2	I	CMOS		
40	GND3	P	M1386 CORE		Double bound
41	BCLK1	O	O_CMOS	8/16 mA	
42	BA25	O	O_CMOS	8/16 mA	pull_high
43	BA24	O	O_CMOS	8/16 mA	pull_high
44	BA23	O	O_CMOS	8/16 mA	pull_high
45	BA2	O	O_CMOS	8/16 mA	pull_high
46	VCC4	P	M1386 CORE		Double bound
47	BUSY7J	I	I_TTL		pull_high
48	WRJ	O	O_CMOS	8/16 mA	pull_high
49	DCJ	O	O_CMOS	8/16 mA	pull_high
50	MIOJ	O	O_CMOS	8/16 mA	pull_high

Numerical Pin List : M6117C(M6117B) : (continued)

Pin No.	Pin Name	I/O	Cell Type	Ioh/Iol	internal pull
51	RSTNP	O	O_CMOS	6/8 mA	
52	GND4	P	IO GND		Double bound
53	VCC5	P	IO VDD		Double bound
54	ADSJ	O	O_CMOS	8/16 mA	pull_high
55	ERR7J	I	I_TTL		pull_high
56	NPRDY7J	I	I_TTL		pull_high
57	PRQ7	I	I_TTL		pull_low
58	BD15	B	I_TTL/O_CMOS	8/16 mA	
59	VCC6	P	M1386 CORE		Double bound
60	BD14	B	I_TTL/O_CMOS	8/16 mA	
61	BD13	B	I_TTL/O_CMOS	8/16 mA	
62	BD12	B	I_TTL/O_CMOS	8/16 mA	
63	BD11	B	I_TTL/O_CMOS	8/16 mA	
64	BD10	B	I_TTL/O_CMOS	8/16 mA	
65	GND5	P	M1386 CORE		Double bound
66	BD9	B	I_TTL/O_CMOS	8/16 mA	
67	BD8	B	I_TTL/O_CMOS	8/16 mA	
68	BD7	B	I_TTL/O_CMOS	8/16 mA	
69	BD6	B	I_TTL/O_CMOS	8/16 mA	
70	BD5	B	I_TTL/O_CMOS	8/16 mA	
71	BD4	B	I_TTL/O_CMOS	8/16 mA	
72	BD3	B	I_TTL/O_CMOS	8/16 mA	
73	BD2	B	I_TTL/O_CMOS	8/16 mA	
74	BD1	B	I_TTL/O_CMOS	8/16 mA	
75	BD0	B	I_TTL/O_CMOS	8/16 mA	
76	VCC7	P	M1386 CORE		Double bound
77	PDH	B	I_TTL/O_CMOS	8/16 mA	
78	PDL	B	I_TTL/O_CMOS	8/16 mA	
79	WEJ	O	O_CMOS	12/24 mA	
80	RAS3J	O	O_CMOS	12/24 mA	
81	GND6	P	M1386 CORE		Double bound
82	RAS2J	O	O_CMOS	12/24 mA	
83	RAS1J	O	O_CMOS	12/24 mA	
84	RAS0J	O	O_CMOS	12/24 mA	
85	VCC8	P	IO VDD		Double bound
86	CAS3HJ	O	O_CMOS	8/16 mA	
87	CAS3LJ	O	O_CMOS	8/16 mA	
88	CAS2HJ	O	O_CMOS	8/16 mA	
89	CAS2LJ	O	O_CMOS	8/16 mA	
90	CAS1HJ	O	O_CMOS	8/16 mA	
91	CAS1LJ	O	O_CMOS	8/16 mA	
92	CAS0HJ	O	O_CMOS	8/16 mA	
93	CAS0LJ	O	O_CMOS	8/16 mA	
94	GND7	P	IO GND		Double bound
95	MA11	O	O_CMOS	12/24 mA	
96	MA10	O	O_CMOS		
97	MA9	O	O_CMOS	12/24 mA	
98	MA8	O	O_CMOS	12/24 mA	
99	MA7	O	O_CMOS	12/24 mA	
100	MA6	O	O_CMOS	12/24 mA	

Acer Laboratories, Inc., USA, 1830B Bering Drive, San Jose, CA 95112; Tel: 408-467-7456; Fax: 408-467-7474
www.acerlabs.com

Numerical Pin List : M6117C(M6117B) : (continued)

Pin No.	Pin Name	I/O	Cell Type	Ioh/Iol	internal pull
101	MA5	O	O_CMOS	12/24 mA	
102	MA4	O	O_CMOS	12/24 mA	
103	MA3	O	O_CMOS	12/24 mA	
104	GND8	P	IO GND		Double bound
105	VCC9	P	IO VDD		Double bound
106	MA2	O	O_CMOS	12/24 mA	
107	MA1	O	O_CMOS	12/24 mA	
108	MA0	O	O_CMOS	12/24 mA	
109	AS	O	O_CMOS	4/4 mA	
110	VBAT	P	RTC CORE		Double bound
111	XTAL1	I	RTC 32K XTAL		
112	XTAL2	I	RTC 32K XTAL		
113	PWG	I	I_TTL/ST		
114	RTCWJ	O	O_CMOS	4/4 mA	
115	RTCRJ	O	O_CMOS	4/4 mA	
116	IRQ8J	I	I_TTL		pull_high
117	OSC14M	I	I_TTL		
118	GND9	P	RTC CORE		Double bound
119	ROMKBCSJ	B	I_TTL/O_CMOS	6/8 mA	pull_high
120	CK7M	O	O_CMOS	6/8 mA	
121	GND10	P	KBC CORE		Double bound
122	KBCLK	B	KBC CLOCK	12/24 mA	pull_high
123	KBDATA	B	KBC DATA	12/24 mA	pull_high
124	IRQ1/KBINH	I	I_TTL		pull_high
125	VCC11	P	KBC CORE		Double bound
126	EXTSW2	I	I_TTL/ST		pull_high
127	EXTSW1	I	I_TTL/ST		pull_high
128	XD7	B	I_TTL/O_CMOS	6/8 mA	
129	XD6	B	I_TTL/O_CMOS	6/8 mA	
130	XD5	B	I_TTL/O_CMOS	6/8 mA	
131	XD4	B	I_TTL/O_CMOS	6/8 mA	
132	GND11	P	IO GND		Double bound
133	XD3	B	I_TTL/O_CMOS	6/8 mA	
134	XD2	B	I_TTL/O_CMOS	6/8 mA	
135	XD1	B	I_TTL/O_CMOS	6/8 mA	
136	XD0	B	I_TTL/O_CMOS	6/8 mA	
137	VCC12	P	IO VDD		Double bound
138	SD7	B	I_TTL/O_CMOS	8/16 mA	pull_high
139	SD6	B	I_TTL/O_CMOS	8/16 mA	pull_high
140	SD5	B	I_TTL/O_CMOS	8/16 mA	pull_high
141	SD4	B	I_TTL/O_CMOS	8/16 mA	pull_high
142	SD3	B	I_TTL/O_CMOS	8/16 mA	pull_high
143	SD2	B	I_TTL/O_CMOS	8/16 mA	pull_high
144	SD1	B	I_TTL/O_CMOS	8/16 mA	pull_high
145	SD0	B	I_TTL/O_CMOS	8/16 mA	pull_high
146	IRQ9	I	I_TTL		pull_high
147	IRQ7	I	I_TTL		pull_high
148	IRQ6	I	I_TTL		pull_high
149	IRQ5	I	I_TTL		pull_high
150	VCC13	P	M1217B CORE		Double bound

Numerical Pin List : M6117C(M6117B) : (continued)

Pin No.	Pin Name	I/O	Cell Type	Ioh/Iol	internal pull
151	IRQ4	I	I_TTL		pull_high
152	IRQ3	I	I_TTL		pull_high
153	IRQ10	I	I_TTL		pull_high
154	IRQ11	I	I_TTL		pull_high
155	IRQ12/MSDATA	B	Mouse DATA	12/24 mA	pull_high
156	GND12	P	IO GND		Double bound
157	VCC14	P	IO VDD		Double bound
158	IRQ14	I	I_TTL		pull_high
159	IRQ15	I	I_TTL		pull_high
160	ATCLK	O	O_CMOS	8/20 mA	
161	GND13	P	IO GND		Double bound
162	RSTDRV	O	O_CMOS	8/20 mA	
163	REFRESHJ	B	I_TTL/O_CMOS	8/16 mA	pull_high
164	IOCHRDY	I	I_TTL		pull_high
165	NOWSJ	I	I_TTL		pull_high
166	IORJ	B	I_TTL/O_CMOS	8/16 mA	pull_high
167	IOWJ	B	I_TTL/O_CMOS	8/16 mA	pull_high
168	SMEMRJ	O	O_CMOS	8/16 mA	
169	GND14	P	M1217B CORE		Double bound
170	SMEMWJ	O	O_CMOS	8/16 mA	
171	MRDJ	B	I_TTL/O_CMOS	8/16 mA	pull_high
172	MWTJ	B	I_TTL/O_CMOS	8/16 mA	pull_high
173	BALE	O	O_CMOS	8/16 mA	
174	SA19/HDCS0J	B	I_TTL/O_CMOS	8/16 mA	pull_high
175	SA18/HDCS1J	B	I_TTL/O_CMOS	8/16 mA	pull_high
176	SA17/HDENJ	B	I_TTL/O_CMOS	8/16 mA	pull_high
177	SA16	B	I_TTL/O_CMOS	8/16 mA	pull_high
178	SA15	B	I_TTL/O_CMOS	8/16 mA	pull_high
179	SA14	B	I_TTL/O_CMOS	8/16 mA	pull_high
180	SA13	B	I_TTL/O_CMOS	8/16 mA	pull_high
181	SA12	B	I_TTL/O_CMOS	8/16 mA	pull_high
182	VCC15	P	M1217B CORE		Double bound
183	SA11	B	I_TTL/O_CMOS	8/16 mA	pull_high
184	SA10	B	I_TTL/O_CMOS	8/16 mA	pull_high
185	SA9	B	I_TTL/O_CMOS	8/16 mA	pull_high
186	SA8	B	I_TTL/O_CMOS	8/16 mA	pull_high
187	SA7	B	I_TTL/O_CMOS	8/16 mA	pull_high
188	GND15	P	M1217B CORE		Double bound
189	SA6	B	I_TTL/O_CMOS	8/16 mA	pull_high
190	SA5	B	I_TTL/O_CMOS	8/16 mA	pull_high
191	SA4	B	I_TTL/O_CMOS	8/16 mA	pull_high
192	SA3	B	I_TTL/O_CMOS	8/16 mA	pull_high
193	SA2	B	I_TTL/O_CMOS	8/16 mA	pull_high
194	SA1	B	I_TTL/O_CMOS	8/16 mA	pull_high
195	SA0	B	I_TTL/O_CMOS	8/16 mA	pull_high
196	IOCHKJ	I	I_TTL		pull_high
197	VCC16	P	IO VDD		Double bound
198	MASTERJ	I	I_TTL		pull_high
199	AEN	B	I_TTL/O_CMOS	8/16 mA	pull_high
200	BHEJ	B	I_TTL/O_CMOS	8/16 mA	pull_high

Acer Laboratories, Inc., USA, 1830B Bering Drive, San Jose, CA 95112; Tel: 408-467-7456; Fax: 408-467-7474
www.acerlabs.com

Numerical Pin List : M6117C(M6117B) : (continued)

Pin No.	Pin Name	I/O	Cell Type	Ioh/Iol	internal pull
201	LA23	B	I_TTL/O_CMOS	8/16 mA	pull_high
202	LA22	B	I_TTL/O_CMOS	8/16 mA	pull_high
203	LA21	B	I_TTL/O_CMOS	8/16 mA	pull_high
204	LA20	B	I_TTL/O_CMOS	8/16 mA	pull_high
205	LA19	B	I_TTL/O_CMOS	8/16 mA	pull_high
206	LA18	B	I_TTL/O_CMOS	8/16 mA	pull_high
207	LA17	B	I_TTL/O_CMOS	8/16 mA	pull_high
208	GND16	P	IO GND		Double bound

2.4 Alphabetical Pin List - M6117C(M6117B)

Pin no.	Pin name	Type
199	AEN	B
54	ADSJ	O
109	AS	O
160	ATCLK	O
42	BA25	O
43	BA24	O
44	BA23	O
45	BA2	O
173	BALE	O
41	BCLK1	O
39	BCLK2	I
58	BD15	B
60	BD14	B
61	BD13	B
62	BD12	B
63	BD11	B
64	BD10	B
66	BD9	B
67	BD8	B
68	BD7	B
69	BD6	B
70	BD5	B
71	BD4	B
72	BD3	B
73	BD2	B
74	BD1	B
75	BD0	B
36	BEHJ	O
37	BELJ	O
200	BHEJ	B
47	BUSY7J	I
86	CAS3HJ	O
87	CAS3LJ	O
88	CAS2HJ	O
89	CAS2LJ	O

Pin no.	Pin name	Type
90	CAS1HJ	O
91	CAS1LJ	O
92	CAS0HJ	O
93	CAS0LJ	O
120	CK7M	O
33	CPUSPD	O
16	DACK0J	B
18	DACK1J	B
20	DACK2J	B
22	DACK3J	B
24	DACK5J	B
26	DACK6J	B
28	DACK7J	B
49	DCJ	O
17	DRQ0	I
19	DRQ1	I
21	DRQ2	I
23	DRQ3	I
25	DRQ5	I
27	DRQ6	I
29	DRQ7	I
35	ENPOWER	O
55	ERR7J	I
126	EXTSW2	I
127	EXTSW1	I
8	GND1	P
31	GND2	P
40	GND3	P
52	GND4	P
65	GND5	P
81	GND6	P
94	GND7	P
104	GND8	P
118	GND9	P
121	GND10	P

Alphabetical Pin List (continued)

Pin no.	Pin name	Type
132	GND11	P
156	GND12	P
161	GND13	P
169	GND14	P
188	GND15	P
208	GND16	P
3	IO16J	I
196	IOCHKJ	I
164	IOCHRDY	I
166	IORJ	B
167	IOWJ	B
124	IRQ1/KBINH	I
151	IRQ4	I
152	IRQ3	I
116	IRQ8J	I
153	IRQ10	I
154	IRQ11	I
155	IRQ12/ MSDATA	B
146	IRQ9	I
147	IRQ7	I
148	IRQ6	I
149	IRQ5	I
158	IRQ14	I
159	IRQ15	I
122	KBCLK	B
123	KBDATA	B
201	LA23	B
202	LA22	B
203	LA21	B
204	LA20	B
205	LA19	B
206	LA18	B
207	LA17	B
95	MA11	O
96	MA10	O
97	MA9	O
98	MA8	O
99	MA7	O
100	MA6	O
101	MA5	O
102	MA4	O
103	MA3	O
106	MA2	O
107	MA1	O
108	MA0	O
198	MASTERJ	I
4	MEM16J	I
50	MIOJ	O
171	MRDJ	B

Pin no.	Pin name	Type
172	MWTJ	B
165	NOWSJ	I
34	MSCLK (NC)	B
56	NPRDY7J	I
117	OSC14M	I
77	PDH	B
78	PDL	B
57	PRQ7	I
113	PWG	I
80	RAS3J	O
82	RAS2J	O
83	RAS1J	O
84	RAS0J	O
30	RDYOJ	O
163	REFRESHJ	B
119	ROMKBCSJ	B
51	RSTNP	O
162	RSTDRV	O
114	RTCWJ	O
115	RTCRJ	O
174	SA19/ HDCS0J	B
175	SA18/ HDCS1J	B
176	SA17/ HDENJ	B
177	SA16	B
178	SA15	B
179	SA14	B
180	SA13	B
181	SA12	B
183	SA11	B
184	SA10	B
185	SA9	B
186	SA8	B
187	SA7	B
189	SA6	B
190	SA5	B
191	SA4	B
192	SA3	B
193	SA2	B
194	SA1	B
195	SA0	B
145	SD0	B
144	SD1	B
143	SD2	B
142	SD3	B
141	SD4	B
140	SD5	B
139	SD6	B
138	SD7	B
5	SD8	B

Acer Laboratories, Inc., USA, 1830B Bering Drive, San Jose, CA 95112; Tel: 408-467-7456; Fax: 408-467-7474
www.acerlabs.com

Alphabetical Pin List (continued)

Pin no.	Pin name	Type
6	SD9	B
7	SD10	B
9	SD11	B
10	SD12	B
11	SD13	B
12	SD14	B
13	SD15	B
168	SMEMRJ	O
170	SMEMWJ	O
32	SPKR	O
14	TC	O
2	TESTJ	I
110	VBAT	P
1	VCC1	P
15	VCC2	P
38	VCC3	P
46	VCC4	P
53	VCC5	P
59	VCC6	P
76	VCC7	P

Pin no.	Pin name	Type
85	VCC8	P
105	VCC9	P
125	VCC11	P
137	VCC12	P
150	VCC13	P
157	VCC14	P
182	VCC15	P
197	VCC16	P
79	WEJ	O
48	WRJ	O
128	XD7	B
129	XD6	B
130	XD5	B
131	XD4	B
133	XD3	B
134	XD2	B
135	XD1	B
136	XD0	B
111	XTAL1	I
112	XTAL2	I

Note : B : Bidirectional P : Power pin
O : Output I : Input

2.5 Hardware Power-On Setup

Hardware Power-On Setup Table of M6117C

Pin No.	Pin Name	Index	Setup	Configuration
26	DACK6J	34H : D[5]	pull high	5V Vdd
22	DACK3J	34H : D[3]	pull low	PS/2 keyboard IRQ1 timing select
20	DACK2J	34H : D[2]	pull high	Internal RTC enable
16	DACK0J	34H : D[0]	pull high	Normal
119	ROMKBCSJ	35H : D[7]	pull low	Normal
24	DACK5J	35H : D[6]	pull low	Normal
28	DACK7J	35H : D[5]	pull high	Internal Keyboard Controller enable
18	DACK1J	35H : D[1]	pull high	Memory parity check enable

Section 3 : Function Description

The M6117C is designed to perform like Intel• 386SX system with deep green features. Aside from the 386SX core, it contains (1) PS2/AT Keyboard Controller and Mouse, (2) Real Time Clock to store system boot data, (3) Integrated System Peripheral to serve the peripheral requests, (4) Power Management Unit to reduce the chip's power consumption efficiently, (5) LS245 : TTL data buffer between ISA data bus SD[7:0] and ROM data bus XD[7:0], (6) DRAM Controller for four banks memory module supporting EDO and Fast Page Mode with page interleave and up to 64M bytes space. The M6117C offers the following blocks :

- Static 386SX Core
- Reset and Clock logic
- CPU Interface logic
- Numerical Coprocessor Interface logic
- DRAM Controller
- Configuration Registers
- ISA Bus Interface logic
- Control logic
- Address Decode and Memory Mapping logic
- Data Buffer
- Address Buffer
- ISP Devices (82C37x2, 82C59x2, 82C54, 74LS612)
- Real Time Clock
- PS2/AT Keyboard Controller
- Keyboard and Speaker logic
- Parity Generation and Checking logic
- Power Management Unit
- WATCHDOG timer
- 16 bits GPI/O
- IDE decoder interface

3.1 Static 386SX Core

The 386SX core is the same as M1386SX of Acer Labs. Inc. and 100% object code compatible with the Intel 386SX microprocessor. System manufacturers can provide 386 CPU based systems optimized for both cost and size. Instruction pipelining and high bus bandwidth ensure short average instruction execution times and high system throughput. Furthermore, it can keep the state internally from charge leakage while external clock to the core is stopped without storing the data in registers. The power consumption here is almost zero when clock stops. The internal structure of this core is 32-bit data and address bus with very low supply current, 116 mA in the conditions of 5.0V, 20MHz, room temperature. Real mode as well as Protected mode are available and can run MS-DOS, MS-Windows, OS/2 and UNIX.

3.2 Reset and Clock logic

The switching power supply sends a PWG (power good signal) to M6117C to generate system reset signals, like RSTDRV, RSTNP, and resets the chip to initial state. Also the reset signal can be generated by internal emulation RC reset and shutdown cycle.

There are two clock inputs: BCLK2 and OSC are 2X system clock and 14.318MHz respectively, and three clock outputs: BCLK1, ATCLK1 and CK7M which provide frequency operation for the system board and devices depending on 1X clock used. The BCLK1 is a half frequency of BCLK2. The CK7M derived from the OSC input (divided by 2) is available as the keyboard controller clock when power is on. To increase system performance, the M6117C supports variable AT clocks for faster ISA add-on cards. When the CPU accesses the register programmed special address range, the AT clock changes to a faster speed. The non-programmed address regions keep the normal speed. There are eight programmable frequencies of the ATCLK1 which can change on fly by different specific addresses and determined by D[2:0] of local port 1EH in both high and normal speed. Please refer to Section 4.2 index 1EH. This optional AT clock can achieve a higher performance when a faster add-on card is used.

3.3 CPU Interface logic

The CPU interface logic decodes the status MIO, DC, WR with different equivalents. It will handle all the CPU-side instruct events and requests the bus ownership from the CPU.

3.4 Numerical Coprocessor Interface Logic

For 386SX systems, M6117C monitors ERRORJ, BUSYJ and PEREQ to support the 80387SX coprocessor. When ERRORJ is active, this indicates that an unmaskable coprocessor exception has occurred and the coprocessor interface will generate an IRQ13 to the CPU core. (If it immediately follows a RESET signal, this indicates that a coprocessor is present in the system.) The coprocessor asserts BUSYJ signal while executing and asserts NPRDYJ to M6117C when it is finished. The coprocessor interface then passes the NPRDYJ signal to RDYOJ signal to the CPU core. This ready signal has to meet the CPU requirement.

Acer Laboratories, Inc., USA, 1830B Bering Drive, San Jose, CA 95112; Tel: 408-467-7456; Fax: 408-467-7474
www.acerlabs.com

3.5 DRAM Controller

The DRAM controller supports Fast Page Mode DRAM and EDO DRAM. The DRAM controller is capable of accessing up to 64 MBytes of local memory, and supporting four banks page interleave of DRAM using 256K, 512K, 1M, 2M, 4M, 16M single sided SIMMs. Page interleave mechanism is able to shorten the memory read/write cycle and raise the data access speed between host and RAM, and works on any two banks with the same DRAM type. Each bank can be disabled through software, please refer to 4.3 memory type configuration and 4.2 index 10H. When using EDO DRAMs, only page mode are enabled. Programmable DRAM timing is provided for RAS pre-charge time and RAS-to-CAS delay to achieve highest performance and reliability, this part is described in 4.2 index 11H and 12H. And they also explain how to use the 256/384KB memory remapping feature in unshadowed RAM region from A0000H to FFFFFH. The A0000H to BFFFFH region can set to shadow enable, please refer to index 3CH and 12H. Programmable shadowing features are supported on 32K boundaries between C0000H and FFFFFH regions (768KB..1MB), please refer to 4.2 index 14H and 15H. It also supports 'RAS only', 'CAS before RAS' refresh type of DRAMs.

3.6 Configuration Registers

The configuration register controls the whole system of the environment under different frequencies. It enables the system to set these configuration registers to meet the compatible, reliable performance and functional requirements.

3.7 ISA Bus Interface Logic

This block includes the ISA bus state machine, 16-bit or 8-bit commands justified, command wait states and control logic. These signals are compatible with PC/AT standards.

3.8 Control logic

The control logic controls the internal data bus and address bus flow. It also generates proper read-select to internal device and uses multiplexer to choose the correct data output. It selects the correct address bus for DMA and refresh cycles to send to system.

3.9 Address Decode and Memory Mapping logic

The 16-bit address decode-circuit fully decodes the BIOS ROM, keyboard controller, internal ISP devices, real time clock, port 61H, and configuration registers. When remap is enabled, it decodes the remap memory to the end of DRAM.

3.10 Data Buffer

This block generates signals which control data transfer between the CPU core data bus, memory data bus and ISA data bus during CPU cycles, ISA bus cycles, DMA cycles and master cycles. Moreover, we added LS245, TTL data buffer between ISA data bus SD[7:0] and ROM data bus XD[7:0], on the ASIC. So that users could save some external TTL logic.

3.11 Address Buffer

The address buffer generated at address SA1, SA0 and BHEJ for ISA bus, initiates the byte-enable signal at DMA and master cycles.

3.12 ISP Devices (82C37x2, 82C59x2, 82C54, 74LS612)

The integrated system peripheral (ISP) devices are built-in, thus no 82C206 is required. There are two 82C37s, two 82C59s, one 82C54 and one 74LS612 built-in devices.

Note : The function of 82C54 has some limitations, please see appendix C

3.13 Real Time Clock

The real time clock (RTC) device is built-in, thus no external RTC is required. If the user does not use the internal RTC for something else, then it can be disabled by hardware setting, please refer to 2.5 Hardware setting.

3.14 PS2/AT Keyboard Controller

The PS2/AT keyboard controller (KBC) device is built-in, and support with Mouse, thus no external KBC is required. If the user does not use the internal KBC for something else, then it can be disabled by hardware setting, please refer to 2.5 Hardware setting.

3.15 Keyboard and Speaker logic

This block emulates the keyboard controller fast-RC and fast gate-A20 functions for maximum performance. It combines with port 61H at this block to generate speaker signal.

3.16 Parity Generating and Checking logic

During a local memory read cycle, M6117C not only monitors bus steering, but also checks the parity bit for each data byte from DRAM to ensure the correct data is read. If a parity error occurs, parity checking logic drives PCHERRJ active to peripheral decoder logic. Then peripheral decoder produces an NMI to CPU core for the parity error. The parity checking can be disabled by hardware setting (pull XDACK1J low). During a local memory write cycle, M6117C monitors the bus steering and uses the accepted data to generate the parity bit sent to DRAM for each data byte.

3.17 Power Management Unit

The M6117C Power management unit includes SMM, I/O trap, APM, external SMI switch control and programmable clock timeout unit for I/O device. The PMU strictly controls and dramatically reduces overall system power consumption. This is accomplished via the activity monitors which detect the system inactivity timer timeout, and signals the power saving device to remove the power sources from various peripherals. The M6117C provides one timer from one-second to 300 minutes to monitor the system states (ON/DOZE/ STANDBY/ SUSPEND modes). The M6117C provides an LED flash control to indicate the system state status. The M6117C supports external SMI switch into suspend mode, SMI setup, and wakeup events (RTC alarm). The M6117C also provides the interaction control for SMIJ and CPURST.

3.17.1 SMM Control Logic

M6117C supports internal 386SX core SMM mode, the M6117C will record these SMI events as :

a. Time-out events : PMU - Mode timeout

b. I/O trap events : VGA device access
harddisk device access
line-printer device access
General I/O port access
General memory ports access

c. External device events : IRQ• active
DRQ• active
Input devices active
External suspend-switch
RTC alarm
SMI setup-switch

d. Software SMI event

3.17.2 APM

The APM (Advanced Power Management interface) creates an interface to allow the OS to communicate with the SMM code. The M6117C provides the configuration index 56H bit 6 to generate the software SMIJ signal for APM applications.

3.18 16 sets of GPI/O power control signals

The M6117C provides 16 expandable GPOs and 16 expandable GPIs. The user can program at most 16 signals to control power, flash disk, IDE, LED... and so on application for peripheral devices. Please refer to Section 4.9.

3.19 WATCHDOG timer

The M6117C has watchdog timer function for monitoring whether the system is still work or not after a period of time. If the system happened some error or hanged up, it cause the timer timed out, then a system reset or NMI or IRQ may happen decided by BIOS programming. The WATCHDOG timer source is 32.768 Khz frequency to counter a 24 bits counter such that the timer range is from 30.5 usecs to 512 secs with resolution 30.5 usecs. Please refer to Section 4.10 .

3.20 IDE decoder interface

The M6117C adds IDE decoder interface for PIO mode signal. It provides one channel connected with external HDD by decoding SA[15:0] with two kinds of channel selectable.

	HDCS0J	HDCS1J
primary channel	1F0 - 1F7	3F6
secondary channel	170 - 177	376

Acer Laboratories, Inc., USA, 1830B Bering Drive, San Jose, CA 95112; Tel: 408-467-7456; Fax: 408-467-7474
www.acerlabs.com

Section 4 : Configuration Registers

4.1 How to read/write to configuration registers

The read/write configuration register is the first index to be processed. On board I/O port 22h is the index register and I/O port 23h is the data register. To read a configuration register, write the index value to I/O port 22h in advance, then read data from I/O port 23h. To write a configuration register, write the index value to I/O port 22h, then write data to I/O port 23h. For instance, if we want to read the data of configuration register which index is 10h, the steps are :

1) Write 10h (index) to I/O port 22h
2) Read data from I/O port 23h

If we want to write data 55h to configuration register which index is 12h, then the steps are :

1) Write 12h (index) to I/O port 22h
2) Write data 55h to I/O port 23h

*The steps of locking/unlocking the configuration registers :

```
OUT 22h, 13h  (Enable 13h)
OUT 23h, C5h  (Unlock)
OUT 22h, XXh  (XX = Configuration Index)
OUT 23h, YYh  (YY = Configuration data)
OUT 22h, XXh
OUT 23h, YYh (Configuration can be written
              repeatedly)
   :
OUT 22h, 13h  (Enable 13h)
OUT 23h, 00h  (Lock)
```

4.2 Hardware Power-On setup

Refer to Section 2.5 Hardware power on setup table.

4.3 Memory Controller

4.3.1 How to enable/disable memory controller

Memory Controller will be enabled if D[7] of index 20h is set to 1, or is disabled if reset D[7]. All original DRAM cycles (local memory cycles) will turn to ISA bus cycles when memory controller is disabled. Meanwhile, WEJ and all RAS and CAS signals are driven high when memory controller does not work. In other words, all memory access will be treated as memory accessing on ISA bus. Related timing waveforms, please refer to Section 6.

4.3.2 How to set memory mode

Memory Controller supports Fast Page Mode and EDO DRAMs. D[0] of index 37h is set to select which kind of DRAM. The default is set to 0 to select Fast Page Mode DRAM. When D[0] of index 37h is set to 1, DRAM is set to EDO DRAM. EDO and Fast Page Mode DRAMs can not mix, so only one type of DRAM can exist at the same time. See Table below. There are 32 modes of memory type configuration which supports up to six DRAM types including 256K, 512K, 1M, 2M, 4M and 16M. Moreover, various 30-pin or 72-pin single-sided SIMMs are available on system design. We also offer a memory autosizing method described by flow chart (Section 5.4 -A) to detect memory mode on board as user• reference.

Table 4-3 Memory Type Configuration

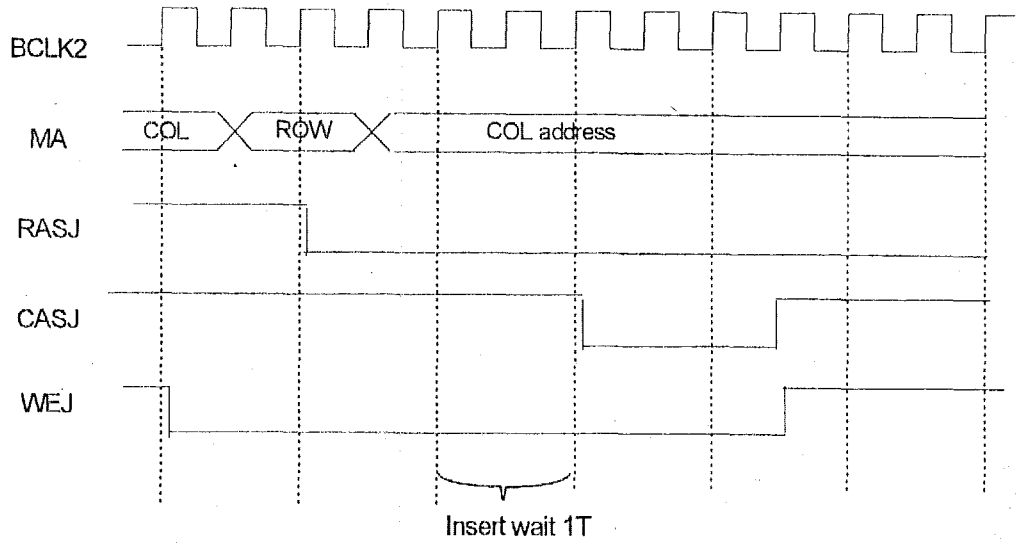
Mode	BANK 0	BANK 1	BANK 2	BANK 3	Total Space	PM(4:0)
0	256K*2	256K*2			1M Bytes	0 0 0 0 0
1	256K*2	256K*2	256K*2	256K*2	2M Bytes	0 0 0 0 1
2	256K*2	256K*2	1M *2		3M Bytes	0 0 0 1 0
3	256K*2	256K*2	1M *2	1M *2	5M Bytes	0 0 0 1 1
4	256K*2	256K*2	4M *2		9M Bytes	0 0 1 0 0
5	512K*2				1M Bytes	0 0 1 0 1
6	512K*2	512K*2			2M Bytes	0 0 1 1 0
7	512K*2	512K*2	1M *2		4M Bytes	0 0 1 1 1
8	512K*2	512K*2	1M *2	1M *2	6M Bytes	0 1 0 0 0
9	512K*2	512K*2	4M *2		10M Bytes	0 1 0 0 1
10	512K*2	512K*2	4M *2	4M *2	18M Bytes	0 1 0 1 0
11	512K*2	1M *2			3M Bytes	0 1 0 1 1
12	512K*2	1M *2	1M *2		5M Bytes	0 1 1 0 0
13	512K*2	4M *2			9M Bytes	0 1 1 0 1
14	1M *2				2M Bytes	0 1 1 1 0
15	1M *2	1M *2			4M Bytes	0 1 1 1 1
16	1M *2	1M *2	1M *2		6M Bytes	1 0 0 0 0
17	1M *2	1M *2	1M *2	1M *2	8M Bytes	1 0 0 0 1
18	1M *2	1M *2	4M *2		12M Bytes	1 0 0 1 0
19	1M *2	1M *2	4M *2	4M *2	20M Bytes	1 0 0 1 1
20	1M *2	4M *2			10M Bytes	1 0 1 0 0
21	1M *2	4M *2	4M *2		18M Bytes	1 0 1 0 1
22	1M *2	4M *2	4M *2	4M *2	26M Bytes	1 0 1 1 0
23	2M *2				4M Bytes	1 0 1 1 1
24	2M *2	2M *2			8M Bytes	1 1 0 0 0
25	2M *2	2M *2	4M *2	4M *2	24M Bytes	1 1 0 0 1
26	2M *2	4M *2			12M Bytes	1 1 0 1 0
27	4M *2				8M Bytes	1 1 0 1 1
28	4M *2	4M *2			16M Bytes	1 1 1 0 0
29	4M *2	4M *2	4M *2		24M Bytes	1 1 1 0 1
30	4M *2	4M *2	4M *2	4M *2	32M Bytes	1 1 1 1 0
31	16M *2	16M *2			64M Bytes	1 1 1 1 1

Ver. 0.93, Document Number: 6117DSC1.doc
Acer Laboratories, Inc., USA, 1830B Bering Drive, San Jose, CA 95112; Tel: 408-467-7456; Fax: 408-467-7474
www.acerlabs.com

4.3.3 DRAM timing control

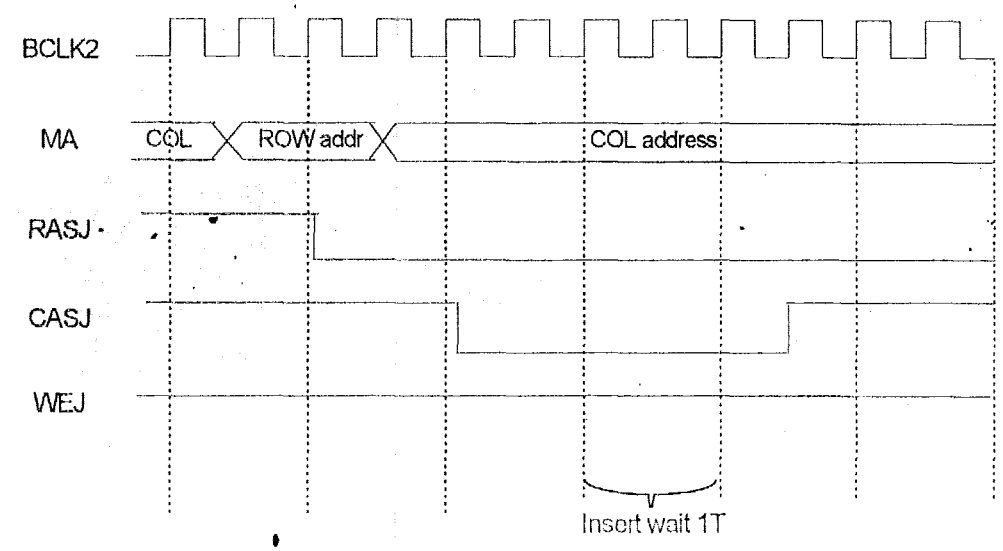
(a) Memory write access time insert wait

CASJ precharge time (high time) will last for one more T-cycle before its falling edge if D[7] of index 11h is set to high.



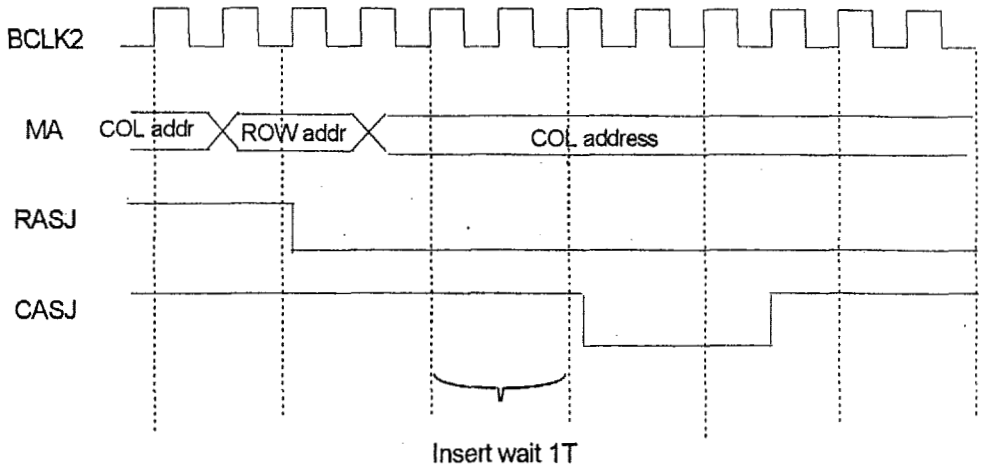
(b) Memory CAS read access time insert wait

When memory read cycles. CASJ active time (low time) will last for one more T-cycle before its rising edge if D[6] of index 11h is set to high.



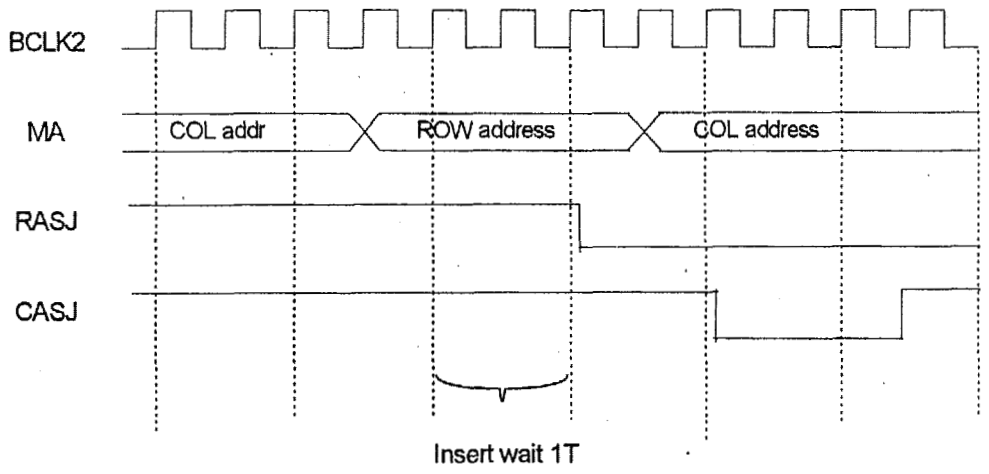
(c) CAS precharge time insert wait

Whatever memory read or write, it will insert 1T wait between the falling edges of both RASJ and CASJ, if D[4] of index 11h is set to high.



(d) RAS active time insert wait

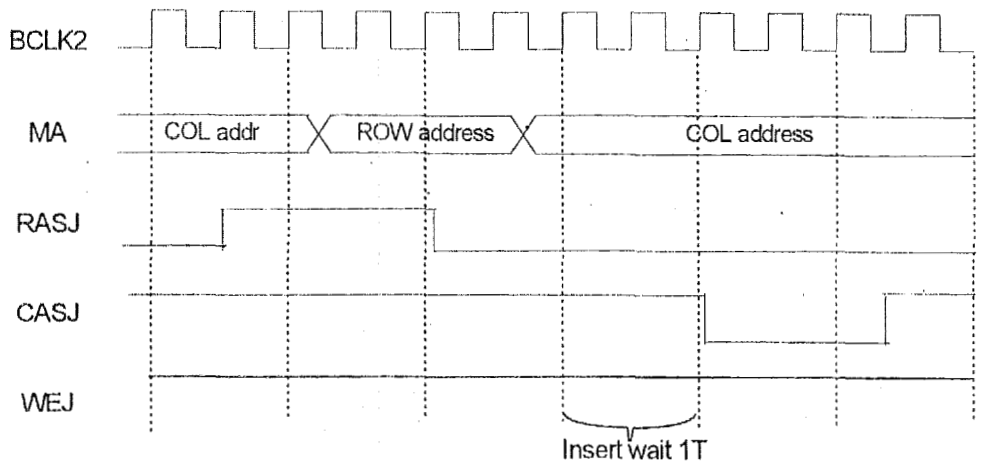
If RAS is originally in miss state, that is RASJ = 1. We are able to prolong the inactive state of RAS for an extra 1T, before its falling edge, if D[3] of index 11h is set to high.



Acer Laboratories, Inc., USA, 1830B Bering Drive, San Jose, CA 95112; Tel: 408-467-7456; Fax: 408-467-7474
www.acerlabs.com

(e) Memory miss read RAS to CAS insert wait

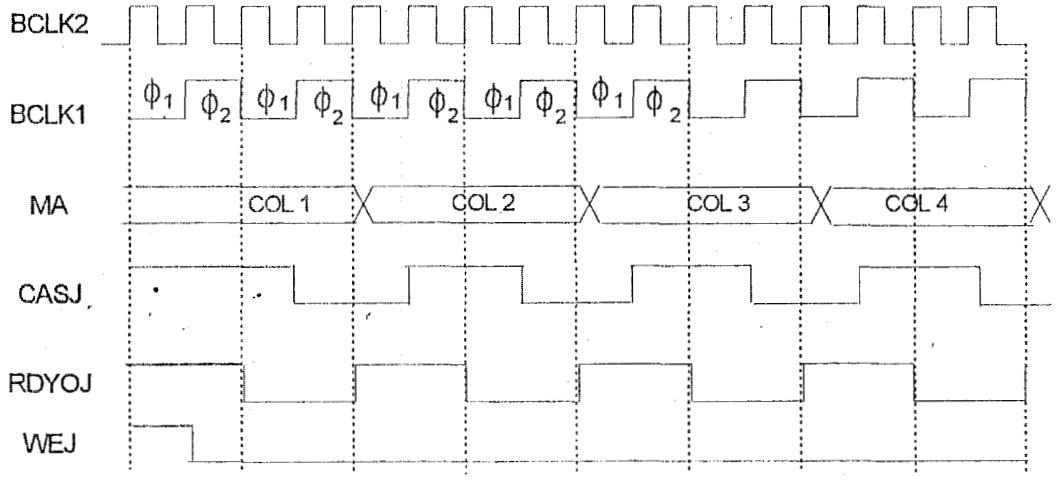
When memory read miss. We can add 1T wait between the falling edges of both RASJ and CASJ, if D[2] of index 12h is set to high.



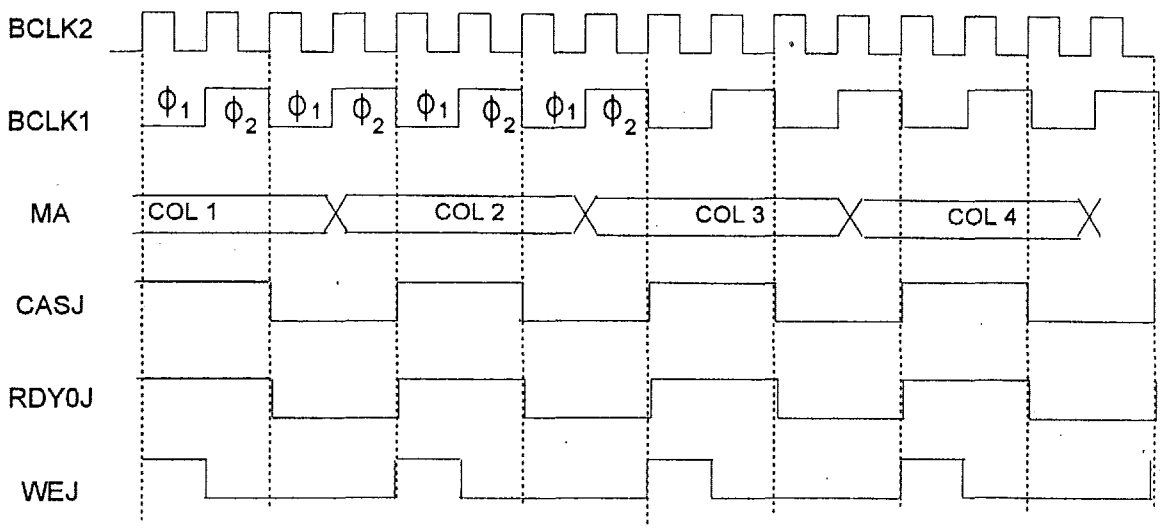
(f) Memory fast write hit insert wait

When memory write hits. This factor is capable of activating CASJ at phase 1 or phase 2 of BCLK2. If D[0] of inde 12h is set to low, M6117C will activate CASJ at phase 2. In other words, the active CASJ will lag active RDYOJ by ha T-cycle. That is early ready timing. If D[0] of index 12h is set to high, chip will activate CASJ at phase 1. So tha active CASJ and RDY0J will be at the same phase.

(A) D[0] of index 12h is low



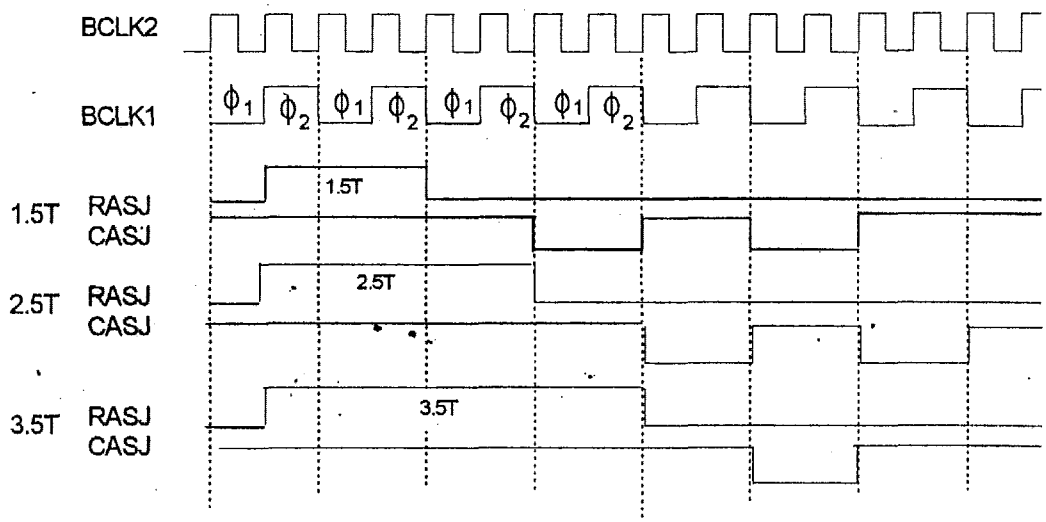
(B) D[0] of index 12h is high



(g) RAS precharge time insert wait

There are two bits to control RAS precharge timing, one is D[5] of index 11h, another is D[0] of the same index. Table as follows is the precharge time related to bit setting.

Index 11h : D[5]	D[0]	Pre-charge time
0	0	2.5T
0	1	1.5T
1	X	3.5T

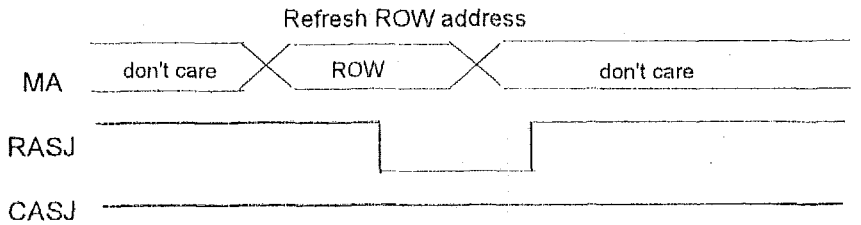


4.3.4 DRAM refreshing

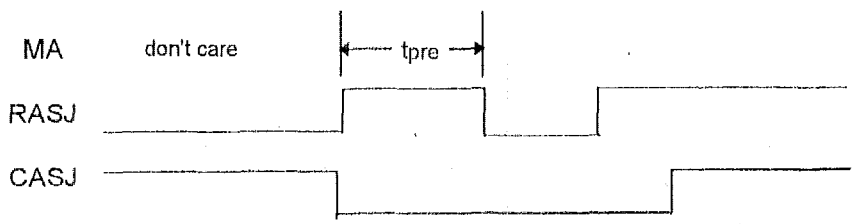
DRAM refresh cycle is 256 for every 4ms. However, for some DRAM products, the data at capacitance can b maintained more than 15 us. To get higher system performance, we can slow down refresh period for some DRAM products. By programming index 36h : D[5-4], we can get slow refresh period as follows:

Index 36h : D[5]	D[4]	Refresh period setting
0	0	15 us
0	1	120 us
1	0	15 us
1	1	60 us

We also support three types of DRAM refresh :
(A) RAS only refresh
This kind of refresh will be selected if D[1] of index 10h is set to '0' (low)



(B) CAS before RAS refresh
This kind of refresh will be selected if D[1] of index 10h is set to '1' (high)



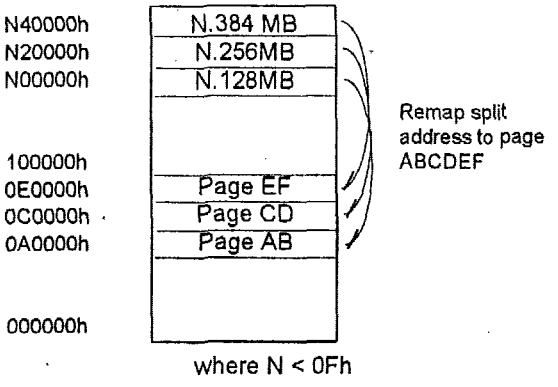
tpre : RAS precharge time

4.3.5 How to remap memory to top of memory

The address 0C0000h ~ 0FFFFFh is for ROM or reserved area. When this range of RAM is not used as shadow RAM, then the memory is wasted. However, we can use this area of RAM by remapping the higher address to the non-shadow RAM. In order to have continuous memory mapping, the remapping address selects the higher area to do so. For example, if we have 12MB DRAM on board, shadow RAM all disabled, then we can select 13MB (00D00000h) to be the address to remap. There are two remapping types determined by D[1] of index 11h, split (D[1]=0) and move-out (D[1]=1). By programming D[7-4] of index 12h, we can set the split address A23-A20. Memory split remapping can be disabled by programming index 11h : D[2] to 0. Please notice that index 11h : D[2] must be set to 1 and index 11h : D[1] must be set to 0 if you want to enable split remap. Otherwise, the move-out remap is selected when index 11h: D[1] is set to 1. Following are the different cases of memory remapping.

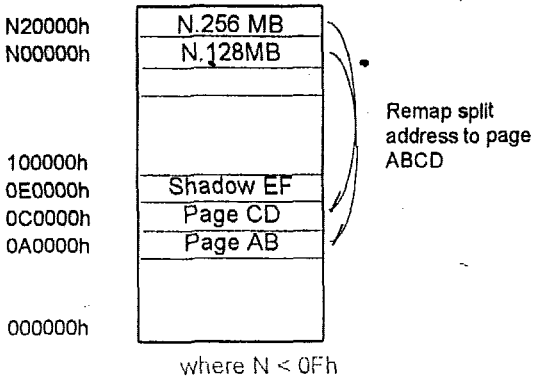
(A) No shadow, split address[23-20] = A[23-20], Remap type = split remap

A19	A18	A17	A16	PA23	PA22	PA21	PA20	PA19	PA18	PA17
0	0	0	x	0	0	0	0	1	0	1
0	0	1	x	0	0	0	0	1	1	0
0	1	0	x	0	0	0	0	1	1	1



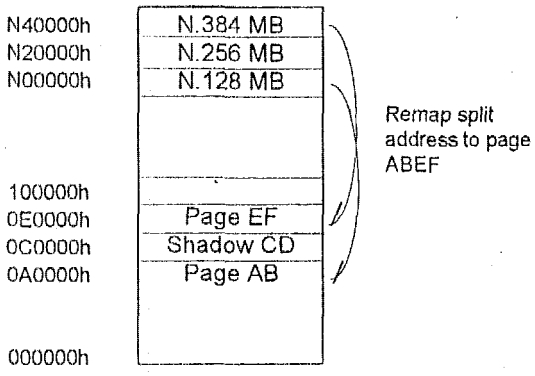
(B) Shadow Page EF, split address[23-20] = A[23-20], Remap type = split remap

A19	A18	A17	A16	PA23	PA22	PA21	PA20	PA19	PA18	PA17
0	0	0	x	0	0	0	0	1	0	1
0	0	1	x	0	0	0	0	1	1	0



(C) Shadow Page CD, split address[23-20] = A[23-20], Remap type = split remap

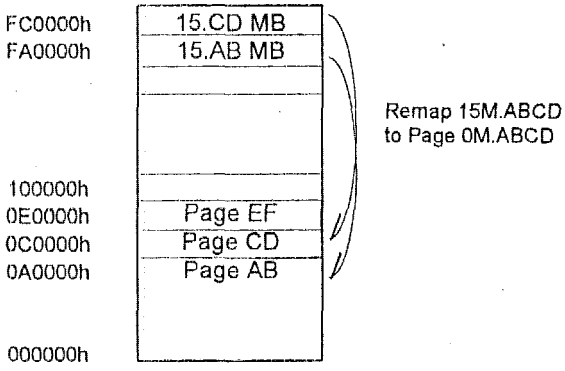
A19	A18	A17	A16	PA23	PA22	PA21	PA20	PA19	PA18	PA17
0	0	0	x	0	0	0	0	1	0	1
0	1	0	x	0	0	0	0	1	1	1



where N < 0Fh

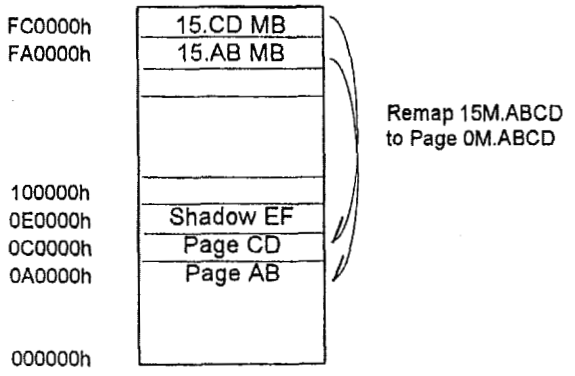
(D) No shadow, A[20-23]=1111, Remap type =Move-out remap and memory mode cannot be 10, 19, 21, 22, 25, 28, 29, 30, 31

A19	A18	A17	A16	PA23	PA22	PA21	PA20	PA19	PA18	PA17
1	0	1	x	0	0	0	0	1	0	1
1	1	0	x	0	0	0	0	1	1	0



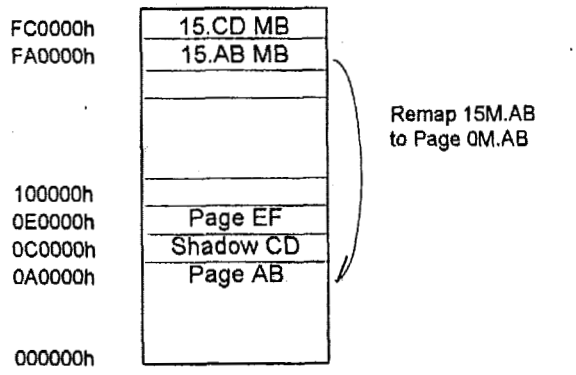
(E) Shadow EF, A[20-23] = 1111, Remap type=Move-out remap and memory mode cannot be 10, 19, 21, 22, 25, 28, 29, 30, 31

A19	A18	A17	A16	PA23	PA22	PA21	PA20	PA19	PA18	PA17
1	0	1	x	0	0	0	0	1	0	1
1	1	0	x	0	0	0	0	1	1	0



(F) Shadow CD, A[20-23]=1111, Remap type=Move-out remap and memory mode cannot be 10, 19, 21, 22, 25, 28, 29, 30, 31

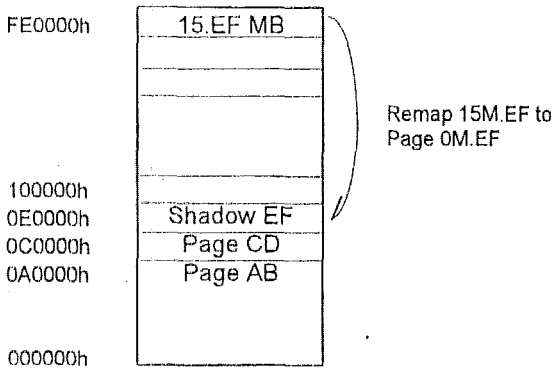
A19	A18	A17	A16	PA23	PA22	PA21	PA20	PA19	PA18	PA17
1	0	1	x	0	0	0	0	1	0	1



Ver. 0.93, Document Number: 6117DSC1.doc
Acer Laboratories, Inc., USA, 1830B Bering Drive, San Jose, CA 95112; Tel: 408-467-7456; Fax: 408-467-7474
www.acerlabs.com

(G) Shadow EF, A[20-23]=1111, or A[20-23] = 0000, Remap type=Move-out remap and memory mode cannot be 10, 19, 21, 22, 25, 28, 29, 30, 31

A19	A18	A17	A16	PA23	PA22	PA21	PA20	PA19	PA18	PA17
1	1	1	x	0	0	0	0	1	1	1



4.3.6 Shadow Control

The address 0C0000h~ 0FFFFFh is for ROM or unused areas. When the range address is used for ROM, the system ca use RAM to map the area to enhance performance. The address 0C0000~0FFFFFh is also called shadow region. Th shadow region can be read/write control. For example, when it is read enable, write disable, the operation of shadow RA is the same as ROM. By programming index 14h, 15h, BIOS programmer can set the read/write control function t initialize the system. Suppose 0C0000h~0C7FFFh video ROM is present, and 0C0000h~ 0C7FFFh region wants to b shadowed, then the setup steps are :

(1) Index 14h : D[1-0] = 00, Shadow region 0C0000h ~ 0C7FFFh read/write both disable. Allow the process to be read from ROM only.

(2) Index 14h : D[1-0] = 10, Shadow region 0C0000h ~ 0C7FFFh read disable, write enable. Allow the process to be rea from ROM and write to DRAM (Shadow RAM). Then video ROM data copies to shadow RAM region.

(3) Index 14h : D[1-0] = 01, Shadow region 0C0000h ~0C7FFFh read enable, write disable. Allow the operation to be rea from DRAM. The video ROM read process is via shadow RAM 0C0000h~ 0C7FFFh region, not via video ROM.

The range of 0C0000h ~ 0FFFFFh partitions to eight blocks, each block is 32KB. Each shadow region block can be shadow read/write disabled or enabled by programming index 14h, 15h.

The A0000h ~ B0000h region can set to shadow enable. If host read/write this region, the data will read/write from local memory when enable shadow.

How to set shadow A/B region ?
Index 3ch:
bit 3 = 0, Disable shadow A/B function
bit 3 = 1, Enable shadow A/B function
Index 12h:
bit 1 = 0, Disable shadow A/B region. All access to A/B memory region will pass to ISA
bit 1 = 1, Enable shadow A/B region. All access to A/B memory region will be at local memory.
Only both the index 3ch and 12h enable, the shadow A/B region will enable.

4.3.7 BIOS ROM control

The size of BIOS ROM can be 64KB or 128KB. When using 64KB BIOS ROM, then index 10h:D[0] sets to '0', and the address used can be 0F0000h~0FFFFFh or 0FF0000h ~ 0FFFFFFh. When 128KB BIOS ROM is used, index 10h : D[0] sets to '1' and the address used is 0E0000h~ 0FFFFFh. The BIOS ROM can be replaced by flash ROM to support ROM BIOS updatable by software program. Flash ROM is writable by activating write enable pin. Before writing data to flash ROM, we have to set index 20h : D[2] to '1'. Otherwise, flash ROM cannot accept it.

4.3.8 On board 15M~ 16M memory enable/disable control

On board 0F00000h~ 0FFFFFFh memory can be enabled/disabled by programming index 10h : D[2]. When D[2] =0, the on board 15M ~16M memory will be recognized as local memory. When D[2] =1, the on board 15M~16M range memory will not be recognized, and will be treated as ISA range.

Acer Laboratories, Inc., USA, 1830B Bering Drive, San Jose, CA 95112; Tel: 408-467-7456; Fax: 408-467-7474
www.acerlabs.com

4.4 ISA Bus Interface Control

The ISA bus controller and ISP devices are developed and verified by ALi's M1487/M1489 series.

4.4.1 ISA ATCLK frequency control

After powering-on, the default value of ISA ATCLK is 7.159 Mhz, this clock is changed by programming index 1Eh : D[2:0] to set to different frequencies to meet the system designer requirements.

Index 1Eh

D[2]	D[1]	D[0]	ATCLK
0	0	0	7.159 MHz (def)
0	0	1	PCLK2/3
0	1	0	PCLK2/4
0	1	1	PCLK2/5
1	0	0	PCLK2/6
1	0	1	PCLK2/8
1	1	0	PCLK2/10
1	1	1	PCLK2/12

Note : PCLK2 means doubled CPU clock

To make sure the system boots normally, the default ATCLK is 7.159 Mhz. So system can boot at any CPU frequency. After powering on, BIOS can detect the CPU frequency, and set the desired AT clock frequency. For example, if CPU running at 40 Mhz, the PCLK2 will be 80 Mhz and if we choose D[2-0] = 110, this means ATCLK =PCLK2/10, then ATCLK is 8 MHz.

Note : The 82C54 has some limitations which require ISA ATCLK set as 7.159 MHz. Please refer to Appendix C.

4.4.2 I/O Recovery Control

For old slow ISA cards, I/O recovery time must be added to back ISA I/O commands. If an I/O writes too fast, the previous I/O write data will be overlaid by the later one, so the card will fail. The I/O recovery time recommends value of 500 ns and set by index 33h : D[7-4]. Notice that you have to enable the I/O recovery time in advance, otherwise index 33h : D[7-4] will not work. We separated on-chip decoded I/O port control and general purpose I/O port control to index 33h : D[2] and D[3] respectively.

Index 33h

D[7-4]	I/O recovery time
0000	0 (default)
0001	250 ns
0010	500 ns
0011	750 ns
0100	1000 ns
0101	1250 ns
0110	1500 ns
0111	1750 ns
1000	2000 ns
1001	2250 ns
1010	2500 ns
1011	2750 ns
1100	3000 ns
1101	3250 ns
1110	3500 ns
1111	3750 ns

D[3]	I/O recovery
0	disable
1	enable

D[2]	On chip I/O recovery
0	disable
1	enable

4.4.3 ISA high speed change on fly

Since ISA bus is slow, our ISA high speed change-on-fly function permits ISA card operating in higher ATCLK during specific I/O or memory accessing cycles. Index 16h ~ 18h are used to define and mask ISA memory address set 1 which will fly to higher ATCLK frequency when setting addresses are matched. Index 19h~ 1Bh are used to define and mask ISA memory address set 2 which will fly to higher ATCLK frequency when setting addresses are matched. Index 1Ch~ 1Dh are used to define and mask ISA I/O address. For instance, if the CPU clock is 40 Mhz, PCLK2 = 80 Mhz and Index 1Eh : D[2:0] = 011, then the AT clock will be PCLK2/5 = 16 Mhz in normal speed address, and PCLK2/4 = 20 Mhz in high speed address.

High frequency	Index 1Eh : D[2:0]	Normal frequency
7.159 MHz	000	7.159 Mhz
PCLK2/2	001	PCLK2/3
PCLK2/3	010	PCLK2/4
PCLK2/4	011	PCLK2/5
PCLK2/5	100	PCLK2/6
PCLK2/6	101	PCLK2/8
PCLK2/8	110	PCLK2/10
PCLK2/10	111	PCLK2/12

Normal frequency goes to high frequency when address matches. High frequency goes back when address does not match.

Acer Laboratories, Inc., USA, 1830B Bering Drive, San Jose, CA 95112; Tel: 408-467-7456; Fax: 408-467-7474
www.acerlabs.com

4.5 Power Management

In M6117C internal circuit, it has an internal signal SMIJ which is used to inform system to enter Power Management (Hyper State Mode; HSM) space if an event happens. However, there are other extra choices to achieve power management. They are NMI, IRQ3, IRQ4, IRQ5, IRQ6, IRQ7, IRQ9, IRQ10, IRQ11, IRQ12, IRQ14, and IRQ15. Chip only switches the SMI signal to internal NMI or IRQs if it is selected when an event happens. Index 55h : D[1-0] and Index 38h : D[3-0] determine which one is selected. *Please refer to Appendix A and B for CPU information in this section.*

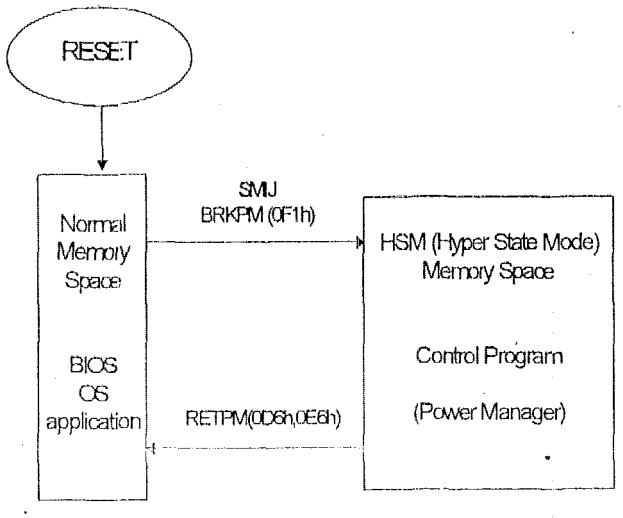
Index 55h

Index 55h : D[1-0]	Index 38h : D[3-0]	timing selection
00	xxxx	SMI timing support
01	xxxx	NMI timing support
10	xxxx	IRQ15 timing support
11	0011	IRQ3 timing support
11	0100	IRQ4 timing support
11	0101	IRQ5 timing support
11	0110	IRQ6 timing support
11	0111	IRQ7 timing support
11	1001	IRQ9 timing support
11	1010	IRQ10 timing support
11	1011	IRQ11 timing support
11	1100	IRQ12 timing support
11	1110	IRQ14 timing support
11	others	reserved

Notice that the system is not going to enter HSM space if you choose NMI or other IRQs timing support. All power management using NMI and other IRQs timing are handled by software.

4.5.1 SMI Structure

Like other green CPUs, M6117 needs a special memory space to place Hyper State Mode (HSM) routine if you would like to use M6117 deep green features. We can enter HSM space by hardware SMIJ (System Management Interrupt) or instruction BRKPM, OP code : 0F1h, and return by instruction RETPM, OP code: 0D6h, 0E6h. How to use power management functions efficiently to minimize the system power consumption is a challenge.



4.5.2 System Management Interrupt (SMI)

System management interrupt has the most priority to cause M6117C entering HSM space. Like non-maskable interrupt (NMI), M6117C will jump to SMI entry point or starting address, ROM area 0FFFFFF90h in default, after accepting SMI, this chip thus has entered HSM space. Depending on different applications, we can also change the page address of SMI entry point by way of updating the value of UGRS' which is a special 32-bit register in M6117 CPU core. For instance, if we write 0A0000h to UGRS before activating SMI, then M6117 will jump to 00AFF90h when SMI asserts. Following is a short sample of assembly code to implement the example above.

```
MOV EAX, 0A0000h
; LDUSR  UGRS, EAX
DB 0D6h, 0CAh, 03h, 0A0h
```

4.5.3 Enter and Exit the HSM(Hyper State Mode) space

If we select SMI to implement Power Management Mode, the CPU will switch memory space to HSM while an event happens. Also, we can use the instruction BRKPM(**opcode - DB 0F1h**) to enter HSM. The PV monitor interrupt (set by ***CR03h PMON***) and opcode trap(set by ***CR0Eh TOP*** and ***CR0Fh TCON***) can do the same operation.

The BRKPM (**opcode - DB 0F1h**) instruction or any equivalent interrupt transfers the values, which have been set before its generation with the registers indispensable in controlling the processor operation, to the high-order general purpose registers(shown below), and then switches the processor space to shift control to the specific physical addresses. Then CPU is brought to the following reset state:

16-bit context
Real Mode Addressing
Paging Off
Interrupt Disable

SMI, BRKPM instruction
PV monitor, and opcode trap

GR31	***CR00h***	- CR0
GR30	***EFLAGS***	
GR29	***AR1***	- CS-LIMIT
GR28	***SR1***	- CS-BASE
GR27	***EIP***	
GR26		
GR25		CS

EXP monitor

GR31	***CR00h***	- CR0
GR30	***EFLAGS***	
GR29	***AR1***	- CS-LIMIT
GR28	***SR1***	- CS-BASE
GR27	***EIP***	
GR26		EXT#
GR25		CS

The RETPM (**opcode - DB 0D6h 0E6h**) instruction causes the reverse operation of the BRKPM instruction. It returns control to the normal context. Note that save/restore is done only for ***CR00h***, ***AR1***, ***EFLAGS***, ***SR1***, ***EIP***, and ***CS*** for both **BRKPM** and **RETPM** instructions. Therefore, the contents of the other registers must be saved and restored by software.

When the original operating system is in protected mode and we enter HSM space, we need to handle the segment base and attribute registers carefully when the segment register is changed. Also, we can get other information from reading the ***GR31 - GR25*** to do our application.

4.6 The way to generate system management interrupt

4.6.1 Mode timer time-out

There is a Mode timer in power management unit of M6117C, this is based on 14.318 Mhz frequency input. There are four time bases 1 sec, 10 secs, 1 min and 10 mins. The timer counter is from 0 to 15, so the combinations are as follows :

Time count	Time base			
	1 sec	10 sec	1 min	10 min
0	0 sec	0 sec	0 min	0 mins
1	1 sec	10 secs	1 min	10 mins
2	2 secs	20 secs	2 mins	20 mins
3	3 secs	30 secs	3 mins	30 mins
4	4 secs	40 secs	4 mins	40 mins
5	5 secs	50 secs	5 mins	50 mins
6	6 secs	60 secs	6 mins	60 mins
7	7 secs	70 secs	7 mins	70 mins
8	8 secs	80 secs	8 mins	80 mins
9	9 secs	90 secs	9 mins	90 mins
10	10 secs	100 secs	10 mins	100 mins
11	11 secs	110 secs	11 mins	110 mins
12	12 secs	120 secs	12 mins	120 mins
13	13 secs	130 secs	13 mins	130 mins
14	14 secs	140 secs	14 mins	140 mins
15	15 secs	150 secs	15 mins	150 mins

e.g. There are two possible settings for two minutes, (a) 10 secs time base, time count is 12.
(b) 1 minute time base, time count is 2.

The BIOS should choose the small time base to reduce time shift. In other words, small time base is more precise than bigger time base.

The configuration register index 64h is the timer setting.

Index 64h

D[7-4]	Time count
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	10
1011	11
1100	12
1101	13
1110	14
1111	15

Index 64h

D[2]	Timer count/reset
0	Timer reset
1	Timer count

D[1-0]	Time base
0 0	1 second
0 1	10 seconds
1 0	1 minute
1 1	10 minutes

Once the mode timer counter is equal to the time count setting, then the time-out occurs. Time-out will generate a system management interrupt and index 5Bh will show 06h to manifest the interrupt cause. Notice that you should set index 59h : D[5] to '1' to activate mode timer even if you have already set the index 64h. We can prolong time delay for double of our time setting by programming index 69h : D[4] to '1' which is double counter time base control bit. When timer is enabled, any access to monitored device range, the corresponding timer will reset and restart counting until time-out is reached. For instance, mode timer will be reset when any key on keyboard is pressed for keyboard is the most possible monitored peripheral device.

4.6.2 Interrupt request active

System will generate SMI as soon as any channel of 8259• IRQ happened if we have already programmed index 5Ch and 5Dh. System• definite instant SMI cause from it has the most priority among all interrupts. Moreover, index 57h:D[3] and index 5Ah :D[5] should be set to '1' to enable IRQ trigger SMI. Index 5Bh will show 08h after this SMI has occurred.

Index	5Ch	Index	5Dh
D[7]	IRQ7 selected	D[7]	IRQ15 selected
D[6]	IRQ6 selected	D[6]	IRQ14 selected
D[5]	IRQ5 selected	D[5]	IRQ13 selected
D[4]	IRQ4 selected	D[4]	IRQ12 selected
D[3]	IRQ3 selected	D[3]	IRQ11 selected
D[2]	NMI selected	D[2]	IRQ10 selected
D[1]	IRQ1 selected	D[1]	IRQ9 selected
D[0]	IRQ0 selected	D[0]	IRQ8 selected

The M6117C has 49 configuration registers, these registers reside at I/O port 23H (read/write), with index at output port 22H. Table 4-1 lists the internal registers summary. Section 4.2 describes the bit function of internal registers. Table 4-3 lists memory type configuration.

4.6.3 DMA channel request active

System will generate SMI as soon as any channel of 8237• DRQ happened if we have already programmed index 5Eh, besides index 57h :D[7] and index 5Ah : D[6] should be set to '1' to enable DRQ trigger SMI. Here, index 5Bh will show 09h after this SMI has occurred.

Index 5Eh	
D[7]	DRQ7 selected
D[6]	DRQ6 selected
D[5]	DRQ5 selected
D[4]	DRQ4 selected
D[3]	DRQ3 selected
D[2]	DRQ2 selected
D[1]	DRQ1 selected
D[0]	DRQ0 selected

4.6.4 IN access

IN access will happen when monitoring IRQ12, IRQ4, IRQ3 and IRQ1 (always enable) are asserting. Index 66h:D[7-5] is to define which IRQ channel is enabled as IN access, notice that IRQ1 is always enabled. SMI occurs when monitoring IN access is activating and then index 5Bh will show 0Ah.

Index 66h	
D[7]	IN monitor IRQ3 select
D[6]	IN monitor IRQ4 select
D[5]	IN monitor IRQ12 select

4.6.5 External switch

There are two external trigger signals to generate SMI, external SMI switch input (EXTSW2) and external suspend switch input (EXTSW1), both of them have the same function-trigger SMI. Index 58h :D[7-6] are the enable bits to EXTSW2 and EXTSW1 respectively. Each input trigger polarity can choose low-to-high active or high-to-low active or both depending on index 67h: D[1-0] respectively. These two input pins has internal debouncing circuit to prevent the miss action. But you can bypass internal debouncing circuit . Index 37H: D[5:4] are enable bits to bypass EXTSW2 and EXTSW1 internal debouncing circuit respectively. Index 5Bh will show 0Ch if SMI cause from EXTSW1 and 10h if SMI cause from EXTSW2.

4.6.6 Real Time Clock alarm

System will generate SMI when IRQ8 assert, if RTC has properly been programmed and the following control bits is set to '1' : index 57h :D[3], index 59h :D[6], index 5Ah :D[5] and index 5Dh:D[0]. Index 5Bh will show 0Dh after this event is asserted.

4.6.7 Software SMI

If index 56h :D[6] is set to '1' and D[7] of the same register is '1', system will generate SMI called software SMI. Index 5Bh will show 0Fh after this event.

4.6.8 VGA access

If index 57h:D[1] and index 5Ah:D[0] are set to '1', then system will generate SMI when memory write address matches 0A0000h~ 0B0000h with index 66h :D[0] is '1', or when I/O write address matches 3B0h~ 3BFh with index 66h :D[1] is '1'. Index 5Bh will show 11h after VGA access event.

4.6.9 Hard Disk Drive access

Hard disk drive operations will trigger SMI by programming following control bits to '1' = Index 57h :D[3], D[2] ; 5Ah:D[5], D[1] and 5Ch : D[5]. 12h will be shown on index 5Bh after HDD event.

4.6.10 Line Printer access

If indices 57h: D[4], D[3], 5Ah:D[5], D[2], and 5Ch:D[7] are set to '1', then it enables line printer access to activate SMI. Index 5Bh will show 13h to manifest SMI cause from line printer operation.

4.6.11 General purpose memory address access

If indices 6Ch, 6Dh, 6Eh, 6Fh were programmed in advance and control bits 58h:D[0], 5Ah:D[3], 6Bh :D[1] were set to '1' (enable), then system will generate SMI when memory read/write address matches the address defined in indices 6Ch~6Fh, called GP0 event. For example, If we write 10h to index 6Ch, 0Fh to index 6Dh, 00h to indices 6Eh and 6Fh, set index 58h: D[0], index 5Ah:D[3], index 6Bh: D[1] to high, then system will generate SMI when memory read/write address is the one during 1MB~2MB. Index 5Bh will show 14h after GP0 event.

4.6.12 General Purpose I/O address access

Similar to CP0 access, SMI will occur when I/O read/write address matches the address defined in index 70h, called GP1 event. In addition to programming 70h and 6Bh, index 58h: D[1], index 5Ah :D[4] and index 6Bh:D[0] should be set to '1', thus GP1 event will be enabled and index 5Bh will show 15h if any SMI occurs by GP1 event. Moreover, index 6Bh:D[7-4] offer three kinds of well defined I/O address group to cause GP1 event.

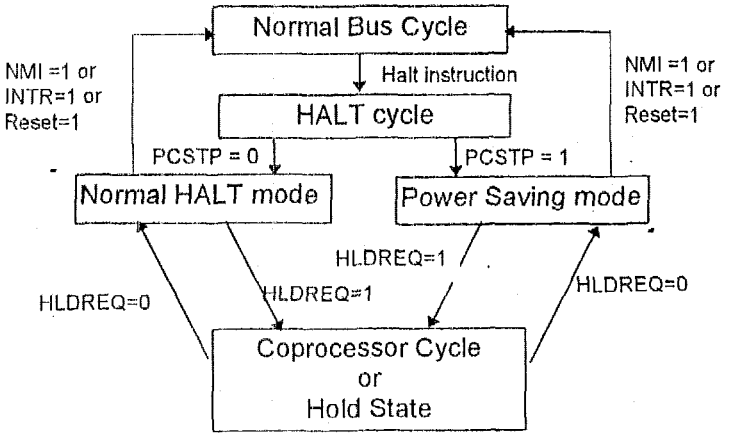
4.6.13 Special instruction to emulate SMI

Instruction BRKPM, OP code = 0F1h, will emulate M6117C entering HSM space as if SMI has occurred. This special instruction is only for emulation or testing, no programming is needed and of course index 5Bh will show nothing when designer uses BRKPM instruction.

4.7 How to enter power saving mode

M6117C can stop internal clock to its CPU core that will reduce almost 80% of its power consumption. Because our chip belongs to pure CMOS process, it will keep the internal states from leaking current when Vcc is still powered on and clock has stopped. There is one control bit, called as power clock stop (PCSTP), in internal control register to determine if M6117C is going to enter power saving mode or not. Here, we have a diagram to show the whole operation on accessing power saving mode. Notice that chip enters power saving mode by executing HALT instruction, and leave by any interrupt or reset.

M6117C Power Saving Mode



Question : How to set PCSTP ? (Power Clock Stop)

Answer : MOV EAX, 00008000h
DB 0D6h, 0FAh, 03h, 02h
/* MOV PWRCR, EA0 */

4.8 Speed LED flash control

System Designer can connect an LED to M6117C• CPUSPD pin to show LED flashing as a phenomenon in green mode. There are many varieties of LED flash combination including duty cycle, just program a value to index 71h. Then LED will start flashing.

Index 71h

D[4-2]	LED flash period
000	0.1 sec
001	0.2 sec
010	0.4 sec
011	0.8 sec
100	1.0 sec
101	2.0 sec
110	4.0 sec
111	8.0 sec

Index 71h

D[1-0]	LED duty cycle
00	disable
01	25%
10	50%
11	75%

4.9 General Purpose Output (GPO) and General Purpose Input (GPI)

M6117C supports 16 expandable GPOs and 16 expandable GPIs. During normal condition, pins XD[7:0] are data bus to peripheral devices. But during cold reset, XD[7:0] is an input pin and latched by internal register - index 68h; the pin ENPOWER is also active at this time to latch XD[7:0] at external 74LS373. Because there is no default value in index 68h and as to XD[7:0] without any pulling resistor. Designer has to connect externally pull-up or pull down resistors to XD[7-0] to initialize index 68h. The index 68h : D[7-0] are both readable and writable. If BIOS wants to change the external 74LS373 latch value. It should first set index 68h :D[7:0] a new value, then write any value to index 73h, that will generate an ENPOWER signal to update 74LS373 latch value. The index value in 68h will appear at XD[7:0] bus and ENPOWER will update the XD value to 74LS373.
Index 3Dh: The high byte GPO value . Default 00h R/W
Index 3Eh: The low byte GPI value. Default 00h Read only.
Index 3Fh: The high byte GPI value. Default 00h Read only.

4.9.1 Generate GPOs method

(1) Use external 2 X 74373 input connect to SD bus. The latch enable pin connects to ENPOWER.
(2) Set index 68h and 3Dh to desired GPO value.
(3) Write index 73h.
(4) Then data stored in index 68h will be sent to SD[7:0] and XD[7:0]. Data stored in Index 3Dh will be sent to SD[15:8] and ENPOWER will be active.
(5) The value will be latched by 74373.

4.9.2 Generate GPIs method

(1) Add external 2 X 74245, the input connects to GPIs, the output connects to ISA SD bus. The OE control connects to ISA REFRESHJ.
(2) When REFRESHJ is active, the SD will become input and M6117C will use MEMRJ rising edge to latch the SD value.
(3) Every 15us, the GPIs value will be updated.
(4) BIOS can read the GPI• value through index 3Fh which store SD[15:0] value.

Acer Laboratories, Inc., USA, 1830B Bering Drive, San Jose, CA 95112; Tel: 408-467-7456; Fax: 408-467-7474
www.acerlabs.com

4.10 Watchdog timer

The watchdog timer uses 32.768 kHz frequency source to count a 24-bit counter so the time range is from 30.5u sec to 512 sec with resolution 30.5u sec. When timer times out, a system reset, NMI or IRQ may happen to be decided by BIOS programming.

4.10.1 How to set the watchdog timer function ?

Index 37h :

Bit 6 = 0, Disable watchdog timer

Bit 6 = 1, Enable watchdog timer

Bit 7 = 0, Counter read mode. When read from index 3Bh, 3Ah, 39h, the return value is the setting counter value

Bit 7 = 1, Counter read mode. When bit 7 set from 0 to 1, the counter present value will be latched to buffer. When read from 3Bh, 3Ah, 39h, the return value is the buffer value. The counter will keep on counting.

Index 3Ch:

Bit 7 = 0, Read only, Watchdog timer time out event does not happen.

Bit 7 = 1, Read only, Watchdog timer time out event happens.

Index 3Bh, 3Ah, 39h : Counter

	3Bh	3Ah	39h
	D7.....D0	D7.....D0	D7.....D0
Counter	Most SBit .....•east SBit		

4.10.2 How to set the watchdog timer counter ?

(1) Set Bit 6 = 0 to disable the timer
(2) Write the desired counter value to 3Bh, 3Ah, 39h.
(3) Set Bit 6 = 1 to enable the timer, the counter will begin to count up.
(4) When counter reaches the setting value, the time out will generate signal setting by index 38h bit[7:4]
(5) BIOS can read index 3Ch Bit 7 to decide whether the Watchdog timeout event will happen or not.

Index 38h :

Bit[7:4] : time out generate signal select

Index 38h D[7:4]	timeout generate signal
0000	Reserved
0001	IRQ3
0010	IRQ4
0011	IRQ5
0100	IRQ6
0101	IRQ7
0110	IRQ9
0111	IRQ10
1000	IRQ11
1001	IRQ12
1010	IRQ14
1011	IRQ15
1100	NMI
1101	System reset
1110	Reserved
1111	Reserved

4.10.3 How to read the watchdog timer counter value when its counting ?

(1) Set Bit 7 = 1 to latch value

(2) Read the value in register index 3Bh,3Ah,39h. Then this is the on going value of counter.

4.10.4 How to clear the watchdog timer counter ?

(1) Set Bit 6 = 0 to disable timer. This will also clear counter at the same time.

4.11 IDE interface

4.11.1 How to enable IDE interface ?

(1) Enable IDE interface by writing index 37h bit [1] to 1 .

Index 37h : D[1]	D[1] = 0	D[1] = 1
Pin 174	SA19	HDCS0J
Pin 175	SA18	HDCS1J
Pin 176	SA17	HDENJ

(2) Write index 3Ch bit[0] to select IDE channel as primary or secondary.

Index 3Ch : D[0]	D [0] = 0	D[0] = 1
Channel	Primary	Secondary
HDCS0J	1F0 - 1F7	170 - 177
HDCS1J	3F6	376

Notice : Be careful with SA[19:17] bus, they must be separated. Please refer to APPLICATION.

Acer Laboratories, Inc., USA, 1830B Bering Drive, San Jose, CA 95112; Tel: 408-467-7456; Fax: 408-467-7474
www.acerlabs.com

4.12 Register Summary

Port	Index	Attribute	Register
22H		R/W	Index register.
23H	10H	R/W	ROM size, refresh type and DRAM mode setting.
23H	11H	R/W	Memory Controller function setting.
23H	12H	R/W	Split address and miscellaneous function
23H	13H	R/W	Lock register
23H	14H	R/W	Page CD shadow area setting
23H	15H	R/W	Page EF shadow area setting
23H	16H	R/W	ISA high speed memory range setting (group 1)
23H	17H	R/W	ISA high speed memory range setting (group 1)
23H	18H	R/W	ISA high speed memory range setting (group 1)
23H	19H	R/W	ISA high speed memory range setting (group 2)
23H	1AH	R/W	ISA high speed memory range setting (group 2)
23H	1BH	R/W	ISA high speed memory range setting (group 2)
23H	1CH	R/W	ISA high speed I/O range setting (group 1)
23H	1DH	R/W	ISA high speed I/O range setting (group 1)
23H	1EH	R/W	ISA AT Clock Definition
23H	20H	R/W	Address remap setting in Power saving mode
23H	30H	R	Clock speed status
23H	31H	R/W	Fast RC and Gate A20 setting
23H	32H	R/W	ISA high speed switching
23H	33H	R/W	I/O recovery setting
23H	34H	Read Only	Hardware power-on configuration port
23H	35H	Read Only	Hardware power-on configuration port
23H	36H	R/W	Chip version and miscellaneous function
23H	37H	R/W	Watchdog timer / external switch / mouse / IDE / EDO DRAM function enable
23H	38H	R/W	Watchdog time out report signal select / SMI relocate select
23H	39H	R/W	Watchdog timer counter value (byte 0)
23H	3AH	R/W	Watchdog timer counter value (byte 1)
23H	3BH	R/W	Watchdog timer counter value (byte 2)
23H	3CH	R/W	Watchdog time out dispatch / Memory shadow AB / IRQ level / EDO timing detect / IDE channel function selection
23H	3DH	R/W	GPO high byte storage register
23H	3EH	Read Only	GPI low byte storage register
23H	3FH	Read Only	GPI high byte storage register
23H	55H	R/W	Power Management Unit Configuration port (First)
23H	56H	R/W	Power Management Unit Configuration port (Second)
23H	57H	R/W	Mode Select of combination of Idle detection
23H	58H	R/W	Generating SMI source selecting
23H	59H	R/W	Timer time-out selection
23H	5AH	R/W	Peripheral access selection
23H	5BH	R/W	SMIJ cause setting
23H	5CH	R/W	IRQ event channel selected control (First)
23H	5DH	R/W	IRQ event channel selected control (Second)
23H	5EH	R/W	DRQ event channel selected control

Table 4-10 Configuration registers (continued)

Port	Index	Attribute	Register
23H	64H	R/W	Mode timer setting
23H	66H	R/W	VGA and IN monitor setting
23H	67H	R/W	EXTSW1/EXTSW2 input polarity select
23H	68H	R/W	Power on latched power control initial status
23H	69H	R/W	Timer counter status setting
23H	6AH	Read Only	DOZE, STANDBY and SUSPEND timer time-out status
23H	6BH	R/W	GP1 address function
23H	6CH	R/W	Define GP0 memory address A[23-16]
23H	6DH	R/W	Define GP0 memory address A[23-16] mask bits
23H	6EH	R/W	Define GP0 memory address A[25-24]
23H	6FH	R/W	Define GP0 memory address A[25-24] mask bits
23H	70H	R/W	Define GP1 I/O port address SA[9-2]
23H	71H	R/W	Mode LED function
23H	72H	R/W	Shadow I/O port for the data of port 70H
23H	73H	Write Only	Power control status output command.

Acer Laboratories, Inc., USA, 1830B Bering Drive, San Jose, CA 95112; Tel: 408-467-7456; Fax: 408-467-7474
www.acerlabs.com

4.13 Register Bit Definition

The details of M6117C configuration registers are described as follows :

PORT 22H default 00H

Bit	Description
7~0	Index of Configuration register

PORT 23H default 00H

Bit	Description
7~0	Data of Configuration register if unlock register unlocked.

INDEX 10H default 00H

Bit	Description
7~3	The five bits are used to set the memory type, the DRAM type is described in memory type table. D[7-4]=PM[3-0] & D3=PM4
2	The on board memory 15M ~ 16M-1 0 : enable 1 : disable
1	Two different refresh types: RAS only or CAS before RAS refresh. 0 : RAS only refresh 1 : CAS before RAS refresh
0	0 : 64KB ROM/EPROM (0F0000~0FFFFF/ FF0000~FFFFFF) 1 : 128K ROM/EPROM (0E0000~0FFFFF/ FE0000~FFFFFF)

INDEX 11H default F8H

Bit	Description
7	Memory write access time insert wait (BWAIT) 0 : disable 1 : enable
6	Memory CAS read access time insert wait (CASLWT) 0 : disable 1 : enable
5	Slow RAS precharge time 0 : see bit 0 below 1 : 3.5T
4	CAS precharge time insert wait for RAS to CAS delay (CASHWT) 0 : disable 1 : enable
3	RAS active time insert wait (RWAIT) 0 : disable 1 : enable
2	Memory remap 0 : disable 1 : enable
1	Select re-map mode 0 : split 1 : move-out
0	Fast RAS pre-charge time 0 : 2.5T 1 : 1.5T

INDEX 12H default 10H

Bit	Description
7~4	Split address SP[23-20]
3	reserved
2	Memory read miss RAS to CAS insert wait (MCASHWT) 0 : disable 1 : enable
1	Shadow RAM 0A0000h~0BFFFFh read/write control 0 : disable 1 : enable
0	Memory fast write hit insert phase (FSTWTHIT) 0 : PH2 1 : PH1

INDEX 13H	default 00H
Bit	Description
7-0	C5h : Unlock configuration Register 00h : Lock Configuration Register

INDEX 14H	default 00H
Bit	Description
7	Shadow RAM 0D8000~0DFFFF write 0 : disable 1 : enable
6	Shadow RAM 0D8000~0DFFFF read 0 : disable 1 : enable
5	Shadow RAM 0D0000~0D7FFF write 0 : disable 1 : enable
4	Shadow RAM 0D0000~0D7FFF read 0 : disable 1 : enable
3	Shadow RAM 0C8000~0CFFFF write 0 : disable 1 : enable
2	Shadow RAM 0C8000~0CFFFF read 0 : disable 1 : enable
1	Shadow RAM 0C0000~0C7FFF write 0 : disable 1 : enable
0	Shadow RAM 0C0000~0C7FFF read 0 : disable 1 : enable

INDEX 15H	default 00H
Bit	Description
7	Shadow RAM 0F8000~0FFFFF write 0 : disable 1 : enable
6	Shadow RAM 0F8000~0FFFFF read 0 : disable 1 : enable
5	Shadow RAM 0F0000~0F7FFF write 0 : disable 1 : enable
4	Shadow RAM 0F0000~0F7FFF read 0 : disable 1 : enable
3	Shadow RAM 0E8000~0EFFFF write 0 : disable 1 : enable
2	Shadow RAM 0E8000~0EFFFF read 0 : disable 1 : enable
1	Shadow RAM 0E0000~0E7FFF write 0 : disable 1 : enable
0	Shadow RAM 0E0000~0E7FFF read 0 : disable 1 : enable

INDEX 16H	default 00H
Bit	Description
7~4	ISA high speed memory address A[23-20] set 1
3~0	ISA high speed memory address A[19-16] set 1

INDEX 17H	default FFH
Bit	Description
7~4	ISA high speed memory address mask A[23-20] set 1
3~0	ISA high speed memory address mask A[19-16] set 1

Acer Laboratories, Inc., USA, 1830B Bering Drive, San Jose, CA 95112; Tel: 408-467-7456; Fax: 408-467-7474
www.acerlabs.com

INDEX 18H default F0H

Bit	Description
7~4	ISA high speed memory address mask A[15-12] set 1
3~0	ISA high speed memory address A[15-12] set 1

INDEX 19H default 00H

Bit	Description
7~4	ISA high speed memory address A[23-20] set 2
3~0	ISA high speed memory address A[19-16] set 2

INDEX 1AH default FFH

Bit	Description
7~4	ISA high speed memory address mask A[23-20] set 2
3~0	ISA high speed memory address mask* A[19-16] set 2

INDEX 1BH default F0H

Bit	Description
7~4	ISA high speed memory address mask* A[15-12] set 2
3~0	ISA high speed memory address A[15-12] set 2

INDEX 1CH default 00H

Bit	Description
7~4	ISA high speed I/O address A[9-6]
3~0	ISA high speed I/O address A[5-2]

INDEX 1DH default FFH

Bit	Description
7~4	ISA high speed I/O address mask* A[9-6]
3~0	ISA high speed I/O address mask* A[5-2]

** Mask Bit = 1 : Compare this address bit.*
0 : Do not compare this address.

INDEX 1EH default 00H

Bit	Description
7~3	reserved
2~0	ATCLK1 Definition

High Freq.	ATCLK[2:0]	Normal Freq.
14.318/2	0 0 0	14.318/2
PCLK2/2	0 0 1	PCLK2/3
PCLK2/3	0 1 0	PCLK2/4
PCLK2/4	0 1 1	PCLK2/5
PCLK2/5	1 0 0	PCLK2/6
PCLK2/6	1 0 1	PCLK2/8
PCLK2/8	1 1 0	PCLK2/10
PCLK2/10	1 1 1	PCLK2/12

INDEX 20H default 80H

Bit	Description
7	DRAM controller 0 : disable 1 : enable
6	reserved
5~4	Allocation remapping when SMI occurs, D3 must be enabled except remapping to ROM area 0 0 : Remap to ROM area 0 1 : Remap to page A,B 1 0 : Remap to page E 1 1 : Remap to page F
3	Remap SMI routine to local memory when this bit is set to high
2	Write to Flash ROM 0 : disable 1 : enable
1	DRAM self refresh (Index 10h D[1] must be 1) 0 : disable 1 : enable
0	Force ROM area remapping, disable remapping SMI routine to A, B page in local memory when this bit is set to high.

INDEX 30H	default 08H
Bit	Description
7~4	reserved, must be 0
3	CPU speed, read only. 0 : low 1 : high
2~0	reserved, must be 0

INDEX 31H	default 01H
Bit	Description
7~6	reserved
5	Fast reset state 0 : enable 1 : disable
4	reserved
3 ~ 1	reserved, must be 0
0	Fast gate A20 0 : disable 1 : enable

INDEX 32H	default 00H
Bit	Description
7	ISA I/O high speed 0 : disable 1 : enable
6	ISA memory high speed 0 : disable 1 : enable
5~1	reserved, must be 0.
0	Port F1 reset NP 0 : disable 1 : enable

INDEX 33H	default 00H
Bit	Description
7~4	I/O recovery period definition (unit : 250ns)
3	I/O recovery 0 : disable 1 : enable
2	On-chip I/O recovery 0 : disable 1 : enable
1	reserved
0	Must be 0

Ver. 0.93, Document Number: 6117DSC1.doc
Acer Laboratories, Inc., USA, 1830B Bering Drive, San Jose, CA 95112; Tel: 408-467-7456; Fax: 408-467-7474
www.acerlabs.com

INDEX 34H (Power-on setup, Read only)

Bit	No., Name	Description
7~6		reserved
5	26, DACK6J	reserved, must be pulled high
4	199, AEN	ISA clock test mode, when TESTJ = 0
3	22, DACK3J	PS/2 Keyboard IRQ1 timing select 1 : direct connect IRQ1 to 8259 0 : send IRQ1 to 8259 after latch bit is reversed when readout
2	20, DACK2J	Internal RTC 0 : disable 1 : enable
1		reserved
0	16, DACK0J	reserved, must be pulled high

INDEX 35H (Power-on setup, Read only)

Bit	No., Name	Description
7	119, ROMKBCSJ	reserved, must be pulled low.
6	24, DACK5J	reserved, must be pulled low.
5	28, DACK7J	Internal Keyboard Controller selection 0 : disable 1 : enable
4~2		reserved, must be 0.
1	18, DACK1J	Memory parity check 0 : disable 1 : enable
0		reserved

INDEX 36H default 00H

Bit	Description
7	16 bit ISA cycle insert 1 wait 0 : disable 1 : enable
6	PS/2 mouse IRQ12 timing 0 : direct connect IRQ12 to 8259 1 : send IRQ12 to 8259 after latch
5~4	Slow refresh control bits : Refresh period 0 0 : 15 us 0 1 : 120 us 1 0 : 15 us 1 1 : 60 us
3	reserved, must be 0
2~0	Chip version (Read only) 000 : M6117B 001 : M6117C

INDEX 37H default 00H

Bit	Description
7	Latch Watchdog timer value and counter read mode. 0: When read from Index3BH,3AH,39H, the read value is the setting counter value 1: When bit 7 set from 0 to 1, the counter present value will be latched to buffer. When read from Index 3BH,3AH,39H, the read value is the buffer value. The counter will keep on counting.
6	Watchdog timer 0: disable 1: enable
5	EXTSW2 de-bouncing circuit 0: enable internal de-bouncing circuit 1: bypass internal de-bouncing circuit
4	EXTSW1 de-bouncing circuit 0: enable internal de-bouncing circuit 1: bypass internal de-bouncing circuit
3	Reserved, must be 0
2	Internal mouse selection 0: disable 1: enable
1	IDE function 0: disable 1: enable
0	EDO DRAM mode 0: disable 1: enable

INDEX 38H default 00H

Bit	Description
7~4	Watchdog timer time out report signal select 0000: Reserved 0001: IRQ3 selected 0010: IRQ4 selected 0011: IRQ5 selected 0100: IRQ6 selected 0101: IRQ7 selected 0110: IRQ9 selected 0111: IRQ10 selected 1000: IRQ11 selected 1001: IRQ12 selected 1010: IRQ14 selected 1011: IRQ15 selected 1100: NMI selected 1101: system reset selected 1110: Reserved 1111: Reserved
3~0	SMI relocate to IRQ 0000: depend on Index 55H setting 0001: Reserved 0010: Reserved 0011: IRQ3 timing support 0100: IRQ4 timing support 0101: IRQ5 timing support 0110: IRQ6 timing support 0111: IRQ7 timing support 1000: Reserved 1001: IRQ9 timing support 1010: IRQ10 timing support 1011: IRQ11 timing support 1100: IRQ12 timing support 1101: Reserved 1110: IRQ14 timing support 1111: Reserved

INDEX 39H default 00H

Bit	Description
7~0	Watchdog timer counter value, byte 0

INDEX 3AH default 00H

Bit	Description
7~0	Watchdog timer counter value, byte 1

INDEX 3BH default 00H

Bit	Description
7~0	Watchdog timer counter value, byte 2

	3Bh	3Ah	39h
	D7.....D0	D7.....D0	D7.....D0
Counter	Most SBit .....•east SBit		

INDEX 3CH default 00H

Bit	Description
7	Watchdog timer time out 0: not happened 1: happened
6	Reserved
5	Reserved
4	Reserved
3	Memory Shadow A, B page function 0: disable 1: enable
2	IRQ Level trigger selection. 0: negative level trigger 1: level trigger
1	EDO DRAM timing detect mode 0: disable 1: enable
0	IDE channel selection 0: primary channel selected 1: secondary channel selected

INDEX 3DH default 00H

Bit	Description
7~0	GPO signals. When write index 73H, Bit 7~0 will sent to SD[15:8]

INDEX 3EH default 00H

Bit	Description
7~0	GPI signals. When REFRESHJ is active, the SD will become input and M6117C will use MEMRJ rising edge to latch the SD[7:0] to Bit 7~0. Read only

INDEX 3FH default 00H

Bit	Description
7~0	When REFRESHJ is active, the SD will become input and M6117C will use MEMRJ rising edge to latch the SD[16:8] to Bit 7~0. Read only.

INDEX 55H	default 00H
Bit	Description
7~2	reserved
1~0	SMI timing selection 0 0 : SMI timing support 0 1 : NMI timing support 1 0 : IRQ15 timing support 1 1 : reserved

INDEX 56H	default 00H
Bit	Description
7	SMI event signal 0 : disable 1 : enable
6	Software SMI 0 : disable 1 : enable
5~3	reserved
2	PMU state 1 : On 0 : Off
1~0	Power management system mode state 0 0 : ON 0 1 : DOZE 1 0 : STANDBY 1 1 : SUSPEND

INDEX 57H	default 00H
Bit	Description
7	DRQ select 1 : selected. If DRQ idle longer than mode timer setting. Mode timer time-out will signal. 0 : not selected.
6	FDD select (3F0H-3F7H) 1 : selected. If FDD cannot be accessed longer than mode timer setting. Mode timer time-out will signal. 0 : not selected.
5	COM select (3F8H-3FFH, 2F8H-2FFH, 3E8H-3EFH, 2E8H-2EEH) 1 : selected. If COM port cannot be accessed longer than mode timer setting. Mode timer time-out will signal. 0 : not selected.
4	LPT select (378H-37FH, 278H-27FH, 3BCH-3BEH) 1 : selected. If LPT port cannot be accessed longer than mode timer setting. Mode timer time-out will signal. 0 : not selected.
3	IRQ select 1 : selected. If IRQ idle longer than mode timer setting. Mode timer time-out will signal. 0 : not selected.
2	HDD select (1F0H-1F7H) 1 : selected. If HDD cannot be accessed longer than mode timer setting. Mode timer time-out will signal. 0 : not selected.
1	VGA select (Memory AB region write, 3B0H-3BFH write) 1 : selected. If VGA cannot be accessed longer than mode timer setting. Mode timer time-out will signal. 0 : not selected.
0	KBD select (60H, 64H) 1 : selected. If KBD cannot be accessed longer than mode timer setting. Mode timer time-out will signal. 0 : not selected.

INDEX 58H	default 00H
Bit	Description
7	EXTSW2 active (external SMI switch input, level trigger) 1 : EXTSW2 issues SMI 0 : no SMI from EXTSW2
6	EXTSW1 active (external suspend switch input, level trigger) 1 : EXTSW1 issues SMI 0 : no SMI from EXTSW1
5~2	reserved, must be 0000b.
1	GP1 select 1 : selected. If GP1 cannot be accessed longer than mode timer setting. Mode timer time-out will signal. 0 : not selected.
0	GP0 select 1 : selected. If GP0 cannot be accessed longer than mode timer setting. Mode timer time-out will signal. 0 : not selected.

INDEX 59H	default 00H
Bit	Description
7	reserved, must be 0.
6	RTC alarm (IRQ8J) SMI 0 : disable 1 : enable
5	MODE timer time-out signaling SMI 1 : enable 0 : disable
4~0	reserved, must be 00000b.

INDEX 5AH	default 00H
Bit	Description
7	IN signaling SMI 1 : enable 0 : disable
6	DRQ active (select DRQ source by register 5E) signaling SMI 1 : enable 0 : disable
5	IRQ active (select IRQ source by register index 5C,5D) signaling SMI 1 : enable 0 : disable
4	GP1 access (R/W GP1 defined area) signaling SMI 1 : enable 0 : disable
3	GP0 access (R/W GP0 defined area) signaling SMI 1 : enable 0 : disable
2	LPT access (R/W 378H-37FH or 278H-27FH) signaling SMI 1 : enable 0 : disable
1	HDD access (R/W port 1F0H-1F7H) signaling SMI 1 : enable 0 : disable
0	VGA access (W A0000H-BFFFFH, port 3B0H-3BFH) signaling SMI 1 : enable 0 : disable

Ver. 0.93, Document Number: 6117DSC1.doc
Acer Laboratories, Inc., USA, 1830B Bering Drive, San Jose, CA 95112; Tel: 408-467-7456; Fax: 408-467-7474
www.acerlabs.com

INDEX 5BH default 00H

Bit	Description
7~5	reserved
4~0	SMI Cause register 0 0 0 0 0 : None 0 0 1 1 0 : Mode Timer Time out 0 1 0 0 0 : IRQ active 0 1 0 0 1 : DRQ active 0 1 0 1 0 : IN access 0 1 1 0 0 : EXTSW1 active (external suspend switch input, level trigger) 0 1 1 0 1 : RTC alarm 0 1 1 1 1 : Software SMI 1 0 0 0 0 : EXTSW2 active (external next SMI switch input, level trigger) 1 0 0 0 1 : VGA access (W A0000H- BFFFFH, port 3B0H-3BFH) 1 0 0 1 0 : HDD access (R/W port 1F0H-1F7H) 1 0 0 1 1 : LPT access (R/W 378H-37FH or 278H-27FH) 1 0 1 0 0 : GP0 access (R/W GP0 defined area) 1 0 1 0 1 : GP1 access (R/W GP1 defined area) Others : reserved

INDEX 5CH default 00H

Bit	Description
	Select source to signal SMI when occur. This selection based on INDEX 5A bit 5 be set.
7	1: IRQ7 event selected 0 : IRQ7 event not selected
6	1: IRQ6 event selected 0 : IRQ6 event not selected
5	1: IRQ5 event selected 0 : IRQ5 event not selected
4	1: IRQ4 event selected 0 : IRQ4 event not selected
3	1: IRQ3 event selected 0 : IRQ3 event not selected
2	1: NMI event selected 0 : NMI event not selected
1	1: IRQ1 event selected 0 : IRQ1 event not selected
0	1: IRQ0 event selected 0 : IRQ0 event not selected

INDEX 5DH default 00H

Bit	Description
	Select source to signal SMI when happening. This selection based on INDEX 5A bit 5 is set.
7	1: IRQ15 event selected 0 : IRQ15 event not selected
6	1: IRQ14 event selected 0 : IRQ14 event not selected
5	1: IRQ13 event selected 0 : IRQ13 event not selected
4	1: IRQ12 event selected 0 : IRQ12 event not selected
3	1: IRQ11 event selected 0 : IRQ11 event not selected
2	1: IRQ10 event selected 0 : IRQ10 event not selected
1	1: IRQ9 event selected 0 : IRQ9 event not selected
0	1: IRQ8 event selected 0 : IRQ8 event not selected

INDEX 5EH default 00H

Bit	Description
	Select source to signal SMI when happened. This selection based on INDEX 5A bit 6 is set.
7	1: DRQ7 event selected 0 : DRQ7 event not selected
6	1: DRQ6 event selected 0 : DRQ6 event not selected
5	1: DRQ5 event selected 0 : DRQ5 event not selected
4	1: DRQ4 event selected 0 : DRQ4 event not selected
3	1: DRQ3 event selected 0 : DRQ3 event not selected
2	1: DRQ2 event selected 0 : DRQ2 event not selected
1	1: DRQ1 event selected 0 : DRQ1 event not selected
0	1: DRQ0 event selected 0 : DRQ0 event not selected

INDEX 64H	default 00H
Bit	Description
7~4	Set the time delay 0 : Timer disable 1-15 : Time count
3	reserved
2	Timer count/reset 1 : Timer count 0 : Timer reset
1~0	Time Base select for Mode timer 0 0 : 1 sec. 0 1 : 10 sec. 1 0 : 1 min. 1 1 : 10 min.

INDEX 66H	default 00H
Bit	Description VGA and IN monitor and signaling SMI. This selection based on INDEX 5A bit 7 is set.
7	IN monitor IRQ3 1: selected 0: not select
6	IN monitor IRQ4 1: selected 0: not select
5	IN monitor IRQ12 1: selected 0: not select
4~2	reserved.
1	VGA monitor I/O write 3B0H-3BFH 1: selected 0: not select
0	VGA monitor memory write A0000H-B0000H 1: selected 0: not select

INDEX 67H	default 00H
Bit	Description
7	SMI event (Read only)
6	EXTSW1 status (Read only)
5	reserved
4	DRQ access (Read only)
3	IRQ access (Read only)
2	EXTSW2 status (Read only)
1-0	EXTSW1/EXTSW2 input polarity setting 00 : disable 01 : rising edge trigger 10 : falling edge trigger 11 : edge trigger

INDEX 68H	default 00H
Bit	Description
7~0	Power ON latched Power Control Initial status from XD[7-0] D[7-0] : PWR[7-0] control pin status

INDEX 69H	default 00H
Bit	Description
7	reserved, must be 0.
6	Read the timer counter 0 : counter setting 1 : current value
5	reserved, must be 0
4	Double counter time base 0 : disable 1 : enable
3~0	reserved.

Ver. 0.93, Document Number: 6117DSC1.doc
Acer Laboratories, Inc., USA, 1830B Bering Drive, San Jose, CA 95112; Tel: 408-467-7456; Fax: 408-467-7474
www.acerlabs.com

INDEX 6AH	default 00H
Bit	Description
7	SUSPEND time-out Mode timer time-out from mode STANDBY to SUSPEND
6	STANDBY time-out Mode timer time-out from mode DOZE to STANDBY
5	DOZE time-out Mode timer time-out from mode ON to DOZE
4~0	reserved, fixed on 0 (Read only)

INDEX 6BH	default 00H
Bit	Description
7	reserved
6	GP1 I/O address 300H-3FFH 0 : disable 1 : enable
5	GP1 I/O address 200H-2FFH 0 : disable 1 : enable
4	GP1 I/O address 100H-1FFH 0 : disable 1 : enable
3~2	Define GP1 I/O address mask A[3-2] respectively. 0 : do not compare this address bit. 1 : compare this address bit.
1	GP0 memory address 0 : disable 1 : enable
0	GP1 I/O address 0 : disable 1 : enable

INDEX 6CH	default 00H
Bit	Description
7~0	Define GP0 memory address A[23-16]

INDEX 6DH	default 00H
Bit	Description
7~0	Define GP0 memory address A[23-16] mask bits respectively. 1 : compare this address bit. 0 : do not compare this address bit.

INDEX 6EH	default 00H
Bit	Description
7~2	reserved, must be 000000b
1~0	Define GP0 memory address A[25-24].

INDEX 6FH	default 00H
Bit	Description
7~2	reserved, must be 000000b
1~0	Define GP0 memory address A[25-24] mask bits respectively. 1 : compare this bit 0 : do not compare this bit

INDEX 70H	default 00H
Bit	Description
7~0	Define GP1 I/O port address SA[9-2]

INDEX 71H	default 00H
Bit	Description
7~5	reserved
4~2	Define Mode LED on/off period 0 0 0 : 0.1 sec 0 0 1 : 0.2 sec 0 1 0 : 0.4 sec 0 1 1 : 0.8 sec 1 0 0 : 1.0 sec 1 0 1 : 2.0 sec 1 1 0 : 4.0 sec 1 1 1 : 8.0 sec
1~0	Define Mode LED duty cycle 0 0 : disable 0 1 : 25% 1 0 : 50% 1 1 : 75%

INDEX 72H	default 00H
Bit	Description
7~0	Shadow I/O port for port 70H data

INDEX 73H	default 00H
Bit	Description
7~0	Power Control status output command Write to this port will generate enpower pulse to update power control status.

Section 5 : Programming Guide

5.1 Basic Procedure and Macro Definition

a) Delay

```
IO_Delay MACRO
        jcxz  $+2
        jcxz  $+2
        ENDM
```

b) Unlock chipset• configure registers

```
Open_Chip MACRO
        mov al, 013h
        out  022h, al
        IO_Delay
        mov al, 0c5h
        out  023h, al
        IO_Delay
        ENDM
```

c) Lock chipset• configure registers

```
Close_Chip MACRO
        mov al, 013h
        out  022h, al
        IO_Delay
        mov al, 000h
        out  023h, al
        IO_Delay
        ENDM
```

d) Write data to configure register

```
; INPUT :   AH     -     INDEX#
; INPUT :   AL     -     Data
; ACTION :  Write the value of AL into the value of AH INDEX
; Interrupt controller and Stack are available
Write_To_Chip PROCEDURE
        cli
        push ax
        Open_Chip
        pop  ax
        out  022h, al
        IO_Delay
        xchg ah, al
        out  023h, al
        IO_Delay
        xchg ah, al
        push ax
        Close_Chip
        pop  ax
        sti
        ret
        ENDP
```

e) Read data from configure register

```
; INPUT :   AL      -       INDEX#
; OUTPUT :  AL      -       Data
; ACTION :  Read data from the value of AL INDEX
; Interrupt controller and Stack are available
Read_From_Chip PROC
        cli
        push ax
        Open_Chip
        pop  ax
        out  022h, al
        IO_Delay
        in   al, 023h
        IO_Delay
        push ax
        Close_Chip
        pop  ax
        sti
        ret
        ENDP
```

5.2 Detection and Setting of Fast Page Mode and EDO DRAMs

```
; BD0 must be pulled high 10K
; Stack must be available
; all routines in this section must be executed at ROM access

DRAM_Type_Detection:
        mov  ax, 01010h             ; Set mode 5 for DRAM detection
        call Read_From_Chip
        and  al, 00000111b
        or   al, 01010000b
        xchg ah, al
        call Write_To_Chip

        mov  ax, 03c3ch             ; Set the bit 1 of INDEX 3Ch to '1'
        call Read_From_Chip                 ; to enable EDO DRAM timing detect
        or   al, 00000010b          ; mode.
        xchg ah, al
        call Write_To_Chip

        push ds
        mov  ax, 0h
        mov  ds, ax
        mov  ds: word ptr[0h], 0aaaah
        mov  ds: word ptr[1000h], 05555h    ; dummy write
        cmp  ds: word ptr[0h], 0aaaah
        je   Is_EDO_type_DRAM
```


Acer Laboratories, Inc., USA, 1830B Bering Drive, San Jose, CA 95112; Tel: 408-467-7456; Fax: 408-467-7474
www.acerlabs.com

```
Is_Fast_Page_Mode_Type_DRAM :
        mov  ax, 03c3ch
        call Read_From_Chip
        and  al, 11111101b          ; Disable EDO DRAM timing detect
        xchg ah, al                 ; mode
        call Write_To_Chip
        jmp  Finish_DRAM_Type_Detection

Is_EDO_Type_DRAM:
        mov  ax, 03c3ch
        call Read_From_Chip
        and  al, 11111101b          ; Disable EDO DRAM timing detect
        xchg ah, al                 ; mode
        call Write_To_Chip

        mov  ax, 03737h
        call Read_From_Chip
        or   al, 00000001b          ; Set EDO DRAM mode
        xchg ah, al
        call Write_To_Chip

Finish_DRAM_Type_Detection:
        pop  ds
        ret
```

5.3 Memory Auto Sizing

```
big_gdt_descriptor label fword
        dw      big_gdt_end - big_gdt - 1  ; limit of gdt
        dw      offset big_gdt
        db      0fh
        db      93h
        dw      0000h

big_gdt label qword
        dq      0
        dq      008f93000000ffffh
        dq      000093000000ffffh
big_gdt_end equ $

mem_config_table label word
        ;       memory mode                     total
        ;       type            pattern         memory          Type
        dw      0011h,          0000h,          0001h,          ; 0
        dw      1111h,          0010h,          0002h,          ; 1
        dw      0311h,          0020h,          0003h,          ; 2
        dw      3311h,          0030h,          0005h,          ; 3
        dw      0511h,          0040h,          0009h,          ; 4
        dw      0002h,          0050h,          0001h,          ; 5
        dw      0022h,          0060h,          0002h,          ; 6
        dw      0322h,          0070h,          0004h,          ; 7
        dw      3322h,          0080h,          0006h,          ; 8
        dw      0522h,          0090h,          000ah,          ; 9
        dw      5522h,          00a0h,          0012h,          ; 10
        dw      0032h,          00b0h,          0003h,          ; 11
        dw      0332h,          00c0h,          0005h,          ; 12
        dw      0052h,          00d0h,          0009h,          ; 13
        dw      0003h,          00e0h,          0002h,          ; 14
        dw      0033h,          00f0h,          0004h,          ; 15
        dw      0333h,          0008h,          0006h,          ; 16
        dw      3333h,          0018h,          0008h,          ; 17
        dw      0533h,          0028h,          000ch,          ; 18
        dw      5533h,          0038h,          0014h,          ; 19
        dw      0053h,          0048h,          000ah,          ; 20
        dw      0553h,          0058h,          0012h,          ; 21
        dw      5553h,          0068h,          001ah,          ; 22
        dw      0004h,          0078h,          0004h,          ; 23
        dw      0044h,          0088h,          0008h,          ; 24
        dw      5544h,          0098h,          0018h,          ; 25
        dw      0054h,          00a8h,          000ch,          ; 26
        dw      0005h,          00b8h,          0008h,          ; 27
        dw      0055h,          00c8h,          0010h,          ; 28
        dw      0555h,          00d8h,          0018h,          ; 29
        dw      5555h,          00e8h,          0020h,          ; 30
        dw      0066h,          00f8h,          0040h,          ; 31

; Stack must be available
; all routines in this section must be executed at ROM access
```


Acer Laboratories, Inc., USA, 1830B Bering Drive, San Jose, CA 95112; Tel: 408-467-7456; Fax: 408-467-7474
www.acerlabs.com

```
Auto_DRAM_sizing :

        call   DRAM_Type_Detection          ; Detect and set the EDO DRAM

        ; enable 8042 address line 20 here

        .386p
        mov  eax, cr0                       ; Set PE
        or   al, 01h                        ; Enter protected mode
        mov  cr0, eax

        cli

        lgdt   cs: big_gdt_descriptor
        jmp    short enter_protected_mode

enter_protected_mode:
        mov  ax, 0008h
        mov  ds, ax

        mov  ax, 01010h                     ; Set mode 30 for DRAM sizing
        call  Read_From_Chip
        and  al, 00000111b
        or   al, 11101000b
        xchg ah, al
        call  Write_To_Chip

        xor   dx, dx                        ; store DRAM mode
        mov   esi, 3000h                    ; A13, A12 enable - bank 3

sizing_bank_23:
        mov  edi, 800h                      ; A11
        and  dl, 0f0h
        or   dl, 5                          ; 5 --- 4M
        mov  ds: word ptr[esi+edi], 0aa99h
        mov  ds: word ptr[esi], 099aah      ; dummy write
        cmp  ds: word ptr[esi+edi], 0aa99h
        jz   sizing_bank_23_end             ; 4M, go to test next bank

        mov  edi, 400h                      ; A10
        and  dl, 0f0h
        or   dl, 3                          ; 3 --- 1M
        mov  ds: word ptr[esi+edi], 0bb88h
        mov  ds: word ptr[esi], 088bbh      ; dummy write
        cmp  ds: word ptr[esi+edi], 0bb88h
        jz   sizing_bank_23_end             ; 1M, go to test next bank

        mov  edi, 2h                        ; A1
        and  dl, 0f0h
        or   dl, 1                          ; 1 --- 256K
        mov  ds: word ptr[esi+edi], 0cc77h
        mov  ds: word ptr[esi], 077cch      ; dummy write
        cmp  ds: word ptr[esi+edi], 0cc77h
        jz   sizing_bank_23_end             ; 256K, go to test next bank

        and  dl, 0f0h                       ; none in this bank
```

```
sizing_bank_23_end:
        cmp  esi, 2000h
        jz   short sizing_bank_01_begin     ; finish sizing bank 3 and 2
        sub  esi, 1000h
        shl  dx, 4
        jmp  sizing_bank_23                 ; go to sizing bank 2

sizing_bank_01_begin:
        mov  ax, 01010h                     ; Set mode 31 for DRAM sizing
        call Read_From_Chip
        and  al, 00000111b
        or   al, 11111000b
        xchg ah, al
        call Write_To_Chip

sizing_bank_01:
        shl  dx, 4                          ; next bank

        mov  edi, 1000h                     ; A12
        and  dl, 0f0h
        or   dl, 6                          ; 6 --- 16M
        mov  ds: word ptr[esi+edi], 0dd66h
        mov  ds: word ptr[esi], 066ddh      ; dummy write
        cmp  ds: word ptr[esi+edi], 0dd66h
        jz   sizing_bank_01_end             ; 16M, go to test next bank

        mov  edi, 800h                      ; A11
        and  dl, 0f0h
        or   dl, 5                          ; 5 --- 4M
        mov  ds: word ptr[esi+edi], 0ee55h
        mov  ds: word ptr[esi], 055eeh      ; dummy write
        cmp  ds: word ptr[esi+edi], 0ee55h
        jz   sizing_bank_01_end             ; 4M, go to test next bank

        mov  edi, 400h                      ; A10
        and  dl, 0f0h
        or   dl, 3                          ; 3 --- 1M
        mov  ds: word ptr[esi+edi], 0ff44h
        mov  ds: word ptr[esi], 044ffh      ; dummy write
        cmp  ds: word ptr[esi+edi], 0ff44h
        jz   short is_1or2M                 ; 1 or 2M, go to is_1or2M to check

        mov  edi, 2                         ; A1
        and  dl, 0f0h
        or   dl, 1                          ; 1 --- 256K
        mov  ds: word ptr[esi+edi], 012abh
        mov  ds: word ptr[esi], 0ab12h      ; dummy write
        cmp  ds: word ptr[esi+edi], 012abh
        jz   short is_2or5K                 ; 256 or 512K, go to is_2or5K to check

        and  dl, 0f0h                       ; none in this bank
        jmp  short sizing_bank_01_end
```


Acer Laboratories, Inc., USA, 1830B Bering Drive, San Jose, CA 95112; Tel: 408-467-7456; Fax: 408-467-7474
www.acerlabs.com

```
is_1or2M:
        mov  edi, 1000000                ; A24
        mov  ds: word ptr[esi+edi], 034cdh
        mov  ds: word ptr[esi], 0ed34h   ; dummy write
        cmp  ds: word ptr[esi+edi], 034cdh
        jnz  short sizing_bank_01_end    ; 1M
        and  dl, 0f0h
        or   dl, 4                       ; 2M
        jmp  short sizing_bank_01_end

is_2or5K:
        mov  edi, 100000                 ; A20
        mov  ds: word ptr[esi+edi], 056efh
        mov  ds: word ptr[esi], 0ef56h   ; dummy write
        cmp  ds: word ptr[esi+edi], 056efh
        jnz  short sizing_bank_01_end    ; 256K
        and  dl, 0f0h
        or   dl, 2                       ; 512K

sizing_bank_01_end:
        cmp  esi, 0
        jz   short sizing_memory_end
        mov  esi, 0                      ; bank 0
        jmp  sizing_bank_01

sizing_memory_end:
        mov  ax, 010h                    ; reset descriptor
        mov  ds, ax

        mov  eax, cr0                    ; reset PE
        and  al, 0feh
        mov  cr0, eax
        jmp  short enter_real_mode

enter_real_mode:
        .286p

        ; disable 8042 address line 20 here

        mov  si, offset cgroup: mem_config_table
        mov  cx, 32

check_mode :
        mov  ax, cs: word ptr[si]
        cmp  dx, ax
        jz   short check_mode_end
        add  si, 6
        loop check_mode

        ; Display memory error here
        ;
        ; if  dx = 0001, set mode 5 for 512K DRAM
        ;
```

```
check_mode_end:
        mov  bx, cs: word ptr[si+2]

        mov  ax, 01010h                 ; Set DRAM Mode
        call Read_From_Chip
        and  al, 00000111b
        or   al, bl
        xchg ah, al
        call Write_To_Chip

        ;
        ; Do your memory remapping setting here.
        ;
        ; Use the total DRAM size to check whether the remapping is
        ; allowable or not.
        ; Use the total DRAM size to set the bit 7-4 of INDEX 12h Split address.
        ; Select the remap mode by setting the bit 1 of INDEX 11h.
```

5.4 Remapping Memory

If enable memory remap(INDEX 11h, bit 2); we have two choices: split or move-out. We can use the split mode when the size of the DRAM is less than 16M owing to the limitation of hardware. In this way, the available memory is increased. When the memory size is less than 16M, we can also choose the move-out mode. Therefore, we should get the DRAM size by the Auto_DRAM_Sizing routine. The details of setting the configure registers are described as follows :

```
; Get the total memory size to bl
;
; 1). Get the total memory size from Auto_DRAM_Sizing

        mov  bl, cs: byte ptr[si+4]

; or 2). Using the following routine to get the total memory size from reading DRAM mode

        mov  al, 010h
        call Read_From_Chip
        and  al, 11111000b
        xor  dx, dx
        or   dl, al
        mov  si, offset cs: mem_config_table
        mov  cx, 32
check_mem_mode:
        mov  ax, cs: word ptr[si+2]
        cmp  dx, ax
        jz   short check_mem_mode_end
        add  si, 6
        loop check_mem_mode
check_mem_mode_end:
        mov  bl, cs: byte ptr[si+4]

        cmp  bl, 16
        ja   do_remap_memory

; if bl >= 16 (the memory size is equal to or greater than 16M ), disable memory remapping.
        mov  ax, 01111h                 ; Disable memory remapping
        call Read_From_Chip
        and  al, 11111011b
        xchg ah, al
        call Write_To_Chip

; Otherwise, doing memory remap in the following routine
; 1) We would like to choose move-out mode
```


Acer Laboratories, Inc., USA, 1830B Bering Drive, San Jose, CA 95112; Tel: 408-467-7456; Fax: 408-467-7474
www.acerlabs.com

```
        mov  ax, 01111h                 ; Enable memory remapping
        call Read_From_Chip                     ; Select move-out mode
        and  al, 11111001b
        or   al, 00000110b
        xchg ah, al
        call Write_To_Chip

        mov  ax, 01212h                 ; Set Split address SP[23-20]
        call Read_From_Chip                     ; to [1111]
        and  al, 00001111b
        or   al, 11110000b
        xchg ah, al
        call Write_To_Chip

; or 2) We would like to choose split mode
        mov  ax, 01111h                 ; Enable memory remapping
        call Read_From_Chip                     ; Select split mode
        and  al, 11111001b
        or   al, 00000100b
        xchg ah, al
        call Write_To_Chip

        mov  ax, 01212h                 ; Set Split address SP[23-20]
        call Read_From_Chip                     ; to [1111]
        and  al, 00001111b
        shl  bl, 4                      ; bl - total memory size
        or   al, bl
        xchg ah, al
        call Write_To_Chip
```

5.5 Interrupt controller edge/level trigger programming

The default setting of M6117C interrupt controller is edge trigger disregarding the value of ICW1. M6117C has the ability of setting each IRQ to be edge or level trigger by decoding I/O ports 04d0h and 04d1h.

To enable this feature, set the bit 2 of INDEX 3Ch to '1'.

```
        mov  ax, 03c3ch                 ; Enable IRQ level trigger selection
        call Read_From_Chip
        or   al, 00000100b
        xchg ah, al
        call Write_To_Chip
```

Then the I/O port 04d0h and 04d1h can set the corresponding IRQ to be level trigger.

```
I/O port 04d0h   -      Interrupt controller 1
        bit 7  -  IRQ7
        bit 6  -  IRQ6
          :
        bit 1  -  IRQ1
        bit 0  -  IRQ0
I/O port 04d1h   -      Interrupt controller 2
        bit 7  -  IRQ15
        bit 6  -  IRQ14
          :
        bit 1  -  IRQ9
        bit 0  -  IRQ8
```

To enable IRQ10 to be level trigger, use

```
        mov  dx, 04d1h
        in   al, dx
        IO_Delay
        or   al, 00000100          ; bit 2 - IRQ10
        out  dx, al
        IO_Delay
```

5.6 PMU Programming Guide

5.6.1 Power Management Mode Selection

If any event happens, M6117C will check the setting of INDEX 55h and INDEX 38h to generate a signal.

1) SMI support :

```
        mov  ax, 05555h
        call Read_From_Chip
        and  al, 11111100b         ; SMI support
        xchg ah, al
        call Write_To_Chip
```

2) NMI support :

```
        mov  ax, 05555h
        call Read_From_Chip
        and  al, 11111100b
        or   al, 00000001b         ; NMI support
        xchg ah, al
        call Write_To_Chip
```

3) IRQ15 support:

```
        mov  ax, 05555h
        call Read_From_Chip
        and  al, 11111100b
        or   al, 00000010b         ; IRQ15 support
        xchg ah, al
        call Write_To_Chip
```


Acer Laboratories, Inc., USA, 1830B Bering Drive, San Jose, CA 95112; Tel: 408-467-7456; Fax: 408-467-7474
www.acerlabs.com

4) Other IRQs support: IRQ3, IRQ4, IRQ5, IRQ6, IRQ7, IRQ9, IRQ10, IRQ11, IRQ12, IRQ14.

```
mov  ax, 05555h
call  Read_From_Chip
or    al, 00000011b                    ; IRQ3, IRQ4,•RQ14 support
xchg  ah, al
call  Write_To_Chip
```

Then set the bit 3-0 of INDEX 38h to IRQ number.

```
mov  ax, 03838h
call  Read_From_Chip
and   al, 11110000b
or    al, 00000011b                    ; IRQ3 support
(or   al, 00000100b                    ; IRQ4 support
(or   al, 00001110b                    ; IRQ14 support
xchg  ah, al
call  Write_To_Chip
```

5.6.2 Structure of Power Management routine

```
mov  ax, 05656h
call  Read_From_Chip
and   al, 01111111b                    ; disable SMI signal
xchg  ah, al
call  Write_To_Chip

; read event from INDEX 5bh
mov  al, 05bh
call  Read_From_Chip
; check the bit4-0 of INDEX 5bh
; disable PMU state
mov  ax, 05656h
call  Read_From_Chip
and   al, 11111011b                    ; disable PUM state
xchg  ah, al
call  Write_To_Chip

; disable original event
; by clearing INDEX 58h, 59h, 5ah
;
; do your power management routine here
;

; set the condition of your next power management event
; by setting INDEX 57h, 58h, 59h, 5ah, 5ch, 5dh, 5eh, 64h, 66h, 67h, 69h, 6bh, 6ch, 6dh,
; 6eh, 6f, 70h
```

5.6.3 Power management example

In the following example, use
LPOETW(INDEX#, DATA) to write chipset• configure register and
LPORTR(INDEX#) to read chipset• configure register

Example 1 : Mode Translation

```
                        /* The DOZE, STANDBY, SUSPEND modes are defined by the BIOS */
LPORTW(056h,084h);      /* Enable SMI, PMU and set the system state at ON mode */
LPORTW(057h,008h);      /* Monitor IRQs in this example */
LPORTW(059h,020h);      /* Enable MODE timer */
LPORTW(064h,067h);      /* Set the time base of the MODE timer as 60 min */
                        /* SMIJ will be generated if no IRQ is active during 60 min */
                        /* Wait for the assertion of SMIACKJ */
LPORTW(056h,004h);      /* Deassert SMIJ */
LPORTR(05Bh);           /* To read the SMI cause */
                        /* It should be the MODE timer time-out in this example */
LPORTR(06Ah);           /* Read the time-out status */
                        /* Start SMI routine */
LPORTW(059h,000h);      /* Clear MODE time-out event */
LPORTW(05Ah,080h);      /* Set IN(standard input) as a wake-up event */
LPORTW(056h,084h);      /* Enable SMI again */
DB 0D6h ;               /* End SMI routine */
DB 0E6h ;               /* RETPM */
```


Acer Laboratories, Inc., USA, 1830B Bering Drive, San Jose, CA 95112; Tel: 408-467-7456; Fax: 408-467-7474
www.acerlabs.com

Example 2 : External Switch

```
LPORTW(056h,084h);      /* Enable SMI and PMU function */
LPORTW(067h,003h);      /* Set external switch both low-to-high and high-to-low active */
LPORTW(058h,000h);      /* Clear external switch */
LPORTW(058h,040h);      /* Enable external switch */
                        /* SMI is generated when external switch is pushed */
                        /* Wait for the assertion of SMIACKJ */
LPORTR(05Bh);           /* To find out the SMI event is caused by which time-out event */
                        /* It should be the external switch active in this example */
LPORTR(067h);           /* Check the external switch status */
                        /* Start SMI routine */
LPORTW(058h,000h);      /* Clear external switch */
LPORTW(058h,040h);      /* Enable external switch */
LPORTW(05Ah,080h);      /* Set IN(standard input) as a wake-up event */
LPORTW(056h,084h);      /* Enable SMI again */
DB 0D6h                 /* End SMI routine */
DB 0E6h                 /* RETPM */
```

Example 3: Usage of the IN Group

IN group is used to monitor the activity of the standard input devices.

IN group is defined as:

IRQ1: default for keyboard
IRQ12: optional for PS/2 mouse (index 66_D5)
IRQ4: optional for COM1 mouse (index 66_D6)
IRQ3: optional for COM2 mouse (index 66_D7)

IN group timer time-out:

(1) Generate power control signal to turn off the screen
(2) Enter SMM by asserting SMIJ

IN group access:

(1) Generate power control signal to turn on the screen
(2) Enter SMM by asserting SMIJ

Hence, monitoring the IN group activity can be used to implement the function of the "screen saver". Besides, it will not impact the performance of the running program, instead the whole power can be reduced dramatically.

Example 4 : Software SMI Event

```
LPORTW(0x56,0xC4);      /* Enable SMI and PMU function, and enable software SMI */
                        /* SMIJ is asserted */
                        /* Wait for the assertion of SMIADSJ */
LPORTR(0x5B);           /* Read SMI cause, it should be software SMI */
LPORTW(0x56,0x04);      /* Deassert SMIJ */
                        /* Start SMI routine */
LPORTW(0x56,0x84);      /* Enable SMI again */
DB 0D6h;                /* End SMI routine */
DB 0E6h;                /* RETPM */
```

Example 5 : SMM Remap Start Address

```
LPORTW(0x20,0xB0);          /* Remap start address to F region, no shadow */
MOV eax, 000F0000h          /* start address = 000F0000 */
DB 0D6h,0CAh,03h,0A0h       /* Load EA0 to ultra SR register*/
DB 0D6h,0C8h,03h,0A0h       /* Load ultra SR register to EA0 */
DB 0F1h                         /* BRKPM, instruction to enter SMM mode */

LPORTW(0x20,0x90);          /* Remap start address to AB region, shadow */
mov eax, 000A0000h          /* start address = 000A0000 */
DB 0D6h,0CAh,03h,0A0h       /* Load EA0 to ultra SR register*/
DB 0D6h,0C8h,03h,0A0h       /* Load ultra SR register to EA0 */
DB 0F1h                         /* BRKPM, instruction to enter SMM mode */

LPORTW(0x20,0xA0);          /* Remap start address to E region, shadow */
mov eax, 000E0000h          /* start address = 000E0000 */
DB 0D6h,0CAh,03h,0A0h       /* Load EA0 to ultra SR register*/
DB 0D6h,0C8h,03h,0A0h       /* Load ultra SR register to EA0 */
DB 0F1h                         /* BRKPM, instruction to enter SMM mode */

LPORTW(0x20,0xB8);          /* Remap start address to F region, shadow */
mov eax, FFFF0000h          /* start address = FFFF0000 */
DB 0D6h,0CAh,03h,0A0h       /* Load EA0 to ultra SR register*/
DB 0D6h,0C8h,03h,0A0h       /* Load ultra SR register to EA0 */
DB 0F1h                         /* BRKPM, instruction to enter SMM mode */
```

Example 6 : Stop Internal CPU Clock

```
mov eax, 00080000h          /* Set  Power Control Register Clock Stop Bit = 1 */
DB 0D6h,0FAh,03h,02h        /* MOV PWRCR, EA0 , where PWRCR is Power control register*/
hlt                         /* Stop internal CPU clock */
                            /* Wake up by INTR, NMI */
```

5.7 Flowcharts

Note : The following notations below have the following meanings.

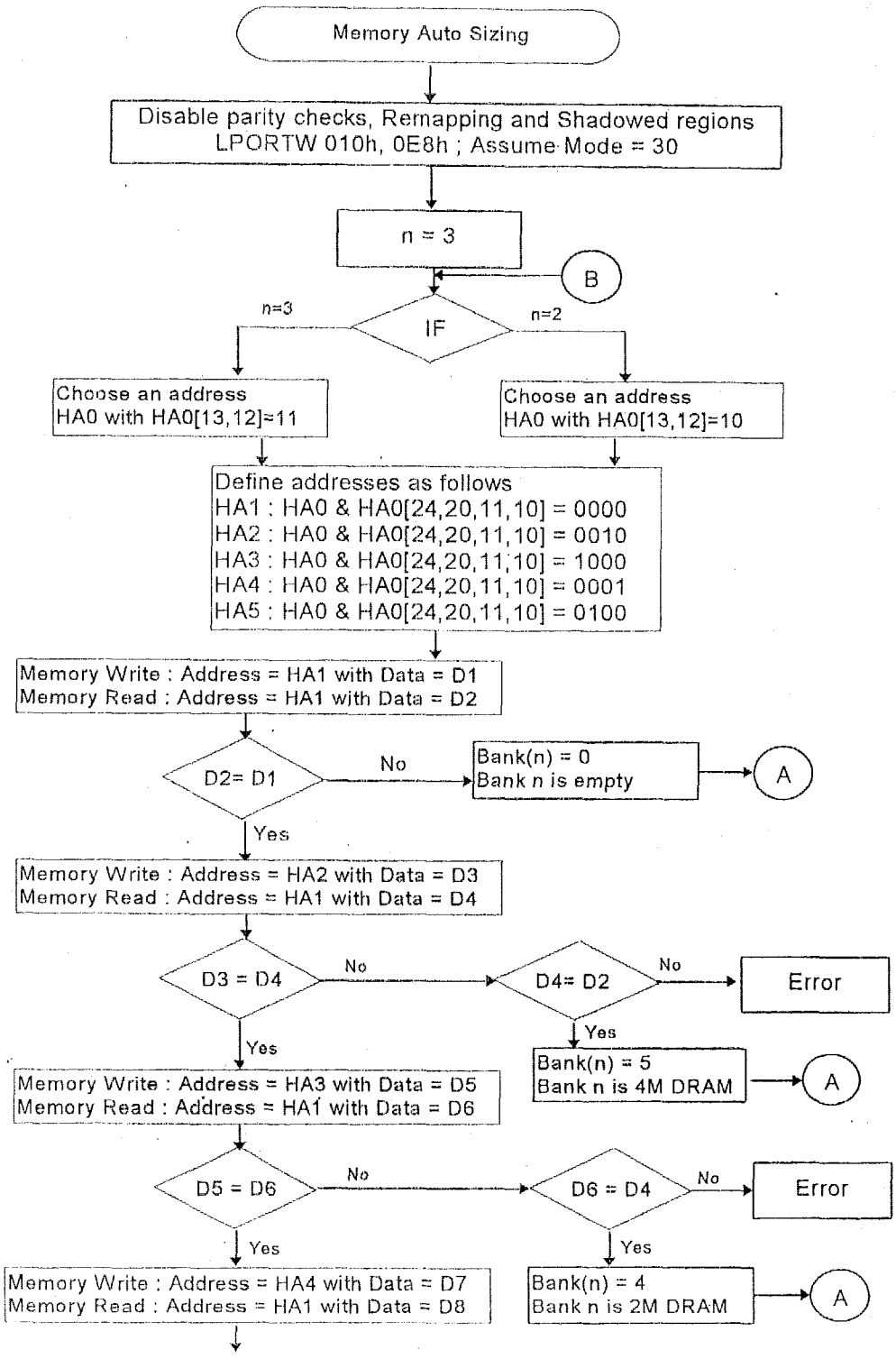
HAi (where I is an integer) denotes a specific CPU address.

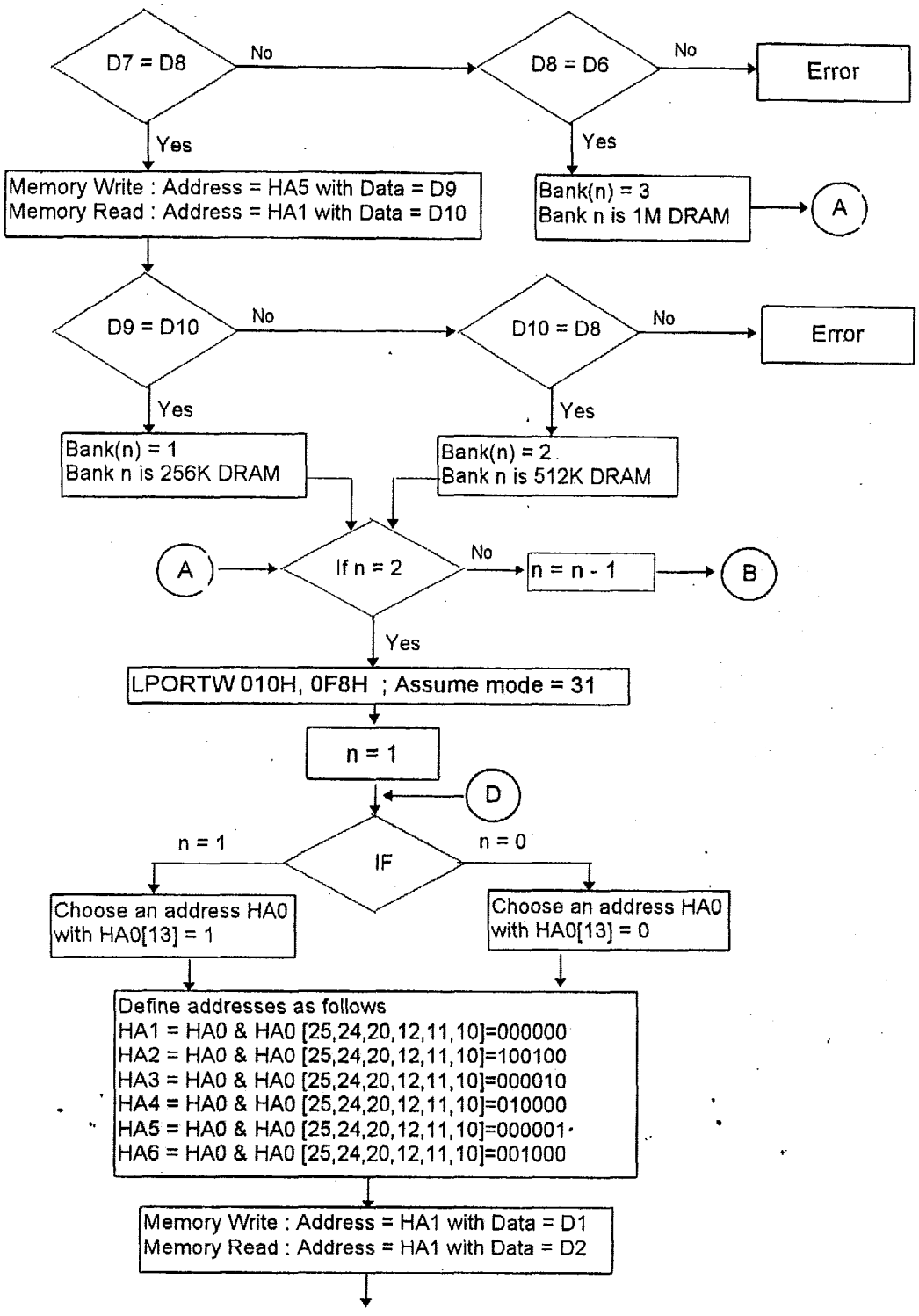
HAi[j] denotes the jth bit of the address lines HAi.

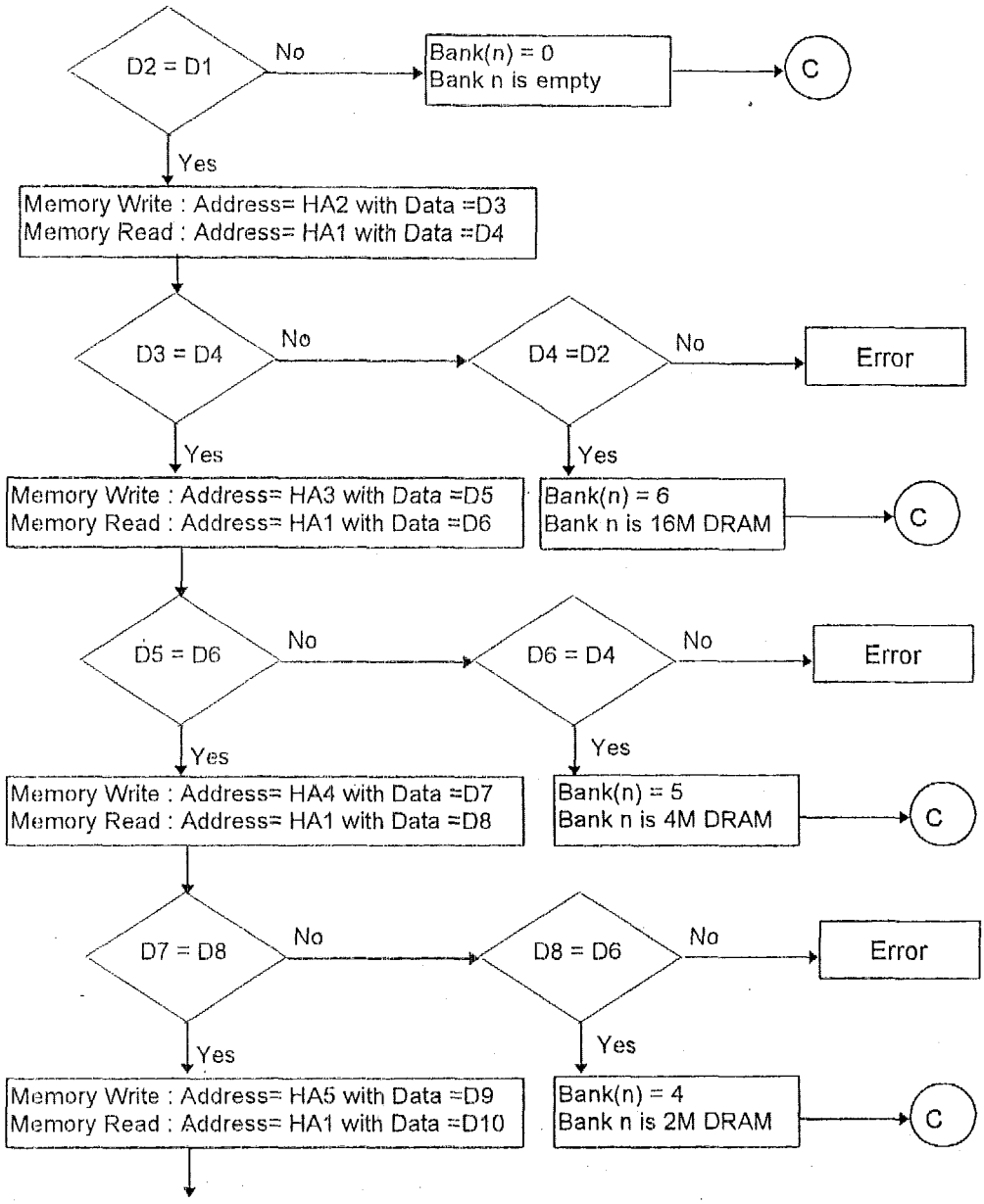
- Ai & HAi[j] = K denote the specific address Hai is with a binary value K assigned to the jth bit.

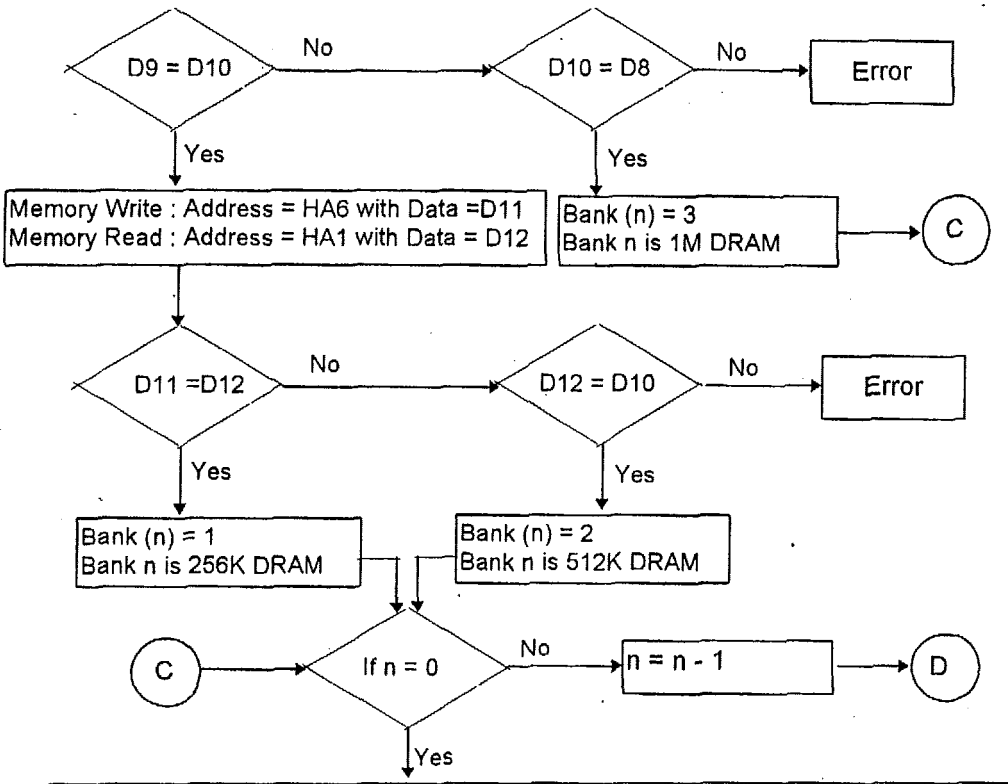
Acer Laboratories, Inc., USA, 1830B Bering Drive, San Jose, CA 95112; Tel: 408-467-7456; Fax: 408-467-7474
www.acerlabs.com

A. Flowchart of Detecting Memory Mode




D7 = D8
No
D8 = D6
No
Error
Yes
Yes
Memory Write : Address = HA5 with Data = D9
Memory Read : Address = HA1 with Data = D10
Bank(n) = 3
Bank n is 1M DRAM
A
D9 = D10
No
D10 = D8
No
Error
Yes
Yes
Bank(n) = 1
Bank n is 256K DRAM
Bank(n) = 2
Bank n is 512K DRAM
A
If n = 2
No
n = n - 1
B
Yes
LPORTW 010H, 0F8H ; Assume mode = 31
n = 1
D
IF
n = 1
n = 0
Choose an address HA0
with HA0[13] = 1
Choose an address HA0
with HA0[13] = 0
Define addresses as follows
HA1 = HA0 & HA0 [25,24,20,12,11,10]=000000
HA2 = HA0 & HA0 [25,24,20,12,11,10]=100100
HA3 = HA0 & HA0 [25,24,20,12,11,10]=000010
HA4 = HA0 & HA0 [25,24,20,12,11,10]=010000
HA5 = HA0 & HA0 [25,24,20,12,11,10]=000001
HA6 = HA0 & HA0 [25,24,20,12,11,10]=001000
Memory Write : Address = HA1 with Data = D1
Memory Read : Address = HA1 with Data = D2


D2 = D1
No
Bank(n) = 0
Bank n is empty
C
Yes
Memory Write : Address= HA2 with Data =D3
Memory Read : Address= HA1 with Data =D4
D3 = D4
No
D4 =D2
No
Error
Yes
Yes
Memory Write : Address= HA3 with Data =D5
Memory Read : Address= HA1 with Data =D6
Bank(n) = 6
Bank n is 16M DRAM
C
D5 = D6
No
D6 = D4
No
Error
Yes
Yes
Memory Write : Address= HA4 with Data =D7
Memory Read : Address= HA1 with Data =D8
Bank(n) = 5
Bank n is 4M DRAM
C
D7 = D8
No
D8 = D6
No
Error
Yes
Yes
Memory Write : Address= HA5 with Data =D9
Memory Read : Address= HA1 with Data =D10
Bank(n) = 4
Bank n is 2M DRAM
C

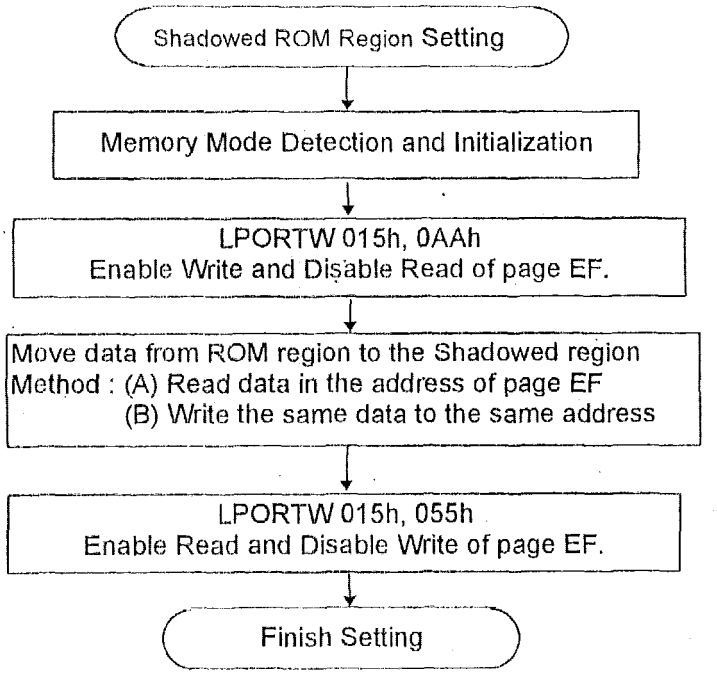


Case Bank (3-0)

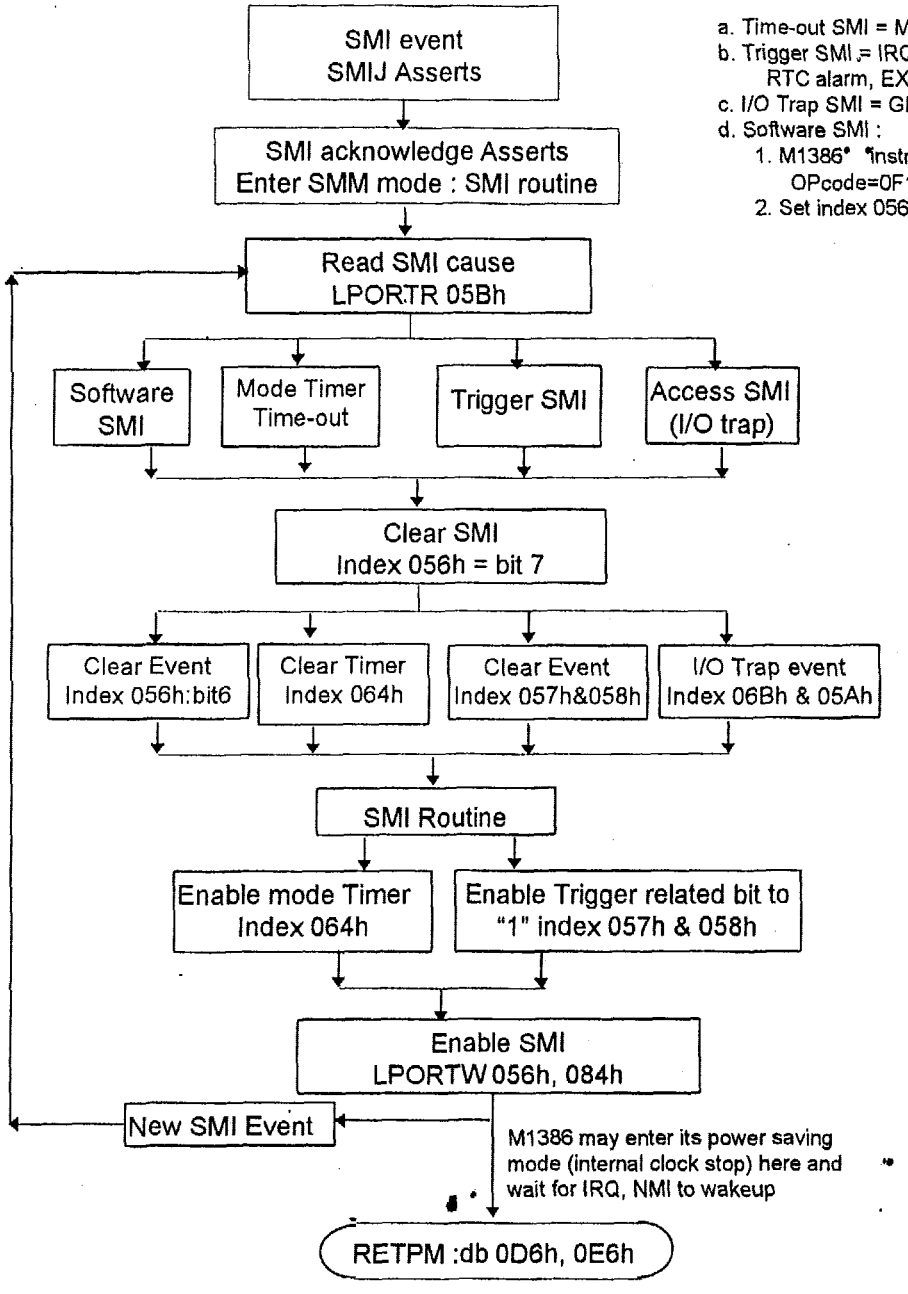
0,0,1,1 : Memory Mode = 0	3,3,3,3 : Memory Mode = 17
1,1,1,1 : Memory Mode = 1	0,5,3,3 : Memory Mode = 18
0,3,1,1 : Memory Mode = 2	5,5,3,3 : Memory Mode = 19
3,3,1,1 : Memory Mode = 3	0,0,5,3 : Memory Mode = 20
0,5,1,1 : Memory Mode = 4	0,5,5,3 : Memory Mode = 21
0,0,0,2 : Memory Mode = 5	5,5,5,3 : Memory Mode = 22
0,0,2,2 : Memory Mode = 6	0,0,0,4 : Memory Mode = 23
0,3,2,2 : Memory Mode = 7	0,0,4,4 : Memory Mode = 24
3,3,2,2 : Memory Mode = 8	5,5,4,4 : Memory Mode = 25
0,5,2,2 : Memory Mode = 9	0,0,5,4 : Memory Mode = 26
5,5,2,2 : Memory Mode = 10	0,0,0,5 : Memory Mode = 27
0,0,3,2 : Memory Mode = 11	0,0,5,5 : Memory Mode = 28
0,3,3,2 : Memory Mode = 12	0,5,5,5 : Memory Mode = 29
0,0,5,2 : Memory Mode = 13	5,5,5,5 : Memory Mode = 30
0,0,0,3 : Memory Mode = 14	0,0,6,6 : Memory Mode = 31
0,0,3,3 : Memory Mode = 15	Others : Type is undefined
0,3,3,3 : Memory Mode = 16	

Acer Laboratories, Inc., USA, 1830B Bering Drive, San Jose, CA 95112; Tel: 408-467-7456; Fax: 408-467-7474
www.acerlabs.com

B. Flowchart of Enabling the Shadowed Regions

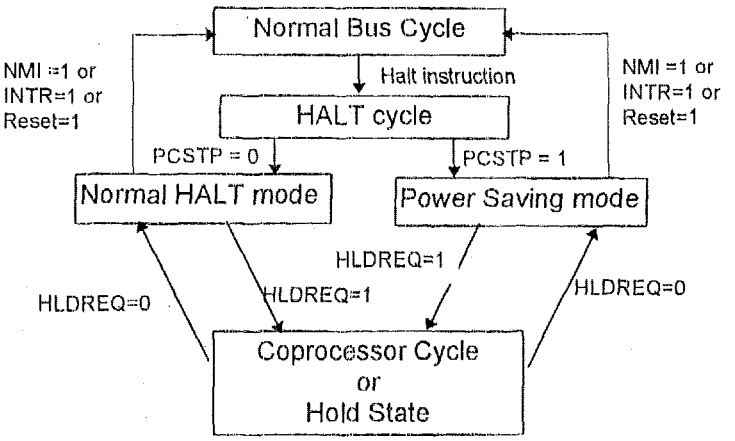


C. SMI Control Flowchart



Acer Laboratories, Inc., USA, 1830B Bering Drive, San Jose, CA 95112; Tel: 408-467-7456; Fax: 408-467-7474
www.acerlabs.com

4. M6117C Power Saving Mode



Question : How to set PCSTP ? (Power Clock Stop)

Answer : MOV EAX, 00080000h
DB 0D6h, 0FAh, 03h, 02h
/* MOV PWRCR, EA0 */

MA Table

DRAM mode	DRAM type	Bank	DRAM addr	MA0 MA1 MA2	MA3 MA4 MA5	MA6 MA7 MA8	MA9 MA10 MA11	Enable Bank
0	256K	0,1	row column	A11 A12 A13 A2 A3 A4	A14 A15 A16 A5 A6 A7	PA17 PA18 PA19 A8 A9 A1		A10
1	256K	0-3	row column	PA20 A12 A13 A2 A3 A4	A14 A15 A16 A5 A6 A7	PA17 PA18 PA19 A8 A9 A1		A11, A10
2	256K	0,1	row column	A11 A12 A13 A2 A3 A4	A14 A15 A16 A5 A6 A7	PA17 A18 PA19 A8 A9 A1		A10
	1M	2	row column	A11 A12 A13 A2 A3 A4	A14 A15 A16 A5 A6 A7	PA17 PA18 PA19 A8 A9 A1	PA20 ___ ___ A10 ___ ___	
3	256K	0,1	row column	A11 A12 A13 A2 A3 A4	A14 A15 A16 A5 A6 A7	PA17 PA18 PA19 A8 A9 A1		A10
	1M	2,3	row column	A21 A12 A13 A2 A3 A4	A14 A15 A16 A5 A6 A7	PA17 PA18 PA19 A8 A9 A1	PA20 ___ ___ A10 ___ ___	A11
4	256K	0,1	row column	A11 A12 A13 A2 A3 A4	A14 A15 A16 A5 A6 A7	PA17 PA18 PA19 A8 A9 A1		A10
	4M	2	row column	PA21 A12 A13 A2 A3 A4	A14 A15 A16 A5 A6 A7	PA17 PA18 PA19 A8 A9 A1	PA20 PA22 ___ A10 A11 ___	
5	512K	0	row column	A11 A12 A13 A2 A3 A4	A14 A15 A16 A5 A6 A7	PA17 PA18 PA19 A8 A9 A1	PA19 ___ ___	
6	512K	0,1	row column	A11 A12 A13 A2 A3 A4	A14 A15 A16 A5 A6 A7	PA17 PA18 PA19 A8 A9 A1	PA20 ___ ___	A10
7	512K	0,1	row column	A11 A12 A13 A2 A3 A4	A14 A15 A16 A5 A6 A7	PA17 PA18 PA19 A8 A9 A1	PA20 ___ ___	A10
	1M	2	row column	A11 A12 A13 A2 A3 A4	A14 A15 A16 A5 A6 A7	PA17 PA18 PA19 A8 A9 A1	PA20 ___ ___ A10 ___ ___	
8	512K	0,1	row column	A11 A12 A13 A2 A3 A4	A14 A15 A16 A5 A6 A7	PA17 PA18 PA19 A8 A9 A1	PA20 _____	A10
	1M	2,3	row column	PA21 A12 A13 A2 A3 A4	A14 A15 A16 A5 A6 A7	PA17 PA18 PA19 A8 A9 A1	PA20 ___ ___ A10 ___ ___	A11
9	512K	0,1	row column	A11 A12 A13 A2 A3 A4	A14 A15 A16 A5 A6 A7	PA17 PA18 PA19 A8 A9 A1	PA20 __ ___	A10
	4M	2	row column	PA21 A12 A13 A2 A3 A4	A14 A15 A16 A5 A6 A7	PA17 PA18 PA19 A8 A9 A1	PA20 PA22 __ A10 A11 ___	
10	512K	0,1	row column	A11 A12 A13 A2 A3 A4	A14 A15 A16 A5 A6 A7	PA17 PA18 PA19 A8 A9 A1	PA20 __ __	A10
	4M	2,3	row column	PA21 PA22 A13 A2 A3 A4	A14 A15 A16 A5 A6 A7	PA17 PA18 PA19 A8 A9 A1	PA20 PA23 ___ A10 A11 ___	A12

Ver. 0.93, Document Number: 6117DSC1.doc
Acer Laboratories, Inc., USA, 1830B Bering Drive, San Jose, CA 95112; Tel: 408-467-7456; Fax: 408-467-7474
www.acerlabs.com

MA Table (continued)

DRAM mode	DRAM type	Bank	DRAM addr	MA0 MA1 MA2	MA3 MA4 MA5	MA6 MA7 MA8	MA9 MA10 MA11	Enable Bank
11	512K	0	row	A11 A12 A13	A14 A15 A16	PA17 PA18 A10	PA19	
			column	A2 A3 A4	A5 A6 A7	A8 A9 A1		
	1M	1	row	A11 A12 A13	A14 A15 A16	PA17 PA18 PA19	PA20	
			column	A2 A3 A4	A5 A6 A7	A8 A9 A1	A10	
12	512K	0	row	A11 A12 A13	A14 A15 A16	PA17 A18 A10	PA19	
			column	A2 A3 A4	A5 A6 A7	A8 A9 A1		
	1M	1,2	row	PA21 A12 A13	A14 A15 A16	PA17 PA18 PA19	PA20	A11
			column	A2 A3 A4	A5 A6 A7	A8 A9 A1	A10	
13	512K	0	row	A11 A12 A13	A14 A15 A16	PA17 PA18 A10	PA19	
			column	A2 A3 A4	A5 A6 A7	A8 A9 A1		
	4M	1	row	PA21 A12 A13	A14 A15 A16	PA17 PA18 PA19	PA20 PA22	
			column	A2 A3 A4	A5 A6 A7	A8 A9 A1	A10 A11	
14	1M	0	row	A11 A12 A13	A14 A15 A16	PA17 PA18 PA19	PA20	
			column	A2 A3 A4	A5 A6 A7	A8 A9 A1	A10	
15	1M	0,1	row	PA21 A12 A13	A14 A15 A16	PA17 PA18 PA19	PA20	A11
			column	A2 A3 A4	A5 A6 A7	A8 A9 A1	A10	
16	1M	0,1	row	PA21 A12 A13	A14 A15 A16	PA17 PA18 PA19	PA20	A11
			column	A2 A3 A4	A5 A6 A7	A8 A9 A1	A10	
	1M	2	row	A11 A12 A13	A14 A15 A16	PA17 PA18 PA19	PA20	
			column	A2 A3 A4	A5 A6 A7	A8 A9 A1	A10	
17	1M	0-3	row	PA21 PA22 A13	A14 A15 A16	PA17 PA18 PA19	PA20	A12,A11
			column	A2 A3 A4	A5 A6 A7	A8 A9 A1	A10	
18	1M	0,1	row	PA21 A12 A13	A14 A15 A16	PA17 PA18 PA19	PA20	A11
			column	A2 A3 A4	A5 A6 A7	A8 A9 A1	A10	
	4M	2	row	PA21 A12 A13	A14 A15 A16	PA17 PA18 PA19	PA20 PA22	
			column	A2 A3 A4	A5 A6 A7	A8 A9 A1	A10 A11	
19	1M	0,1	row	PA21 A12 A13	A14 A15 A16	PA17 PA18 PA19	PA20	A11
			column	A2 A3 A4	A5 A6 A7	A8 A9 A1	A10	
	4M	2,3	row	PA21 PA22 A13	A14 A15 A16	PA17 PA18 PA19	PA20 PA23	A12
			column	A2 A3 A4	A5 A6 A7	A8 A9 A1	A10 A11	
20	1M	0	row	A11 A12 A13	A14 A15 A16	PA17 PA18 PA19	PA20 A10	
			column	A2 A3 A4	A5 A6 A7	A8 A9 A1		
	4M	1	row	PA21 A12 A13	A14 A15 A16	PA17 PA18 PA19	PA20 PA22	
			column	A2 A3 A4	A5 A6 A7	A8 A9 A1	A10 A11	

MA Table (continued)

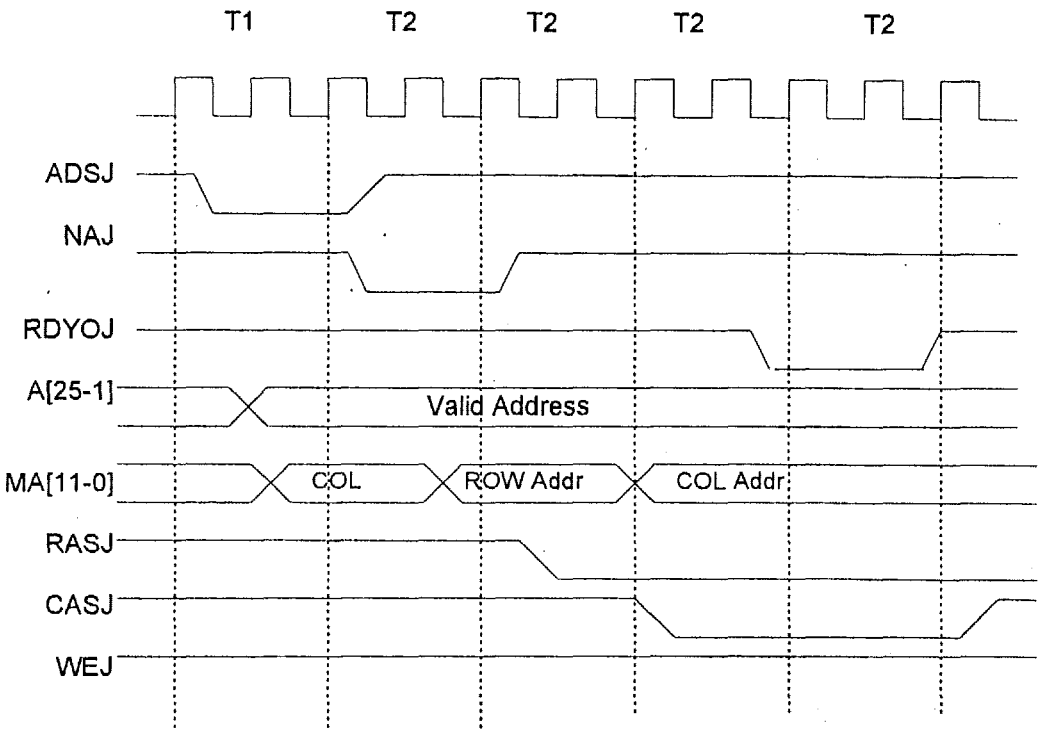
DRAM mode	DRAM type	Bank	DRAM addr	MA0 MA1 MA2	MA3 MA4 MA5	MA6 MA7 MA8	MA9 MA10 MA11	Enable Bank
21	1M	0	row	A11 A12 A13	A14 A15 A16	PA17 PA18 PA19	PA20	
			column	A2 A3 A4	A5 A6 A7	A8 A9 A1	A10	
	4M	1,2	row	PA21 PA22 A13	A14 A15 A16	PA17 PA18 PA19	PA20 PA23	A12
			column	A2 A3 A4	A5 A6 A7	A8 A9 A1	A10 A11	
22	1M	0	row	A11 A12 A13	A14 A15 A16	PA17 PA18 PA19	PA20	
			column	A2 A3 A4	A5 A6 A7	A8 A9 A1	A10	
	4M	1	row	PA21 A12 A13	A14 A15 A16	PA17 PA18 PA19	PA20 PA22	
			column	A2 A3 A4	A5 A6 A7	A8 A9 A1	A10 A11	
	4M	2,3	row	PA21 PA22 A13	A14 A15 A16	PA17 PA18 PA19	PA20 PA23	A12
			column	A2 A3 A4	A5 A6 A7	A8 A9 A1	A10 A11	
23	2M	0	row	A11 A12 A13	A14 A15 A16	PA17 PA18 PA19	PA20 PA21	
			column	A2 A3 A4	A5 A6 A7	A8 A9 A1	A10	
24	2M	0,1	row	PA21 A12 A13	A14 A15 A16	PA17 PA18 PA19	PA20 PA22	A11
			column	A2 A3 A4	A5 A6 A7	A8 A9 A1	A10	
25	2M	0,1	row	PA21 A12 A13	A14 A15 A16	PA17 PA18 PA19	PA20 PA22	A11
			column	A2 A3 A4	A5 A6 A7	A8 A9 A1	A10	
	4M	2,3	row	PA21 PA22 A13	A14 A15 A16	PA17 PA18 PA19	PA20 PA23	A12
			column	A2 A3 A4	A5 A6 A7	A8 A9 A1	A10 A11	
26	2M	0	row	A11 A12 A13	A14 A15 A16	PA17 PA18 PA19	PA20 PA21	
			column	A2 A3 A4	A5 A6 A7	A8 A9 A1	A10	
	4M	1	row	PA21 A12 A13	A14 A15 A16	PA17 PA18 PA19	PA20 PA22	
			column	A2 A3 A4	A5 A6 A7	A8 A9 A1	A10 A11	
27	4M	0	row	PA21 A12 A13	A14 A15 A16	PA17 PA18 PA19	PA20 PA22	
			column	A2 A3 A4	A5 A6 A7	A8 A9 A1	A10 A11	
28	4M	0,1	row	PA21 PA22 A13	A14 A15 A16	PA17 PA18 PA19	PA20 PA23	A12
			column	A2 A3 A4	A5 A6 A7	A8 A9 A1	A10 A11	
29	4M	0,1	row	PA21 PA22 A13	A14 A15 A16	PA17 PA18 PA19	PA20 PA23	A12
			column	A2 A3 A4	A5 A6 A7	A8 A9 A1	A10 A11	
	4M	2	row	PA21 A12 A13	A14 A15 A16	PA17 PA18 PA19	PA20 PA22	
			column	A2 A3 A4	A5 A6 A7	A8 A9 A1	A10 A11	
30	4M	0-3	row	PA21 PA22 PA23	A14 A15 A16	PA17 PA18 PA19	PA20 PA24	A13, A12
			column	A2 A3 A4	A5 A6 A7	A8 A9 A1	A10 A11	
31	16M	0,1	row	PA21 PA22 PA23	A14 A15 A16	PA17 PA18 PA19	PA20 PA24 PA25	A13
			column	A2 A3 A4	A5 A6 A7	A8 A9 A1	A10 A11 A12	

Refresh Address (RAS only)	RA2 RA3 RA4 RA5 RA6 RA7 RA0 RA1 RA8 RA9 RA10

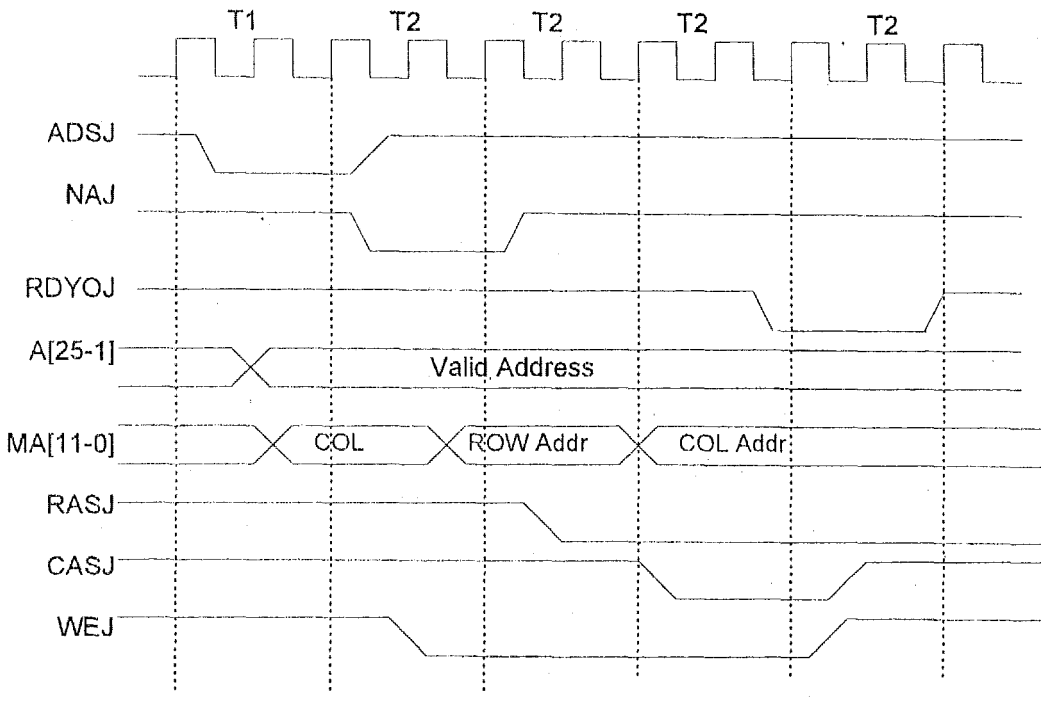
Acer Laboratories, Inc., USA, 1830B Bering Drive, San Jose, CA 95112; Tel: 408-467-7456; Fax: 408-467-7474
www.acerlabs.com

Section 6 : Timing Diagrams

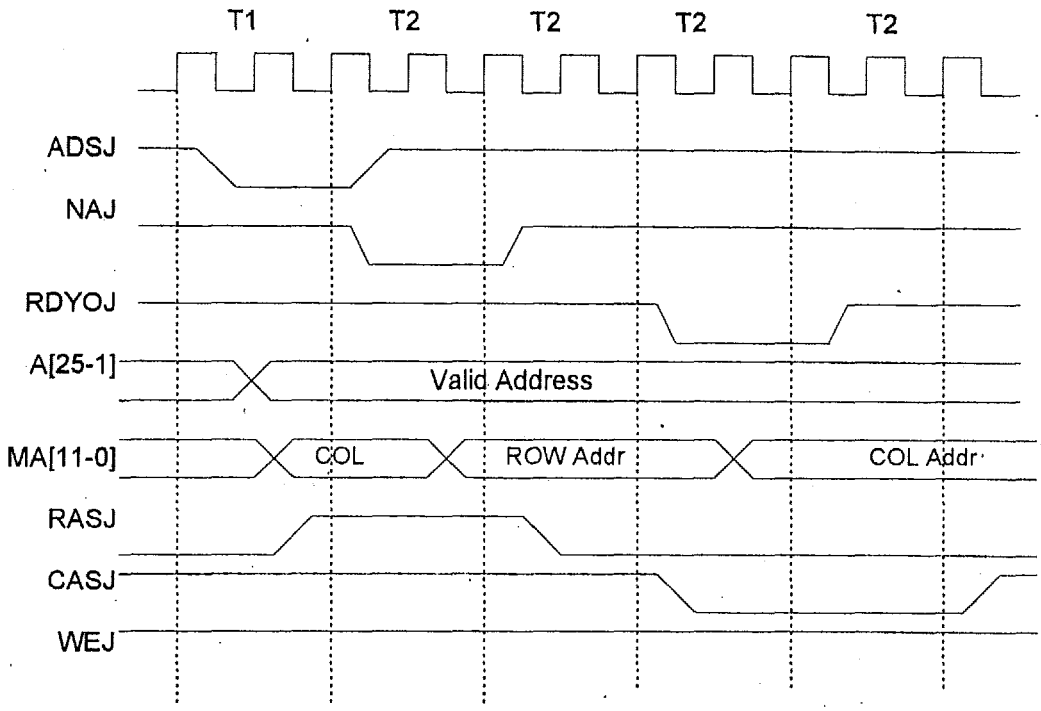
(A) **P9 Non-pipelined Read... Inactive .... 3 waits**



(B) P9 Non-pipelined Write... Inactive .... 2 waits

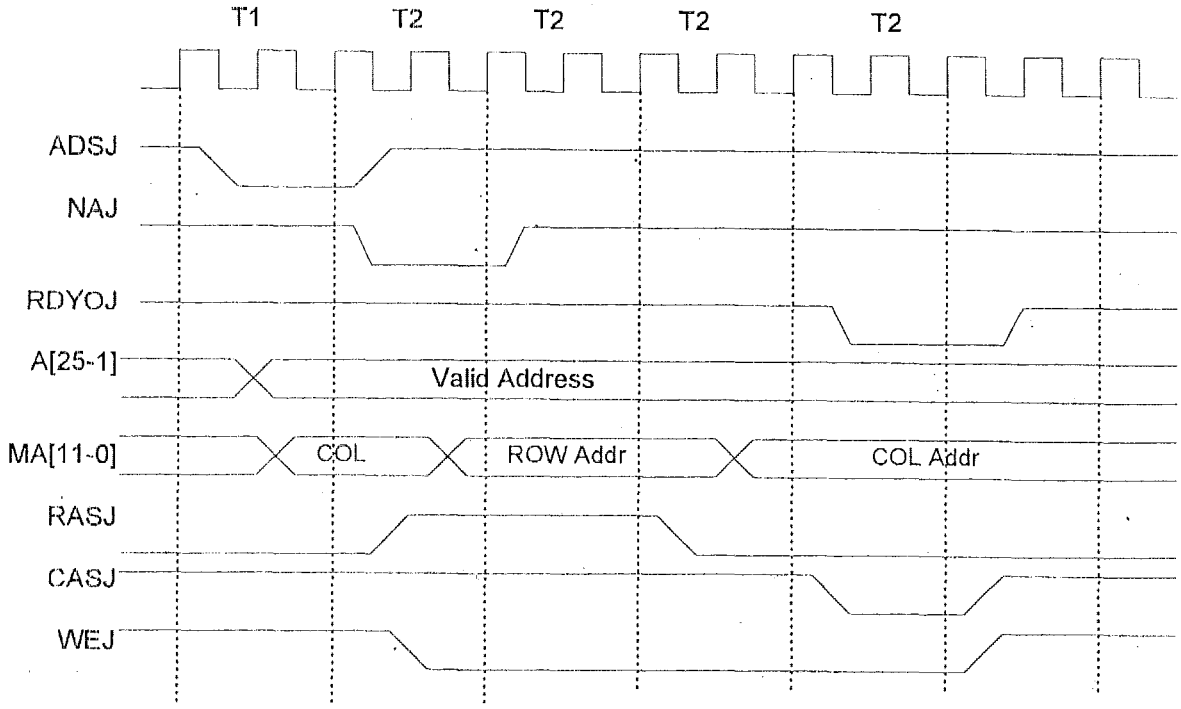


(C) **P9 Non-pipelined Read... Miss .... 4 waits**

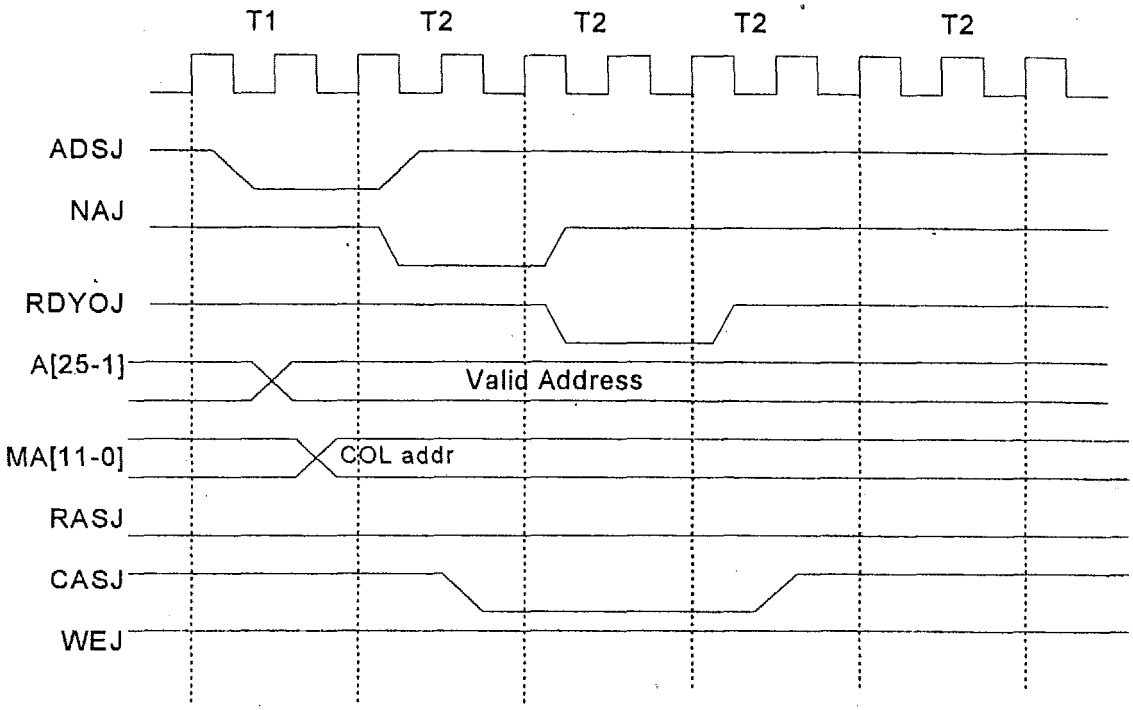


Ver. 0.93, Document Number: 6117DSC1.doc
Acer Laboratories, Inc., USA, 1830B Bering Drive, San Jose, CA 95112; Tel: 408-467-7456; Fax: 408-467-7474
www.acerlabs.com

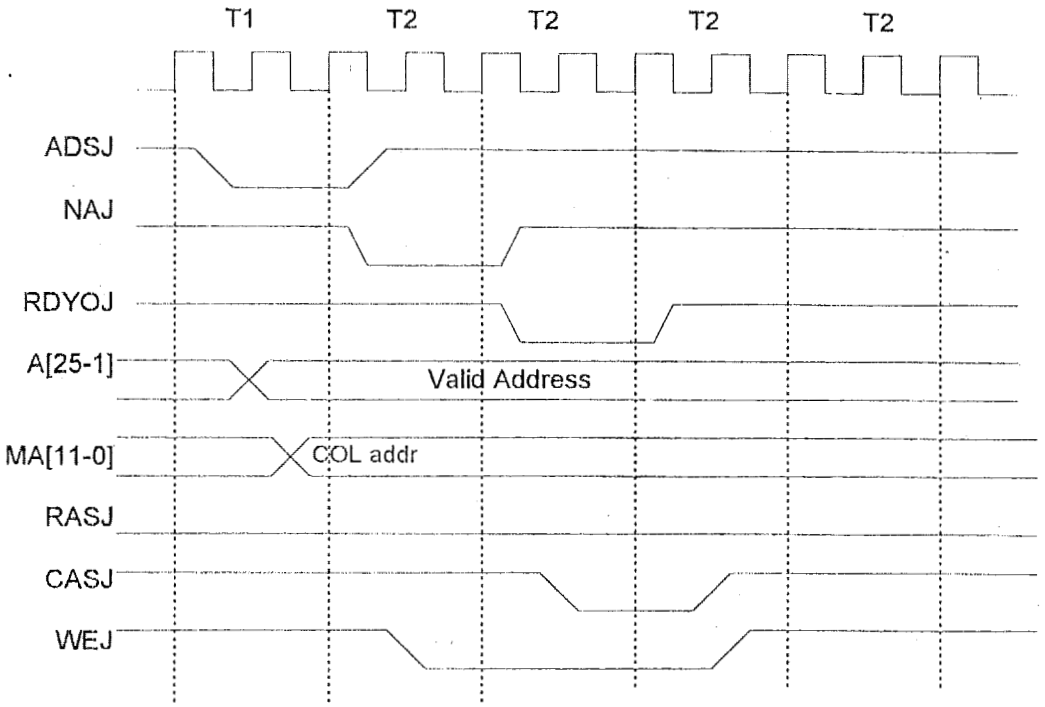
(D) **P9 Non-pipelined Write... Miss .... 3 waits**



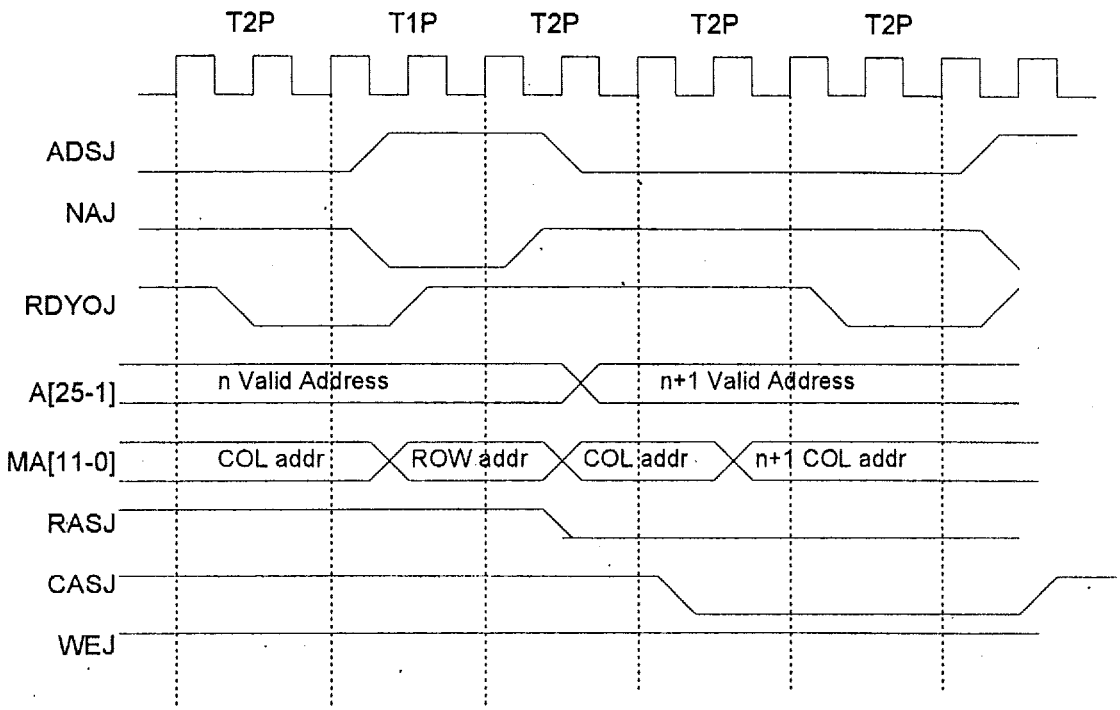
(E) **P9 Non-pipelined Read... Hit .... 1 wait**



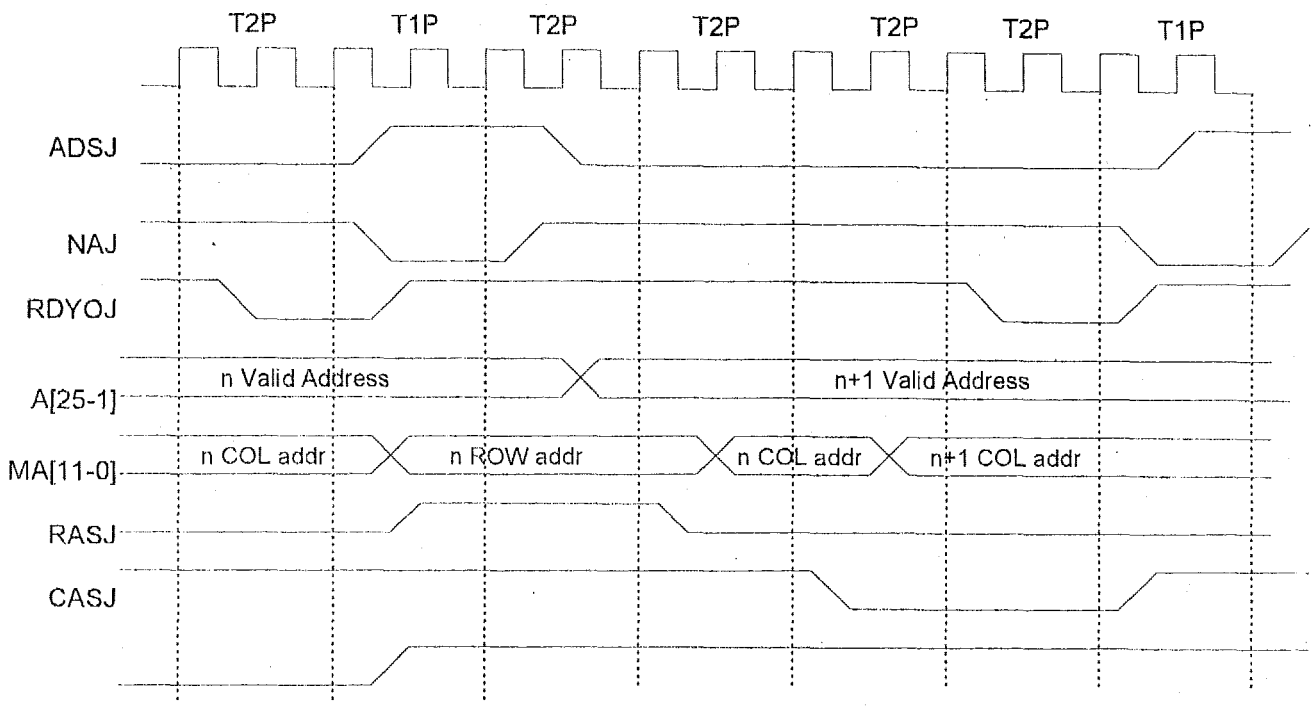
(F) **P9 Non-pipelined Write... Hit .... 1 wait**



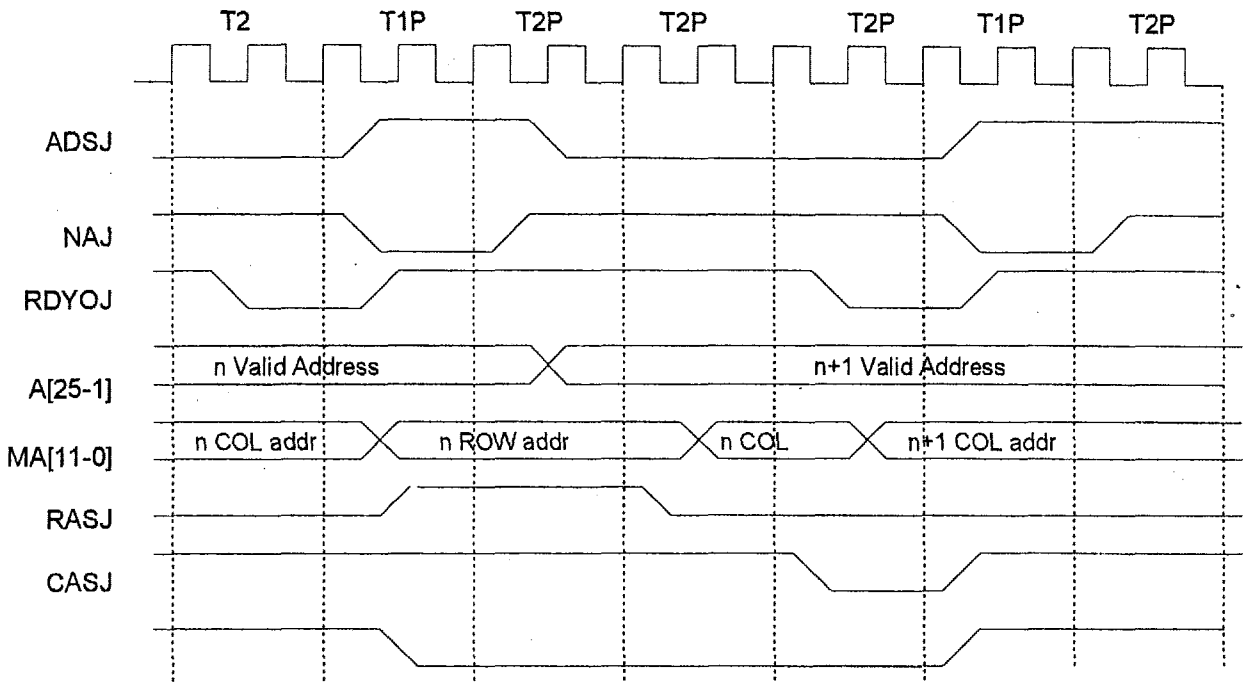
(G) **P9 Pipelined Read... Inactive .... 2 waits**



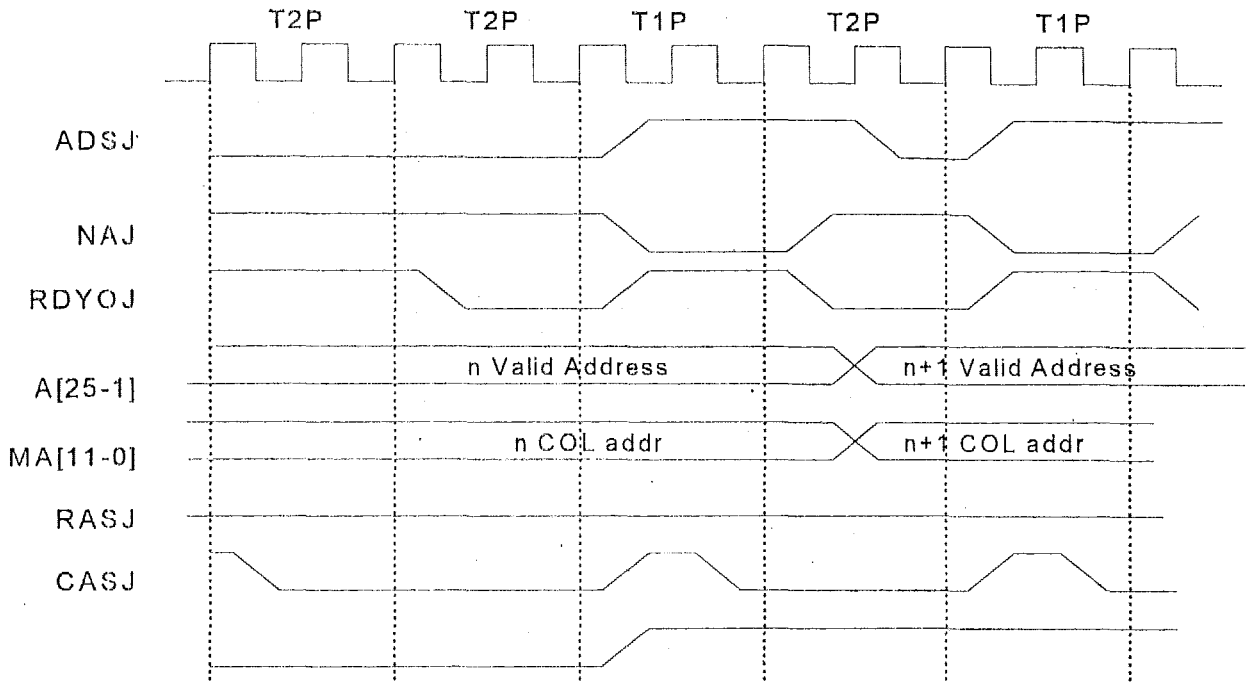
(H) **P9 Pipelined Read... Miss .... 3 waits**



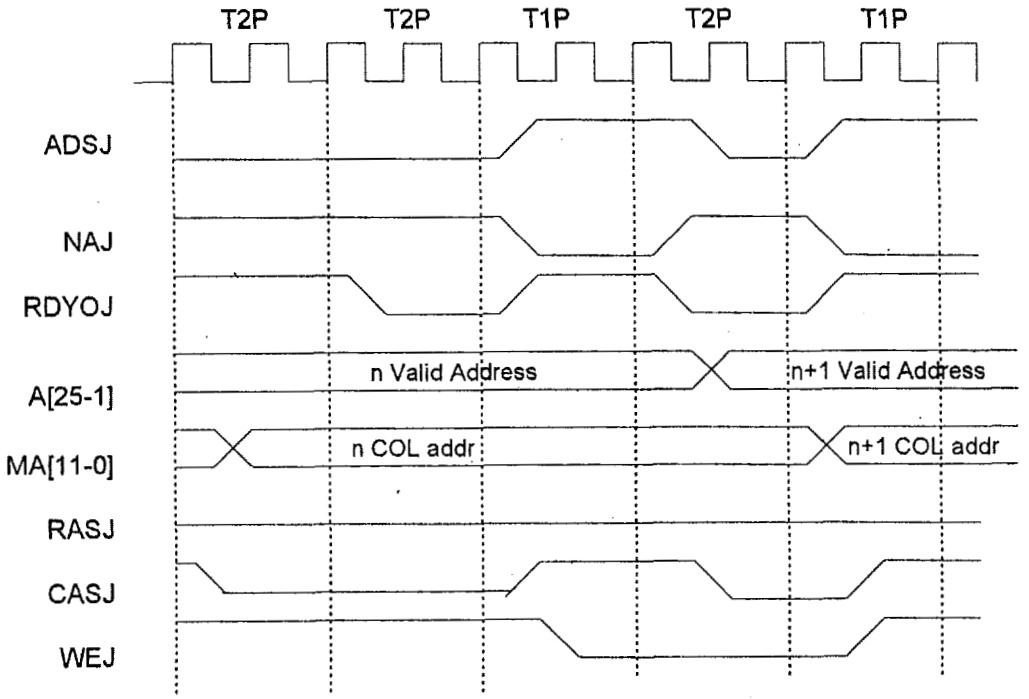
(l) **P9 Pipelined Write... Miss .... 2 waits**



(J) **P9 Pipelined Read... Hit .... 0 wait**

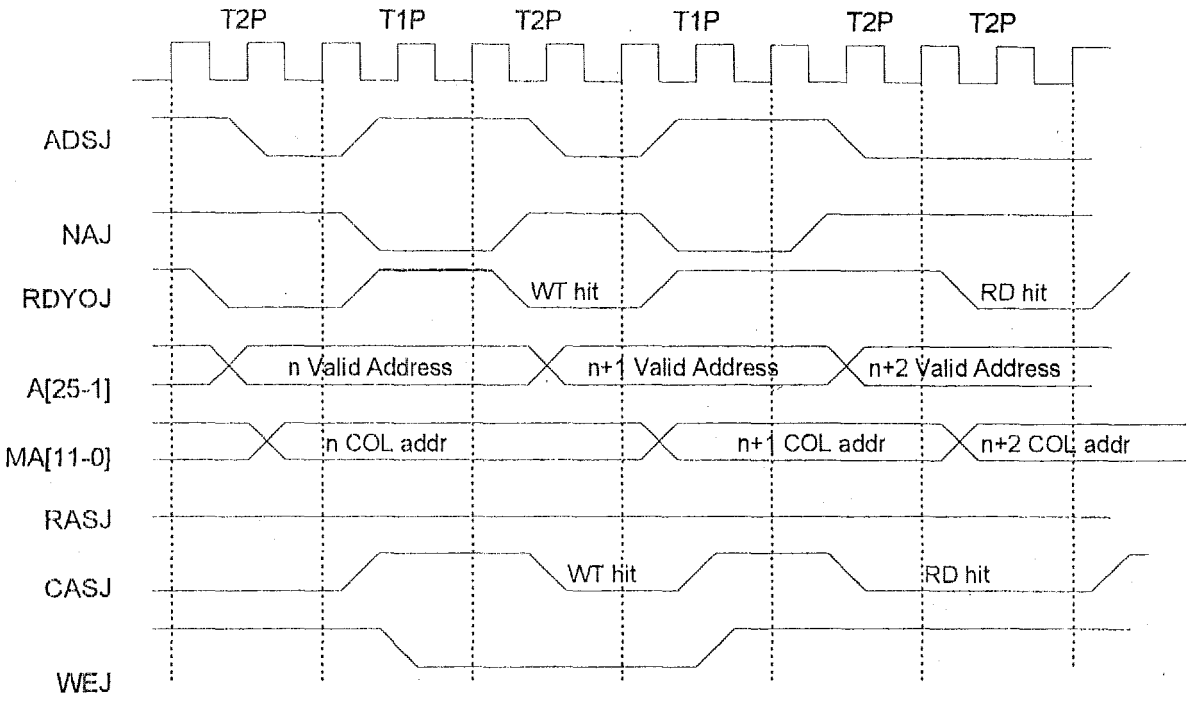


(K) **P9 Pipelined Write... Without Fast Write Hit .... 0 wait**

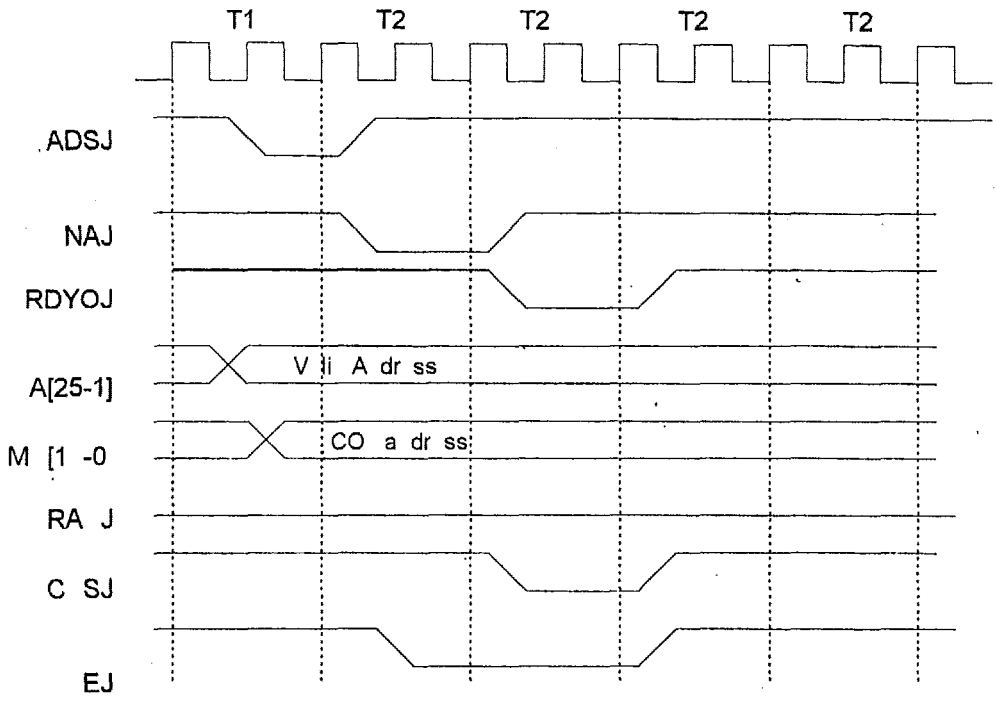


Ver. 0.93, Document Number: 6117DSC1.doc
Acer Laboratories, Inc., USA, 1830B Bering Drive, San Jose, CA 95112; Tel: 408-467-7456; Fax: 408-467-7474
www.acerlabs.com

(L) P9 Pipelined Read Hit after Write Hit... Add 1 wait

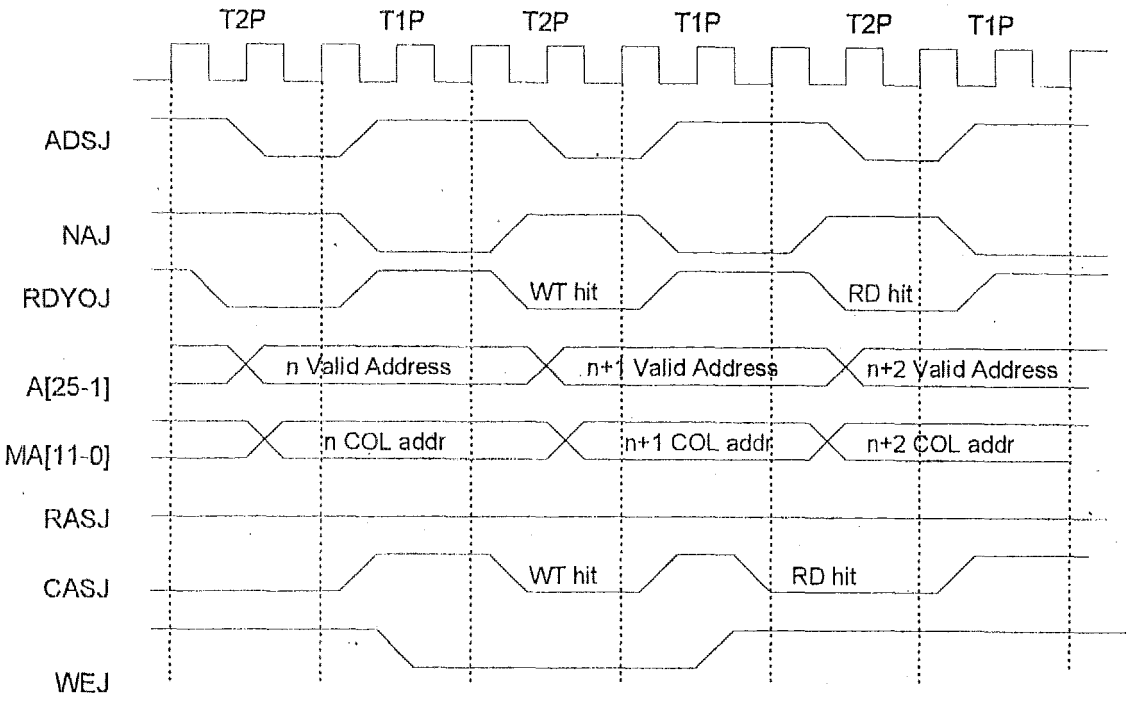


(M) **P9 Pipelined Write Hit... with Fast Write Hit... 1 wait**

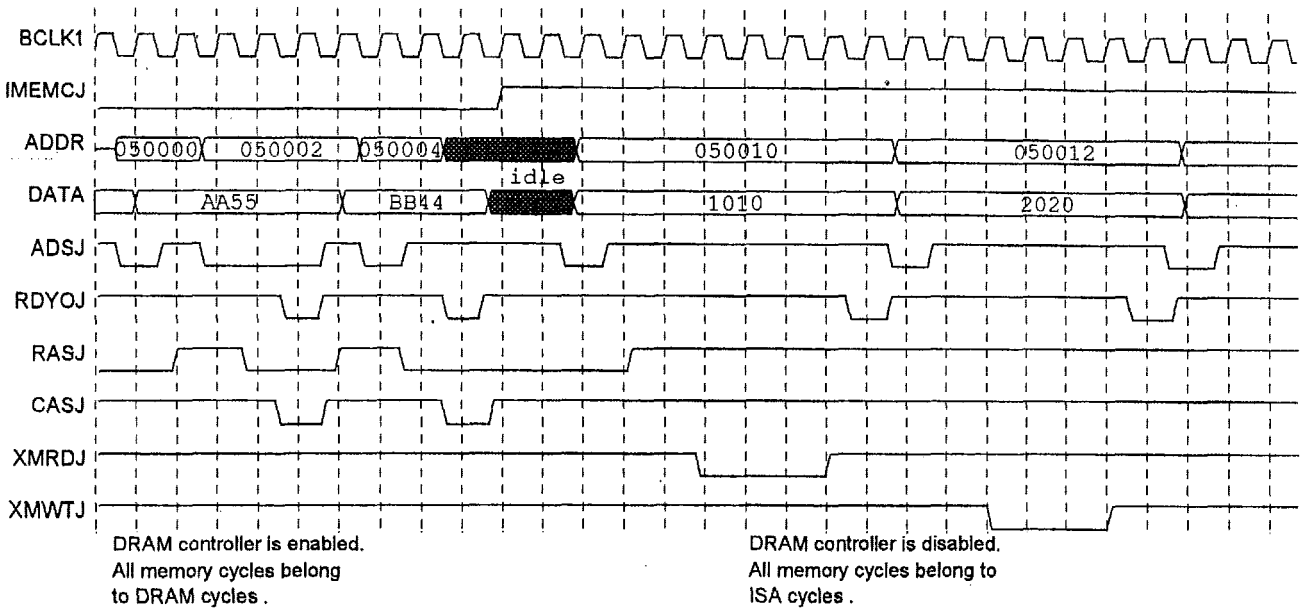


Ver. 0.93, Document Number: 6117DSC1.doc
Acer Laboratories, Inc., USA, 1830B Bering Drive, San Jose, CA 95112; Tel: 408-467-7456; Fax: 408-467-7474
www.acerlabs.com

(N) **P9 Pipelined Read Hit after Fast Write Hit... 0 wait**

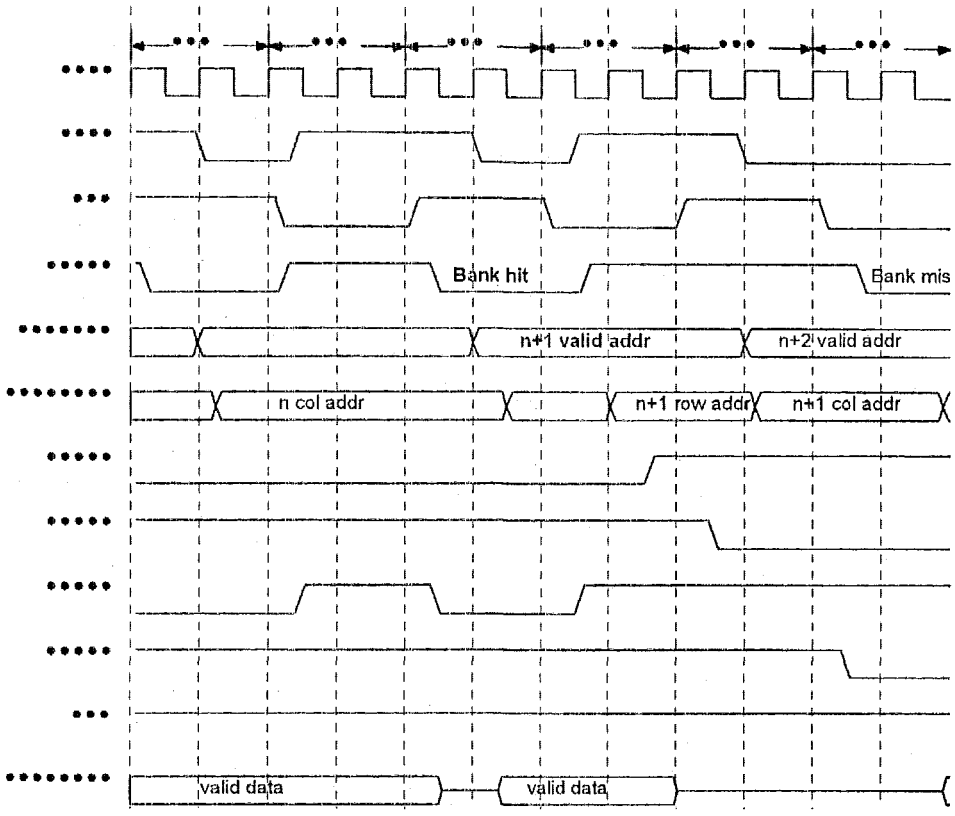


(O) Waveform of Disabling DRAM Controller



Acer Laboratories, Inc., USA, 1830B Bering Drive, San Jose, CA 95112; Tel: 408-467-7456; Fax: 408-467-7474

(P) Bank Miss for EDO DRAM Timing



Section 7 : Electrical Characteristics

Although all inputs are protected against ESD or inadvertent connection to high voltages, exposure to stresses exceeding absolute maximum ratings may permanently damage the device or seriously affect reliability.

	Symbol	Min.		Unit
Supply Voltage		- 0.5	+ 7.0	
Input Voltage	V	- 0.5	Vcc + 0.5	
Operating Temperature	T	0	+ 70	C
Storage Temperature	Stg	- 65		°C

Functional operating range: Tcase = 0 ° °C

Parameter	Symbol		Max.	Unit	
Input Low Voltage	V	- 0.3	+ 0.8		
Input High Voltage	V	2.0	Vcc+0.3		
BCLK2 Input Low Voltage	V	- 0.3	+ 0.8		
BCLK2 Input High Voltage	V	Vcc-0.8	Vcc+0.3		
Output Low Voltage	V	-	0.45		
Output High Voltage	V	2.4	-		
Input Leakage Current	I	-	±15	•	≤ V ≤
Input Pull Up Current	I	-	- 250	•	IL = 0.45V
	ILD		- 250	•	VIH = 2.4V
Output Leakage Current	ILO	-	±15	•	
Power Supply Current	ICC PS	-	 50	mA	CLKT = 80 MHz CLKT = 80 MHz

Acer Laboratories, Inc., USA, 1830B Bering Drive, San Jose, CA 95112; Tel: 408-467-7456; Fax: 408-467-7474
www.acerlabs.com

7.3 AC Characteristics

Symbol	Parameter	Minimum	Maximum	Unit	Notes
D	Data Flow from BD to SD delay			ns	Note 4
D	Data Flow from SD to BD delay			ns	Note 4
VD	16-bit MWTJ, MRDJ valid delay			ns	AT timing
FD	16-bit MWTJ, MRDJ float delay		18	ns	
t_{VD}		49		ns	
t_{FD}		9	18		AT timing
t	IO16J Input setup time	10			AT timing
t	IO16J Input hold time	5			AT timing
t	MEM16J Input setup time	10			AT timing
t	MEM16J Input hold time	5			AT timing
t	IOCHRDY Input setup time	15			AT timing
t	IOCHRDY Input hold time	5			AT timing
t	NOWSJ, MASTERJ Setup time	15			AT timing
t	NOWSJ, MASTERJ Hold time	5			AT timing
t	BALE Valid delay	54			AT timing
t	BALE Float delay	3		ns	AT timing
S	REFRESHJ Input setup time			ns	AT timing
h	REFRESHJ Input hold time			ns	AT timing
VD	REFRESHJ Output valid delay			ns	AT timing
FD	REFRESHJ Output float delay		8	ns	
t_{VD}		48		ns	
t_{FD}		33	40		Note 3
t	IOCHKJ Input setup time	15			AT timing
t	IOCHKJ Input Hold time	5			AT timing
t	RTCRJ, RTCWJ Valid delay			ns	
t_{FD}				ns	Note 3
S	IRQ Group Setup time			ns	AT timing
h	IRQ Group Hold time			ns	AT timing
S	DRQ Group Setup time			ns	AT timing
h	DRQ Group Hold time			ns	AT timing

AC Characteristics (continued)

Symbol	Parameter	Minimum	Maximum	Unit	Notes
	BCLK2		80	MHz	Clock signal
t_{VD}	BA2, BA23, BA24, BA25, BEHJ, BELJ, ADSJ, MIOJ, DCJ, WRJ, valid delay		10	ns	NP timing
t_{FD}	BA2, BA23, BA24, BA25, BEHJ, BELJ, ADSJ, MIOJ, DCJ, WRJ, float delay		10	ns	NP timing
t_S	BD0-BD15 Input setup time	5		ns	NP timing
t_h	BD0-BD15 Input hold time	3		ns	NP timing
t_{VD}	BD0-BD15 Output valid delay	20		ns	NP timing
t_{FD}	BD0-BD15 Output float delay	11		ns	NP timing
t_S	NPRDY7J Setup time			ns	NP timing
t_h	NPRDY7J Hold time			ns	NP timing
t_{VD}	RDY0J Valid delay	12		ns	NP timing
t_{FD}	RDY0J Float delay	10	20	ns	NP timing
t_S	PDL, PDH Input setup time	10		ns	Note 1
t_h	PDL, PDH Input hold time	3		ns	Note 1
t_{VD}	PDL, PDH Output valid delay		25	ns	Note 2
t_{FD}	PDL, PDH Output float delay	6		ns	Note 2
t_{VD}	RAS[3-0]J Valid delay	15		ns	DRAM timing
t_{FD}	RAS[3-0]J Float delay	5	11	ns	DRAM timing
t_{VD}	CAS[3-0][HL]J Valid delay	16		ns	DRAM timing
t_{FD}	CAS[3-0][HL]J Float delay	3	8	ns	DRAM timing
t_{VD}	MA[11-0] Valid delay	6		ns	DRAM timing
t_{FD}	MA[11-0] Float delay	3		ns	DRAM timing
t_{VD}	WEJ Valid delay	3		ns	DRAM timing
t_{FD}	WEJ Float delay	3	8	ns	DRAM timing
t_S	SA0, BHEJ Input setup time	4		ns	Note 3
t_h	SA0, BHEJ Input hold time	2		ns	Note 3
t_{VD}	SA[19,0], BHEJ Output Valid delay		2	ns	Note 3
t_{FD}	SA[19,0], BHEJ Output Float delay		8	ns	Note 3
t_{VD}	SD[15-0] Output Valid delay	0		ns	CPU timing
t_{FD}	SD[15-0] Output Float delay	3		ns	CPU timing
t_{VD}	DACKJ Group Valid delay	90		ns	AT timing
t_{FD}	DACKJ Group Float delay	15		ns	AT timing
t_{VD}	RSTNP Valid delay	8		ns	CPU timing
t_{FD}	RSTNP Float delay	3		ns	CPU timing
t_{VD}	RSTDRV Valid delay	5		ns	CPU timing
t_{FD}	RSTDRV Float delay	4		ns	CPU timing
t_{skew1}	Time skew between ATCLK and BCLK2	9	20	ns	
t_{skew2}	Time skew between CK7M and OSC14M	6	12	ns	

Acer Laboratories, Inc., USA, 1830B Bering Drive, San Jose, CA 95112; Tel: 408-467-7456; Fax: 408-467-7474
www.acerlabs.com

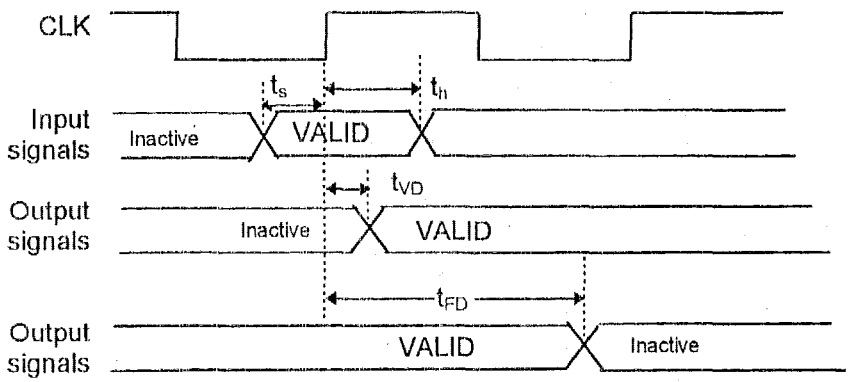
Meaning of symbols :

t_D	Data Flow delay
t_S	Input Setup time
t_h	Input Hold time
t_{VD}	Output Valid delay
t_{FD}	Output Float delay
t_{skew1}	Clock skew time between BCLK2 and ATCLK
t_{skew2}	Clock skew time between OSC14M and CK7M

Notes :

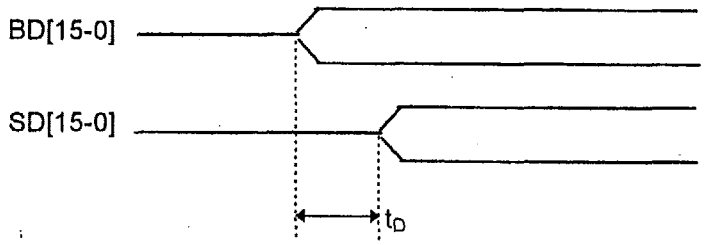
1. Parity data is only recognized in non-CPU memory read cycles, the timing requirements are related to command ending
2. Parity data are only generated in non-CPU memory write cycles, the timing are related to the stable ISA data. The memory cycles in notes 2 and 3 refer to the on-board local memory cycles.
3. The timing refers to the generated-delay after the CPU stable address.
4. The timing refers to propagating delay from BD to SD.
5. The timing refers to propagating delay from SD to BD.

The following pages show the input waveforms :
Setup, Hold, Valid, Float Delay time description

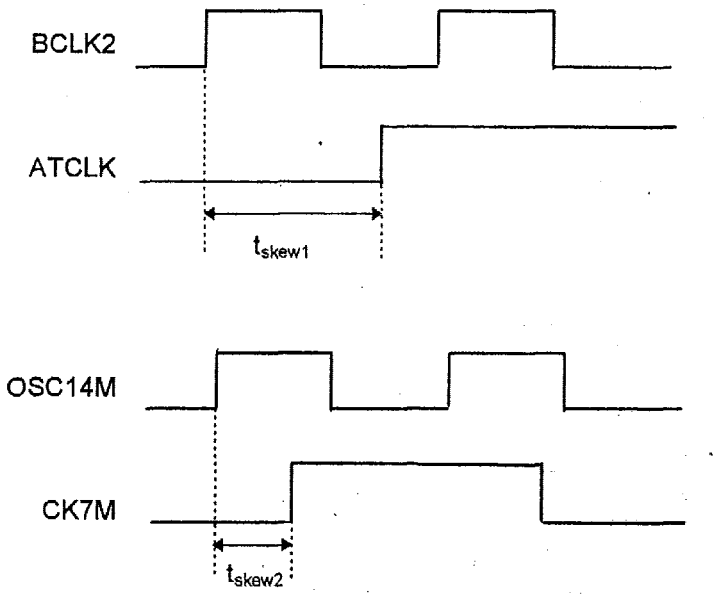


Note : 1. For coprocessor and DRAM side signals, CLK = CLK2
2. For ISA side signals, CLK = ATCLK
3. Signal reference level = 1.5 V
4. Environment : loading 50 pF

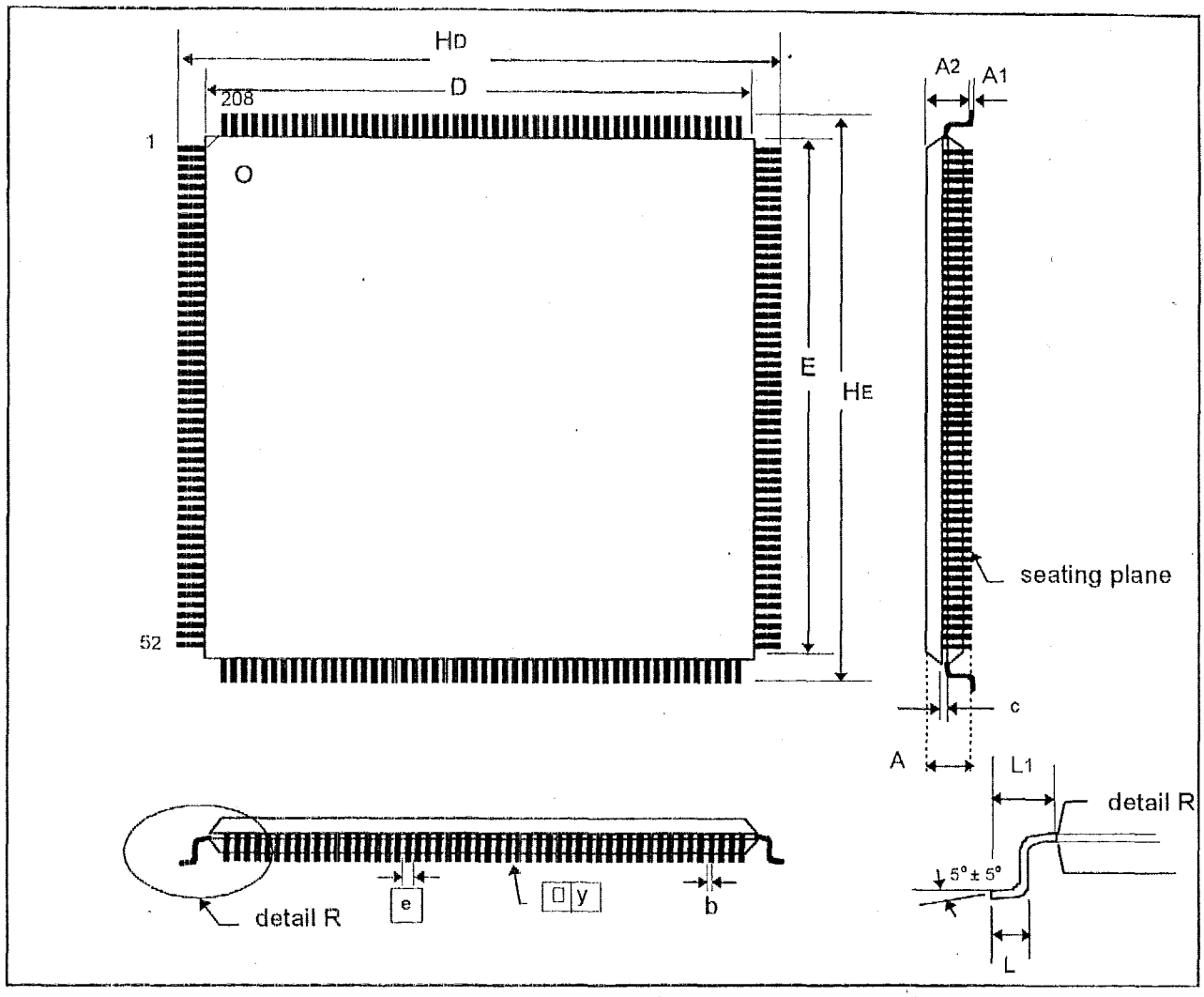
Data Flow Delay Description



Clock Skew Time Description



Section 8 : Packaging Information



Symbol	Dimensions in Millimeters (nom)	Dimensions in Inches (nom.)
A	3.5 (max)	0.137 (max)
A_1	0.2 (min)	0.008 (min)
A_2	3.0	0.118
b	0.18	0.007
c	0.15	0.006
D	28.0	1.102
E	28.0	1.102
e	0.5	0.020
H_D	30.6	1.205
H_E	30.6	1.205
L_1	1.3	0.051
L	0.5	0.020
y	0.15 (max)	0.006 (max)

Section 9 : Revision History

i.v.	11/11/97
p.15,29	03/13/98
p.18	05/28/98
p.29	06/30/98

Appendix
Subject : New Improvement of Power Good Circuit on M6117
Date : October 1, 1997
Part & Version : Total Pages : 4

To : All Distributors & M6117 Customers
From : ALi

Note : M6117 needs delay of 150ms for PWG signal.

0. Please refer to Fig.1 for PWG circuit.

1. PWG form power supply:
- Power good delay time depends on the power supply spec.
- PWG rise time depends on R8 / C2 value, warm reset delay time depends on C2 / R9 value.
- Please refer to Fig.2 - Fig. 4 for scope timing.

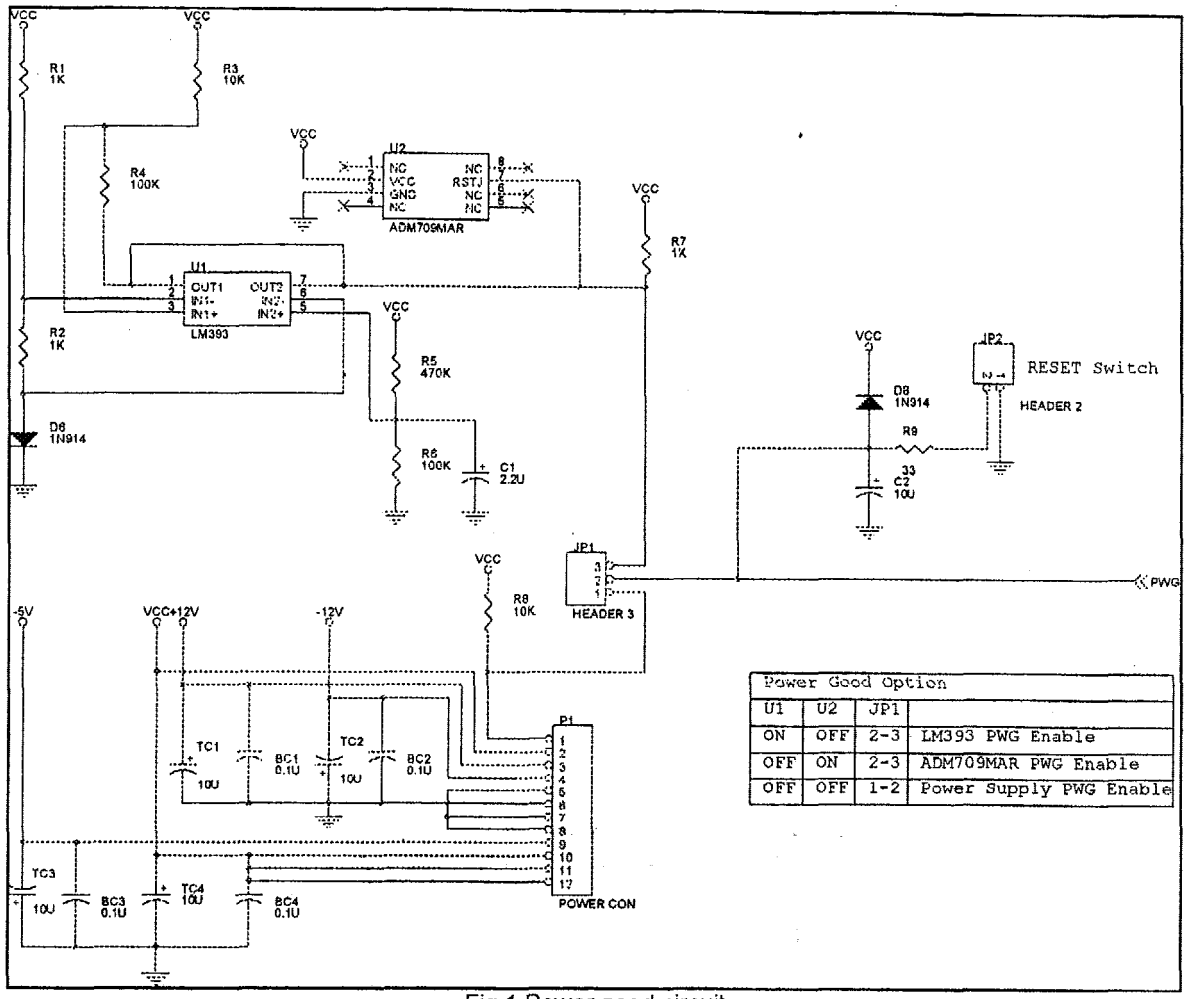
2. PWG form LM393
- PWG rise time, delay time, warm reset delay time all depend on R / C value.
- PWG delay time depends on R5,R6 / C1 value (see table1 for delay time setting), PWG rise time depends on R7 / C2 value, warm reset delay time depends on C2 & R9 value.
- Please ***add C1*** for LM393 circuit and modify R5 & R6 to 470K & 100K. If we do not modify LM393 circuit for PWG, there may be some power-on failure on high speed application (such as 40Mhz). This change will insert delay time for VCC rise to valid voltage.
- Please refer to Fig.5 - Fig. 9 for scope timing.

R5	R6	C1	Delay Time
470K	100K	10U	1080ms
470K	100K	4.7U	350ms
470K	100K	2.2U	306ms
47K	10K	10U	150ms
47K	10K	4.7U	98ms

Table1 LM393 delay time

3. PWG form ADM709MAR

- PWG delay time fix in 184 ms by ADI.
- PWG rise time depends on R7 / C2 value, warm reset delay time depends on C2 / R9 value.
- Please refer to Fig.10 - Fig. 12 for scope timing.



Power Good Option			
U1	U2	JP1	
ON	OFF	2-3	LM393 PWG Enable
OFF	ON	2-3	ADM709MAR PWG Enable
OFF	OFF	1-2	Power Supply PWG Enable

Fig.1 Power good circuit

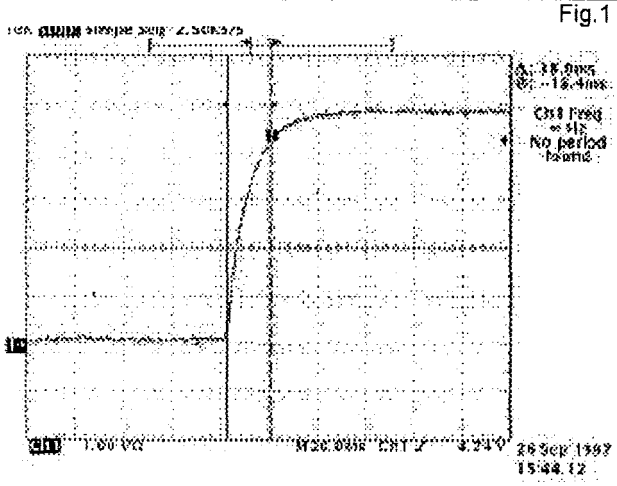


Fig.2 Power supply PWG rise(18ms)

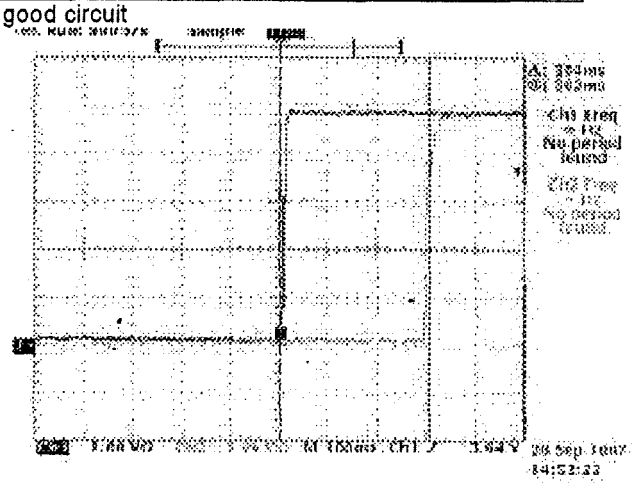


Fig.3 Power supply PWG delay time(304ms)

Acer Laboratories, Inc., USA, 1830B Bering Drive, San Jose, CA 95112; Tel: 408-467-7456; Fax: 408-467-7474
www.acerlabs.com

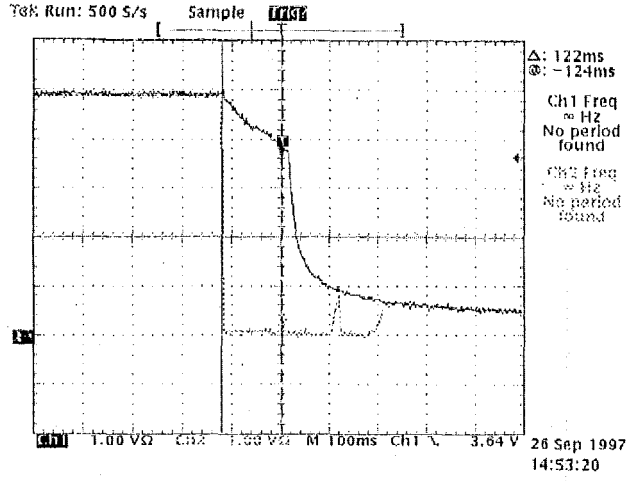


Fig.4 Power supply Power Fail Delay time(122ms)

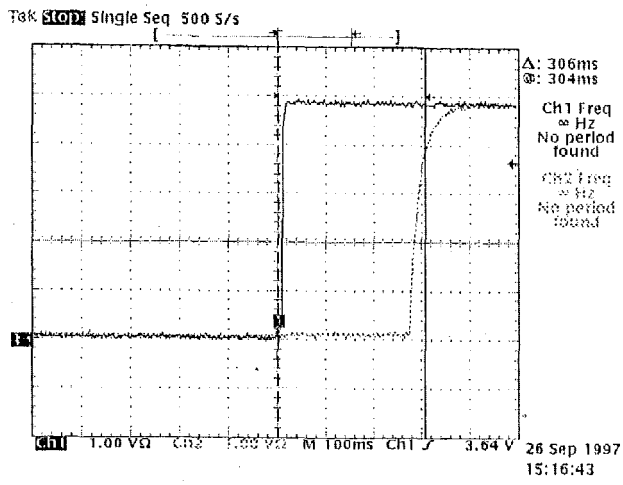


Fig.5 LM393 PWG delay time (org. , no delay)

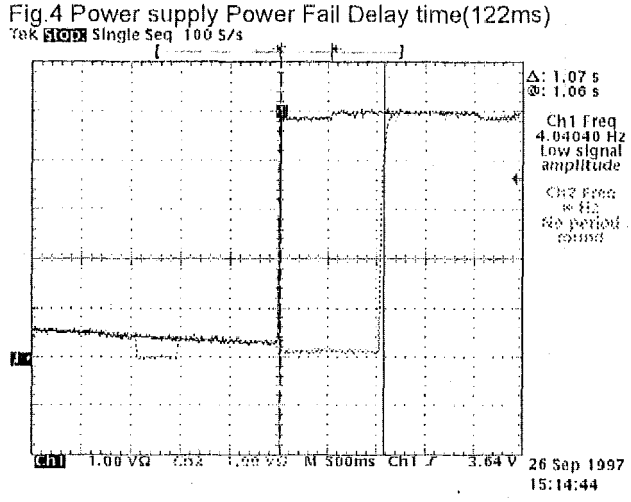


Fig.6 LM393 PWG Delay time(1.07s by 10U)

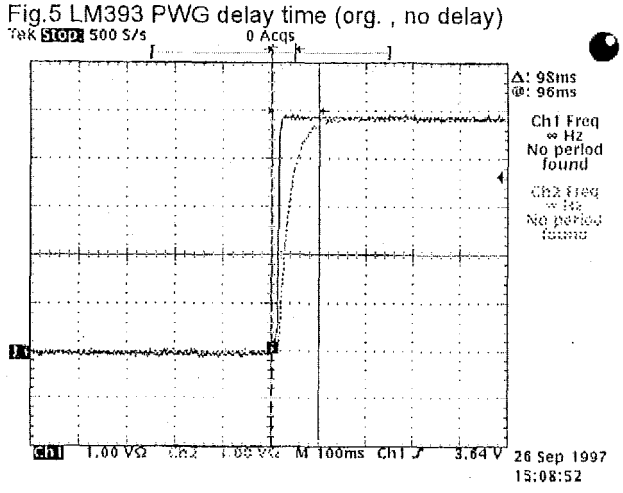


Fig.7 LM393 PWG Delay time(306ms by 2.2U)

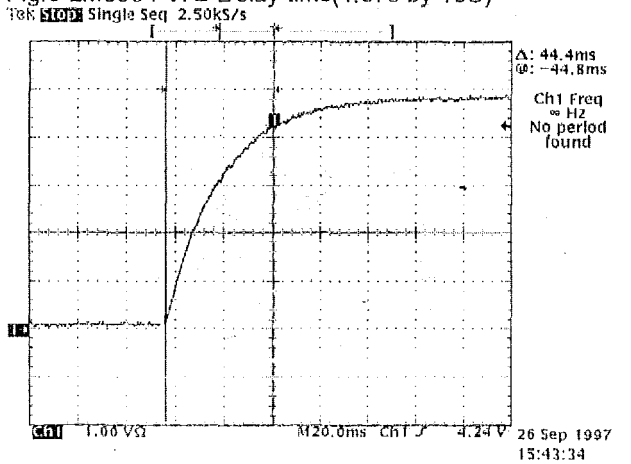


Fig.8 LM393 PWG rise(44.4ms)

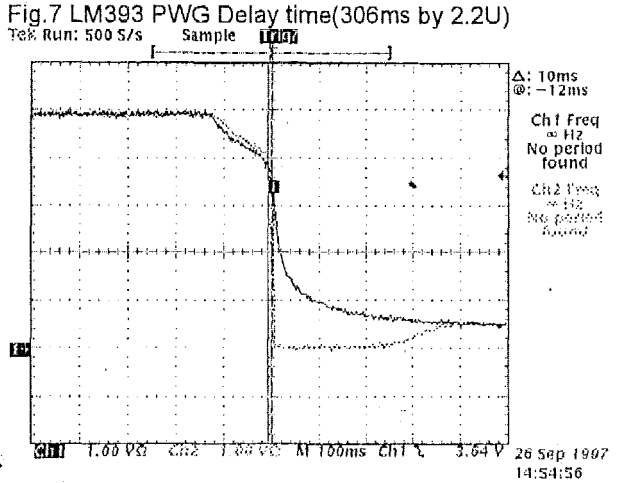


Fig.9 LM393 PF Delay time(0ms)

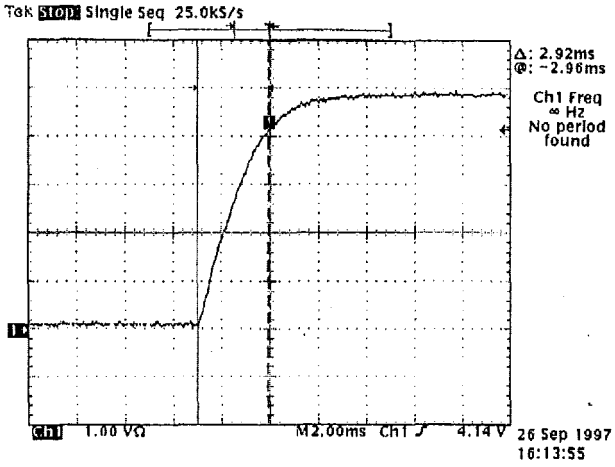


Fig.10 ADM709 PWG rise(2.92ms)

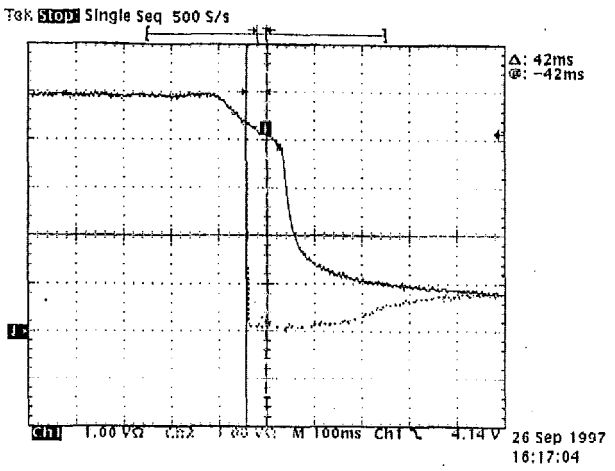


Fig.11 ADM709 Power Fail delay time(42ms)

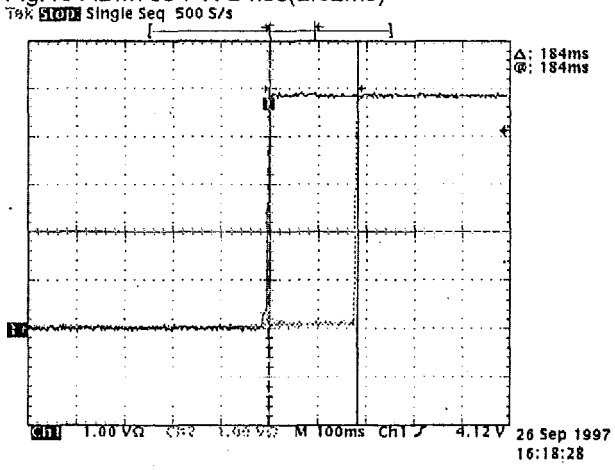


Fig.12 ADM709 PWG delay time(184ms)

Acer Laboratories, Inc., USA, 1830B Bering Drive, San Jose, CA 95112; Tel: 408-467-7456; Fax: 408-467-7474
www.acerlabs.com

This material is recyclable.

Pentium is a trademark of Intel Corp.
Windows is a trademark of Microsoft Corp.
Other brands and names are the property of their respective owners.

Acer Labs products are not licensed for use in medical applications, including, but not limited to, use in life support devices without proper authorization from medical officers. Buyers are requested to inform ALi sales office when planning to use the products for medical applications.

Product names used in this publication are for identification purposes only and may be trademarks of their respective companies.

Acer Laboratories Inc. makes no warranty for the use of its products and assumes no responsibility for any errors which may appear in this document nor does it make a commitment to update the information contained herein.

Acer Laboratories Inc. retains the right to make changes to these specifications at any time, without notice.

Contact your local sales office to obtain the latest specifications before placing your order.

ALi is a registered trademark of Acer Laboratories Incorporated and may only be used to identify ALi's products.

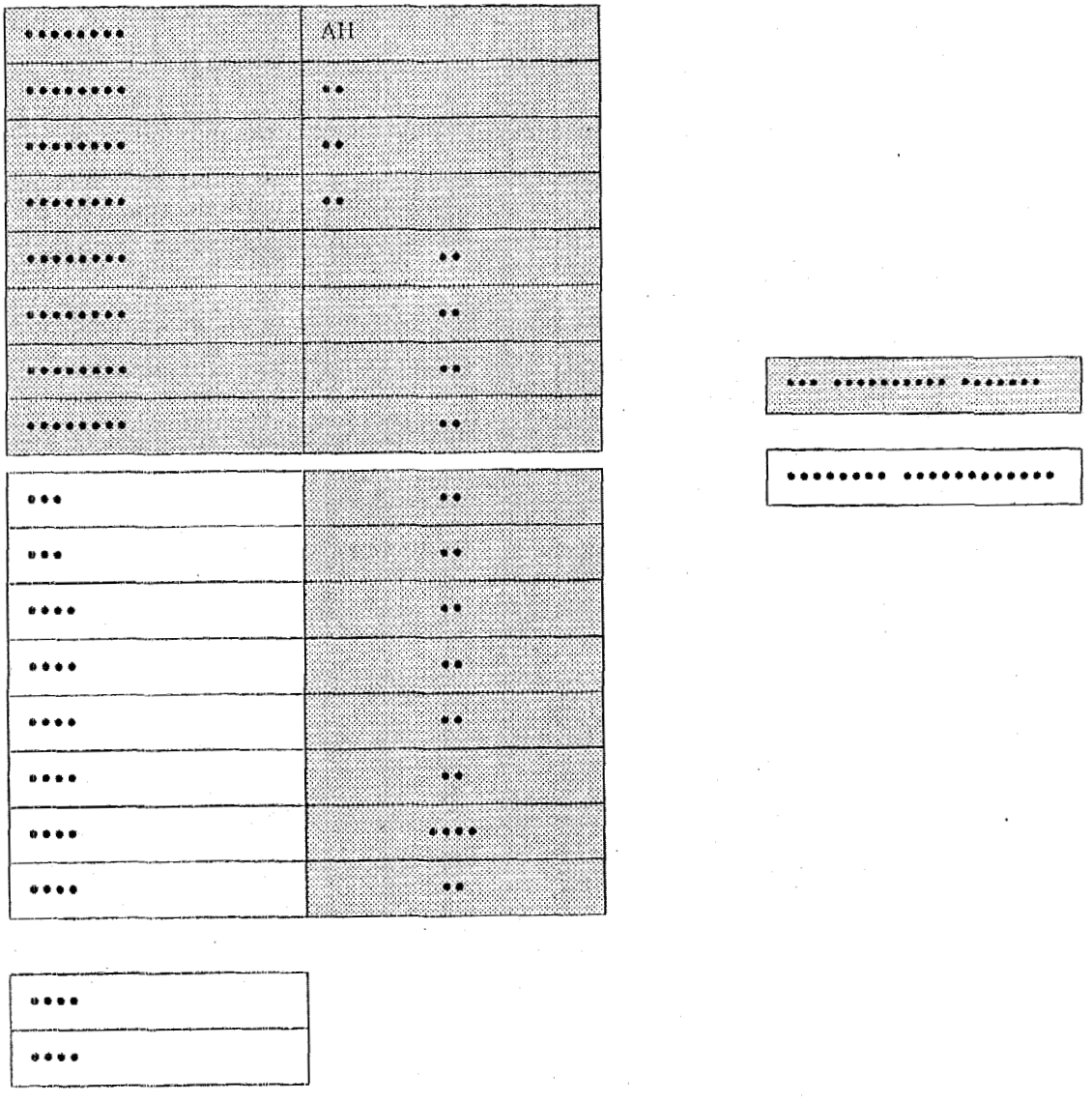
Acer Laboratories, Inc., USA, 1830B Bering Drive, San Jose, CA 95112; Tel: 408-467-7456; Fax: 408-467-7474
www.acerlabs.com

APPENDIX A. M6117C's CPU Registers

The CPU registers are compatible with 80386 registers. They also provide more powerful operations on register programming.

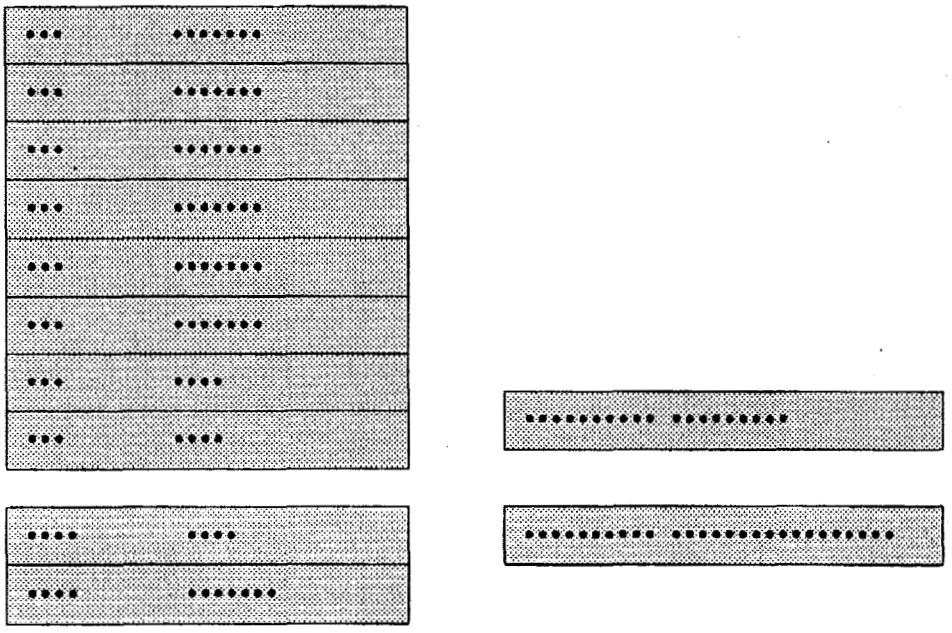
A.1 General Purpose Registers (GR)

The CPU incorporates 32 general purpose 32-bit registers. In arithmetic and logic operations, these registers can be used as operands. The registers then are used as data registers. In address calculations, the registers are used as base registers or index registers.



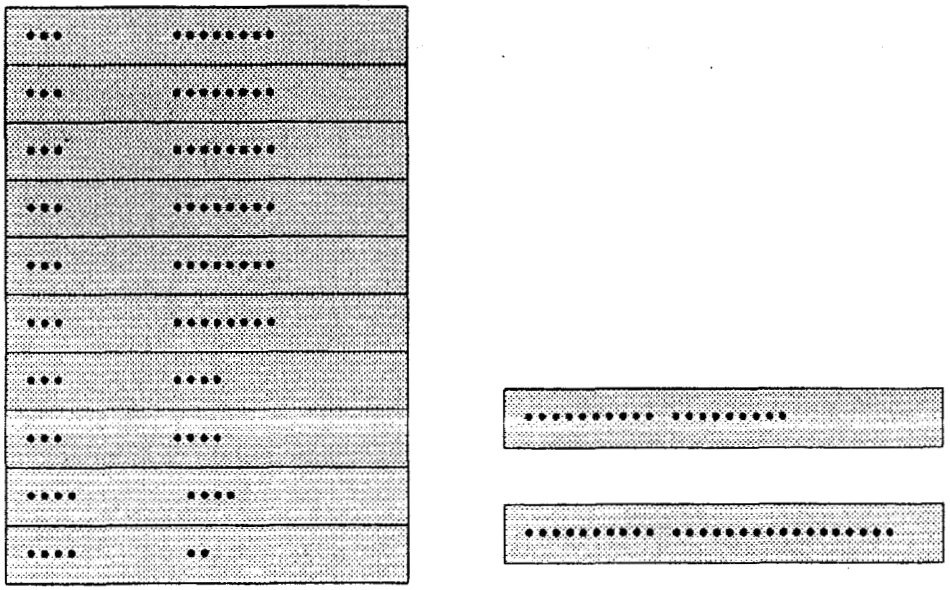
A.2 Base Registers (SR)

The CPU has 10 32-bit segment base registers to locate a block of program or data in the memory area. When the 80386 object code is used, these registers operate automatically. They require no operator intervention.



A.3 Attribute Registers (AR)

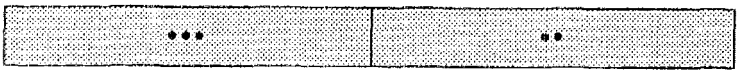
The CPU has 10 32-bit segment attribute registers. Each attribute register holds an attribute of the corresponding segment register. When the 80386 object code is used, these registers operate automatically. They require no operator intervention.



Acer Laboratories, Inc., USA, 1830B Bering Drive, San Jose, CA 95112; Tel: 408-467-7456; Fax: 408-467-7474
www.acerlabs.com

A.4 Instruction Pointer

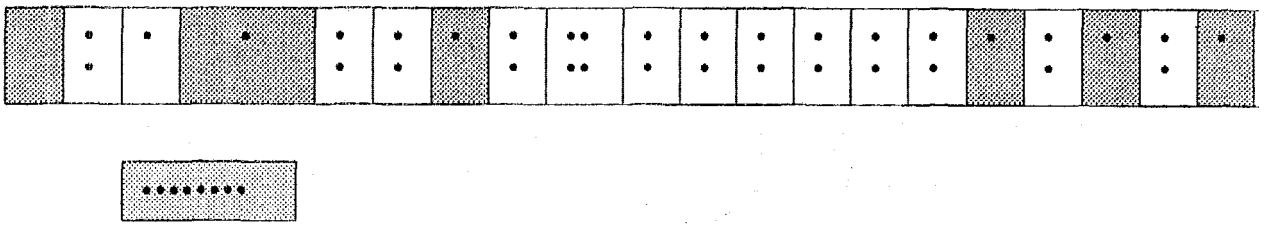
The instruction pointer (EIP) register contains the offset in the current code segment for the next instruction to execute. It is controlled implicitly by control-transfer instructions (jumps, returns, etc.), interrupts, and exceptions.



According to the CPU mode or operand size, the program counter is used as either a 32-bit register EIP, or a 16-bit register IP. The initial value of EIP register is 0000FFF0h.

A.5 Flags Register

The CPU has flags to indicate current modes and operation results, EFLAGS. The initial value right after reset is 00000002h.

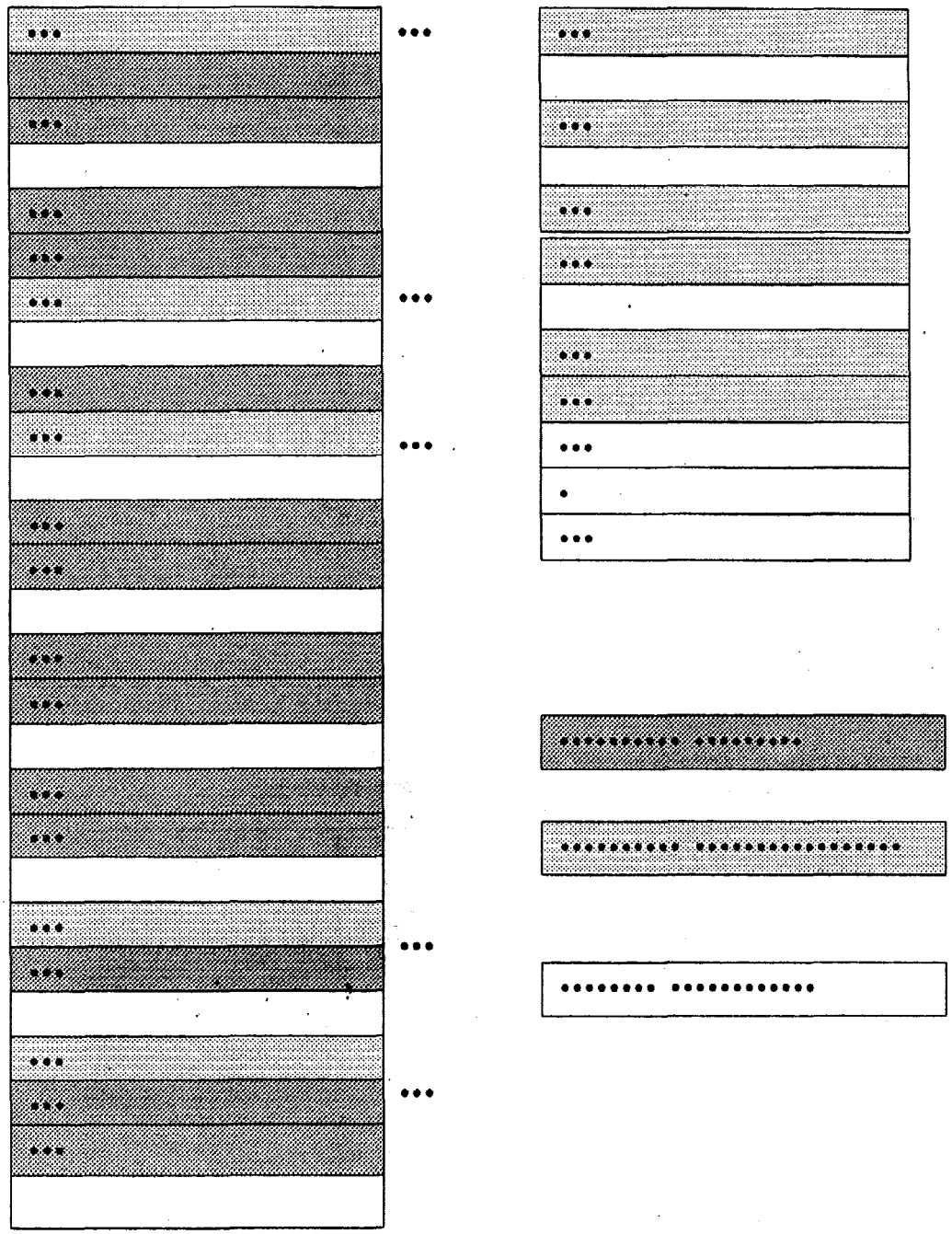


BIT 30 EA : Extended Address Generation Flag
This bit controls paging.
BIT 29 D : Default Flag
Cleared by external reset signal. Set/Clear by special OP codes :
BRDD/BRDW instructions (Change Context to 32/16 bit and jump).

D-Flag	0	0	0	0	1	1	1	1
Operand Size Prefix	N	Y	N	Y	N	Y	N	Y
Address Size Prefix	N	N	Y	Y	N	N	Y	Y
Operand Size in Selection	8/16	8/32	8/16	8/32	8/32	8/16	8/32	8/16
Address Size in Selection	16	16	32	32	32	32	16	16

A.6 Control Registers(CR)

The CPU incorporates up to 256 32-bit control registers. Some registers is assigned to the 80386-compatible functions and can be accessed by 80386-compatible instructions.



These registers can be divided into 5 groups :

1. Processor Control Registers,
2. Instruction Decoding Unit Control Registers,
3. Protected-Mode Instruction Unit Control Registers,
4. Paging Unit Control Registers,
5. SiEMU-II Unit Control Registers.

Acer Laboratories, Inc., USA, 1830B Bering Drive, San Jose, CA 95112; Tel: 408-467-7456; Fax: 408-467-7474
www.acerlabs.com

A.6.1 Processor Control Registers

CR00h CR0

The CR00h contains those bits to control an interface of numeric coprocessor and execution modes. The initial value right after reset is "7FFFFFD0h." The meaning of every bit except bit 5 IOB is same as normal definition.

bit 5 IOB

(A) 16-bit I/O bus mode(IOB = 0, default setting)

Byte operation at even address. (DBHE# = 1, DBLE# = 0)
DATA BUS

Byte operation at odd address. (DBHE# = 0, DBLE# = 1)
DATA BUS

Word operation at even address. (DBHE# = 0, DBLE# = 0)
DATA BUS

Word operation at odd address.
DATA BUS

(B) 8 bit I/O bus mode(IOB = 1)

Byte operation at even address. (DBHE# = 1, DBLE# = 0)
DATA BUS

Byte operation at odd address. (DBHE# = 0, DBLE# = 1)
DATA BUS

Word operation at even address. (DBHE# = 0, DBLE# = 0)
DATA BUS

Word operation at odd address.
DATA BUS
DATA BUS

CR02h(EPWRCR)special

This register controls Power Save Mode. The initial value right after reset is FFFF00FFh.

bit 15 PCSTP

When set, the CPU enters Power Save Mode and main clock stops functioning after HALT instruction.
If HLDREQ input becomes active in the Power Save Mode. CPU enters a hold state. When HLDREQ becomes inactive. CPU returns to the Power Save Mode.
Main clock starts again when NMI, INTREQ, or RESET becomes active. When clear, HALT instruction is executed normally. PCSTP is cleared after external reset signal.

CR03h(PMON) special

This register controls processor monitor function. The initial value right after reset is EAAFFFFFh.

bit 28 PMM: Protected Mode Instruction Monitor

When set, the following instructions cause an operation equivalent to the BRKPM interrupt:
PUSHF, POPF, CLI, STI, IRET, LGDT, SGDT, LIDT, SIDT, LLDT, SLDT,
LTR, STR, LMSW, SMSW, ARPL, CLRTS, LDAC, LDLM, TRDE, TWRE,
move to/from CRx, DRx, TRx(CR00h, 08h, 0Ch, 19h, 1Dh, 30h, 34h, 38h,
39h, 3Ch, 3Dh) from/to GRx.

This allows the software to emulate the instructions used for a protected-mode OS, and a protected-mode OS can run as a guest OS under the virtual machine OS. The Protected Mode Instruction Monitor can then be used for debugging the protected-mode OS or BIOS. It can also be used for processor testing.

bit 26 VMZM#: VMZ Instruction Monitor

If set at '0', when a special instruction (move immediate to CS, etc. which is not compatible with 386 instruction) is attempted in protected mode, an invalid operand exception (exception 6) occurs. If set at '1', such instructions become valid in protected mode. After reset, this bit is set at '0' to make possible a protection function. The bit 31 - 29 of EFLAGS can be written only when VMZM# is '1'.

bit 24 EXPM: Exception Monitor

If set, when either a hardware or software exception is detected in protected mode, an execution similar to BRKPM occurs.

bit 22 TM: Test Mode

If set, when an exception occurs, CPU stores the exception number in GR26 and jumps indirectly using GR27 as a new IP.

bit 20 DPAC: Cycle Steal Address Calculation Disable

When set, cycle steal address calculation is disabled and when clear, it is enabled. Cycle steal address calculation is an address operand calculation of the next instruction using the unused time slot in internal buses for pre-fetching.

bit 18 IOM: I/O Instruction Monitor

When set, the instructions below cause an operation equivalent to the BRKPM interrupt. This forces the CPU to enter Real Mode, regardless of the mode that it was in.

IN, OUT, INS, OUTS

A.6.2 Instruction Decoding Unit Control Registers

CR0Eh(TOP) special

The register controls the 'opcode trap function', together with ***CR0Fh***. It is operated only through Real Mode instructions. The initial value right after reset is undefined. Each field of 'OPCODE 3' to 'OPCODE 0' stores an opcode to be trapped. The relevant opcode will cause the same operation as BRKPM.

CR0Fh(TCON) special

The register controls the 'opcode trap function', together with ***CR0Eh***. It is operated only through Real Mode instructions. The initial value right after reset is 00000000h. Each field of 'OPCODE 3' to 'OPCODE 0' stores an opcode to be trapped. The relevant opcode will cause the same operation as BRKPM.

bit 16 TCE: Opcode Trap Enable

When set, the opcode trap is enable. Clears when an opcode trap occurs.

bit 15 - 8: Trap Opcode Mask

When set, masks compares to the bits set for OPCODE BREAK 0.

bit 7 - 0 : Map 3 - 0

00 : No trap
01 : 1st opcode
10 : 2nd opcode following 00Fh
11 : 2nd opcode following 0D6h

Acer Laboratories, Inc., USA, 1830B Bering Drive, San Jose, CA 95112; Tel: 408-467-7456; Fax: 408-467-7474
www.acerlabs.com

A.6.3 Protected-Mode Instruction Unit Control Registers

CR12h(TDESH) special

This register is used under Protected Mode. It is accessible when the CPU is in Real Mode or VMZ# is set. However, operation by software is not normally required. The initial value right after reset is undefined.

CR13h(TDESL) special

This register is used under Protected Mode. It is accessible when the CPU is in Real Mode or VMZ# is set. However, operation by software is not normally required. The initial value right after reset is undefined.

CR16h(TZ) special

This register is used under Protected Mode. It is accessible when the CPU is in Real Mode or VMZ# is set. However, operation by software is not normally required. The initial value right after reset is undefined.

CR17h(TY) special

This register is used under Protected Mode. It is accessible when the CPU is in Real Mode or VMZ# is set. However, operation by software is not normally required. The initial value right after reset is undefined.

CR1Ah(TX) special

This register is used under Protected Mode. It is accessible when the CPU is in Real Mode or VMZ# is set. However, operation by software is not normally required. The initial value right after reset is undefined.

CR1Eh(TSEL) special

This register is used under Protected Mode. It is accessible when the CPU is in Real Mode or VMZ# is set. However, operation by software is not normally required. The initial value right after reset is undefined.

CR1Fh(TERR) special

This register is used under Protected Mode. It is accessible when the CPU is in Real Mode or VMZ# is set. However, operation by software is not normally required. The initial value right after reset is undefined.

A.6.4 Paging Unit Control Registers

CR06h(PDER) special

The page directory entry register (PDER) is loaded with the value of a page directory entry at the time of a hit or miss by the paging unit. It is accessible when the Real Mode instructions are available. However, operation by software is not normally required. The initial value right after reset is undefined.

CR07h(PTER) special

The page table entry register (PTER) is loaded with the value of a page directory entry at the time of a hit or miss by the paging unit. It is accessible when the Real Mode instructions are available. However, operation by software is not normally required. The initial value right after reset is undefined.

CR08h(PFAR) CR2

PFAR is compatible with 386 CR2 and accessible by either Real or Protected Mode instructions. A linear address which is generated a page fault is transferred from the page fault address buffer and stored in this register. The initial value right after reset is undefined.

CR0Bh(PECR) special

PECR is provided for paged virtual memory under Real Mode. Its lowest 3 bits are loaded with the error code for a page fault. However, the error code is pushed onto the stack under Protected Mode. The initial value right after reset is undefined. Error codes to be set with the lowest 3 bits conform to those under Protected Mode.

CR0Ch(PDBR) CR3

PDBR is compatible with 386 CR3 and accessible by either Real or Protected Mode instructions. The initial value right after reset is undefined. The base address of a page directory is 32-bit and can be read/written.

CR19h(LAR) TR6

LAR is compatible with 386 CR6 and accessible by either Real or Protected Mode instructions. The initial value right after reset is undefined.

bit 31-12: Linear Address

On a TLB write: To be written to the TLB
On a TLB lookup: To be looked up in the TLB

bit 11: V

Valid Bit

bit 10 - 9: D, D#

Dirty bit and its complement

bit 8 - 7: U, U#

U/S bit and its complement

bit 8 - 7: U, U#

U/S bit and its complement

bit 6 - 5: W, W#

R/W bit and its complement

bit 0: C

Command Bit

C = 1: TLB write
C = 0: TLB lookup

Ver. 0.93, Document Number: 6117DSC1.doc
Acer Laboratories, Inc., USA, 1830B Bering Drive, San Jose, CA 95112; Tel: 408-467-7456; Fax: 408-467-7474
www.acerlabs.com

CR1Dh(PAR) TR7

PAR is compatible with 386 TR7 and accessible by either Real or Protected Mode instructions. The initial value right after reset is undefined.

bit 31-12: Physical Address
On a TLB write: To be written to the TLB
On a TLB lookup: Value from the TLB

bit 4: PL
On a TLB write:
PL = 0: Write to the internal pointer
PL = 1: Write to the REP pointer
On a TLB lookup:
PL = 0: Miss
PL = 1: Hit

bit 3: REP1
For a TLB write, selects which block is to be written.

bit 2: REP0
For a TLB lookup, selects in which block the look-up entry was found.

A.6.5 SiEMU-II Unit Control Registers

CR30h(SPA0) DR0

SPA0 is compatible with 386 DR0 and holds the address for Breakpoint 0. The initial value right after reset is undefined.

CR34h(SPA1) DR1

SPA1 is compatible with 386 DR1 and holds the address for Breakpoint 1. The initial value right after reset is undefined.

CR38h(SPA2) DR2

SPA2 is compatible with 386 DR2 and holds the address for Breakpoint 2. The initial value right after reset is undefined.

CR39h(SBRK) DR6

SBRK is compatible with 386 DR6 and to be set when any break is detected. The initial value right after reset is 00000000h.

bit 15: BT
Task switch break detect

bit 14: BS
Single step break detect

bit 13: BD
Register protection violation break detect

bit 3: B3
Breakpoint 3 break detect

bit 2: B2
Breakpoint 2 break detect

bit 1: B1
Breakpoint 1 break detect

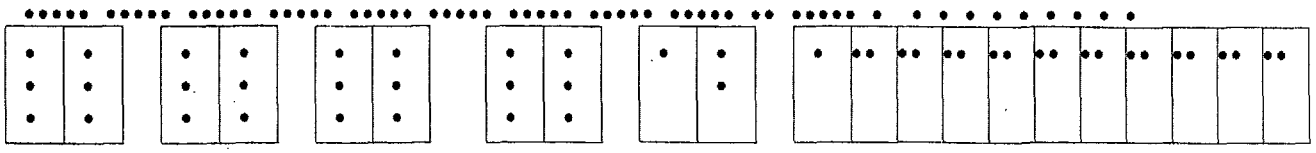
bit 0: B0
Breakpoint 0 break detect

CR3Ch(SPA3) DR3

SPA0 is compatible with 386 DR3 and holds the address for Breakpoint 3. The initial value right after reset is undefined.

CR3Dh(SCON) DR7

SCON is compatible with 386 DR7 and controls breakpoints 3 to 1. The initial value right after reset is 00000000h.



bit 31, 30, 27, 26, 23, 22, 19, 18: LENx

Breakpoint location size

•••	
0 0	Byte
0 1	Word
1 0	Double • undefined in the 386
1 1	Double

bit 29, 28, 25, 24, 21, 20, 17, 16: R/Wx

Break condition

•••	
0 0	Instruction Execution Only
0 1	Data Write
1 0	No Break undefined in the 386
1 1	Data read / write

bit 13: GD

Debug Register Protection Condition Enable (To be cleared by the processor)

bit 9: GE

Fault Mode Enable (Global)

bit 8: LE

Fault Mode Enable (To be cleared by the local or task switch)

bit 7, 5, 3, 1: Gx

Address Break x Enable (Global)

bit 6, 4, 2, 0: Lx

Address Break x Enable (To be cleared by the local or task switch)

Acer Laboratories, In 'SA, 1830B Bering Drive, San Jose, CA 95112; Tel: 408-467-7456; Fax: 408-467-7474
www.acerlabs.com

APPENDIX B. Special instruction format

The following instruction set is useful for processing SMI routine if the original operating system is in protected mode. If we use IRQ15 to implement power management mode or the original operating system is in real mode, we do not need to do extra work.

B.1 Move Data

This function is provided to move data between register and register/memory. It is fully compatible with 386 instructions. Also, it includes the following instructions which are more powerful than 386 instructions are.

General register to General register/memory

11010110	01001000	greg bm1	bm2 greg/im disp W

greg : 5 bit, from 00000 to 11111, GR0 to GR31

bm1	greg
000	GB#(Use GR# as Byte Register, bit 0 to 7)
001	GL#(Use GR# as Word Register, bit 0 to 15)
010	GH#(Use GR# as Word Register, bit 16 to 31)
011	GR#(Use GR# as Double Word Register, bit 0 to 31)
100	GB#, bm2 gr/im is not required, next two bytes is disp W
101	GL#, bm2 gr/im is not required, next two bytes is disp W
110	GH#, bm2 gr/im is not required, next two bytes is disp W
111	GR#, bm2 gr/im is not required, next two bytes is disp W

bm2	greg/im
000	GB#
001	GL#
010	GH#
011	GR#
100	post-increment*
101	pre-decrement*
110	5-bit short immediate with zero extend
111	5-bit short immediate with sign extend

*: If GR4 or GR5 is selected as GBASE, the stack segment is selected in default. It can be changed by a segment override prefix or postfix. In other memory access, EDS segment register will be selected in default.

Control register to General register

11010110	11111000	greg 011	creg

creg : 8 bit, from 00000000 to 11111111, CR00h to CRFFh

General register to Control register

11010110	11111010	greg 011	creg

Base register to General register

11010110	11011000	greg 011	sreg 00000

sreg : 3 bit, from 000 to 111, SR0 to SR7

General register to Base register

11010110	11011010	greg 011	sreg 00000

Attribute register to General register

11010110 11011001 greg 011 areg 00000

areg : 3 bit, from 000 to 111, AR0 to AR7

General register to Attribute register
11010110 11011011 greg 011 areg 00000

User Base register to General register
11010110 11001000 greg 011 usreg 00000

usreg : 3 bit, 100 eusp, 010 SR2u, 111 SR7u

General register to User Base register
11010110 11001010 greg 011 usreg 00000

User Attribute register to General register
11010110 11001001 greg 011 uareg 00000

uareg : 3 bit, 010 AR2u, 111 AR7u

General register to User Attribute register
11010110 11001011 greg 011 uareg 00000

B.2 Push/Pop Data

This function is provided to push/pop data into/from stack. It is fully compatible with 386 instructions. If we pop segment register (DS, ES, and so on), the base and attribute register may be changed (please refer to Intel'• instruction set). Also, it includes the following instructions which are more powerful than 386 instructions are.

push General register
11010110 10110100 greg bm1

pop General register
11010110 10111100 greg bm1

push segment register
11010110 11110100 sreg 00000

pop segment register
11010110 11111100 sreg 00000

push all General register GR0 GR31
11010110 11100100

pop all General register GR0GR31
11010110 11101100

APPENDIX C. 82C54 behavior limitation

The 82C54 has the capability of read channel counter value. When the counter is programmed as 16-bit, the programmer has a way to obtain channel counter value by reading the channel twice. It happens in very rare situations. The phenomenon below.

When a 16-bit counter value reads counter twice, which obtains low counter value followed by high counter value. Sometimes the counter happen at the time frame near carry happened between high/low byte. The counter value may be wrong on the high byte.

Suggestion :

Case 1. Customers calculate time distance by reading 82C54 counter value, and have full control of the programming code. The suggested method is to change ISA AT clock to 7.159MHz temporarily during reading 82C54 counter value.

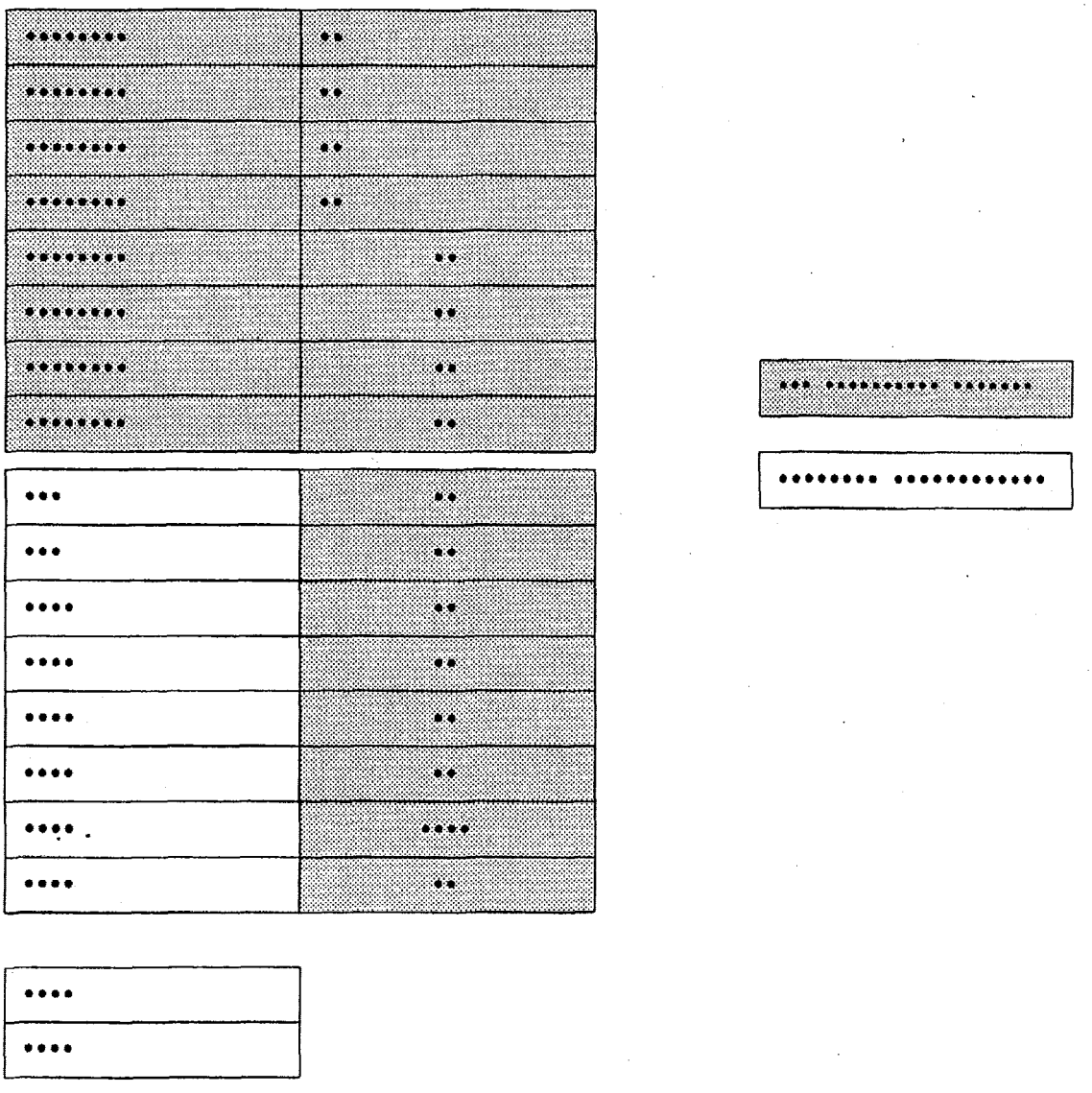
Case 2. Customers have no control of programming code related to access 82C54 counter value. E.g. Landmark and Windows' media player, which will access 82C54 counter in order to calculate timing, will cause this problem. The suggestion is to fix ISA AT clock as 7.159MHz.

APPENDIX A. M6117C's CPU Registers

The CPU registers are compatible with 80386 registers. They also provide more powerful operations on register programming.

A.1 General Purpose Registers (GR)

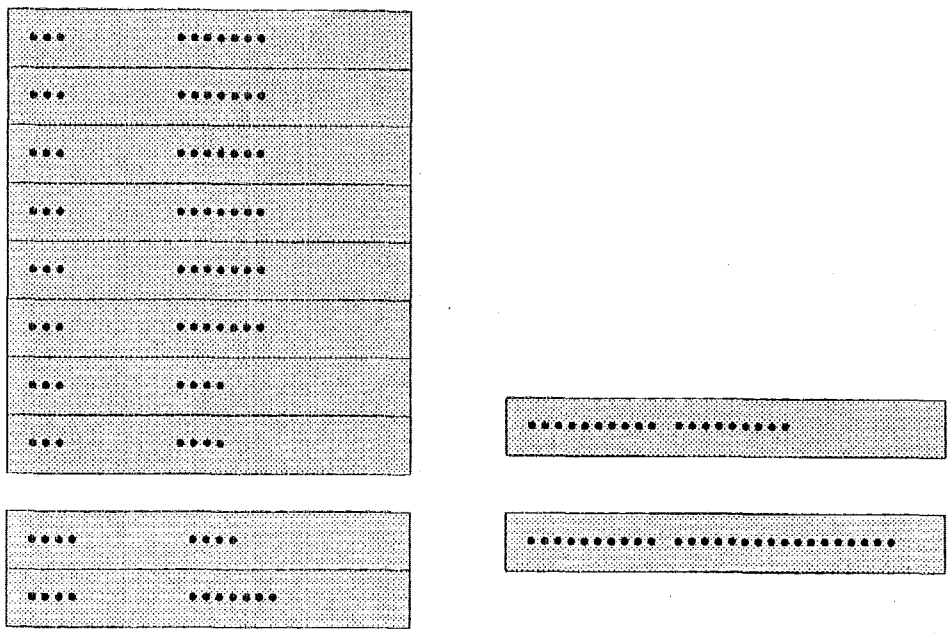
The CPU incorporates 32 general purpose 32-bit registers. In arithmetic and logic operations, these registers can be used as operands. The registers then are used as data registers. In address calculations, the registers are used as base registers or index registers.



Acer Laboratories, Inc., USA, 1830B Bering Drive, San Jose, CA 95112; Tel: 408-467-7456; Fax: 408-467-7474
www.acerlabs.com

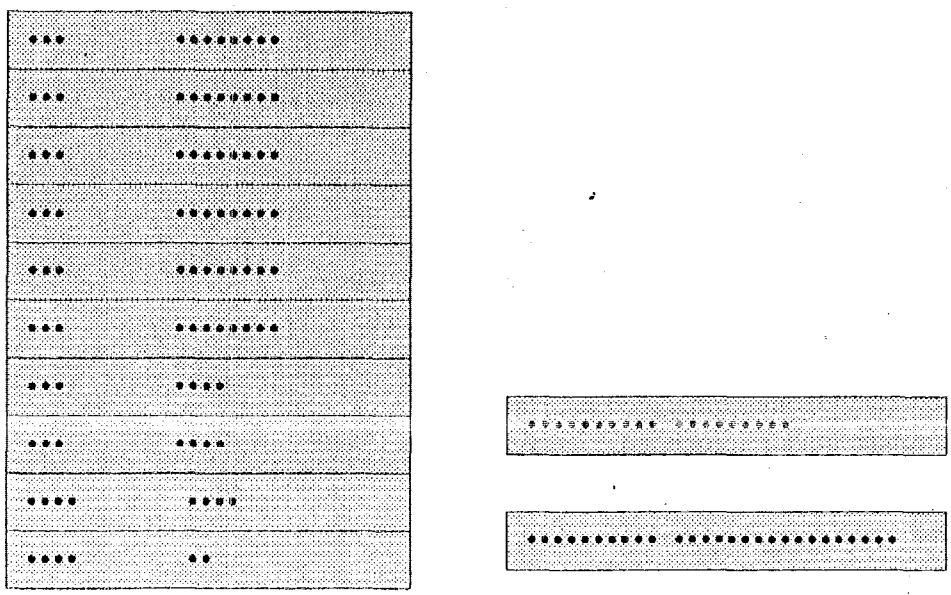
A.2 Base Registers (SR)

The CPU has 10 32-bit segment base registers to locate a block of program or data in the memory area. When the 80386 object code is used, these registers operate automatically. They require no operator intervention.



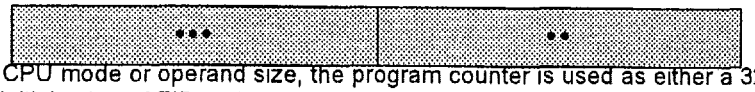
A.3 Attribute Registers (AR)

The CPU has 10 32-bit segment attribute registers. Each attribute register holds an attribute of the corresponding segment register. When the 80386 object code is used, these registers operate automatically. They require no operator intervention.



A.4 Instruction Pointer

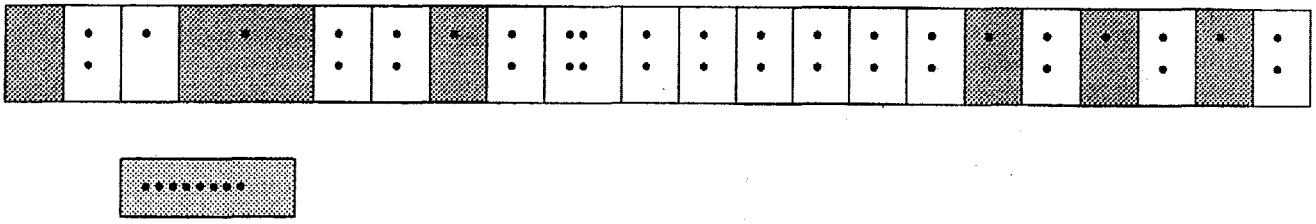
The instruction pointer (EIP) register contains the offset in the current code segment for the next instruction to execute. It is controlled implicitly by control-transfer instructions (jumps, returns, etc.), interrupts; and exceptions.



According to the CPU mode or operand size, the program counter is used as either a 32-bit register EIP, or a 16-bit register IP. The initial value of EIP register is 0000FFF0h.

A.5 Flags Register

The CPU has flags to indicate current modes and operation results, EFLAGS. The initial value right after reset is 00000002h.



BIT 30 EA : Extended Address Generation Flag
This bit controls paging.

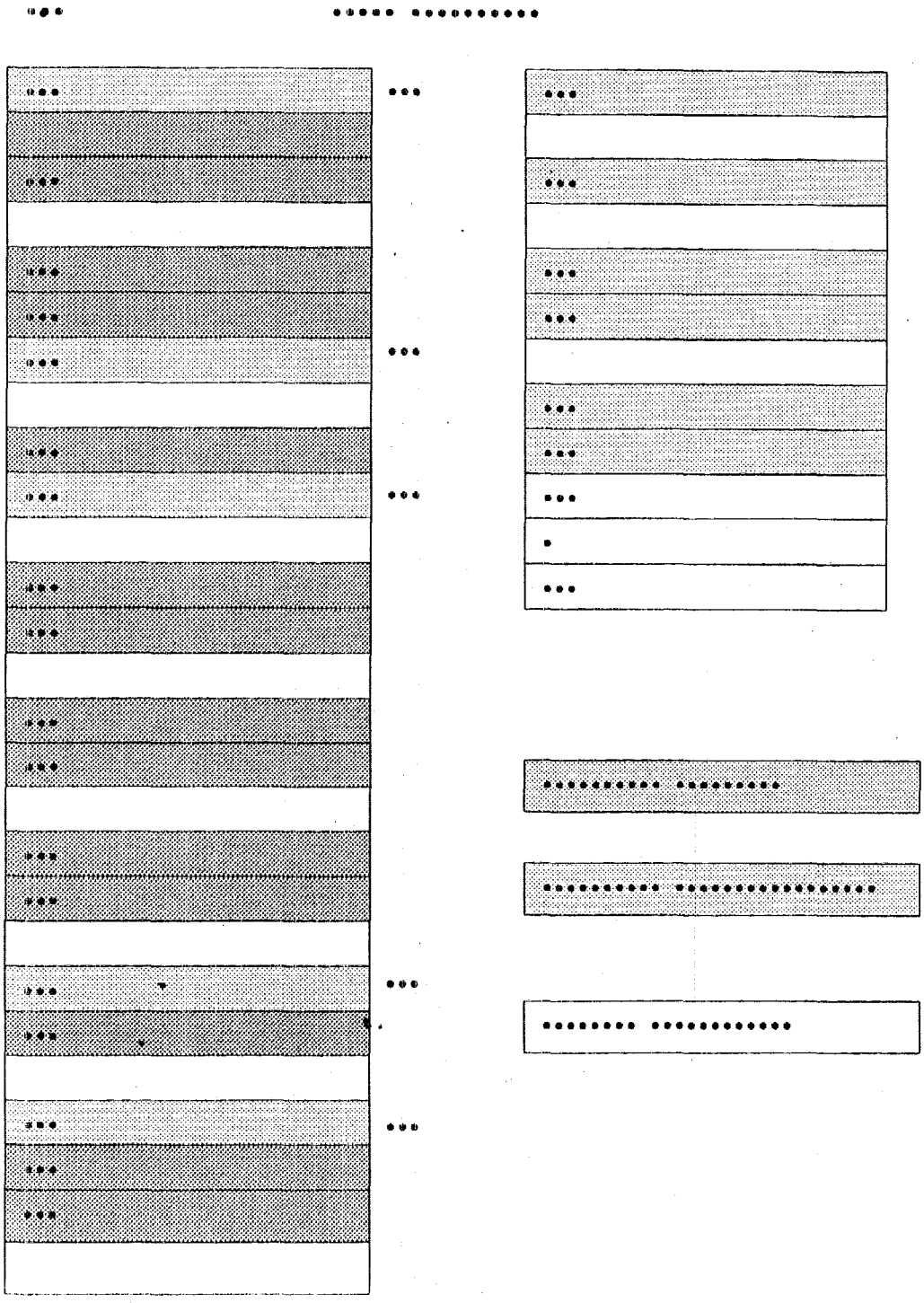
BIT 29 D : Default Flag
Cleared by external reset signal. Set/Clear by special OP codes :
BRDD/BRDW instructions (Change Context to 32/16 bit and jump).

D-Flag	0	0	0	0	1	1	1	1
Operand Size Prefix	N	Y	N	Y	N	Y	N	Y
Address Size Prefix	N	N	Y	Y	N	N	Y	Y
Operand Size in Selection	8/16	8/32	8/16	8/32	8/32	8/16	8/32	8/16
Address Size in Selection	16	16	32	32	32	32	16	16

Acer Laboratories, Inc., USA, 1830B Bering Drive, San Jose, CA 95112; Tel: 408-467-7456; Fax: 408-467-7474
www.acerlabs.com

A.6 Control Registers(CR)

The CPU incorporates up to 256 32-bit control registers. Some registers is assigned to the 80386-compatible functions and can be accessed by 80386-compatible instructions.



These registers can be divided into 5 groups :

1. Processor Control Registers,
2. Instruction Decoding Unit Control Registers,
3. Protected-Mode Instruction Unit Control Registers,

4. Paging Unit Control Registers, 5. SiEMU-II Unit Control Registers.

Acer Laboratories, Inc., USA, 1830B Bering Drive, San Jose, CA 95112; Tel: 408-467-7456; Fax: 408-467-7474
www.acerlabs.com

A.6.1 Processor Control Registers

CR00h symbol 190 \f "Symbol" \s 9—} CR0

The CR00h contains those bits to control an interface of numeric coprocessor and execution modes. The initial value right after reset is "7FFFFFD0h." The meaning of every bit except bit 5 IOB is same as normal definition.

bit 5 IOB

(A) 16-bit I/O bus mode(IOB = 0, default setting)

Byte operation at even address. (DBHE# = 1, DBLE# ◘ 0)

DATA BUS

Byte operation at odd address. (DBHE# = 0, DBLE# = 1)

DATA BUS

Word operation at even address. (DBHE# = 0, DBLE# = 0)

DATA BUS

Word operation at odd address.

DATA BUS

(B) 8 bit I/O bus mode(IOB = 1)

Byte operation at even address. (DBHE# = 1, DBLE# = 0)

DATA BUS

Byte operation at odd address. (DBHE# = 0, DBLE# = 1)

DATA BUS

Word operation at even address. (DBHE# = 0, DBLE# = 0)

DATA BUS

Word operation at odd address.

DATA BUS

DATA BUS

CR02h(EPWRCR) symbol 190 \f "Symbol" \s 10—} special

This register controls Power Save Mode. The initial value right after reset is FFFF00FFh.

bit 15 PCSTP

When set, the CPU enters Power Save Mode and main clock stops functioning after HALT instruction.

If HLDREQ input becomes active in the Power Save Mode. CPU enters a hold state. When HLDREQ becomes inactive. CPU returns to the Power Save Mode.

Main clock starts again when NMI, INTREQ, or RESET becomes active. When clear, HALT instruction is executed normally. PCSTP is cleared after external reset signal.

CR03h(PMON) symbol 190 \f "Symbol" \s 9—} special

This register controls processor monitor function. The initial value right after reset is EAAFFFFFh.

bit 28 PMM: Protected Mode Instruction Monitor

When set, the following instructions cause an operation equivalent to the BRKPM interrupt:

PUSHF, POPF, CLI, STI, IRET, LGDT, SGDT, LIDT, SIDT, LLDT, SLDT,
LTR, STR, LMSW, SMSW, ARPL, CLRTS, LDAC, LDLM, TRDE, TWRE,
move to/from CRx, DRx, TRx(CR00h, 08h, 0Ch, 19h, 1Dh, 30h, 34h, 38h,
39h, 3Ch, 3Dh) from/to GRx.

This allows the software to emulate the instructions used for a protected-mode OS, and a protected-mode OS can run as a guest OS under the virtual machine OS. The Protected Mode Instruction Monitor can then be used for debugging the protected-mode OS or BIOS. It can also be used for processor testing.

bit 26 VMZM#: VMZ Instruction Monitor

If set at '0', when a special instruction (move immediate to CS, etc. which is not compatible with 386 instruction) is attempted in protected mode, an invalid operand exception (exception 6) occurs. If set at '1', such instructions become valid in protected mode. After reset, this bit is set at '0' to make possible a protection function. The bit 31 - 29 of EFLAGS can be written only when VMZM# is '1'.

bit 24 EXPM: Exception Monitor

If set, when either a hardware or software exception is detected in protected mode, an execution similar to BRKPM occurs.

bit 22 TM: Test Mode

If set, when an exception occurs, CPU stores the exception number in GR26 and jumps indirectly using GR27 as a new IP.

bit 20 DPAC: Cycle Steal Address Calculation Disable

When set, cycle steal address calculation is disabled and when clear, it is enabled. Cycle steal address calculation is an address operand calculation of the next instruction using the unused time slot in internal buses for pre-fetching.

bit 18 IOM: I/O Instruction Monitor

When set, the instructions below cause an operation equivalent to the BRKPM interrupt. This forces the CPU to enter Real Mode, regardless of the mode that it was in.

IN, OUT, INS, OUTS

A.6.2 Instruction Decoding Unit Control Registers

CR0Eh(TOP) symbol 190 \f "Symbol" \s 10—} special

The register controls the 'opcode trap function', together with ***CR0Fh***. It is operated only through Real Mode instructions. The initial value right after reset is undefined. Each field of 'OPCODE 3' to 'OPCODE 0' stores an opcode to be trapped. The relevant opcode will cause the same operation as BRKPM.

CR0Fh(TCON) symbol 190 \f "Symbol" \s 10—} special

The register controls the 'opcode trap function', together with ***CR0Eh***. It is operated only through Real Mode instructions. The initial value right after reset is 00000000h. Each field of 'OPCODE 3' to 'OPCODE 0' stores an opcode to be trapped. The relevant opcode will cause the same operation as BRKPM.

bit 16 TCE: Opcode Trap Enable

When set, the opcode trap is enable. Clears when an opcode trap occurs.

bit 15 - 8: Trap Opcode Mask

When set, masks compares to the bits set for OPCODE BREAK 0.

bit 7 - 0 : Map 3 - 0

00 : No trap
01 : 1st opcode
10 : 2nd opcode following 00Fh
11 : 2nd opcode following 0D6h

Acer Laboratories, Inc., USA, 1830B Bering Drive, San Jose, CA 95112; Tel: 408-467-7456; Fax: 408-467-7474
www.acerlabs.com

A.6.3 Protected-Mode Instruction Unit Control Registers

CR12h(TDESH) symbol 190 \f "Symbol" \s 10—} special

This register is used under Protected Mode. It is accessible when the CPU is in Real Mode or VMZ# is set. However, operation by software is not normally required. The initial value right after reset is undefined.

CR13h(TDESL) symbol 190 \f "Symbol" \s 10—} special

This register is used under Protected Mode. It is accessible when the CPU is in Real Mode or VMZ# is set. However, operation by software is not normally required. The initial value right after reset is undefined.

CR16h(TZ) symbol 190 \f "Symbol" \s 10—} special

This register is used under Protected Mode. It is accessible when the CPU is in Real Mode or VMZ# is set. However, operation by software is not normally required. The initial value right after reset is undefined.

CR17h(TY) symbol 190 \f "Symbol" \s 10—} special

This register is used under Protected Mode. It is accessible when the CPU is in Real Mode or VMZ# is set. However, operation by software is not normally required. The initial value right after reset is undefined.

CR1Ah(TX) symbol 190 \f "Symbol" \s 10—} special

This register is used under Protected Mode. It is accessible when the CPU is in Real Mode or VMZ# is set. However, operation by software is not normally required. The initial value right after reset is undefined.

CR1Eh(TSEL) symbol 190 \f "Symbol" \s 10—} special

This register is used under Protected Mode. It is accessible when the CPU is in Real Mode or VMZ# is set. However, operation by software is not normally required. The initial value right after reset is undefined.

CR1Fh(TERR) symbol 190 \f "Symbol" \s 10—} special

This register is used under Protected Mode. It is accessible when the CPU is in Real Mode or VMZ# is set. However, operation by software is not normally required. The initial value right after reset is undefined.

A.6.4 Paging Unit Control Registers

CR06h(PDER) symbol 190 \f "Symbol" \s 10—} special

The page directory entry register (PDER) is loaded with the value of a page directory entry at the time of a hit or miss by the paging unit. It is accessible when the Real Mode instructions are available. However, operation by software is not normally required. The initial value right after reset is undefined.

CR07h(PTER) symbol 190 \f "Symbol" \s 10—} special

The page table entry register (PTER) is loaded with the value of a page directory entry at the time of a hit or miss by the paging unit. It is accessible when the Real Mode instructions are available. However, operation by software is not normally required. The initial value right after reset is undefined.

CR08h(PFAR) symbol 190 \f "Symbol" \s 10—} CR2

PFAR is compatible with 386 CR2 and accessible by either Real or Protected Mode instructions. A linear address which is generated a page fault is transferred from the page fault address buffer and stored in this register. The initial value right after reset is undefined.

CR0Bh(PECR) symbol 190 \f "Symbol" \s 10—} special

PECR is provided for paged virtual memory under Real Mode. Its lowest 3 bits are loaded with the error code for a page fault. However, the error code is pushed onto the stack under Protected Mode. The initial value right after reset is undefined. Error codes to be set with the lowest 3 bits conform to those under Protected Mode.

CR0Ch(PDBR) symbol 190 \f "Symbol" \s 10—} CR3

PDBR is compatible with 386 CR3 and accessible by either Real or Protected Mode instructions. The initial value right after reset is undefined. The base address of a page directory is 32-bit and can be read/written.

CR19h(LAR) symbol 190 \f "Symbol" \s 10—} TR6

LAR is compatible with 386 CR6 and accessible by either Real or Protected Mode instructions. The initial value right after reset is undefined.

bit 31-12: Linear Address

On a TLB write: To be written to the TLB

On a TLB lookup: To be looked up in the TLB

bit 11: V

Valid Bit

bit 10 - 9: D, D#

Dirty bit and its complement

bit 8 - 7: U, U#

U/S bit and its complement

bit 8 - 7: U, U#

U/S bit and its complement

bit 6 - 5: W, W#

R/W bit and its complement

bit 0: C

Command Bit

C = 1: TLB write

C = 0: TLB lookup

Acer Laboratories, Inc., USA, 1830B Bering Drive, San Jose, CA 95112; Tel: 408-467-7456; Fax: 408-467-7474

www.acerlabs.com

CR1Dh(PAR) TR7

PAR is compatible with 386 TR7 and accessible by either Real or Protected Mode instructions. The initial value right after reset is undefined.

bit 31-12: Physical Address
On a TLB write: To be written to the TLB
On a TLB lookup: Value from the TLB

bit 4: PL
On a TLB write:
PL = 0: Write to the internal pointer
PL = 1: Write to the REP pointer
On a TLB lookup:
PL = 0: Miss
PL = 1: Hit

bit 3: REP1
For a TLB write, selects which block is to be written.

bit 2: REP0
For a TLB lookup, selects in which block the look-up entry was found.

A.6.5 SiEMU-II Unit Control Registers

CR30h(SPA0) DR0

SPA0 is compatible with 386 DR0 and holds the address for Breakpoint 0. The initial value right after reset is undefined.

CR34h(SPA1) DR1

SPA1 is compatible with 386 DR1 and holds the address for Breakpoint 1. The initial value right after reset is undefined.

CR38h(SPA2) DR2

SPA2 is compatible with 386 DR2 and holds the address for Breakpoint 2. The initial value right after reset is undefined.

CR39h(SBRK) DR6

SBRK is compatible with 386 DR6 and to be set when any break is detected. The initial value right after reset is 00000000h.

bit 15: BT
Task switch break detect

bit 14: BS
Single step break detect

bit 13: BD
Register protection violation break detect

bit 3: B3
Breakpoint 3 break detect

bit 2: B2
Breakpoint 2 break detect

bit 1: B1
Breakpoint 1 break detect

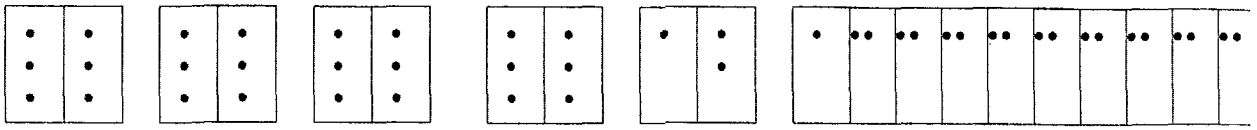
bit 0: B0
Breakpoint 0 break detect

CR3Ch(SPA3) DR3

SPA0 is compatible with 386 DR3 and holds the address for Breakpoint 3. The initial value right after reset is undefined.

CR3Dh(SCON) DR7

SCON is compatible with 386 DR7 and controls breakpoints 3 to 1. The initial value right after reset is 0(



bit 31, 30, 27, 26, 23, 22, 19, 18: LENx

Breakpoint location size

0 0	Byte
0 1	Word
1 0	Double • undefined in the 386
1 1	Double

bit 29, 28, 25, 24, 21, 20, 17, 16: R/Wx.

Break condition

0 0	Instruction Execution Only
0 1	Data Write
1 0	No Break undefined in the 386
1 1	Data read / write

bit 13: GD

Debug Register Protection Condition Enable (To be cleared by the processor)

bit 9: GE

Fault Mode Enable (Global)

bit 8: LE

Fault Mode Enable (To be cleared by the local or task switch)

bit 7, 5, 3, 1: Gx

Address Break x Enable (Global)

bit 6, 4, 2, 0: Lx

Address Break x Enable (To be cleared by the local or task switch)

Ver. 0.93, Document Number: 6117DSC1.doc
Acer Laboratories, Inc., USA, 1830B Bering Drive, San Jose, CA 95112; Tel: 408-467-7456; Fax: 408-467-7474
www.acerlabs.com

APPENDIX B. Special instruction format

The following instruction set is useful for processing SMI routine if the original operating system is in protected mode. If we use IRQ15 to implement power management mode or the original operating system is in real mode, we do not need to do extra work.

B.1 Move Data

This function is provided to move data between register and register/memory. It is fully compatible with 386 instructions. Also, it includes the following instructions which are more powerful than 386 instructions are.

General register to General register/memory

11010110	01001000	greg bm1	bm2 greg/im disp W

greg : 5 bit, from 00000 to 11111, GR0 to GR31

bm1	greg
000	GB#(Use GR# as Byte Register, bit 0 to 7)
001	GL#(Use GR# as Word Register, bit 0 to 15)
010	GH#(Use GR# as Word Register, bit 16 to 31)
011	GR#(Use GR# as Double Word Register, bit 0 to 31)
100	GB#, bm2 gr/im is not required, next two bytes is disp W
101	GL#, bm2 gr/im is not required, next two bytes is disp W
110	GH#, bm2 gr/im is not required, next two bytes is disp W
111	GR#, bm2 gr/im is not required, next two bytes is disp W

bm2	greg/im
000	GB#
001	GL#
010	GH#
011	GR#
100	post-increment*
101	pre-decrement*
110	5-bit short immediate with zero extend
111	5-bit short immediate with sign extend

*: If GR4 or GR5 is selected as GBASE, the stack segment is selected in default. It can be changed by a segment override prefix or postfix. In other memory access, EDS segment register will be selected in default.

Control register to General register

11010110	11111000	greg 011	creg

creg : 8 bit, from 00000000 to 11111111, CR00h to CRFFh

General register to Control register

11010110	11111010	greg 011	creg

Base register to General register

11010110	11011000	greg 011	sreg 00000

sreg : 3 bit, from 000 to 111, SR0 to SR7

General register to Base register

11010110	11011010	greg 011	sreg 00000

Attribute register to General register

11010110 11011001 greg 011 areg 00000

areg : 3 bit, from 000 to 111, AR0 to AR7

General register to Attribute register
11010110 11011011 greg 011 areg 00000

User Base register to General register
11010110 11001000 greg 011 usreg 00000

usreg : 3 bit, 100 eusp, 010 SR2u, 111 SR7u

General register to User Base register
11010110 11001010 greg 011 usreg 00000

User Attribute register to General register
11010110 11001001 greg 011 uareg 00000

uareg : 3 bit, 010 AR2u, 111 AR7u

General register to User Attribute register
11010110 11001011 greg 011 uareg 00000

B.2 Push/Pop Data

This function is provided to push/pop data into/from stack. It is fully compatible with 386 instructions. If we pop segment register (DS, ES, and so on), the base and attribute register may be changed (please refer to Intel's instruction set). Also, it includes the following instructions which are more powerful than 386 instructions are.

push General register
11010110 10110100 greg bm1

pop General register
11010110 10111100 greg bm1

push segment register
11010110 11110100 sreg 00000

pop segment register
11010110 11111100 sreg 00000

push all General register GR0 GR31
11010110 11100100

pop all General register GR0GR31
11010110 11101100

Acer Laboratories, Inc., USA, 1830B Bering Drive, San Jose, CA 95112; Tel: 408-467-7456; Fax: 408-467-7474
www.acerlabs.com

APPENDIX C. 82C54 behavior limitation

The 82C54 has the capability of read channel counter value. When the counter is programmed as 16-bit, the programmer has a way to obtain channel counter value by reading the channel twice. It happens in very rare situations. The phenomenon below.

When a 16-bit counter value reads counter twice, which obtains low counter value followed by high counter value. Sometimes the counter happen at the time frame near carry happened between high/low byte. The counter value may be wrong on the high byte.

Suggestion :

Case 1. Customers calculate time distance by reading 82C54 counter value, and have full control of the programming code. The suggested method is to change ISA AT clock to 7.159MHz temporarily during reading 82C54 counter value.

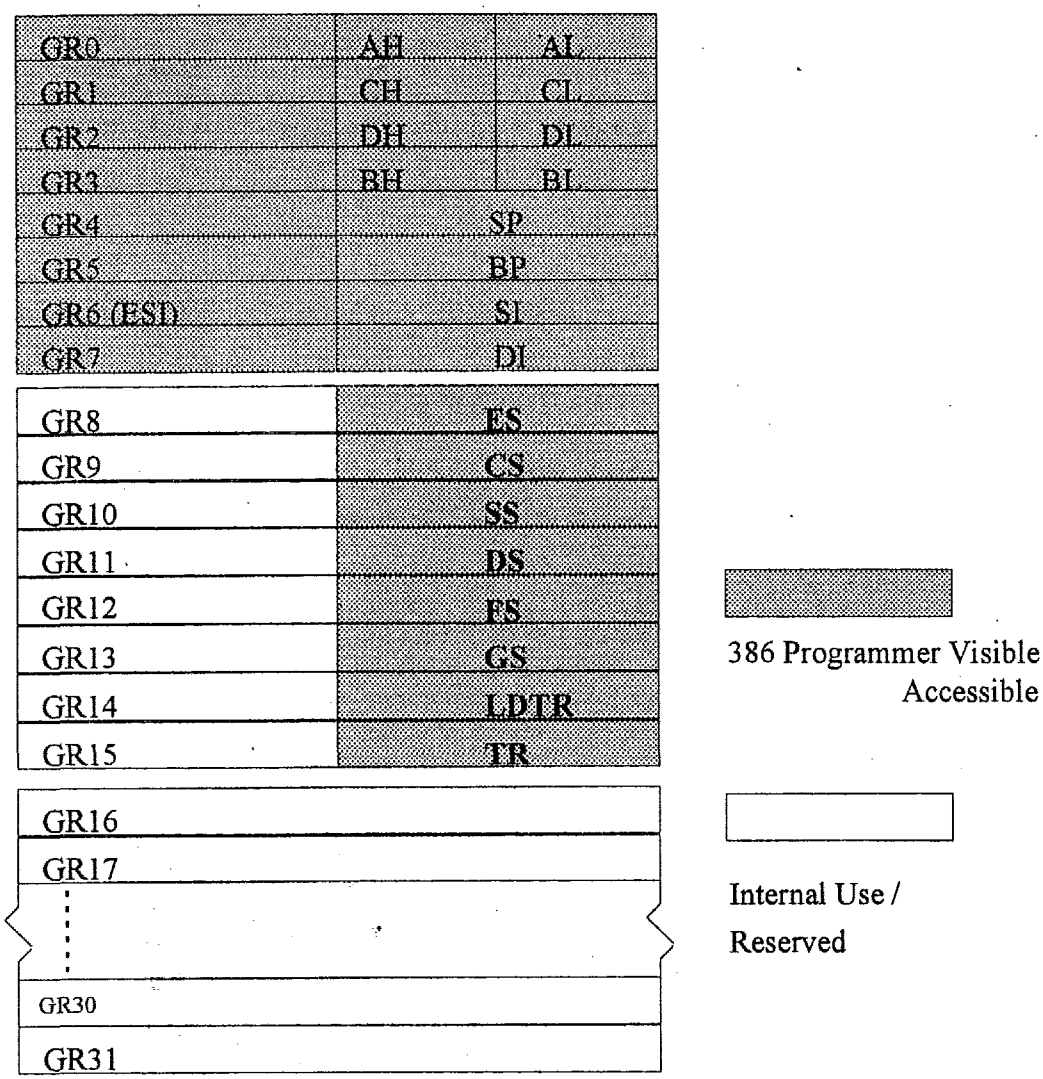
Case 2. Customers have no control of programming code related to access 82C54 counter value. E.g. Landmark and Windows' media player, which will access 82C54 counter in order to calculate timing, will cause this problem. The suggestion is to fix ISA AT clock as 7.159MHz.

APPENDIX A. M6117C's CPU Registers

The CPU registers are compatible with 80386 registers. They also provide more powerful operation on register programming.

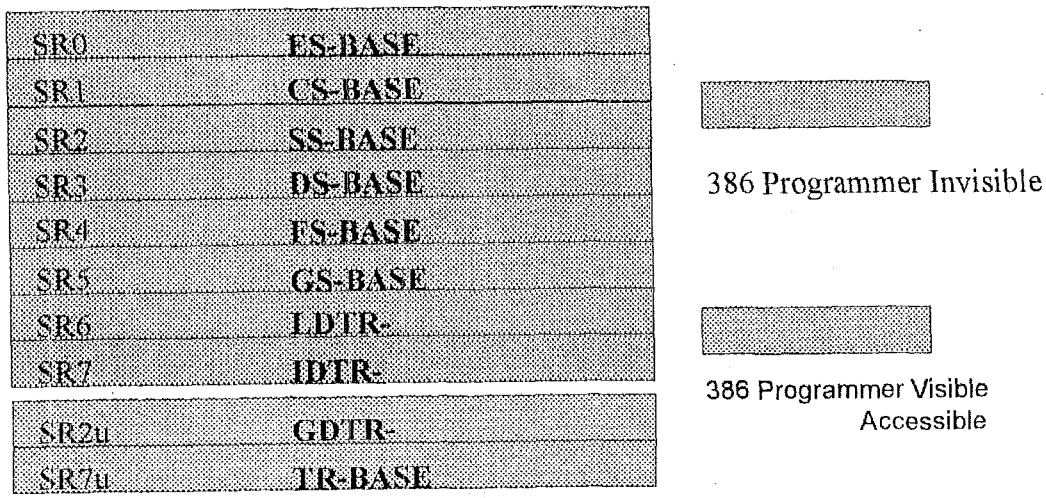
A.1 General Purpose Registers(GR)

The CPU incorporates 32 general purpose 32-bit registers. In arithmetic and logic operations, these registers can be used as operands. The registers then are used as the data registers. In address calculations, the registers are used as the base registers or index registers.



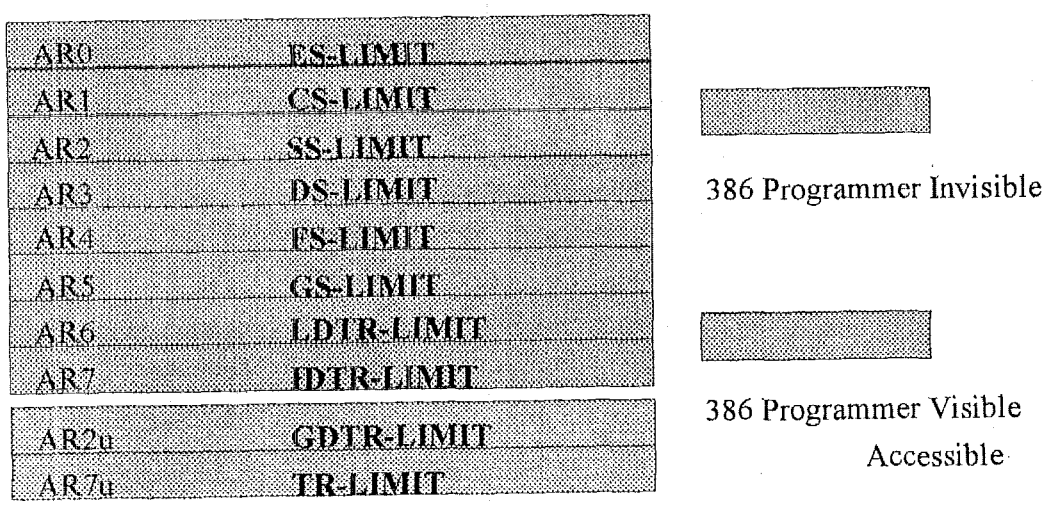
A.2 Base Registers(SR)

The CPU has 10 32-bit segment base registers to locate a block of program or data in the memory area. When the 80386 object code is used, these registers operate automatically. They require no operator intervention.



A.3 Attribute Registers (AR)

The CPU has 10 32-bit segment attribute registers. Each attribute register holds an attribute of the corresponding segment register. When the 80386 object code is used, these registers operate automatically. They require no operator intervention.



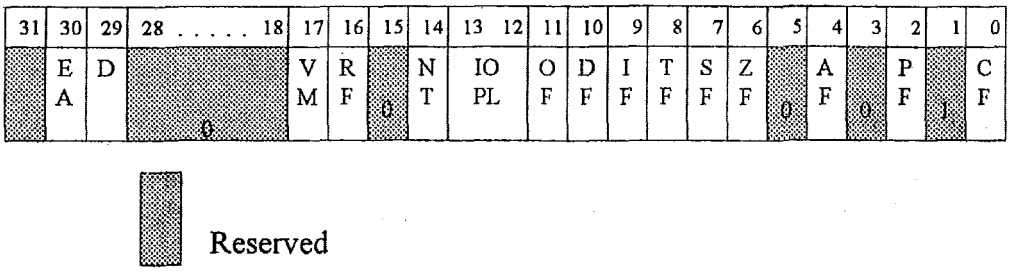
A.4 Instruction Pointer

The instruction pointer (EIP) register contains the offset in the current code segment for the next instruction to execute. It is controlled implicitly by control-transfer instructions (jumps, returns, etc.), interrupts, and exceptions.

According to the CPU mode or operand size, the program counter is used as either a 32-bit register — EIP, or a 16-bit register — IP. The initial value of EIP register is 0000FFF0h.

A.5 Flags Register

The CPU has flags to indicate current modes and operation results, EFLAGS. The initial value right after reset is 00000002h.



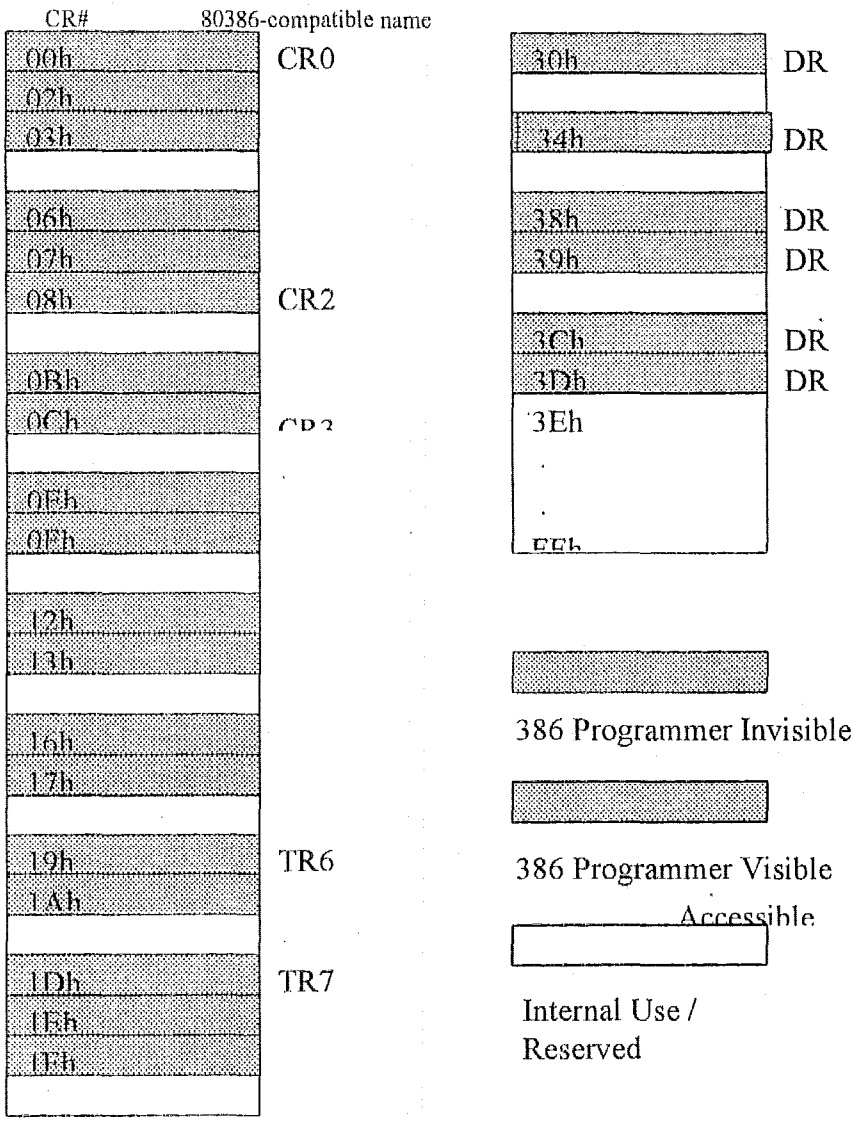
BIT 30 EA: Extended Address Generation Flag
This bit controls paging.

BIT 29 D: Default Flag
Cleared by external reset signal. Set/Clear by special OP codes:
BRDD/BRDW instructions (Change Context to 32/16 bit and jmp).

D-Flag	0	0	0	0	1	1	1	1
Operand Size Prefix	N	Y	N	Y	N	Y	N	Y
Address Size Prefix	N	N	Y	Y	N	N	Y	Y
Operand Size in Selection	8/16	8/32	8/16	8/32	8/32	8/16	8/32	8/16
Address Size in Selection	16	16	32	32	32	32	16	16

A.6 Control Registers (CR)

The CPU incorporates up to 256 32-bit control registers. Some registers is assigned to the 80386-compatible functions and can be accessed by 80386-compatible instructions.



These registers can be divided into 5 groups:

1. Processor Control Registers,
2. Instruction Decoding Unit Control Registers,
3. Protected-Mode Instruction Unit Control Registers,
4. Paging Unit Control Registers,
5. SiEMU-II Unit Control Registers.

A.6.1 Processor Control Registers

CR00h — CR0

The CR00h contains those bits to control an interface of numeric coprocessor and execution modes. The initial value right after reset is "7FFFFFD0h." The meaning of every bits except bit 5 IOB is same as normal definition.

bit 5 IOB

(A) 16 bit I/O bus mode(IOB = 0, default setting)

Byte operation at even address. (DBHE# = 1, DBLE# = 0)

DATA BUS

Byte operation at odd address. (DBHE# = 0, DBLE# = 1)

DATA BUS

Word operation at even address. (DBHE# = 0, DBLE# = 0)

DATA BUS

Word operation at odd address.

DATA BUS

(B) 8 bit I/O bus mode(IOB = 1)

Byte operation at even address. (DBHE# = 1, DBLE# = 0)

DATA BUS

Byte operation at odd address. (DBHE# = 0, DBLE# = 1)

DATA BUS

Word operation at even address. (DBHE# = 0, DBLE# = 0)

DATA BUS

Word operation at odd address.

DATA BUS

DATA BUS

CR02h(EPWRCR) — special

This register controls Power Save Mode. The initial value right after reset is FFFF00FFh.

bit 15 PCSTP

When set, the CPU enters Power Save Mode and main clock stops functioning after HALT instruction.
If HLDREQ input becomes active in the Power Save Mode. CPU enters a hold state. When HLDREQ becomes inactive. CPU returns to the Power Save Mode.
Main clock starts again when NMI, INTREQ, or RESET becomes active. When clear, HALT instruction is executed normally. PCSTP is cleared after external reset signal.

CR03h(PMON) — special

This register controls processor monitor function. The initial value right after reset is EAAFFFFFh.

bit 28 PMM: Protected Mode Instruction Monitor

When set, the following instructions cause an operation equivalent to the BRKPM interrupt:
PUSHF, POPF, CLI, STI, IRET, LGDT, SGDT, LIDT, SIDT, LLDT, SLDT,
LTR, STR, LMSW, SMSW, ARPL, CLRTS, LDAC, LDLM, TRDE, TWRE,
move to/from CRx, DRx, TRx(CR00h, 08h, 0Ch, 19h, 1Dh, 30h, 34h, 38h,
39h, 3Ch, 3Dh) from/to GRx.
This allows the software to emulate the instructions used for a protected-mode OS, and a protected-mode OS can run as a guest OS under the virtual machine OS. The Protected Mode Instruction Monitor can then be used for debugging the protected-mode OS or BIOS. It can also be used for processor testing.

bit 26 VMZM#: VMZ instruction Monitor

If set at '0', when a special instruction (move immediate to CS, etc. which is not compatible with 386 instruction) is attempted in protected mode, an invalid operand exception (exception 6) occurs. If set at '1', such instructions become valid in protected mode. After reset, this bit is set at '0' to make possible a protection function. The bit 31 - 29 of EFLAGS can be written only when VMZM# is '1'.

bit 24 EXPM: Exception Monitor

If set, when either a hardware or software exception is detected in protected mode, an execution similar to BRKPM occurs.

bit 22 TM: Test Mode

If set, when an exception occurs, CPU stores the exception number in GR26 and jumps indirectly using GR27 as a new IP.

bit 20 DPAC: Cycle Steal Address Calculation Disable

When set, cycle steal address calculation is diabled and when clear, it is enabled. Cycle steal address calculation is an address operand calculation of the next instruction using the unused time slot in internal buses for prefetching.

bit 18 IOM: I/O Instruction Monitor

When set, the instructions below cause an operation equivalent to the BRKPM interrupt. This forces the CPU to enter Real Mode, regardless of the mode that it was in.

IN, OUT, INS, OUTS

A.6.2 Instruction Decoding Unit Control Registers

CR0Eh(TOP) — special

The register controls the 'opcode trap function', together with ***CR0Fh***. It is operated only through Real Mode instructions. The initial value right after reset is undefined. Each field of 'oPCODE 3' to 'oPCODE 0' stores an opcode to be trapped. The relevant opcode will cause the same operation as BRKPM.

CR0Fh(TCON) — special

The register controls the 'opcode trap function', together with ***CR0Eh***. It is operated only through Real Mode instructions. The initial value right after reset is 00000000h. Each field of 'oPCODE 3' to 'oPCODE 0' stores an opcode to be trapped. The relevant opcode will cause the same operation as BRKPM.

bit 16 TCE: Opcode Trap Enable
When set, the opcode trap is enable. Clears when an opcode trap occurs.
bit 15 - 8: Trap Opcode Mask
When set, masks compares to the bits set for OPCODE BREAK 0.
bit 7 - 0: Map 3 - 0
00: No trap
01: 1st opcode
10: 2nd opcode following 00Fh
11: 2nd opcode following 0D6h

A.6.3 Protected-Mode Instruction Unit Control Registers

CR12h(TDESH) — special

This register is used under Protected Mode. It is accessible when the CPU is in Real Mode or VMZ# is set. However, operation by software is not normally required. The initial value right after reset is undefined.

CR13h(TDESL) — special

This register is used under Protected Mode. It is accessible when the CPU is in Real Mode or VMZ# is set. However, operation by software is not normally required. The initial value right after reset is undefined.

CR16h(TZ) — special

This register is used under Protected Mode. It is accessible when the CPU is in Real Mode or VMZ# is set. However, operation by software is not normally required. The initial value right after reset is undefined.

CR17h(TY) — special

This register is used under Protected Mode. It is accessible when the CPU is in Real Mode or VMZ# is set. However, operation by software is not normally required. The initial value right after reset is undefined.

CR1Ah(TX) — special

This register is used under Protected Mode. It is accessible when the CPU is in Real Mode or VMZ# is set. However, operation by software is not normally required. The initial value right after reset is undefined.

CR1Eh(TSEL) — special

This register is used under Protected Mode. It is accessible when the CPU is in Real Mode or VMZ# is set. However, operation by software is not normally required. The initial value right after reset is undefined.

CR1Fh(TERR) — special

This register is used under Protected Mode. It is accessible when the CPU is in Real Mode or VMZ# is set. However, operation by software is not normally required. The initial value right after reset is undefined.

A.6.4 Paging Unit Control Registers

CR06h(PDER) — special

The page directory entry register (PDER) is loaded with the value of a page directory entry at the time of a hit or miss by the paging unit. It is accessible when the Real Mode instructions are available. However, operation by software is not normally required. The initial value right after reset is undefined.

CR07h(PTER) — special

The page table entry register (PTER) is loaded with the value of a page directory entry at the time of a hit or miss by the paging unit. It is accessible when the Real Mode instructions are available. However, operation by software is not normally required. The initial value right after reset is undefined.

CR08h(PFAR) — CR2

PFAR is compatible with 386 CR2 and accessible by either Real or Protected Mode instructions. A linear address which is generated a page fault is transferred from the page fault address buffer and stored in this register. The initial value right after reset is undefined.

CR0Bh(PECR) — special

PECR is provided for paged virtual memory under Real Mode. Its lowest 3 bits are loaded with the error code for a page fault. However, the error code is pushed onto the stack under Protected Mode. The initial value right after reset is undefined. Error codes to be set with the lowest 3 bits conform to those under Protected Mode.

CR0Ch(PDBR) — CR3

PDBR is compatible with 386 CR3 and accessible by either Real or Protected Mode instructions. The initial value right after reset is undefined. The base address of a page directory is 32-bit and can be read/written.

CR19h(LAR) — TR6

LAR is compatible with 386 CR6 and accessible by either Real or Protected Mode instructions. The initial value right after reset is undefined.

bit 31-12: Linear Address

On a TLB write: To be written to the TLB

On a TLB lookup: To be looked up in the TLB

bit 11: V

Valid Bit

bit 10 - 9: D, D#

Dirty bit and its complement

bit 8 - 7: U, U#

U/S bit and its complement

bit 8 - 7: U, U#

U/S bit and its complement

bit 6 - 5: W, W#

R/W bit and its complement

bit 0: C

Command Bit

C = 1: TLB write

C = 0: TLB lookup

CR1Dh(PAR) — TR7

PAR is compatible with 386 TR7 and accessible by either Real or Protected Mode instructions. The initial value right after reset is undefined.

bit 31-12: Physical Address
On a TLB write: To be written to the TLB
On a TLB lookup: Value from the TLB
bit 4: PL
On a TLB write:
PL = 0: Write to the internal pointer
PL = 1: Write to the REP pointer
On a TLB lookup:
PL = 0: Miss
PL = 1: Hit
bit 3: REP1
For a TLB write, selects which block is to be written.
bit 2: REP0
For a TLB lookup, selects in which block the look-up entry was found.

A.6.5 SiEMU-II Unit Control Registers

CR30h(SPA0) — DR0
SPA0 is compatible with 386 DR0 and holds the address for Breakpoint 0. The initial value right after reset is undefined.

CR34h(SPA1) — DR1
SPA1 is compatible with 386 DR1 and holds the address for Breakpoint 1. The initial value right after reset is undefined.

CR38h(SPA2) — DR2
SPA2 is compatible with 386 DR2 and holds the address for Breakpoint 2. The initial value right after reset is undefined.

CR39h(SBRK) — DR6
SBRK is compatible with 386 DR6 and to be set when any break is detected. The initial value right after reset is 00000000h.

bit 15: BT
Task switch break detect
bit 14: BS
Single step break detect

bit 13: BD
Register protection violation break detect
bit 3: B3
Breakpoint 3 break detect
bit 2: B2
Breakpoint 2 break detect
bit 1: B1
Breakpoint 1 break detect
bit 0: B0
Breakpoint 0 break detect

CR3Ch(SPA3) — DR3
SPA0 is compatible with 386 DR3 and holds the address for Breakpoint 3. The initial value right after reset is undefined.

CR3Dh(SCON) — DR7
SCON is compatible with 386 DR7 and controls breakpoints 3 to 1. The initial value right after reset is 00000000h.

31 30	29 28	27 26	25 24	23 22	21 20	19 18	17 16	15 14	13	12 11 10	9	8	7	6	5	4	3	2	1	0
LEN	R/W	LEN	R/W	LEN	R/W	LEN	R/W	0	GD	0	GE	LE	G3	L3	G2	L2	G1	L1	G0	L0

bit 31, 30, 27, 26, 23, 22, 19, 18: LENx
Breakpointed location size

LEN	
0 0	Byte
0 1	Word
1 0	Double — undefined in the 386
1 1	Double

bit 29, 28, 25, 24, 21, 20, 17, 16: R/Wx
Break condition

R/W	
0 0	Instruction Execution Only
0 1	Data Write
1 0	No Break — undefined in the 386
1 1	Data read / write

bit 13: GD
Debug Register Protection Condition Enable (To be cleared by the processor)
bit 9: GE
Fault Mode Enable (Global)
bit 8: LE
Fault Mode Enable (To be cleared by the local or task switch)
bit 7, 5, 3, 1: Gx
Address Break x Enable (Gobal)
bit 6, 4, 2, 0: Lx
Address Break x Enable (To be cleared by the local or task switch)

APPENDIX B. Special instruction format

The following instruction set is useful for processing SMI routine if the original operating system is in protected mode. If we use IRQ15 to implement power management mode or the original operating system is in real mode, we do not need to do extra work.

B.1 Move Data

This function is provided to move data between register and register/memory. It is fully compatible with 386 instructions. Also, it includes the following instructions which are more powerful than 386 instructions are.

General register to General register/memory

11010110	01001000	greg bm1	bm2 greg/im disp W

greg : 5 bit, from 00000 to 11111, GR0 to GR31

bm1	greg
000	GB#(Use GR# as Byte Register, bit 0 to 7)
001	GL#(Use GR# as Word Register, bit 0 to 15)
010	GH#(Use GR# as Word Register, bit 16 to 31)
011	GR#(Use GR# as Double Word Register, bit 0 to 31)
100	GB#, bm2 gr/im is not required, next two bytes is disp W
101	GL#, bm2 gr/im is not required, next two bytes is disp W
110	GH#, bm2 gr/im is not required, next two bytes is disp W
111	GR#, bm2 gr/im is not required, next two bytes is disp W

bm2	greg/im
000	GB#
001	GL#
010	GH#
011	GR#
100	post-increment*
101	pre-decrement*
110	5-bit short immediate with zero extend
111	5-bit short immediate with sign extend

*: If GR4 or GR5 is selected as GBASE, the stack segment is selected in default. It can be changed by a segment override prefix or postfix. In other memory access, EDS segment register will be selected in default.

Control register to General register

11010110	11111000	greg 011	creg

creg : 8 bit, from 00000000 to 11111111, CR00h to CRFFh

General register to Control register

11010110	11111010	greg 011	creg

Base register to General register

11010110	11011000	greg 011	sreg 00000

sreg : 3 bit, from 000 to 111, SR0 to SR7

General register to Base register

11010110	11011010	greg 011	sreg 00000

Attribute register to General register

11010110	11011001	greg 011	areg 00000

areg : 3 bit, from 000 to 111, AR0 to AR7

General register to Attribute register

11010110	11011011	greg 011	areg 00000

User Base register to General register

11010110	11001000	greg 011	usreg 00000

usreg : 3 bit, 100 — eusp, 010 — SR2u, 111 — SR7u

General register to User Base register

11010110	11001010	greg 011	usreg 00000

User Attribute register to General register

11010110	11001001	greg 011	uareg 00000

uareg : 3 bit, 010 — AR2u, 111 — AR7u

General register to User Attribute register

11010110	11001011	greg 011	uareg 00000

B.2 Push/Pop Data

This function is provided to push/pop data into/from stack. It is fully compatible with 386 instructions. If we pop segment register(DS, ES, and so on), the base and attribute register may be changed (please refer Intel's instruction set). Also, it includes the following instructions which are more powerful than 386 instructions are.

push General register

11010110	10110100	greg bm1

pop General register

11010110	10111100	greg bm1

push segment register

11010110	11110100	sreg 00000

pop segment register

11010110	11111100	sreg 00000

push all General register GR0 — GR31

11010110	11100100

pop all General register GR0 — GR31

11010110	11101100

2.4 Atmel AT27C080 – 1-Mbyte EPROM

The AT27C080-12PC is a critical memory component of the Ampro CoreModule 3SXi card and contains the system and application software, which determines most CPU operations. The specifications and data sheet for this IC are included for extra clarity.

Two types of memory ICs from Atmel have been used.

- 27C080-12 PC — Once programmable read-only memory – PROM
- 27C080-12 DC — UV erasable programmable read-only memory – EPROM

The UV erasable version is useful during program development. The once programmable version is used in the information barrier system to reduce the capability to change the system. The PC suffix indicates a plastic DIP package with a commercial temperature range. The DC suffix indicates a cerdip package with a commercial temperature range.

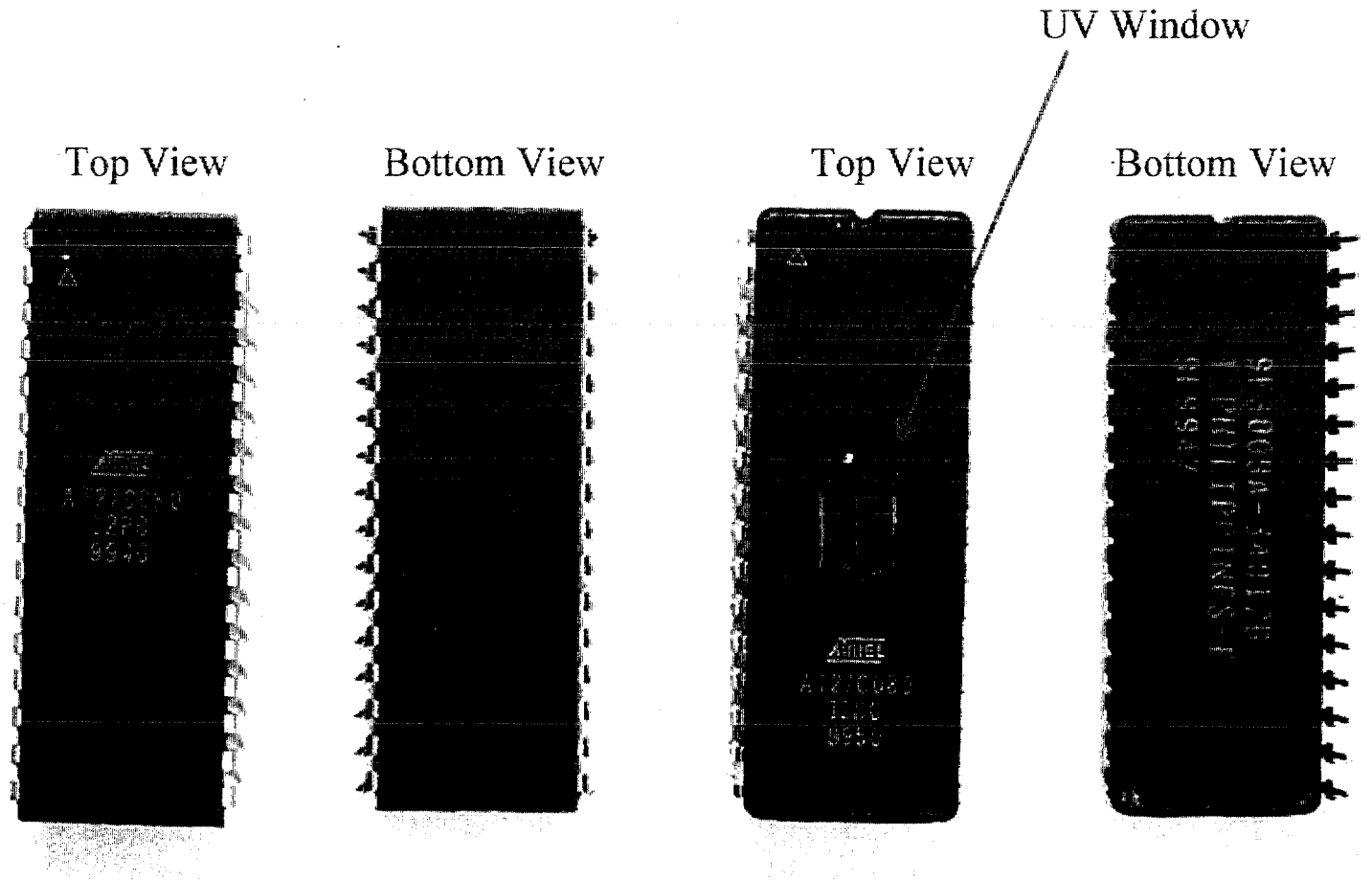
2.4.1 Photograph of the AT27C080-12PC and AT27C080-12DC

This section contains:

- A photograph with both top and bottom views of both the once-programmable (PROM) version and the UV erasable (EPROM) version.

Notice the distinctive window on the EPROM version. The PROM version, without the window, is used in the computational block during the demonstration. Notice also the longer shadow associated with the top view when the legs offset the IC more from the white background sheet.

PROM Integrated Circuits



Once-Programmable PROM ICs
AT27C080-12PC

UV-Erasable PROM ICs
AT27C080-12DC

2.4.2 X-Ray of the AT27C080-12PC and AT27C080-12DC

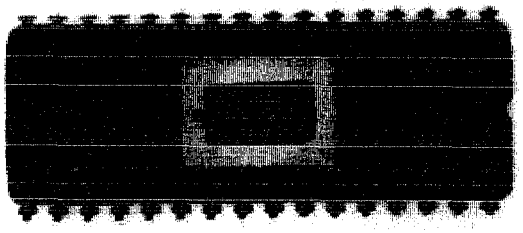
This section contains:

- A x-ray with both top and bottom views of both the once-programmable (PROM) version and the UV erasable (EPROM) version.

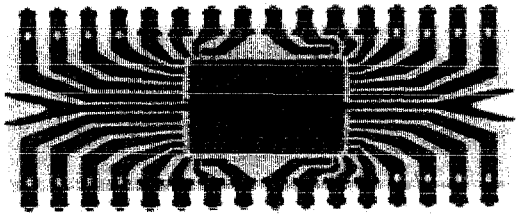
Notice the plastic case of the once-programmable PROM is much more transparent to x-rays than the ceramic case of the UV erasable version. In the raw image of the UV-erasable version, one can see a circle, which is slightly whiter than surroundings, corresponding to the clear window.

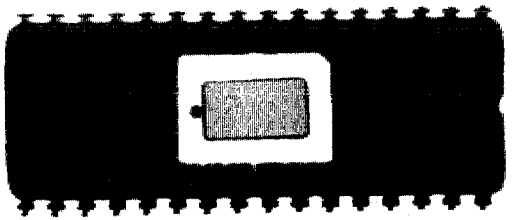
Notice the distinctive window on the EPROM. The PROM version is used in the IB system during the demonstration. Notice also the longer shadow associated with the top view when the legs offset the IC more from the white background sheet.

PROM Integrated Circuits – X-ray

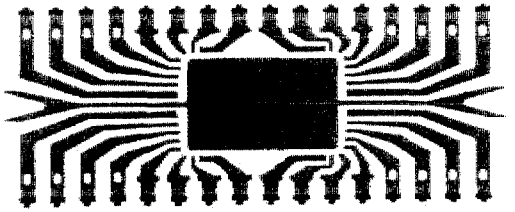


Raw
Image





Brightened
Image



UV-Erasable PROM ICs
AT27C080-12DC

Once-Programmable PROM ICs
AT27C080-12PC

2.4.3 Data Sheet for the AT27C080

The following is the vendor's data sheet for the Atmel AT27C080 1-Mbyte PROM memory, which was downloaded from the Atmel Internet site at:

http://www.atmel.com/atmel/acrobat/doc0360.pdf.

The data sheet does not distinguish between the various versions.

Features

- Fast Read Access Time - 90 ns
- Low Power CMOS Operation
  - 100 µA max. Standby
  - 40 mA max. Active at 5 MHz
- JEDEC Standard Packages
  - 32 Lead PLCC
  - 32-Lead 600-mil PDIP and Cerdip
  - 32-Lead 450-mil SOIC (SOP)
  - 32-Lead TSOP
- 5V ± 10% Supply
- High-Reliability CMOS Technology
  - 2,000V ESD Protection
  - 200 mA Latchup Immunity
- Rapid™ Programming Algorithm - 50 µs/byte (typical)
- CMOS and TTL Compatible Inputs and Outputs
- Integrated Product Identification Code
- Industrial and Commercial Temperature Ranges

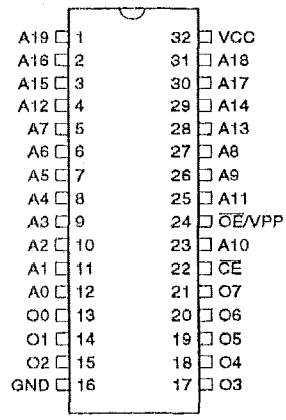
Description

The AT27C080 chip is a low-power, high-performance 8,388,608-bit ultraviolet erasable programmable read only memory (EPROM) organized as 1M by 8 bits. The AT27C080 requires only one 5V power supply in normal read mode operation. Any byte can be accessed in less than 90 ns, eliminating the need for speed reducing WAIT states on high-performance microprocessor systems. *(continued)*

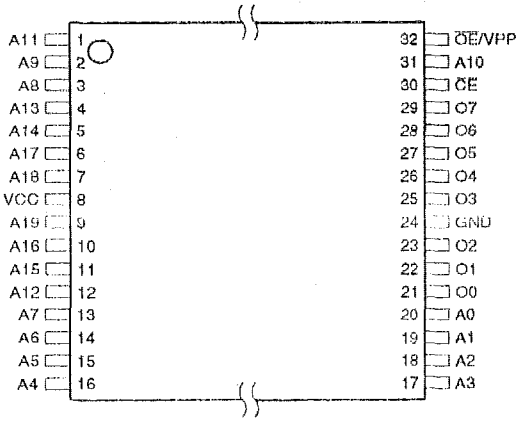
Pin Configurations

Pin Name	Function
A0 - A19	Addresses
O0 - O7	Outputs
$\overline{CE}$	Chip Enable
$\overline{OE}$/VPP	Output Enable/ Program Supply

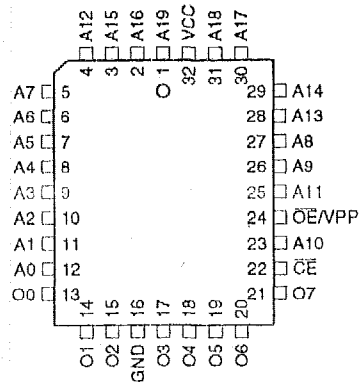
CDIP, PDIP, SOIC Top View



TSOP Top View
Type 1



PLCC Top View



8-Megabit (1M x 8) UV Erasable EPROM

AT27C080

Rev. 0360G–10/98

Atmel's scaled CMOS technology provides low active power consumption and fast programming. Power consumption is typically 10 mA in active mode and less than 10 µA in standby mode.

The AT27C080 is available in a choice of packages, including; one-time programmable (OTP) plastic PLCC, PDIP, SOIC (SOP), and TSOP, as well as windowed ceramic Cerdip. All devices feature two-line control ($\overline{CE}$, $\overline{OE}$) to give designers the flexibility to prevent bus contention.

With high density 1M byte storage capability, the AT27C080 allows firmware to be stored reliably and to be accessed by the system without the delays of mass storage media.

Atmel's 27C080 has additional features to ensure high quality and efficient production use. The Rapid™ Programming Algorithm reduces the time required to program the part and guarantees reliable programming. Programming time is typically only 50 µs/byte. The Integrated Product Identification Code electronically identifies the device and manufacturer. This feature is used by industry standard programming equipment to select the proper programming algorithms and voltages.

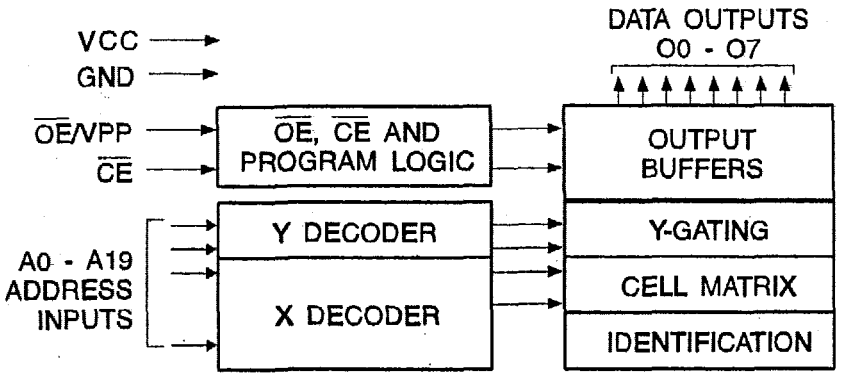
Erasure Characteristics

The entire memory array of the AT27C080 is erased (all outputs read as V_{OH}) after exposure to ultraviolet light at a wavelength of 2,537Å. Complete erasure is assured after a minimum of 20 minutes of exposure using 12,000 µW/cm^2 intensity lamps spaced one inch away from the chip. Minimum erase time for lamps at other intensity ratings can be calculated from the minimum integrated erasure dose of 15 W.sec/cm^2. To prevent unintentional erasure, an opaque label is recommended to cover the clear window on any UV erasable EPROM that will be subjected to continuous flourescent indoor lighting or sunlight.

System Considerations

Switching between active and standby conditions via the Chip Enable pin may produce transient voltage excursions. Unless accommodated by the system design, these transients may exceed data sheet limits, resulting in device non-conformance. At a minimum, a 0.1 µF high frequency, low inherent inductance, ceramic capacitor should be utilized for each device. This capacitor should be connected between the V_{CC} and Ground terminals of the device, as close to the device as possible. Additionally, to stabilize the supply voltage level on printed circuit boards with large EPROM arrays, a 4.7 µF bulk electrolytic capacitor should be utilized, again connected between the V_{CC} and Ground terminals. This capacitor should be positioned as close as possible to the point where the power supply is connected to the array.

Block Diagram



\bsolute Maximum Ratings*

Temperature Under Bias	-55°C to +125°C
Storage Temperature	-65°C to +150°C
Voltage on Any Pin with Respect to Ground	-2.0V to +7.0V(1)
Voltage on A9 with Respect to Ground	-2.0V to +14.0V(1)
V_{PP} Supply Voltage with Respect to Ground	-2.0V to +14.0V(1)
Integrated UV Erase Dose	7258 W•sec/cm^2

*NOTICE: Stresses beyond those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions beyond those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

Note: 1. Minimum voltage is -0.6V DC which may undershoot to -2.0V for pulses of less than 20 ns. Maximum output pin voltage is V_{CC} + 0.75V DC which may overshoot to +7.0V for pulses of less than 20 ns.

Operating Modes

Mode/Pin	$\overline{CE}$	$\overline{OE}/V_{PP}$	Ai	Outputs
Read	V_{IL}	V_{IL}	Ai	D_{OUT}
Output Disable	X	V_{IH}	X(1)	High Z
Standby	V_{IH}	X	X	High Z
Rapid Program(2)	V_{IL}	V_{PP}	Ai	D_{IN}
PGM Verify	V_{IL}	V_{IL}	Ai	D_{OUT}
PGM Inhibit	V_{IH}	V_{PP}	X	High Z
Product Identification(4)	V_{IL}	V_{IL}	A9 = V_H(3) A0 = V_{IH} or V_{IL} A1 - A19 = V_{IL}	Identification Code

Notes: 1. X can be V_{IL} or V_{IH}.
2. Refer to Programming Characteristics.
3. V_H = 12.0 ± 0.5V.
4. Two identifier bytes may be selected. All Ai inputs are held low (V_{IL}), except A9 which is set to V_H and A0 which is toggled low (V_{IL}) to select the Manufacturer's Identification byte and high (V_{IH}) to select the Device Code byte.

DC and AC Operating Conditions for Read Operation

		AT27C080-90	AT27C080-10	AT27C080-12	AT27C080-15
Operating Temperature (Case)	Com.	0°C - 70°C	0°C - 70°C	0°C - 70°C	0°C - 70°C
	Ind.	-40°C - 85°C	-40°C - 85°C	-40°C - 85°C	-40°C - 85°C
V_{CC} Power Supply		5V ± 10%	5V ± 10%	5V ± 10%	5V ± 10%

DC and Operating Characteristics for Read Operation

Symbol	Parameter	Condition	Min	Max	Units
I_{LI}	Input Load Current	V_{IN} = 0V to V_{CC} (Com., Ind.)		±1.0	µA
I_{LO}	Output Leakage Current	V_{OUT} = 0V to V_{CC} (Com., Ind.)		±5.0	µA
I_{SB}	V_{CC}[1] Standby Current	I_{SB1} (CMOS), $\overline{CE}$ = V_{CC} ± 0.3V		100	µA
		I_{SB2} (TTL), $\overline{CE}$ = 2.0 to V_{CC} + 0.5V		1.0	mA
I_{CC}	V_{CC} Active Current	f = 5 MHz, I_{OUT} = 0 mA, $\overline{CE}$ = V_{IL}		40	mA
V_{IL}	Input Low Voltage		-0.6	0.8	V
V_{IH}	Input High Voltage		2.0	V_{CC} + 0.5	V
V_{OL}	Output Low Voltage	I_{OL} = 2.1 mA		0.4	V
V_{OH}	Output High Voltage	I_{OH} = -400 µA	2.4		V

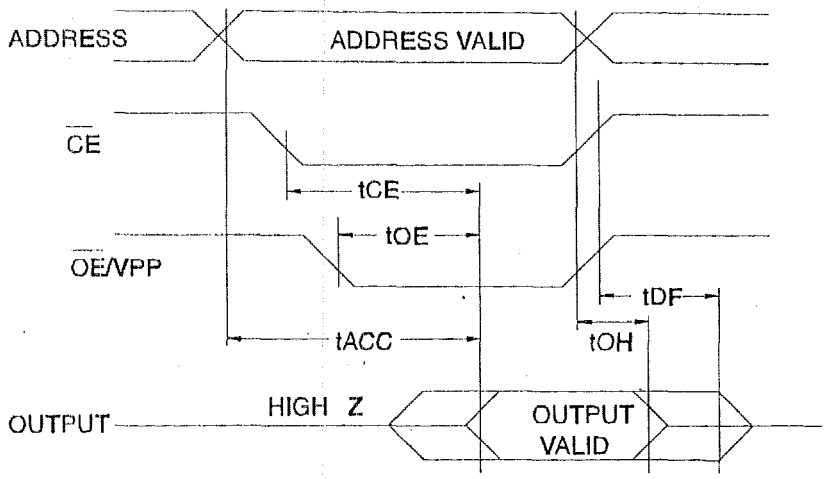
Note: 1. V_{CC} must be applied simultaneously or before $\overline{OE}$/ V_{PP}, and removed simultaneously or after $\overline{OE}$/V_{PP}.

AC Characteristics for Read Operation

			AT27C080								
			-90		-10		-12		-15		
Symbol	Parameter	Condition	Min	Max	Min	Max	Min	Max	Min	Max	Units
t_{ACC}[4]	Address to Output Delay	$\overline{CE}$ = $\overline{OE}$/V_{PP} = V_{IL}		90		100		120		150	ns
t_{CE}[3]	$\overline{CE}$ to Output Delay	$\overline{OE}$ = V_{IL}		90		100		120		150	ns
t_{OE}[3][4]	$\overline{OE}$ to Output Delay	$\overline{CE}$ = V_{IL}		20		20		30		35	ns
t_{DF}[2][5]	$\overline{OE}$ or $\overline{CE}$ High to Output Float, whichever occurred first			30		30		35		40	ns
t_{OH}	Output Hold from Address, $\overline{CE}$ or $\overline{OE}$/V_{PP} whichever occurred first		0		0		0		0		ns

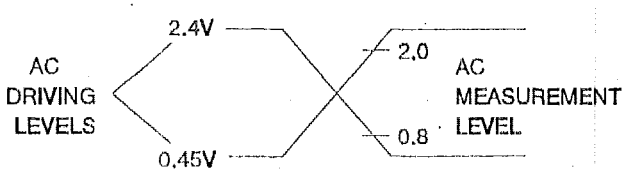
Notes: 1. 2, 3, 4, 5. See AC Waveforms for Read Operation.

\C Waveforms for Read Operation[1]



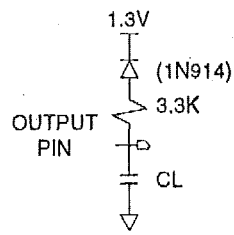
Notes:
1. Timing measurement references are 0.8V and 2.0V. Input AC drive levels are 0.45V and 2.4V, unless otherwise specified.
2. t_{DF} is specified form OE/VPP or CE, whichever occurs first. Output float is defined as the point when data is no longer driven.
3. $\overline{OE}/V_{PP}$ may be delayed up to t_{CE} - t_{OE} after the falling edge of $\overline{CE}$ without impact on t_{CE}.
4. $\overline{OE}/V_{PP}$ may be delayed up to t_{ACC}- t_{OE} after the address is valid without impact on t_{ACC}.
5. This parameter is only sampled and is not 100% tested.

Input Test Waveform and Measurement Levels



t_R, t_F < 20 ns (10% to 90%)

Output Test Load



Note: CL = 100 pF including jig capacitance.

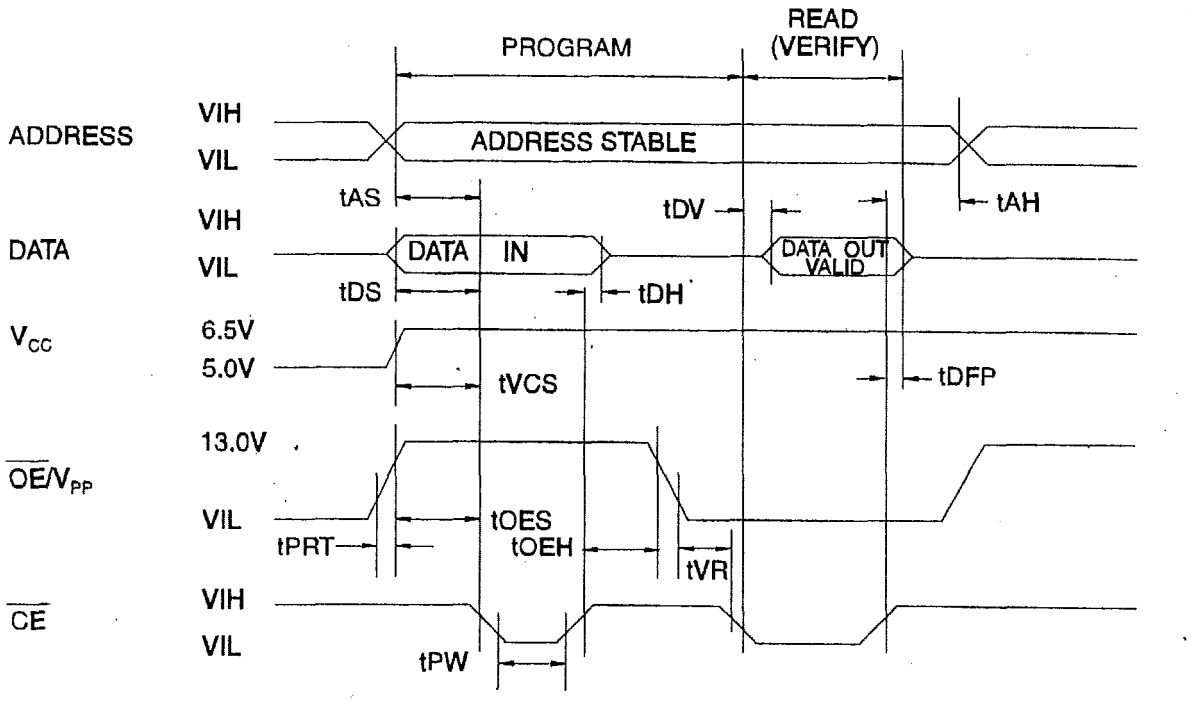
Pin Capacitance

f = 1 MHz, T = 25°C[1]

Symbol	Typ	Max	Units	Conditions
C_{IN}	4	8	pF	V_{IN} = 0V
C_{OUT}	8	12	pF	V_{OUT} = 0V

Note: 1. Typical values for nominal supply voltage. This parameter is only sampled and is not 100% tested.

Programming Waveforms



Notes: 1. The Input Timing reference is 0.8V for V_{IL} and 2.0V for V_{IH}.

2. t_{OE} and t_{DFP} are characteristics of the device but must be accommodated by the programmer.

DC Programming Characteristics

T_A = 25 ± 5°C, V_{CC} = 6.5 ± 0.25V, $\overline{OE}/V_{PP}$ = 13.0 ± 0.25V

Symbol	Parameter	Test Conditions	Limits		Units
			Min	Max	
I_{LI}	Input Load Current	$V_{IN} = V_{IL}, V_{IH}$		±10	µA
V_{IL}	Input Low Level		-0.6	0.8	V
V_{IH}	Input High Level		2.0	V_{CC} + 1.0	V
V_{OL}	Output Low Voltage	I_{OL} = 2.1 mA		0.4	V
V_{OH}	Output High Voltage	I_{OH} = -400 µA	2.4		V
I_{CC2}	V_{CC} Supply Current (Program and Verify)			40	mA
I_{PP2}	$\overline{OE}/V_{PP}$ Supply Current	$\overline{CE} = V_{IL}$		25	mA
V_{ID}	A9 Product Identification Voltage		11.5	12.5	V

AC Programming Characteristics

T_A = 25 ± 5°C, V_{CC} = 6.5 ± 0.25V, $\overline{OE}/V_{PP}$ = 13.0 ± 0.25V

Symbol	Parameter	Test Conditions[1]	Limits		Units
			Min	Max	
t_{AS}	Address Setup Time	Input Rise and Fall Times: (10% to 90%) 20 ns Input Pulse Levels: 0.45V to 2.4V Input Timing Reference Level: 0.8V to 2.0V Output Timing Reference Level: 0.8V to 2.0V	2.0		µs
t_{OES}	$\overline{OE}/V_{PP}$ Setup Time		2.0		µs
t_{OEH}	$\overline{OE}/V_{PP}$ Hold Time		2.0		µs
t_{DS}	Data SetupTime		2.0		µs
t_{AH}	Address Hold Time		0.0		µs
t_{DH}	Data Hold Time		2.0		µs
t_{DFP}	$\overline{CE}$ High to Output Float Delay[2]		0.0	130	ns
t_{VCS}	V_{CC} Setup Time		2.0		µs
t_{PW}	$\overline{CE}$ Program Pulse Width[3]		47.5	52.5	µs
t_{DV}	Data Valid from $\overline{CE}$			1.0	µs
t_{VR}	$\overline{OE}/V_{PP}$ Recovery Time		2.0		ns
t_{PRT}	$\overline{OE}/V_{PP}$ Pulse Rise Time During Programming		50		ns

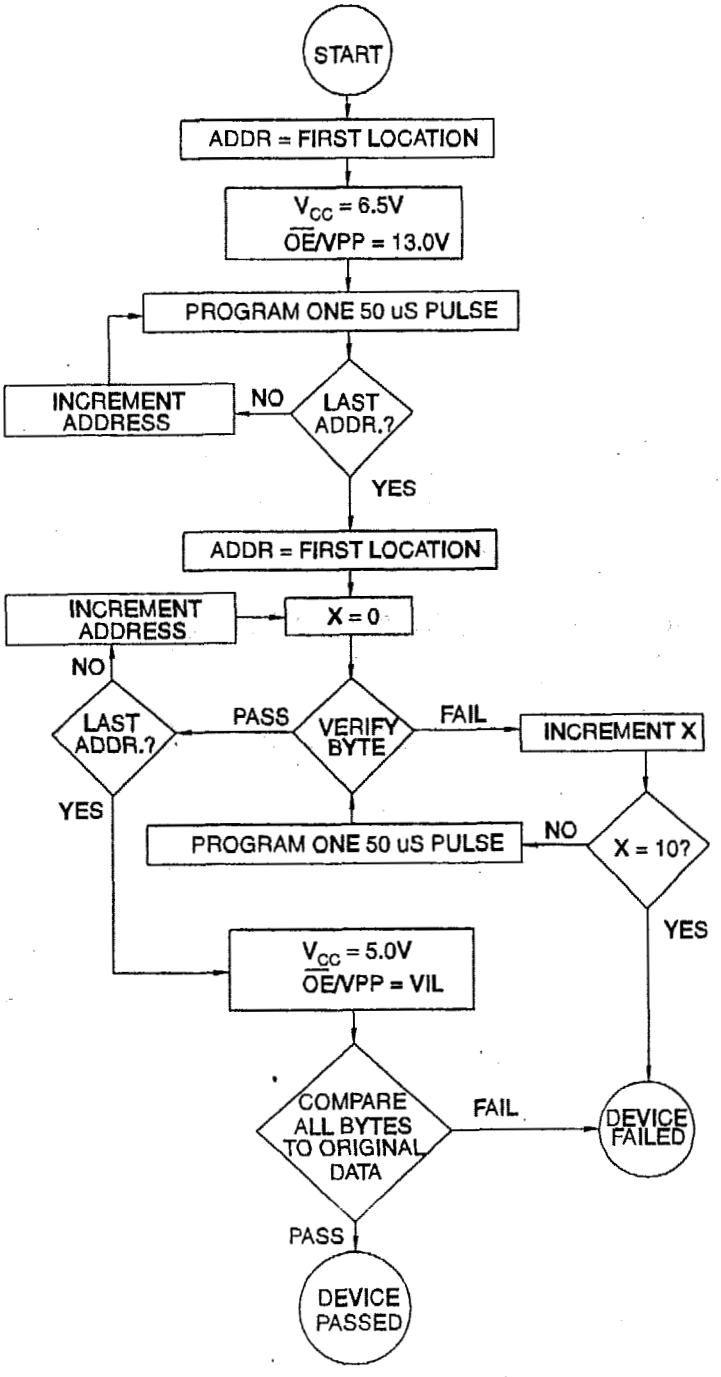
Notes:
1. V_{CC} must be applied simultaneously or before $\overline{OE}/V_{PP}$ and removed simultaneously or after $\overline{OE}/V_{PP}$
2. This parameter is only sampled and is not 100% tested. Output Float is defined as the point where data is no longer driven—see timing diagram.
3. Program Pulse width tolerance is 50 µs ± 5%.

Atmel's 27C080 Integrated Product Identification Code

Codes	Pins									Hex Data
	A0	O7	O6	O5	O4	O3	O2	O1	O0	
Manufacturer	0	0	0	0	1	1	1	1	0	1E
Device Type	1	1	0	0	0	1	0	1	0	8A

Rapid Programming Algorithm

A 50 µs $\overline{CE}$ pulse width is used to program. The address is set to the first location. V_{CC} is raised to 6.5V and $\overline{OE}/V_{PP}$ is raised to 13.0V. Each address is first programmed with one 50 µs $\overline{CE}$ pulse without verification. Then a verification reprogramming loop is executed for each address. In the event a byte fails to pass verification, up to 10 successive 50 µs pulses are applied with a verification after each pulse. If the byte fails to verify after 10 pulses have been applied, the part is considered failed. After the byte verifies properly, the next address is selected until all have been checked. $\overline{OE}/V_{PP}$ is then lowered to V_{IL} and V_{CC} to 5.0V. All bytes are read again and compared with the original data to determine if the device passes or fails.



Ordering Information

t_{ACC} (ns)	I_{CC} (mA)		Ordering Code	Package	Operation Range
	Active	Standby			
90	40	0.1	AT27C080-90DC	32DW6	Commercial (0°C to 70°C)
			AT27C080-90JC	32J	
			AT27C080-90PC	32P6	
			AT27C080-90RC	32R	
			AT27C080-90TC	32T	
	40	0.1	AT27C080-90DI	32DW6	Industrial (-40°C to 85°C)
			AT27C080-90JI	32J	
			AT27C080-90PI	32P6	
			AT27C080-90RI	32R	
			AT27C080-90TI	32T	
100	40	0.1	AT27C080-10DC	32DW6	Commercial (0°C to 70°C)
			AT27C080-10JC	32J	
			AT27C080-10PC	32P6	
			AT27C080-10RC	32R	
			AT27C080-10TC	32T	
	40	0.1	AT27C080-10DI	32DW6	Industrial (-40°C to 85°C)
			AT27C080-10JI	32J	
			AT27C080-10PI	32P6	
			AT27C080-10RI	32R	
			AT27C080-10TI	32T	

Package Type	
32DW6	32-Lead, 0.600" Windowed, Ceramic Dual Inline Package (Cerdip)
32J	32-Lead,Plastic J-Leaded Chip Carrier (PLCC)
32P6	32-Lead, 0.600" Wide, Plastic Dual Inline Package (PDIP)
32R	32-Lead, 0.450" Wide, Plastic Gull Wing Small Outline (SOIC)
32T	32-Lead, Plastic Thin Small Outline Package (TSOP)

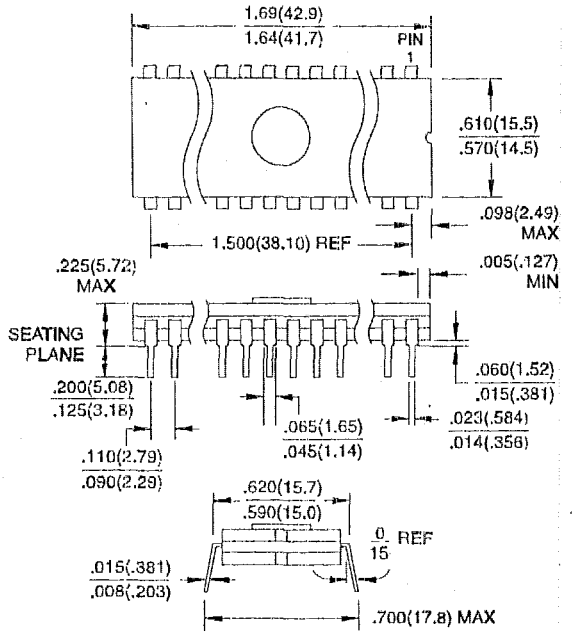
Ordering Information (Continued)

t_{ACC} (ns)	I_{CC} (mA)		Ordering Code	Package	Operation Range
	Active	Standby			
120	40	0.1	AT27C080-12DC AT27C080-12JC AT27C080-12PC AT27C080-12RC AT27C080-12TC	32DW6 32J 32P6 32R 32T	Commercial (0°C to 70°C)
	40	0.1	AT27C080-12DI AT27C080-12JI AT27C080-12PI AT27C080-12RI AT27C080-12TI	32DW6 32J 32P6 32R 32T	Industrial (-40°C to 85°C)
150	40	0.1	AT27C080-15DC AT27C080-15JC AT27C080-15PC AT27C080-15RC AT27C080-15TC	32DW6 32J 32P6 32R 32T	Commercial (0°C to 70°C)
	40	0.1	AT27C080-15DI AT27C080-15JI AT27C080-15PI AT27C080-15RI AT27C080-15TI	32DW6 32J 32P6 32R 32T	Industrial (-40°C to 85°C)

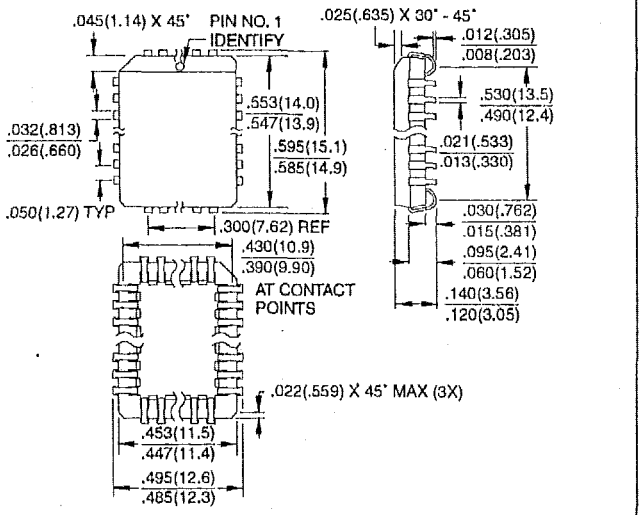
Package Type	
32DW6	32-Lead, 0.600" Windowed, Ceramic Dual Inline Package (Cerdip)
32J	32-Lead,Plastic J-Leaded Chip Carrier (PLCC)
32P6	32-Lead, 0.600" Wide, Plastic Dual Inline Package (PDIP)
32R	32-Lead, 0.450" Wide, Plastic Gull Wing Small Outline (SOIC)
32T	32-Lead, Plastic Thin Small Outline Package (TSOP)

Packaging Information

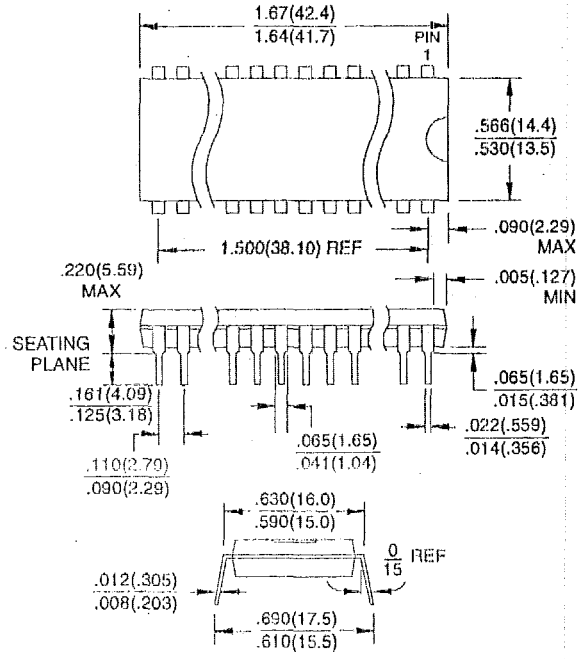
32DW6, 32-Lead, 0.600" Wide, Windowed, Ceramic Dual Inline Package (Cerdip)
Dimensions in Inches and (Millimeters)
MIL-STD-1835 D-16 CONFIG A



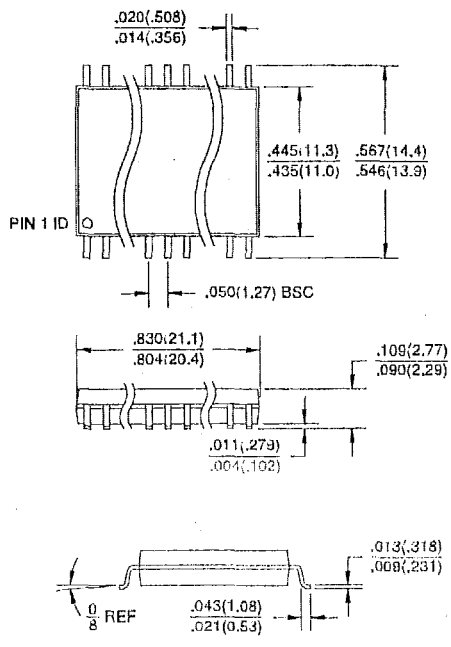
32J, 32-Lead, Plastic J-Leaded Chip Carrier (PLCC)
Dimensions in Inches and (Millimeters)
JEDEC STANDARD MS-016 AE



32P6, 32-Lead, 0.600" Wide, Plastic Dual Inline Package (PDIP)
Dimensions in Inches and (Millimeters)

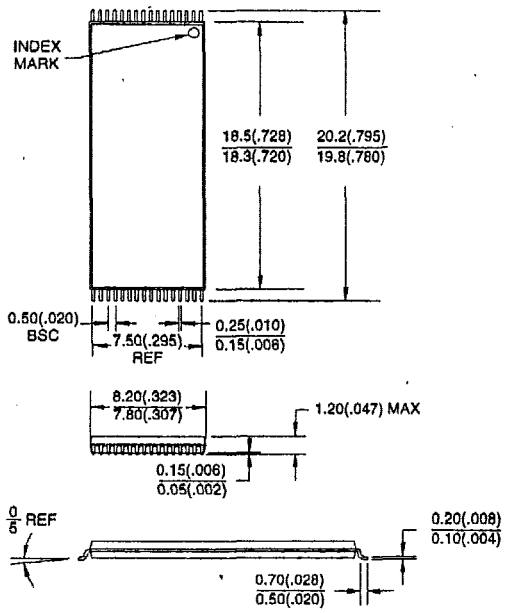


32R, 32-Lead, 0.440" Wide, Plastic Gull Wing Small Outline (SOIC)
Dimensions in Inches and (Millimeters)



Packaging Information

32T, 32-Lead, Plastic Thin Small Outline Package (TSOP)
Dimensions in Millimeters and (Inches)*
JEDEC OUTLINE MO-142 BD



Atmel Headquarters

Corporate Headquarters
2325 Orchard Parkway
San Jose, CA 95131
TEL (408) 441-0311
FAX (408) 487-2600

Europe
Atmel U.K., Ltd.
Coliseum Business Centre
Riverside Way
Camberley, Surrey GU15 3YL
England
TEL (44) 1276-686677
FAX (44) 1276-686697

Asia
Atmel Asia, Ltd.
Room 1219
Chinachem Golden Plaza
77 Mody Road
Tsimshatsui East
Kowloon, Hong Kong
TEL (852) 27219778
FAX (852) 27221369

Japan
Atmel Japan K.K.
Tonetsu Shinkawa Bldg., 9F
1-24-8 Shinkawa
Chuo-ku, Tokyo 104-0033
Japan
TEL (81) 3-3523-3551
FAX (81) 3-3523-7581

Atmel Operations

Atmel Colorado Springs
1150 E. Cheyenne Mtn. Blvd.
Colorado Springs, CO 80906
TEL (719) 576-3300
FAX (719) 540-1759

Atmel Rousset
Zone Industrielle
13106 Rousset Cedex, France
TEL (33) 4 42 53 60 00
FAX (33) 4 42 53 60 01

Fax-on-Demand
North America:
1-(800) 292-8635
International:
1-(408) 441-0732

e-mail
literature@atmel.com

Web Site
http://www.atmel.com

BBS
1-(408) 436-4309

0360G–10/98/xM

2.4.4 Part Number Codes for the AT27C080

The following is the vendor's product number codes for the Atmel AT27C080 1-Mbyte PROM memory, which was downloaded from the Atmel Internet site at:

http://www.atmel.com/atmel/acrobat/doc0538.pdf.

When purchased the AT27C080-12PC comes in a plastic case without the window to erase the memory on either the top or the bottom of the IC. The top is labeled Atmel | AT27C080 | 12PC | 9946 and the bottom is labeled 9E4254-34012B | 1M 9E9931.

When purchased the AT27C080-12DC comes in a ceramic case with a circular window to allow UV light to erase the memory centered on the topside of the IC. The top is labeled Atmel | AT27C080 | 12PC | 9950 and the bottom is labeled 9E3009A-34012B | 1 PHILIPPINES-F | 9E9947.

Explanation of Atmel's Part Number Code

All Atmel part numbers begin with the prefix "AT". The next four to nine digits are the part number. In addition, Atmel parts can be ordered in particular speeds, in specific packages, for particular temperature ranges and with the option of 883C level B military compliance. The available options for each part are listed at the back of its data sheet in its "Ordering Information" table. These options are designated by the following suffixes placed at the end of the Atmel part number, in the order given:

Prefix	Device-	Suffix			
AT	XXXXX	X	X	X	X

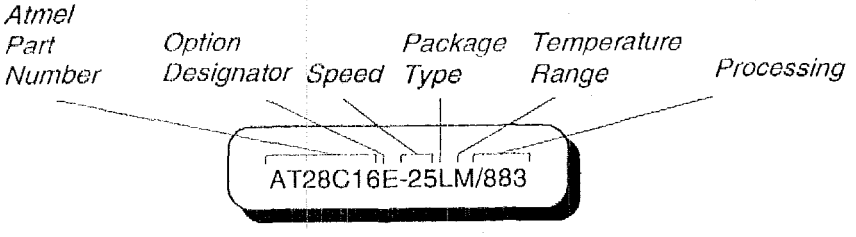
Processing
- Blank = Standard
- /883 = MIL-STD-883, Class B Fully Compliant
- B = MIL-STD-883, Class B Non-Compliant

Temperature Range
- C = Com Temp (0°C to 70°C)
- I = Ind Temp (-40°C to 85°C)
- M = Mil Temp (-55°C to 125°C)

Package
- A = TQFP
- B = Ceramic Side Braze Dual Inline
- C = CBGA
- D = Cerdip
- F = Flatpack
- G = Cerdip, One Time Programmable
- J = Plastic J-Lead Chip Carrier
- K = Ceramic J-Lead Chip Carrier
- L = Leadless Chip Carrier
- M = MSOP
- N = Leadless Chip Carrier, One Time Programmable
- P = Plastic DIP
- Q = Plastic Quad Flatpack
- R = SOIC
- S = SOIC
- T = TSOP (TSSOP for Serial EEPROM)
- U = µBGA
- V = VSOP (Small TSOP)
- W = Die
- X = TSSOP
- Y = Cerpack
- Z = Ceramic Multi-Chip Module

Speed

Here is an example Atmel part number:



Rev. 0538B–10/98

EPROMs

***Battery-Voltage™* (2.7 to 3.6V)**				
AT27BV256	32K x 8	70 - 150 ns	256K-bit, 2.7-volt to 3.6-volt EPROM	Now
AT27BV512	64K x 8	70 - 150 ns	512K-bit, 2.7-volt to 3.6-volt EPROM	Now
AT27BV010	128K x 8	90 - 150 ns	1M-bit, 2.7-volt to 3.6-volt EPROM	Now
AT27BV1024	64K x 16	90 - 150 ns	1M-bit, 2.7-volt to 3.6-volt EPROM	Now
AT27BV020	256K x 8	90 - 150 ns	2M-bit, 2.7-volt to 3.6-volt EPROM	Now
AT27BV040	512K x 8	120 - 150 ns	4M-bit, 2.7-volt to 3.6-volt EPROM	Now
AT27BV4096	256K x 16	120 - 150 ns	4M-bit, 2.7-volt to 3.6-volt EPROM	Now
AT27BV400	512K x 8 / 256K x 16	150 ns	4M-bit, Byte-selectable 2.7-volt to 3.6-volt EPROM	Now
AT27BV800	1M x 8 / 512K x 16	150 ns	8M-bit, Byte-selectable 2.7-volt to 3.6-volt EPROM	Now
Low-voltage (3.0 to 3.6V)				
AT27LV256A	32K x 8	55 - 150 ns	256K-bit, 3-volt EPROM	Now
AT27LV512A	64K x 8	70 - 150 ns	512K-bit, 3-volt EPROM	Now
AT27LV520	64K x 8	90 ns	512K-bit, Latched 3-volt EPROM	Now
AT27LV010A	128K x 8	70 - 150 ns	1M-bit, 3-volt EPROM	Now
AT27LV1026	2 x 32K x 16	35 - 55 ns	1M-bit, Interleaved 3-volt EPROM	Now
AT27LV020A	256K x 8	90 - 150 ns	2M-bit, 3-volt EPROM	Now
AT27LV040A	512K x 8	90 - 150 ns	4M-bit, 3-volt EPROM	Now
Standard Voltage (5.0V)				
AT27C256R	32K x 8	45 - 150 ns	256K-bit, 5-volt EPROM	Now
AT27C512R	64K x 8	45 - 150 ns	512K-bit, 5-volt EPROM	Now
AT27C516	32K x 16	45 - 100 ns	512K-bit, 5-volt EPROM	Now
AT27C520	64K x 8	70 - 90 ns	512K-bit, Latched 5-volt EPROM	Now
AT27C010(L)	128K x 8	45 - 150 ns	1M-bit, 5-volt EPROM Standard and Low-power	Now
AT27C1024	64K x 16	45 - 150 ns	1M-bit, 5-volt EPROM	Now
AT27C020	256K x 8	55 - 150 ns	2M-bit, 5-volt EPROM	Now
AT27C2048	128K x 16	55 - 150 ns	2M-bit, 5-volt EPROM	Now
AT27C040	512K x 8	70 - 150 ns	4M-bit, 5-volt EPROM	Now
AT27C4096	256K x 16	55 - 150 ns	4M-bit, 5-volt EPROM	Now
AT27C400	512K x 8 / 256K x 16	90 - 150 ns	4M-bit, Byte-selectable 5-volt EPROM	Now
AT27C080	1M x 8	90 - 150 ns	8M-bit, 5-volt EPROM	Now
AT27C800	1M x 8 / 512K x 16	100 - 150 ns	8M-bit, Byte-selectable 5-volt EPROM	Now

2.5 Intel, 28F008SA – 1-Mbyte Flash Memory

The 28F008SA is a critical memory component of the Ampro CoreModule 3SXi card and contains the system BIOS software, which controls low-level interactions between the CPU and peripheral ICs. The specifications and data sheet for this IC are included for extra clarity.

2.5.1 Data Sheet for the 28F008SA

The following is the vendor's data sheet for the Intel 28F008SA 1-Mbyte flash memory, which was downloaded from the Intel Internet site at:

ftp://download.intel.com/design/flcomp/datashts/29042908.pdf.

PRELIMINARY

5 VOLT FlashFile™ MEMORY

28F008SA (x8)

- High-Density Symmetrically-Blocked Architecture
  - Sixteen 64-Kbyte Blocks
- Extended Cycling Capability
  - 100,000 Block Erase Cycles
  - 1.6 Million Block Erase Cycles per Chip
- Automated Byte Write and Block Erase
  - Command User Interface
  - Status Register
- System Performance Enhancements
  - RY/BY# Status Output
  - Erase Suspend Capability
- Deep Power-Down Mode
  - 0.20 µA I_{CC} Typical
- Very High-Performance Read
  - 85 ns Maximum Access Time
- SRAM-Compatible Write Interface
- Hardware Data Protection Feature
  - Erase/Write Lockout during Power Transitions
- Industry Standard Packaging
  - 40-Lead TSOP, 44-Lead PSOP
- ETOX™ V Nonvolatile Flash Technology
  - 12 V Byte Write/Block Erase

The 5 Volt FlashFile™ memory 28F008SA's extended cycling, symmetrically blocked architecture, fast access time, write automation and low power consumption provide a more reliable, lower power, lighter weight and higher performance alternative to traditional rotating disk technology. The 28F008SA brings new capabilities to portable computing. Application and operating system software stored in resident flash memory arrays provide instant-on, rapid eXecute-In-Place (XIP) and protection from obsolescence through in-system software updates. Resident software also extends system battery life and increases reliability by reducing disk drive accesses.

For high-density data acquisition applications, the 28F008SA offers a more cost-effective and reliable alternative to SRAM and battery. Traditional high-density embedded applications, such as telecommunications, can take advantage of the 28F008SA's nonvolatility, blocking and minimal system code requirements for flexible firmware and modular software designs.

The 28F008SA is offered in 40-lead TSOP and 44-lead PSOP packages. Pin assignments simplify board layout when integrating multiple devices in a flash memory array or subsystem. This device uses an integrated Command User Interface and state machine for simplified block erasure and byte write. The 28F008SA memory map consists of 16 separately erasable 64-Kbyte blocks.

Intel® 28F008SA employs advanced CMOS circuitry for systems requiring low power consumption and noise immunity. Its 85 ns access time provides superior performance when compared with magnetic storage media. A deep power-down mode lowers power consumption to 1 µW typical through V_{CC}, crucial in portable computing, handheld instrumentation and other low-power applications. The RP# power control input also provides absolute data protection during system power-up/down.

Manufactured on Intel® 0.4 micron ETOX V process technology, the 28F008SA provides the highest levels of quality, reliability and cost-effectiveness.

NOTE: This document formerly known as *28F008SA 8-Mbit (1-Mbit x 8) FlashFile™ Memory.*

December 1998

Order Number: 290429-008

Information in this document is provided in connection with Intel products. No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted by this document. Except as provided in Intel's Terms and Conditions of Sale for such products, Intel assumes no liability whatsoever, and Intel disclaims any express or implied warranty, relating to sale and/or use of Intel products including liability or warranties relating to fitness for a particular purpose, merchantability, or infringement of any patent, copyright or other intellectual property right. Intel products are not intended for use in medical, life saving, or life sustaining applications.

Intel may make changes to specifications and product descriptions at any time, without notice.

The 28F008SA may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Copies of documents which have an ordering number and are referenced in this document, or other Intel literature, may be obtained from:

Intel Corporation
P.O. Box 5937
Denver, CO 80217-9808

or call 1-800-548-4725
or visit Intel's website at http://www.intel.com

CG-041493

CONTENTS

PRELIMINARY

REVISION HISTORY

Number	Description
-002	Revised from Advanced Information to Preliminary Modified Erase Suspend Flowchart Removed -90 speed bin Integrated -90 characteristics into -85 speed bin Combined V_{PP} Standby current and V_{PP} Read current into one V_{PP} Standby current spec with two test conditions (DC Characteristics table) Lowered V_{LKO} from 2.2 V to 2.0 V.
-004	PWD renamed to RP# for JEDEC standardization compatibility. Changed I_{PPS} Standby current spec from ±10 µA to ±15 µA in DC Characteristics table.
-005	Added Extended Temperature Specs for 28F008SA Added I_{PPR} Spec Corrected I_{PPS} Spec Type Added V_{OHZ} (Output High Voltage—CMOS) Spec Added Byte Write Time Spec
-006	Minor changes throughout document. Added reset specifications. All components used prior to the publication date of this datasheet are not affected by the new specification. Only devices used after this date must adhere to this new specification.
-007	Removed references to reverse pinout throughout document. Added section numbers.
-008	Changed document title from *28F008SA 8-Mbit (1-Mbit x 8) FlashFile™ Memory.*

1.0 PRODUCT OVERVIEW

The 28F008SA is a high-performance 8-Mbit (8,388,608 bit) memory organized as 1 Mbyte (1,048,576 bytes) of 8 bits each. Sixteen 64-Kbyte (65,536 byte) blocks are included on the 28F008SA. A memory map is shown in Figure 5 of this specification. A block erase operation erases one of the sixteen blocks of memory in typically 1.6 seconds, independent of the remaining blocks. Each block can be independently erased and written 100,000 cycles. Erase suspend mode allows system software to suspend block erase to read data or execute code from any other block of the 28F008SA.

The 28F008SA is available in the 40-lead TSOP (Thin Small Outline Package, 1.2 mm thick) and 44-lead PSOP (Plastic Small Outline) packages. Pinouts are shown in Figures 2 and 3 of this specification.

The Command User Interface (CUI) serves as the interface between the microprocessor or microcontroller and the internal operation of the 28F008SA.

Byte Write and Block Erase Automation allow byte write and block erase operations to be executed using a two-write command sequence to the CUI. The internal Write State Machine (WSM) automatically executes the algorithms and timings necessary for byte write and block erase operations, including verifications, thereby unburdening the microprocessor or microcontroller. Writing of memory data is performed in byte increments typically within 9 µs—an 80% improvement over current flash memory products. I_{PP} byte write and block erase currents are 10 mA typical, 30 mA maximum. V_{PP} byte write and block erase voltage is 11.4 V to 12.6 V.

The status register indicates the status of the WSM and when the WSM successfully completes the desired byte write or block erase operation.

The RY/BY# output gives an additional indicator of WSM activity, providing capability for both hardware signal of status (versus software polling) and status masking (interrupt masking for background erase, for example). Status polling using RY/BY# minimizes both CPU overhead and system power consumption. When low, RY/BY# indicates that the WSM is performing a block erase or byte write operation. RY/BY# high indicates that the WSM is ready for new commands, block erase is suspended or the device is in deep power-down mode.

Maximum access time is 85 ns (t_{ACC}) over the commercial temperature range (0 °C to +70 °C) and over V_{CC} supply voltage range (4.5 V to 5.5 V and 4.75 V to 5.25 V). I_{CC} active current (CMOS Read) is 20 mA typical, 35 mA maximum at 8 MHz.

When the CE# and RP# pins are at V_{CC}, the I_{CC} CMOS standby mode is enabled.

A deep power-down mode is enabled when the RP# pin is at GND, minimizing power consumption and providing write protection. I_{CC} current in deep power-down is 0.20 µA typical. Reset time of 400 ns is required from RP# switching high until outputs are valid to read attempts. Equivalently, the device has a wake time of 1 µs from RP# high until writes to the CUI are recognized by the 28F008SA. With RP# at GND, the WSM is reset and the status register is cleared.

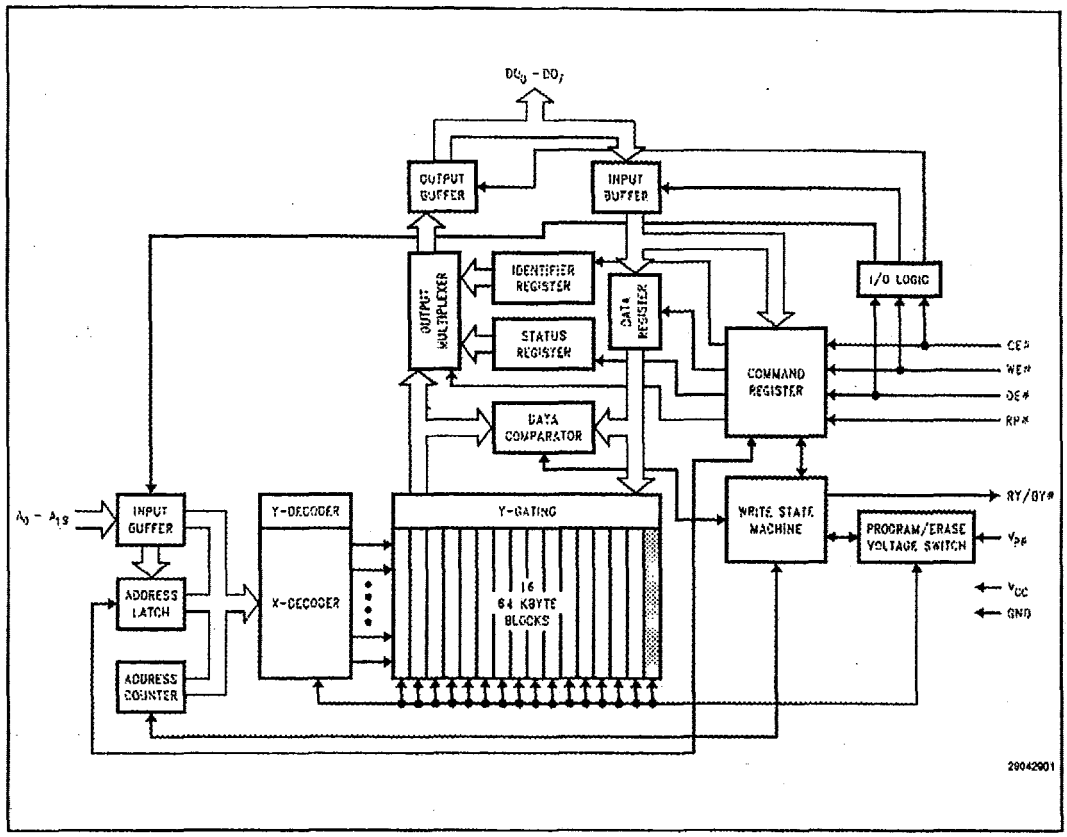


Figure 1. Block Diagram

Table 1. Pin Descriptions

Symbol	Type	Name and Function
A_0–A_{19}	INPUT	**ADDRESS INPUTS** for memory addresses. Addresses are internally latched during a write cycle.
DQ_0–DQ_7	INPUT/OUTPUT	**DATA INPUT/OUTPUTS:** Inputs data and commands during CUI write cycles; outputs data during memory array, status register and identifier read cycles. The data pins are active high and float to tri-state off when the chip is deselected or the outputs are disabled. Data is internally latched during a write cycle.
CE#	INPUT	**CHIP ENABLE:** Activates the device's control logic, input buffers, decoders, and sense amplifiers. CE# is active low; CE# high deselects the memory device and reduces power consumption to standby levels.
RP#	INPUT	**RESET/DEEP POWER-DOWN:** Puts the device in deep power-down mode. RP# is active low; RP# high gates normal operation. RP# also locks out block erase or byte write operations when active low, providing data protection during power transitions. RP# active resets internal automation. Exit from deep power-down sets device to read-array mode.
OE#	INPUT	**OUTPUT ENABLE:** Gates the device's outputs through the data buffers during a read cycle. OE# is active low.
WE#	INPUT	**WRITE ENABLE:** Controls writes to the CUI and array blocks. WE# is active low. Addresses and data are latched on the rising edge of the WE# pulse.
RY/BY#	OUTPUT	**READY/BUSY#:** Indicates the status of the internal Write State Machine. When low, it indicates that the WSM is performing a block erase or byte write operation. RY/BY# high indicates that the WSM is ready for new commands, block erase is suspended or the device is in deep power-down mode. RY/BY# is always active and does **NOT** float to tri-state off when the chip is deselected or data outputs are disabled.
V_{PP}		**BLOCK ERASE/BYTE WRITE POWER SUPPLY** for erasing blocks of the array or writing bytes of each block. **NOTE:** With $V_{PP} < V_{PPLMAX}$, memory contents cannot be altered.
V_{CC}		**DEVICE POWER SUPPLY (5 V ±10%, 5 V ±5%)**
GND		**GROUND**

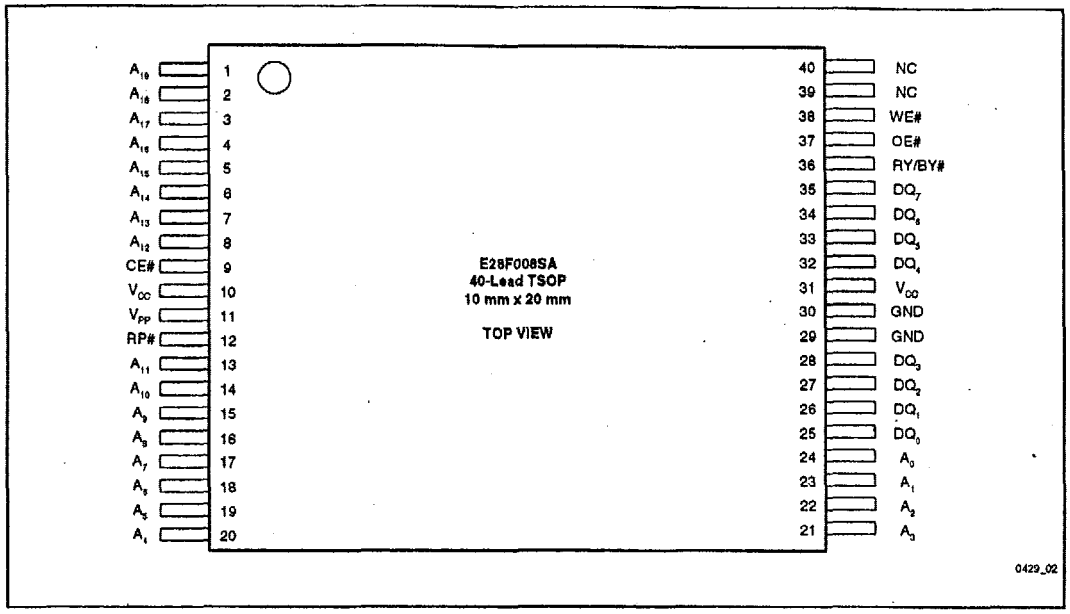


Figure 2. TSOP Lead Configurations

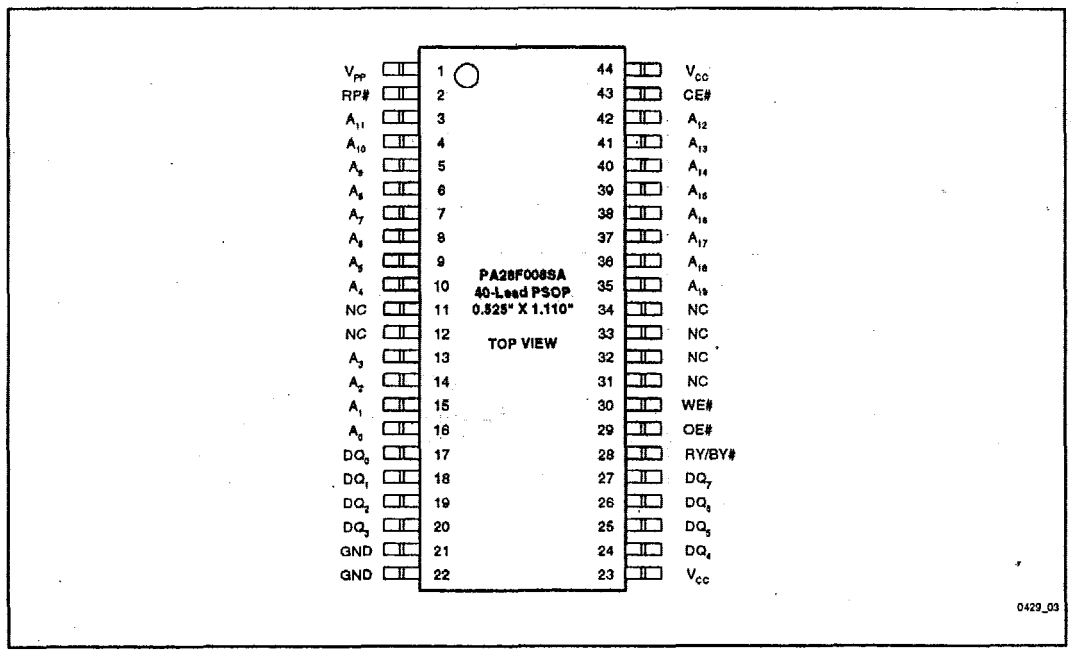


Figure 3. PSOP Lead Configuration

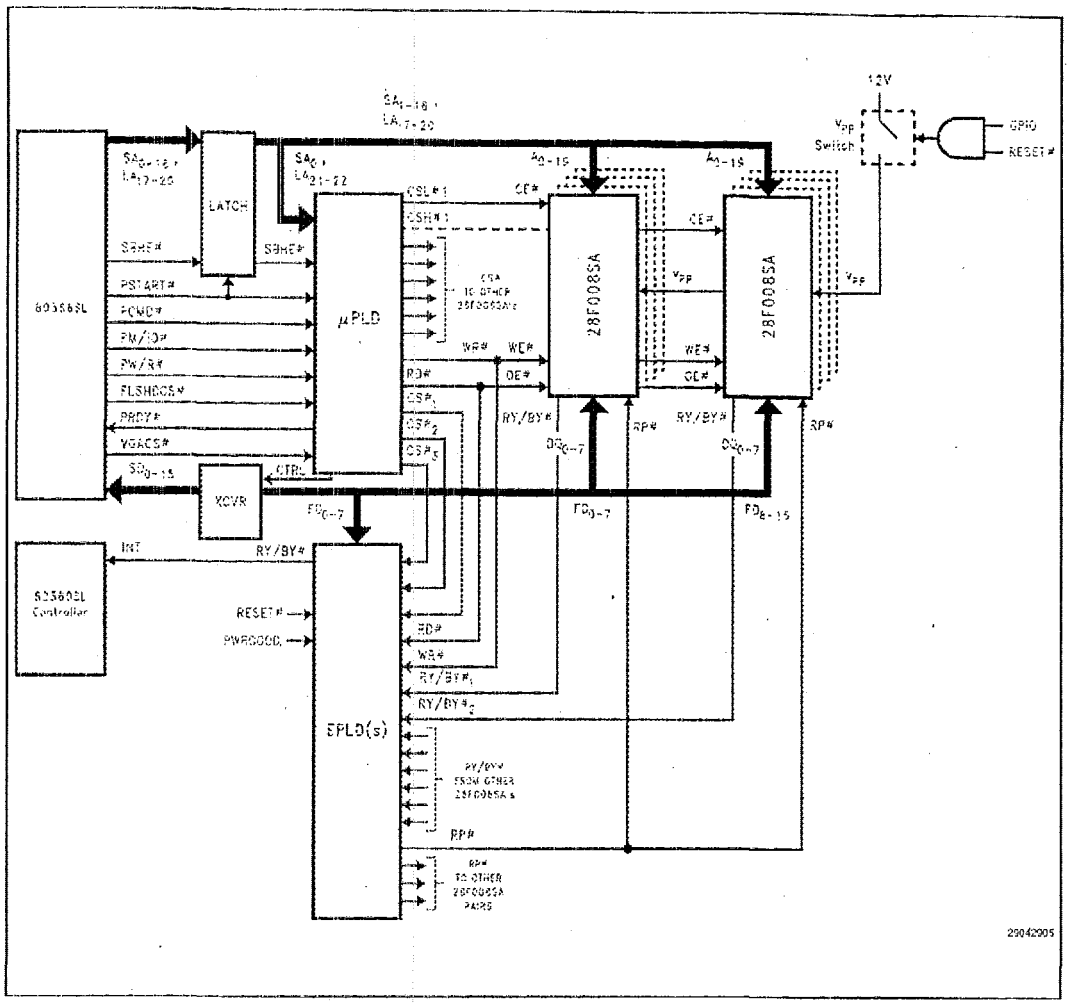


Figure 4. 28F008SA Array Interface to Intel386SL Microprocessor Superset through PI Bus (Including RY/BY# Masking and Selective Power-Down), for DRAM Backup during System SUSPEND, Resident O/S and Applications and Motherboard Solid-State Disk.

2.0 PRINCIPLES OF OPERATION

The 28F008SA includes on-chip write automation to manage write and erase functions. The Write State Machine (WSM) allows for 100% TTL-level control inputs, fixed power supplies during block erasure and byte write, and minimal processor overhead with RAM-like interface timings.

After initial device power-up, or after return from deep power-down mode (see *Bus Operations*), the 28F008SA functions as a read-only memory. Manipulation of external memory-control pins allow array read, standby and output disable operations. Both status register and intelligent identifiers can also be accessed through the CUI when $V_{PP} = V_{PPL}$.

This same subset of operations is also available when high voltage is applied to the V_{PP} pin. In addition, high voltage on V_{PP} enables successful block erasure and byte writing of the device. All functions associated with altering memory contents—byte write, block erase, status and intelligent identifier—are accessed via the CUI and verified through the status register.

Commands are written using standard microprocessor write timings. CUI contents serve as input to the WSM, which controls the block erase and byte write circuitry. Write cycles also internally latch addresses and data needed for byte write or block erase operations. With the appropriate command written to the register, standard microprocessor read timings output array data, access the intelligent identifier codes, or output byte write and block erase status for verification.

Interface software to initiate and poll progress of internal byte write and block erase can be stored in any of the 28F008SA blocks. This code is copied to, and executed from, system RAM during actual flash memory update. After successful completion of byte write and/or block erase, code/data reads from the 28F008SA are again possible via the Read Array command. Erase suspend/resume capability allows system software to suspend block erase to read data and execute code from any other block.

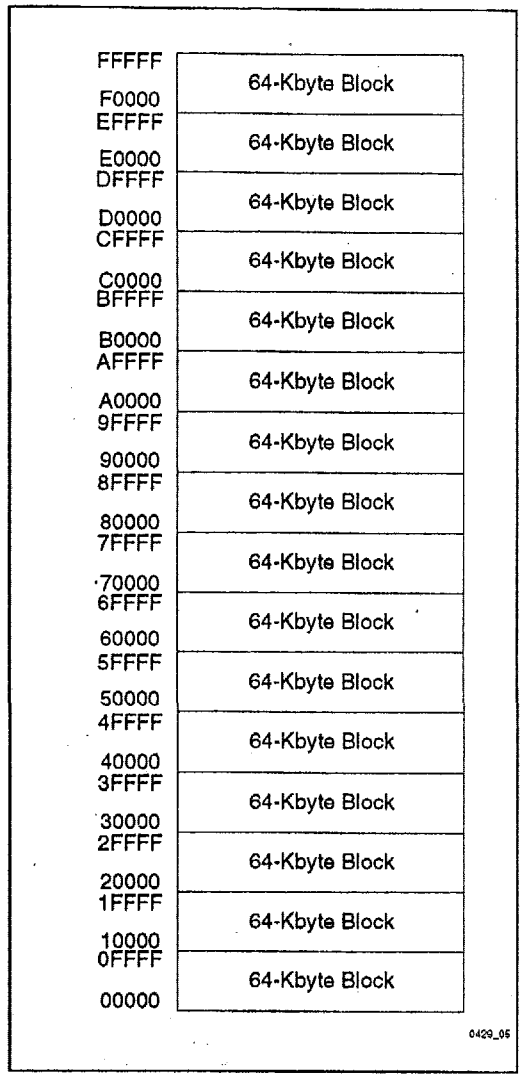


Figure 5. Memory Map

2.1 Command User Interface and Write Automation

An on-chip state machine controls block erase and byte write, freeing the system processor for other tasks. After receiving the Erase Setup and Erase Confirm commands, the state machine controls block pre-conditioning and erase, returning progress via the status register and RY/BY# output. Byte write is similarly controlled, after destination address and expected data are supplied. The program and erase algorithms of past Intel® Flash memories are now regulated by the state machine, including pulse repetition where required and internal verification and margining of data.

2.2 Data Protection

Depending on the application, the system designer may choose to make the V_{PP} power supply switchable (available only when memory byte writes/block erases are required) or hardwired to V_{PPH}. When $V_{PP} = V_{PPL}$, memory contents cannot be altered. The 28F008SA CUI architecture provides protection from unwanted byte write or block erase operations even when high voltage is applied to V_{PP}. Additionally, all functions are disabled whenever V_{CC} is below the write lockout voltage V_{LKO}, or when RP# is at V_{IL}. The 28F008SA accommodates either design practice and encourages optimization of the processor-memory interface.

The two-step byte write/block erase CUI write sequence provides additional software write protection.

3.0 BUS OPERATION

Flash memory reads, erases and writes in-system via the local CPU. All bus cycles to or from the flash memory conform to standard microprocessor bus cycles.

3.1 Read

The 28F008SA has three read modes. The memory can be read from any of its blocks, and information can be read from the intelligent identifier or status register. V_{PP} can be at either V_{PPL} or V_{PPH}.

The first task is to write the appropriate read mode command to the CUI (array, intelligent identifier, or status register). The 28F008SA automatically resets to read array mode upon initial device power-up or after exit from deep power-down. The 28F008SA has four control pins, two of which must be logically active to obtain data at the outputs. Chip Enable (CE#) is the device selection control, and when active enables the selected memory device. Output Enable (OE#) is the data input/output (DQ_0–DQ_7) direction control, and when active drives data from the selected memory onto the I/O bus. RP# and WE# must also be at V_{IH}. Figure 13 illustrates read bus cycle waveforms.

3.2 Output Disable

With OE# at a logic-high level (V_{IH}), the device outputs are disabled. Output pins (DQ_0–DQ_7) are placed in a high-impedance state.

3.3 Standby

CE# at a logic-high level (V_{IH}) places the 28F008SA in standby mode. Standby operation disables much of the 28F008SA's circuitry and substantially reduces device power consumption. The outputs (DQ_0–DQ_7) are placed in a high-impedance state independent of the status of OE#. If the 28F008SA is deselected during block erase or byte write, the device will continue functioning and consuming normal active power until the operation completes.

Table 2. Bus Operations

Mode	Notes	RP#	CE#	OE#	WE#	A_0	V_{PP}	DQ_{0-7}	RY/BY#
Read	1, 2, 3	V_{IH}	V_{IL}	V_{IL}	V_{IH}	X	X	D_{OUT}	X
Output Disable	1, 2, 3	V_{IH}	V_{IL}	V_{IH}	V_{IH}	X	X	High Z	X
Standby	1, 2, 3	V_{IH}	V_{IH}	X	X	X	X	High Z	X
Deep Power-Down	1, 2	V_{IL}	X	X	X	X	X	High Z	V_{OH}
Intelligent Identifier (Mfr)	1, 2	V_{IH}	V_{IL}	V_{IL}	V_{IH}	V_{IL}	X	89H	V_{OH}
Intelligent Identifier (Device)	1, 2	V_{IH}	V_{IL}	V_{IL}	V_{IH}	V_{IH}	X	A2H	V_{OH}
Write	1,2,3,4,5	V_{IH}	V_{IL}	V_{IH}	V_{IL}	X	X	D_{IN}	X

NOTES:

1. Refer to *DC Characteristics*. When $V_{PP} = V_{PPL}$, memory contents can be read but not written or erased.
2. X can be V_{IL} or V_{IH} for control pins and addresses, and V_{PPL} or V_{PPH} for V_{PP}. See *DC Characteristics* for V_{PPL} and V_{PPH} voltages.
3. RY/BY# is V_{OL} when the Write State Machine is executing internal block erase or byte write algorithms. It is V_{OH} when the WSM is not busy, in erase suspend mode or deep power-down mode.
4. Command writes involving block erase or byte write are only successfully executed when $V_{PP} = V_{PPH}$.
5. Refer to Table 3 for valid D_{IN} during a write operation.

3.4 Deep Power-Down

The 28F008SA offers a deep power-down feature, entered when RP# is at V_{IL}. Current draw through V_{CC} is 0.20 µA typical in deep power-down mode, with current draw through V_{PP} typically 0.1 µA. During read modes, RP#-low deselects the memory, places output drivers in a high-impedance state and turns off all internal circuits. The 28F008SA requires time t_{PHQV} (see *AC Characteristics-Read-Only Operations*) after return from power-down until initial memory access outputs are valid. After this wakeup interval, normal operation is restored. The CUI is reset to Read Array, and the upper 5 bits of the status register are cleared to value 10000, upon return to normal operation.

During block erase, program or lock-bit configuration, RP#-low will abort the operation. RY/BY# remains low until the reset operation is complete. Memory contents being altered are no longer valid; the data may be partially erased or written. Time t_{PHWL} is required after RP# goes to logic-high (V_{IH}) before another command can be written.

This use of RP# during system reset is important with automated write/erase devices. When the system comes out of reset it expects to read from the flash memory. Automated flash memories provide status information when accessed during write/erase modes. If a CPU reset occurs with no flash memory reset, proper CPU initialization would not occur because the flash memory would be providing the status information instead of array data. Intel's Flash memories allow proper CPU initialization following a system reset through the use of the RP# input. In this application RP# is controlled by the same RESET# signal that resets the system CPU.

3.5 Intelligent Identifier Operation

The intelligent identifier operation outputs the manufacturer code, 89H; and the device code, A2H for the 28F008SA. The system CPU can then automatically match the device with its proper block erase and byte write algorithms.

The manufacturer- and device-codes are read via the CUI. Following a write of 90H to the CUI, a read from address location 00000H outputs the manufacturer code (89H). A read from address 00001H outputs the device code (A2H). It is not necessary to have high voltage applied to V_{PP} to read the intelligent identifiers from the CUI.

3.6 Write

Writes to the CUI enable reading of device data and Intelligent Identifiers. They also control inspection and clearing of the status register. Additionally, when $V_{PP} = V_{PPH}$, the CUI controls block erasure and byte write. The contents of the interface register serve as input to the internal state machine.

The CUI itself does not occupy an addressable memory location. The interface register is a latch used to store the command and address and data information needed to execute the command. Erase Setup and Erase Confirm commands require both appropriate command data and an address within the block to be erased. The Byte Write Setup command requires both appropriate command data and the address of the location to be written, while the Byte Write command consists of the data to be written and the address of the location to be written.

The CUI is written by bringing WE# to a logic-low level (V_{IL}) while CE# is low. Addresses and data are latched on the rising edge of WE#. Standard microprocessor write timings are used.

Refer to *AC Write Characteristics* and the AC *Waveforms for Write Operations*, Figure 15, for specific timing parameters.

4.0 COMMAND DEFINITIONS

When V_{PPL} is applied to the V_{PP} pin, read operations from the status register, intelligent identifiers, or array blocks are enabled. Placing V_{PPH} on V_{PP} enables successful byte write and block erase operations as well.

Device operations are selected by writing specific commands into the CUI. Table 3 defines the 28F008SA commands.

4.1 Read Array Command

Upon initial device power-up and after exit from deep power-down mode, the 28F008SA defaults to read array mode. This operation is also initiated by writing FFH into the CUI. Microprocessor read cycles retrieve array data. The device remains enabled for reads until the CUI contents are altered. Once the internal WSM has started a block erase or byte write operation, the device will not recognize the Read Array command, until the WSM has completed its operation. The Read Array command is functional when $V_{PP} = V_{PPL}$ or V_{PPH}.

Table 3. Command Definitions

Command	Bus Cycles Req'd	Notes	First Bus Cycle			Second Bus Cycle		
			Oper(1)	Addr (2)	Data(3)	Oper(1)	Addr (2)	Data(3)
Read Array/Reset	1		Write	X	FFH			
Intelligent Identifier	3	4	Write	X	90H	Read	IA	IID
Read Status Register	2		Write	X	70H	Read	X	SRD
Clear Status Register	1		Write	X	50H			
Erase Setup/Erase Confirm	2		Write	BA	20H	Write	BA	D0H
Erase Suspend/Erase Resume	2		Write	X	B0H	Write	X	D0H
Byte Write Setup/Write	2	5	Write	WA	40H	Write	WA	WD
Alternate Byte Write Setup/Write	2	5	Write	WA	10H	Write	WA	WD

NOTES:

1. Bus operations are defined in Table 2.
2. IA = Identifier Address: 00H for manufacturer code, 01H for device code.
   BA = Address within the block being erased.
   WA = Address of memory location to be written.
3. SRD = Data read from status register. See Table 4 for a description of the status register bits.
   WD = Data to be written at location WA. Data is latched on the rising edge of WE#.
   IID = Data read from Intelligent Identifiers.
4. Following the Intelligent Identifier command, two read operations access manufacture and device codes.
5. Either 40H or 10H are recognized by the WSM as the Byte Write Setup command.
6. Commands other than those shown above are reserved by Intel for future device implementations and should not be used.

4.2 Intelligent Identifier Command

The 28F008SA contains an intelligent identifier operation, initiated by writing 90H into the CUI. Following the command write, a read cycle from address 00000H retrieves the manufacturer code of 89H. A read cycle from address 00001H returns the device code of A2H. To terminate the operation, it is necessary to write another valid command into the register. Like the Read Array command, the Intelligent Identifier command is functional when $V_{PP} = V_{PPL}$ or V_{PPH}.

4.3 Read Status Register Command

The 28F008SA contains a status register which may be read to determine when a byte write or block erase operation is complete, and whether that operation completed successfully. The status register may be read at any time by writing the Read Status Register command (70H) to the CUI. After writing this command, all subsequent read operations output data from the status register, until another valid command is written to the CUI. The contents of the status register are latched on the falling edge of OE# or CE#, whichever occurs last in the read cycle. OE# or CE# must be toggled to V_{IH} before further reads to update the status register latch. The Read Status Register command functions when $V_{PP} = V_{PPL}$ or V_{PPH}.

4.4 Clear Status Register Command

The erase status and byte write status bits are set to "1"s by the Write State Machine and can only be reset by the Clear Status Register command. These bits indicate various failure conditions (see Table 4). By allowing system software to control the resetting of these bits, several operations may be performed (such as cumulatively writing several bytes or erasing multiple blocks in sequence). The status register may then be polled to determine if an error occurred during that sequence. This adds flexibility to the way the device may be used.

Additionally, the V_{PP} status bit (SR.3) **must** be reset by system software before further byte writes or block erases are attempted. To clear the Status Register, the Clear Status Register command (50H) is written to the CUI. The Clear Status Register command is functional when $V_{PP} = V_{PPL}$ or V_{PPH}.

4.5 Erase Setup/Erase Confirm Commands

Erase is executed one block at a time, initiated by a two-cycle command sequence. An Erase Setup command (20H) is first written to the CUI, followed by the Erase Confirm command (D0H). These commands require both appropriate sequencing and an address within the block to be erased to FFH. Block preconditioning, erase and verify are all handled internally by the WSM, invisible to the system. After the two-command erase sequence is written to it, the 28F008SA automatically outputs status register data when read (see Figure 6; *Automated Block Erase Flowchart*). The CPU can detect the completion of the erase event by analyzing the output of the RY/BY# pin, or the WSM status bit of the status register.

When erase is completed, the erase status bit should be checked. If erase error is detected, the status register should be cleared. The CUI remains in read status register mode until further commands are issued to it.

This two-step sequence of set-up followed by execution ensures that memory contents are not accidentally erased. Also, reliable block erasure can only occur when $V_{PP} = V_{PPH}$. In the absence of this high voltage, memory contents are protected against erasure. If block erase is attempted while $V_{PP} = V_{PPL}$, the V_{PP} status bit will be set to "1." Erase attempts while $V_{PPL} < V_{PP} < V_{PPH}$ produce spurious results and should not be attempted.

4.6 Erase Suspend/Erase Resume Commands

The Erase Suspend command allows block erase interruption in order to read data from another block of memory. Once the erase process starts, writing the erase suspend command (B0H) to the CUI requests that the WSM suspend the erase sequence at a predetermined point in the erase algorithm. The 28F008SA continues to output status register data when read, after the Erase Suspend command is written to it. Polling the WSM status and erase suspend status bits will determine when the erase operation has been suspended (both will be set to "1"). RY/BY# will also transition to V_{OH}.

At this point, a Read Array command can be written to the CUI to read data from blocks other than that which is suspended. The only other valid commands at this time are Read Status Register (70H) and Erase Resume (D0H), at which time the WSM will continue with the erase process. The erase suspend status and WSM status bits of the status register will be automatically cleared and RY/BY# will return to V_{OL}. After the Erase Resume command is written to it, the 28F008SA automatically outputs status register data when read (see Figure 7; *Erase Suspend/Resume Flowchart*). V_{PP} must remain at V_{PPH} while the 28F008SA is in Erase Suspend.

4.7 Byte Write Setup/Write Commands (40H or 10H)

Byte write is executed by a two-command sequence. The Byte Write Setup command (40H or 10H) is written to the CUI, followed by a second write specifying the address and data (latched on the rising edge of WE#) to be written. The WSM then takes over, controlling the byte write and write verify algorithms internally. After the two-command byte write sequence is written to it, the 28F008SA automatically outputs status register data when read (see Figure 8; *Automated Byte Write Flowchart*). The CPU can detect the completion of the byte write event by analyzing the output of the RY/BY# pin, or the WSM status bit of the status register. Only the Read Status Register command is valid while byte write is active.

When byte write is complete, the byte write status bit should be checked. If byte write error is detected, the status register should be cleared. The internal WSM verify only detects errors for "1"s that do not successfully write to "0"s. The CUI remains in read status register mode until further commands are issued to it. If byte write is attempted while $V_{PP} = V_{PPL}$, the V_{PP} status bit will be set to "1." Byte write attempts while $V_{PPL} < V_{PP} < V_{PPH}$ produce spurious results and should not be attempted.

5.0 EXTENDED BLOCK ERASE/BYTE WRITE CYCLING

Intel has designed extended cycling capability into its ETOX flash memory technologies. The 28F008SA is designed for 100,000 byte write/block erase cycles on each of the sixteen 64-Kbyte blocks. Low electric fields, advanced oxides and minimal oxide area per cell subjected to the tunneling electric field combine to greatly reduce oxide stress and the probability of failure. A 20-Mbyte solid-state drive using an array of 28F008SAs has a MTBF (Mean Time Between Failure) of 33.3 million hours[1], over 600 times more reliable than equivalent rotating disk technology.

6.0 AUTOMATED BYTE WRITE

The 28F008SA integrates the Quick-Pulse programming algorithm of prior Intel Flash devices on-chip, using the CUI, status register and WSM. On-chip integration dramatically simplifies system software and provides processor interface timings to the CUI and status register. WSM operation, internal verify and V_{PP} high voltage presence are monitored and reported via the RY/BY# output and appropriate status register bits. Figure 8, *Automated Byte Write Flowchart*, shows a system software flowchart for device byte write. The entire sequence is performed with V_{PP} at V_{PPH}. Byte write abort occurs when RP# transitions to V_{IL}, or V_{PP} drops to V_{PPL}. Although the WSM is halted, byte data is partially written at the location where byte write was aborted. Block erasure, or a repeat of byte write, is required to initialize this data to a known value.

7.0 AUTOMATED BLOCK ERASE

As above, the Quick-Erase algorithm of prior Intel Flash devices is now implemented internally, including all preconditioning of block data. WSM operation, erase success and V_{PP} high voltage presence are monitored and reported through RY/BY# and the status register. Additionally, if a command other than Erase Confirm is written to the device following Erase Setup, both the Erase status and Byte Write status bits will be set to "1"s. When issuing the Erase Setup and Erase Confirm commands, they should be written to an address within the address range of the block to be erased. Figure 6, *Automated Block Erase Flowchart*, shows a system software flowchart for block erase.

---

[1] Assumptions: 10-Kbyte file written every 10 minutes. (20-Mbyte array)/(10-Kbyte file) = 2,000 file writes before erase required.

(2000 files writes/erase) x (100,000 cycles per 28F008SA block) = 200 million file writes.

(200 x 10^4 file writes) x (10 min/write) x (1 hr/60 min) = 33.3 x 10^4 MTBF.

Erase typically takes 1.6 seconds per block. The Erase Suspend/Erase Resume command sequence allows suspension of this erase operation to read data from a block other than that in which erase is being performed. A system software flowchart is shown in Figure 7, *Erase Suspend/Resume Flowchart*.

The entire sequence is performed with V_{PP} at V_{PPH}. Abort occurs when RP# transitions to V_{IL} or V_{PP} falls to V_{PPL}, while erase is in progress. Block data is partially erased by this operation, and a repeat of erase is required to obtain a fully erased block.

8.0 DESIGN CONSIDERATIONS

8.1 Three-Line Output Control

The 28F008SA will often be used in large memory arrays. Intel provides three control inputs to accommodate multiple memory connections. Three-line control provides for:

a) lowest possible memory power dissipation

b) complete assurance that data bus contention will not occur

To efficiently use these control inputs, an address decoder should enable CE#, while OE# should be connected to all memory devices and the system's READ# control line. This assures that only selected memory devices have active outputs while deselected memory devices are in standby mode. RP# should be connected to the system POWERGOOD signal to prevent unintended writes during system power transitions. POWERGOOD should also toggle during system reset.

8.2 RY/BY# and Byte Write/Block Erase Polling

RY/BY# is a full CMOS output that provides a hardware method of detecting byte write and block erase completion. It transitions low time t_{WHRL} after a write or erase command sequence is written to the 28F008SA, and returns to V_{OH} when the WSM has finished executing the internal algorithm.

RY/BY# can be connected to the interrupt input of the system CPU or controller. It is active at all times, not tri-stated if the 28F008SA CE# or OE# inputs are brought to V_{IH}. RY/BY# is also V_{OH} when the device is in erase suspend or deep power-down modes.

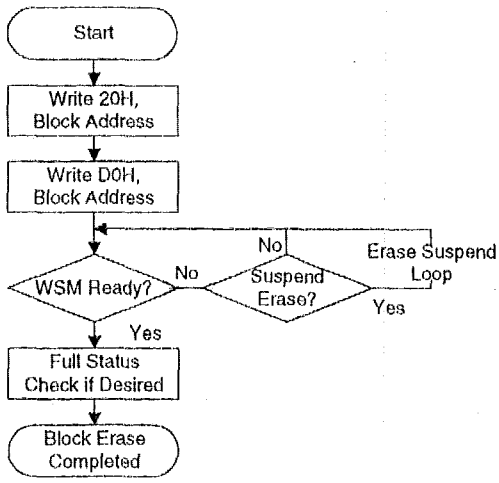
8.3 Power Supply Decoupling

Flash memory power switching characteristics require careful device decoupling. System designers are interested in three supply current issues; standby current levels (I_{SB}), active current levels (I_{CC}) and transient peaks produced by falling and rising edges of CE#. Transient current magnitudes depend on the device outputs' capacitive and inductive loading. Two-line control and proper decoupling capacitor selection will suppress transient voltage peaks. Each device should have a 0.1 µF ceramic capacitor connected between each V_{CC} and GND, and between its V_{PP} and GND. These high frequency, low inherent-inductance capacitors should be placed as close as possible to package leads. Additionally, for every eight devices, a 4.7 µF electrolytic capacitor should be placed at the array's power supply connection between V_{CC} and GND. The bulk capacitor will overcome voltage slumps caused by PC board trace inductances.

Table 4. Status Register Definitions

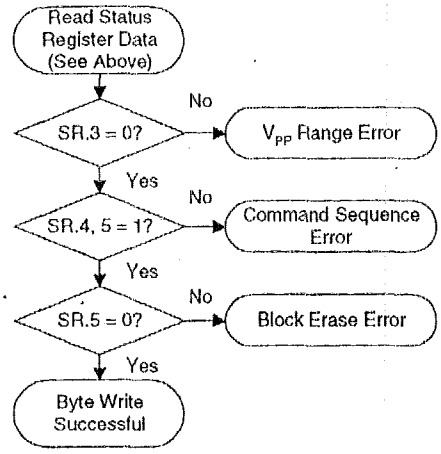
WSMS	ESS	ES	BWS	VPPS	R	R	R
7	6	5	4	3	2	1	0

	NOTES:
R.7 = WRITE STATE MACHINE STATUS 1 = Ready 0 = Busy	RY/BY# or the Write State Machine status bit must first be checked to determine byte write or block erase completion, before the Byte Write or Erase status bit are checked for success.
SR.6 = ERASE SUSPEND STATUS 1 = Erase Suspended 0 = Erase in Progress/Completed	
SR.5 = ERASE STATUS 1 = Error in Block Erasure 0 = Successful Block Erase	
SR.4 = BYTE WRITE STATUS 1 = Error in Byte Write 0 = Successful Byte Write	If the Byte Write **and** Erase status bits are set to "1"s during a block erase attempt, an improper command sequence was entered. Attempt the operation again.
SR.3 = V_{PP} STATUS 1 = V_{PP} Low Detect; Operation Abort 0 = V_{PP} OK	If V_{PP} low status is detected, the status register must be cleared before another byte write or block erase operation is attempted. The V_{PP} status bit, unlike an A/D converter, does not provide continuous indication of V_{PP} level. The WSM interrogates the V_{PP} level only after the byte write or block erase command sequences have been entered and informs the system if V_{PP} has not been switched on. The V_{PP} status bit is not guaranteed to report accurate feedback between V_{PPL} and V_{PPH}.
SR.2–SR.0 = RESERVED FOR FUTURE ENHANCEMENTS	These bits are reserved for future use and should be masked out when polling the status register.



Bus Operation	Command	Comments
Write	Erase Setup	Data = 40H (10H) Addr = Within Block to Be Erased
Write	Erase	Data = D0H Addr = Within Block to Be Erased
Standby/Read		Check RY/BY# (V_{OH} = Ready, V_{OL} = Busy) or Read Status Register (Check SR.7 1 = Ready, 0 = Busy Toggle OE# or CE# to Update Status Register)
Repeat for subsequent bytes. Full status check can be done after each byte, or after a sequence of bytes. Write FFH after the last byte write operation to reset the device to read array mode.		

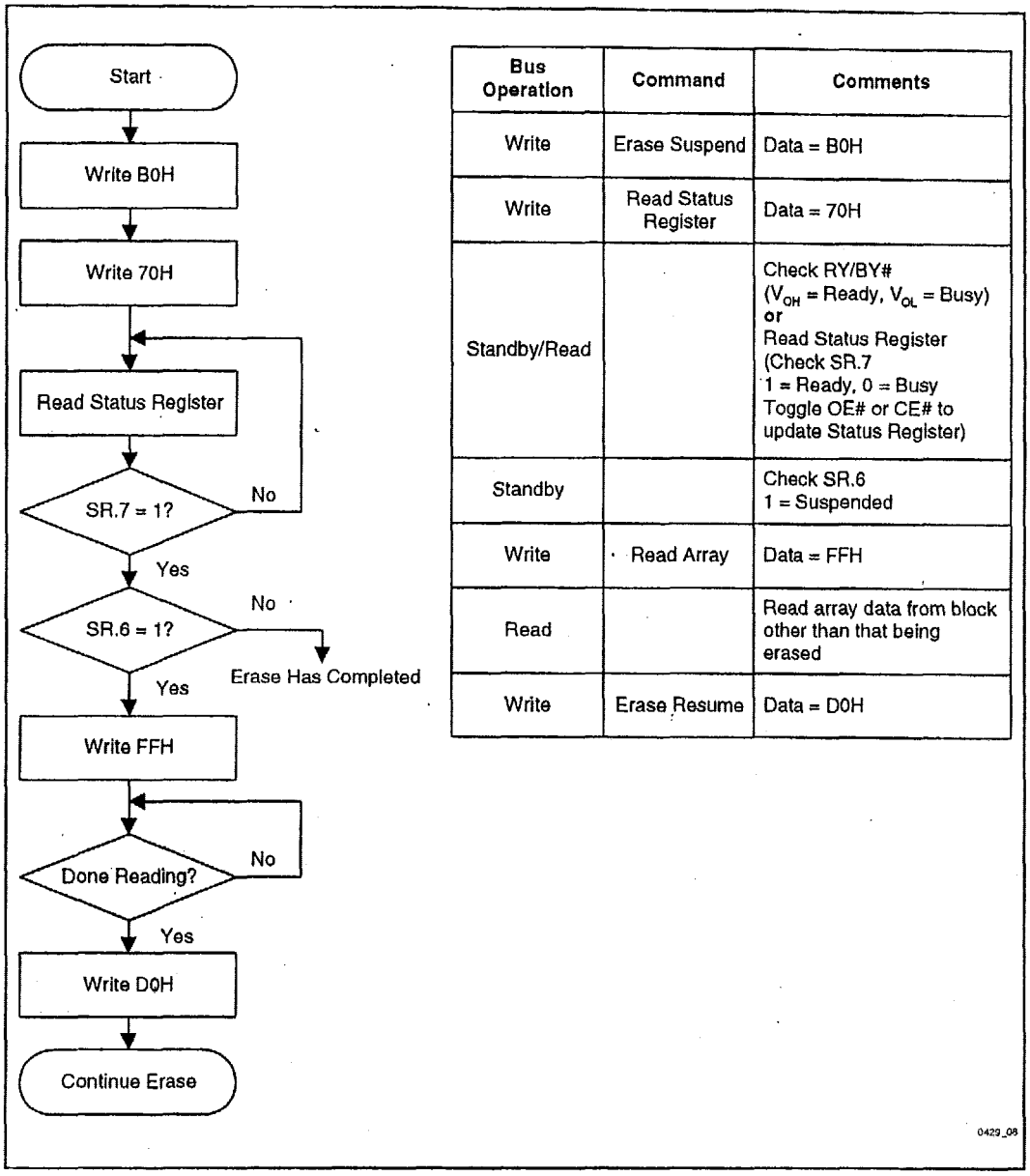
FULL STATUS CHECK PROCEDURE



Bus Operation	Command	Comments
Optional Read		CPU may already have read status register data in WSM Ready polling above
Standby		Check SR.3 1 = V_{PP} Low Detect
Standby		Check SR.4, 5 Both 1 = Command Sequence Error
Standby		Check SR.5 1 = Block Erase Error
SR.3 must be cleared, if set during a block erase attempt, before further attempts are allowed by the Write State Machine. SR.5 is only cleared by the clear status register command in cases where multiple bytes are erased before full status is checked. If an error is detected, clear the status register before attempting retry or other error recovery.		

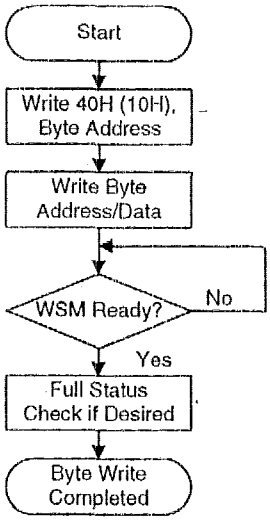
0429_07

Figure 6. Automated Block Erase Flowchart



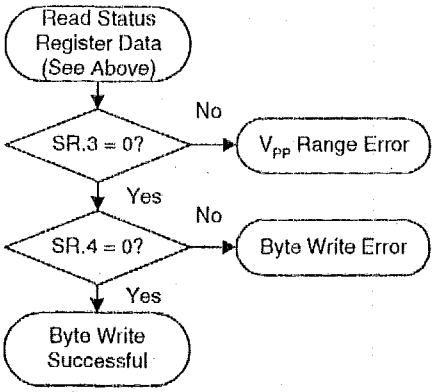
Bus Operation	Command	Comments
Write	Erase Suspend	Data = B0H
Write	Read Status Register	Data = 70H
Standby/Read		Check RY/BY# (V_{OH} = Ready, V_{OL} = Busy) **or** Read Status Register (Check SR.7 1 = Ready, 0 = Busy Toggle OE# or CE# to update Status Register)
Standby		Check SR.6 1 = Suspended
Write	Read Array	Data = FFH
Read		Read array data from block other than that being erased
Write	Erase Resume	Data = D0H

Figure 7. Erase Suspend/Resume Flowchart



Bus Operation	Command	Comments
Write	Byte Write Setup	Data = 40H (10H) Addr = Byte to Be Written
Write	Byte Write	Data to Be Wriiten Addr = Byte to Be Written
Standby/Read		Check RY/BY# (V_{OH} = Ready, V_{OL} = Busy) **or** Read Status Register (Check SR.7 1 = Ready, 0 = Busy Toggle OE# or CE# to Update Status Register)
Repeat for subsequent bytes. Full status check can be done after each byte, or after a sequence of bytes. Write FFH after the last byte write operation to reset the device to read array mode.		

FULL STATUS CHECK PROCEDURE



Bus Operation	Command	Comments
Optional Read		CPU may already have read status register data in WSM Ready polling above
Standby		Check SR.3 1 = V_{PP} Low Detect
Standby		Check SR.4 1 = Byte Write Error
SR.3 must be cleared, if set during a byte write attempt, before further attempts are allowed by the Write State Machine. SR.4 is only cleared by the clear status register command in cases where multiple bytes are written before full status is checked. If an error is detected, clear the status register before attempting retry or other error recovery.		

0420_06

Figure 8. Automated Byte Write Flowchart

8.4 V_{PP} Trace on Printed Circuit Boards

Writing flash memories, while they reside in the target system, requires that the printed circuit board designer pay attention to the V_{PP} power supply trace. The V_{PP} pin supplies the memory cell current for writing and erasing. Use similar trace widths and layout considerations given to the V_{CC} power bus. Adequate V_{PP} supply traces and decoupling will decrease V_{PP} voltage spikes and overshoots.

8.5 V_{CC}, V_{PP}, RP# Transitions and the Command/Status Registers

Byte write and block erase completion are not guaranteed if V_{PP} drops below V_{PPH}. If the V_{PP} status bit of the status register (SR.3) is set to "1," a Clear Status Register command **must** be issued before further byte write/block erase attempts are allowed by the WSM. Otherwise, the byte write (SR.4) or erase (SR.5) status bits of the status register will be set to "1"s if error is detected. If RP# transitions to V_{IL} during byte write and block erase, RY/BY# will remain low until the reset operation is complete. Data is partially altered in either case, and the command sequence must be repeated after normal operation is restored. Device power-off, or RP# transitions to V_{IL}, clear the status register to initial value 10000 for the upper 5 bits.

The CUI latches commands as issued by system software and is not altered by V_{PP} or CE# transitions or WSM actions. Its state upon power-up, after exit from deep power-down or after V_{CC} transitions below V_{LKO}, is read array mode.

After byte write or block erase is complete, even after V_{PP} transitions down to V_{PPL}, the CUI must be reset to read array mode via the Read Array command if access to the memory array is desired.

8.6 Power Up/Down Protection

The 28F008SA is designed to offer protection against accidental block erasure or byte writing during power transitions. Upon power-up, the 28F008SA is indifferent as to which power supply, V_{PP} or V_{CC}, powers up first. Power supply sequencing is not required. Internal circuitry in the 28F008SA ensures that the CUI is reset to the read array mode on power-up.

A system designer must guard against spurious writes for V_{CC} voltages above V_{LKO} when V_{PP} is active. Since both WE# and CE# must be low for a command write, driving either to V_{IH} will inhibit writes. The CUI architecture provides an added level of protection since alteration of memory contents only occurs after successful completion of the two-step command sequences.

Finally, the device is disabled until RP# is brought to V_{IH}, regardless of the state of its control inputs. This provides an additional level of memory protection.

8.7 Power Dissipation

When designing portable systems, designers must consider battery power consumption not only during device operation, but also for data retention during system idle time. Flash nonvolatility increases usable battery life, because the 28F008SA does not consume any power to retain code or data when the system is off.

In addition, the 28F008SA's deep power-down mode ensures extremely low power dissipation even when system power is applied. For example, portable PCs and other power sensitive applications, using an array of 28F008SAs for solid-state storage, can lower RP# to V_{IL} in standby or sleep modes, producing negligible power consumption. If access to the 28F008SA is again needed, the part can again be read, following the t_{PHQV} and t_{PHWL} wakeup cycles required after RP# is first raised back to V_{IH}. See AC Characteristics—Read-Only and Write Operations and Figures 13 and 15 for more information.

9.0 ELECTRICAL SPECIFICATIONS

NOTICE: This datasheet contains preliminary information on new products in production. The specifications are subject to change without notice. Verify with your local Intel Sales office that you have the latest datasheet before finalizing a design.

9.1 Absolute Maximum Ratings*

Operating Temperature
- During Read ..... 0 °C to +70 °C(1)
- During Block Erase/Byte Write ... 0 °C to +70 °C

Temperature Under Bias .....–10 °C to +80 °C

Storage Temperature .....–65 °C to +125 °C

Voltage on Any Pin
(except V_{CC} and V_{PP})
with Respect to GND..... –2.0 V to +7.0 V(2)

V_{PP} Program Voltage with
Respect to GND during
Block Erase/Byte Write ....–2.0 V to +14.0 V(2, 3)

V_{CC} Supply Voltage
with Respect to GND ..... –2.0 V to +7.0 V(2)

Output Short Circuit Current.....100 mA(4)

"WARNING: Stressing the device beyond the "Absolute Maximum Ratings" may cause permanent damage. These are stress ratings only. Operation beyond the "Operating Conditions" is not recommended and extended exposure beyond the "Operating Conditions" may affect device reliability.

NOTES:
1. Operating temperature is for commercial product defined by this specification.
2. Minimum DC voltage is –0.5 V on input/output pins. During transitions, this level may undershoot to –2.0 V for periods <20 ns. Maximum DC voltage on input/output pins is V_{CC} + 0.5 V which, during transitions, may overshoot to V_{CC} + 2.0 V for periods <20 ns.
3. Maximum DC voltage on V_{PP} may overshoot to +14.0 V for periods <20 ns.
4. Output shorted for no more than one second. No more than one output shorted at a time.
5. 5% V_{CC} specifications reference the 28F008SA-85 in its High Speed configuration. 10% V_{CC} specifications reference the 28F008SA-85 in its Standard configuration, and the 28F008SA-120.

9.2 Operating Conditions

Symbol	Parameter	Notes	Min	Max	Unit
T_A	Operating Temperature		0	70	°C
V_{CC}	V_{CC} Supply Voltage (10%)	5	4.50	5.50	V
V_{CC}	V_{CC} Supply Voltage (5%)	5	4.75	5.25	V

9.3 Capacitance(1)

T_A = 25 °C, f = 1 MHz

Symbol	Parameter	Typ	Max	Unit	Condition
C_{IN}	Input Capacitance	6	8	pF	V_{IN} = 0 V
C_{OUT}	Output Capacitance	8	12	pF	V_{OUT} = 0 V

NOTE:
1. Sampled, not 100% tested.

9.4 DC Characteristics

Symbol	Parameter	Notes	Min	Typ	Max	Unit	Test Condition
I_{LI}	Input Load Current	1			±1.0	µA	$V_{CC} = V_{CC}$ Max $V_{IN} = V_{CC}$ or GND
I_{LO}	Output Leakage Current	1			±10	µA	$V_{CC} = V_{CC}$ Max $V_{OUT} = V_{CC}$ or GND
I_{CCS}	V_{CC} Standby Current	1, 3		1.0	2.0	mA	$V_{CC} = V_{CC}$ Max CE# = RP# = V_{IH}
				30	100	µA	$V_{CC} = V_{CC}$ Max CE# = RP# = V_{CC} ±0.2 V
I_{CCD}	V_{CC} Deep Power-Down Current	1		0.20	1.2	µA	RP# = GND ±0.2 V I_{OUT} (RY/BY#) = 0 mA
I_{CCR}	V_{CC} Read Current	1		20	35	mA	$V_{CC} = V_{CC}$ Max, CE# = GND f = 8 MHz, I_{OUT} = 0 mA CMOS Inputs
				25	50	mA	$V_{CC} = V_{CC}$ Max, CE# = V_{IL} f = 8 MHz, I_{OUT} = 0 mA TTL Inputs
I_{CCW}	V_{CC} Byte Write Current	1		10	30	mA	Byte Write In Progress
I_{CCE}	V_{CC} Block Erase Current	1		10	30	mA	Block Erase In Progress
I_{CCES}	V_{CC} Erase Suspend Current	1, 2		5	10	mA	Block Erase Suspended CE# = V_{IH}
I_{PPS}	V_{PP} Standby Current	1		±1	±15	µA	$V_{PP} \leq V_{CC}$
I_{PPD}	V_{PP} Deep Power-Down Current	1		0.10	5.0	µA	RP# = GND ±0.2 V
I_{PPR}	V_{PP} Read Current				200	µA	$V_{PP} > V_{CC}$
I_{PPW}	V_{PP} Byte Write Current	1		10	30	mA	$V_{PP} = V_{PPH}$ Byte Write in Progress
I_{PPE}	V_{PP} Block Erase Current	1		10	30	mA	$V_{PP} = V_{PPH}$ Block Erase in Progress
I_{PPES}	V_{PP} Erase Suspend Current	1		90	200	µA	$V_{PP} = V_{PPH}$ Block Erase Suspended
V_{IL}	Input Low Voltage		–0.5		0.8	V	
V_{IH}	Input High Voltage		2.0		V_{CC} + 0.5	V	

9.4 DC Characteristics (Continued)

Symbol	Parameter	Notes	Min	Typ	Max	Unit	Test Condition
V_{OL}	Output Low Voltage	3			0.45	V	$V_{CC} = V_{CC}$ Min I_{OL} = 5.8 mA
V_{OH1}	Output High Voltage (TTL)	3	2.4			V	$V_{CC} = V_{CC}$ Min I_{OH} = −2.5 mA
V_{OH2}	Output High Voltage (CMOS)		0.85 V_{CC}			V	$V_{CC} = V_{CC}$ Min I_{OH} = −2.5 µA
			V_{CC} − 0.4				$V_{CC} = V_{CC}$ Min I_{OH} = −100 µA
V_{PPL}	V_{PP} during Normal Operations	4	0.0		6.5	V	
V_{PPH}	V_{PP} during Erase/Write Operations		11.4	12.0	12.6	V	
V_{LKO}	V_{CC} Erase/Write Lock Voltage		2.0			V	

NOTES:

1. All currents are in RMS unless otherwise noted. Typical values at V_{CC} = 5.0 V, V_{PP} = 12.0 V, T_A = 25 °C. These currents are valid for all product versions (packages and speeds).
2. I_{CCES} is specified with the device deselected. If the 28F008SA is read while in erase suspend mode, current draw is the sum of I_{CCES} and I_{CCR}.
3. Includes RY/BY#.
4. Block erases/byte writes are inhibited when $V_{PP} = V_{PPL}$ and not guaranteed in the range between V_{PPH} and V_{PPL}.
5. Sampled, not 100% tested.

9.5 Extended Temperature Operating Conditions

Symbol	Parameter	Notes	Min	Max	Unit
T_A	Operating Temperature		−40	+85	°C
V_{CC}	V_{CC} Supply Voltage (10%)	5	4.50	5.50	V
V_{CC}	V_{CC} Supply Voltage (5%)	5	4.75	5.25	V

9.6 DC Characteristics—Extended Temperature Operation

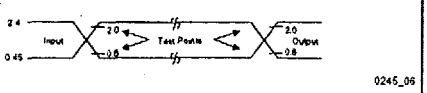
Symbol	Parameter	Notes	Min	Typ	Max	Unit	Test Condition
I_{LI}	Input Load Current	1			±1.0	µA	$V_{CC} = V_{CC}$ Max $V_{IN} = V_{CC}$ or GND
I_{LO}	Output Leakage Current	1			±10	µA	$V_{CC} = V_{CC}$ Max $V_{OUT} = V_{CC}$ or GND
I_{CCS}	V_{CC} Standby Current	1, 3		1.0	2.0	mA	$V_{CC} = V_{CC}$ Max CE# = RP# = V_{IH}
				30	100	µA	$V_{CC} = V_{CC}$ Max CE# = RP# = V_{CC} ±0.2 V
I_{CCD}	V_{CC} Deep Power-Down Current	1		0.20	20	µA	RP# = GND ±0.2 V I_{OUT} (RY/BY#) = 0 mA
I_{CCR}	V_{CC} Read Current	1		20	35	mA	$V_{CC} = V_{CC}$ Max, CE# = GND f = 8 MHz, I_{OUT} = 0 mA CMOS Inputs
				25	50	mA	$V_{CC} = V_{CC}$ Max, CE# = V_{IL} f = 8 MHz, I_{OUT} = 0 mA TTL Inputs
I_{CCW}	V_{CC} Byte Write Current	1		10	30	mA	Byte Write In Progress
I_{CCE}	V_{CC} Block Erase Current	1		10	30	mA	Block Erase In Progress
I_{CCES}	V_{CC} Erase Suspend Current	1, 2		5	10	mA	Block Erase Suspended CE# = V_{IH}
I_{PPS}	V_{PP} Standby Current	1		±1	±15	µA	$V_{PP} \leq V_{CC}$
I_{PPD}	V_{PP} Deep Power-Down Current	1		0.10	5.0	µA	RP# = GND ±0.2 V
I_{PPR}	V_{PP} Read Current				200	µA	$V_{PP} > V_{CC}$
I_{PPW}	V_{PP} Byte Write Current	1		10	30	mA	$V_{PP} = V_{PPH}$ Byte Write in Progress
I_{PPE}	V_{PP} Block Erase Current	1		10	30	mA	$V_{PP} = V_{PPH}$ Block Erase in Progress
I_{PPES}	V_{PP} Erase Suspend Current	1		90	200	µA	$V_{PP} = V_{PPH}$ Block Erase Suspended
V_{IL}	Input Low Voltage		−0.5		0.8	V	
V_{IH}	Input High Voltage		2.0		V_{CC} + 0.5	V	

9.6 DC Characteristics—Extended Temperature Operation (Continued)

Symbol	Parameter	Notes	Min	Typ	Max	Unit	Test Condition
V_{OL}	Output Low Voltage	3			0.45	V	$V_{CC} = V_{CC}$ Min I_{OL} = 5.8 mA
V_{OH1}	Output High Voltage (TTL)	3	2.4			V	$V_{CC} = V_{CC}$ Min I_{OH} = −2.5 mA
V_{OH2}	Output High Voltage (CMOS)		0.85 V_{CC}			V	$V_{CC} = V_{CC}$ Min I_{OH} = −2.5 µA
			V_{CC} − 0.4				$V_{CC} = V_{CC}$ Min I_{OH} = −100 µA
V_{PPL}	V_{PP} during Normal Operations	4	0.0		6.5	V	
V_{PPH}	V_{PP} during Erase/Write Operations		11.4	12.0	12.6	V	
V_{LKO}	V_{CC} Erase/Write Lock Voltage		2.0			V	

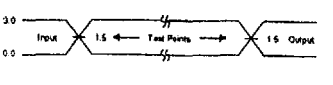
NOTES:

1. All currents are in RMS unless otherwise noted. Typical values at V_{CC} = 5.0 V, V_{PP} = 12.0 V, T_A = 25 °C. These currents are valid for all product versions (packages and speeds).
2. I_{CCES} is specified with the device deselected. If the 28F008SA is read while in erase suspend mode, current draw is the sum of I_{CCES} and I_{CCR}.
3. Includes RY/BY#.
4. Block erases/byte writes are inhibited when $V_{PP} = V_{PPL}$ and not guaranteed in the range between V_{PPH} and V_{PPL}.
5. Sampled, not 100% tested.



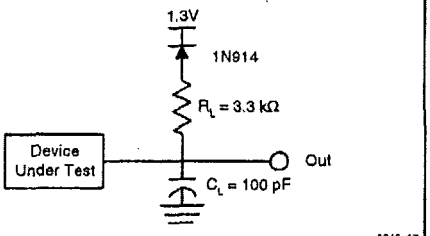
AC test inputs are driven at V_{OH} (2.4 V_{TTL}) for a Logic "1" and V_{OL} (0.45 V_{TTL}) for a Logic "0." Input timing begins at V_{IH} (2.0 V_{TTL}) and V_{IL} (0.8 V_{TTL}). Output timing ends at V_{IH} and V_{IL}. Input rise and fall times (10% to 90%) <10 ns.

Figure 9. Testing Input/Output Waveform(1)



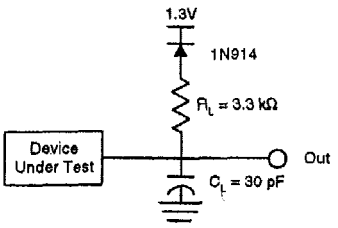
AC test inputs are driven at 3.0 V for a Logic "1" and 0.0 V for a Logic "0." Input timing begins, and output timing ends, at 1.5 V. Input rise and fall times (10% to 90%) <10 ns.

Figure 11. High Speed AC Testing Input/Output Waveforms(2)



NOTE:
C_L includes Jig Capacitance

Figure 10. AC Testing Load Circuit(1)



NOTE:
C_L includes Jig Capacitance

Figure 12. High Speed AC Testing Load Circuit(2)

NOTES:
1. Testing characteristics for 28F008SA-85 in Standard configuration, and 28F008SA-120.
2. Testing characteristics for 28F008SA-85 in High Speed configuration.

9.7 AC Characteristics—Read-Only Operations(1)

Versions			$V_{CC} \pm 5\%$	28F008SA-85(4)		—		—		Unit
			$V_{CC} \pm 10\%$	—		28F008SA-85(5)		28F008SA-120(5)		
Symbol		Parameter	Notes	Min	Max	Min	Max	Min	Max	
t_{AVAV}	t_{RC}	Read Cycle Time		85		90		120		ns
t_{AVQV}	t_{ACC}	Address to Output Delay			85		90		120	ns
t_{ELQV}	t_{CE}	CE# to Output Delay	2		85		90		120	ns
t_{PHQV}	t_{PWH}	RP# High to Output Delay			400		400		400	ns
t_{GLQV}	t_{OE}	OE# to Output Delay	2		40		45		50	ns
t_{ELQX}	t_{LZ}	CE# to Output Low Z	3	0		0		0		ns
t_{EHQZ}	t_{HZ}	CE# High to Output High Z	3		55		55		55	ns
t_{GLQX}	t_{OLZ}	OE# to Output Low Z	3	0		0		0		ns
t_{GHQZ}	t_{DF}	OE# High to Output High Z	3		30		30		30	ns
	t_{OH}	Output Hold from Addresses, CE# or OE# Change, Whichever is First	3	0		0		0		ns

NOTES:
1. See AC Input/Output Reference Waveform for timing measurements.
2. OE# may be delayed up to $t_{CE}-t_{OE}$ after the falling edge of CE# without impact on t_{CE}.
3. Sampled, not 100% tested.
4. See High Speed AC Input/Output Reference Waveforms and High Speed AC Testing Load Circuits for testing characteristics.
5. See AC Input/Output Reference Waveforms and AC Testing Load Circuits for testing characteristics.

9.8 AC Characteristics—Read-Only Operations[1]— Extended Temperature Operation

Versions			$V_{CC} \pm 10\%$	28F008SA-100[5]		Unit
Symbol		Parameter	Notes	Min	Max	
t_{AVAV}	t_{RC}	Read Cycle Time		100		ns
t_{AVQV}	t_{ACC}	Address to Output Delay			100	ns
t_{ELQV}	t_{CE}	CE# to Output Delay	2		100	ns
t_{PHQV}	t_{PWH}	RP# High to Output Delay			400	ns
t_{GLQV}	t_{OE}	OE# to Output Delay	2		55	ns
t_{ELQX}	t_{LZ}	CE# to Output Low Z	3	0		ns
t_{EHQZ}	t_{HZ}	CE# High to Output High Z	3		55	ns
t_{GLQX}	t_{OLZ}	OE# to Output Low Z	3	0		ns
t_{GHQZ}	t_{DF}	OE# High to Output High Z	3		30	ns
	t_{OH}	Output Hold from Addresses, CE# or OE# Change, Whichever is First	3	0		ns

NOTES:

1. See AC Input/Output Reference Waveform for timing measurements.
2. OE# may be delayed up to $t_{CE}-t_{OE}$ after the falling edge of CE# without impact on t_{CE}.
3. Sampled, not 100% tested.
4. See High Speed AC Input/Output Reference Waveforms and High Speed AC Testing Load Circuits for testing characteristics.
5. See AC Input/Output Reference Waveforms and AC Testing Load Circuits for testing characteristics.

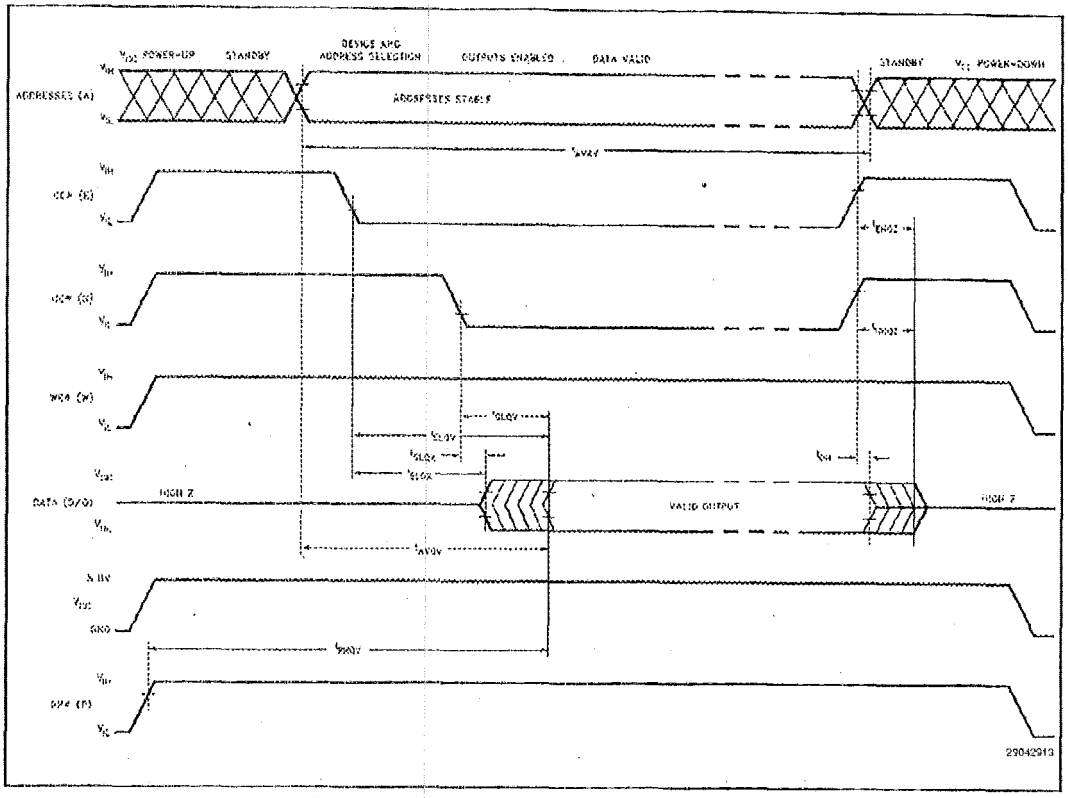


Figure 13. AC Waveform for Read Operations

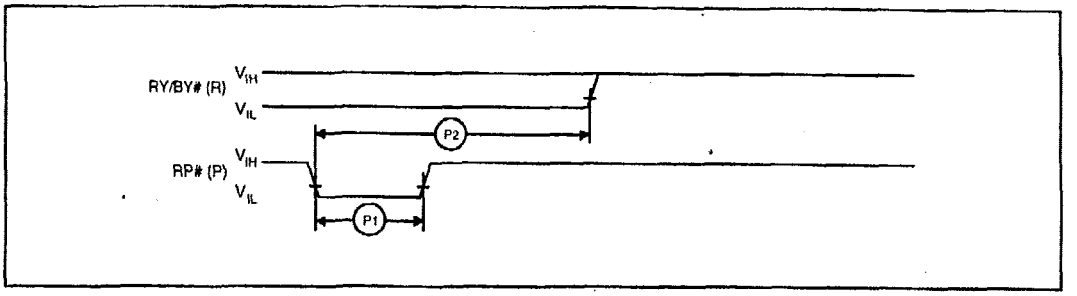


Figure 14. AC Waveform for Reset Operation

#	Sym	Parameter	Notes	Min	Max	Unit
P1	t_{PLPH}	RP# Pulse Low Time (If RP# is tied to V_{CC}, this specification is not applicable)		100		ns
P2	t_{PLRH}	RP# Low to Reset during Block Erase, Program, or Lock-Bit Configuration	2, 3		12	µs

NOTES:

1. These specifications are valid for all product versions (packages and speeds).
2. If RP# is asserted when the WSM is not busy (RY/BY# = "1"), the reset will complete within 100 ns.
3. A reset time, t_{PHQV}, is required from the latter of RY/BY# or RP# going high until outputs are valid.

9.9 AC Characteristics—Write Operations[1]

Versions			V_{CC} ±5%	28F008SA-85[7]		—		—		Unit
			V_{CC} ±10%	—		28F008SA-85[8]		28F008SA-120[8]		
Symbol		Parameter	Notes	Min	Max	Min	Max	Min	Max	
t_{AVAV}	t_{WC}	Write Cycle Time		85		90		120		ns
t_{PHWL}	t_{PS}	RP# High Recovery to WE# Going Low	2	1		1		1		µs
t_{ELWL}	t_{CS}	CE# Setup to WE# Going Low		10		10		10		ns
t_{WLWH}	t_{WP}	WE# Pulse Width		40		40		40		ns
t_{VPWH}	t_{VPS}	V_{PP} Setup to WE# Going High	2	100		100		100		ns
t_{AVWH}	t_{AS}	Address Setup to WE# Going High	3	40		40		40		ns
t_{DVWH}	t_{DS}	Data Setup to WE# Going High	4	40		40		40		ns
t_{WHDX}	t_{DH}	Data Hold from WE# High		5		5		5		ns
t_{WHAX}	t_{AH}	Address Hold from WE# High		5		5		5		ns
t_{WHEH}	t_{CH}	CE# Hold from WE# High		10		10		10		ns
t_{WHWL}	t_{WPH}	WE# Pulse Width High		30		30		30		ns
t_{WHRL}		WE# High to RY/BY# Going Low			100		100		100	ns
t_{WHQV1}		Duration of Byte Write Operation	5, 6	6		6		6		µs
t_{WHQV2}		Duration of Block Erase Operation	5, 6	0.3		0.3		0.3		sec
t_{WHGL}		Write Recovery before Read		0		0		0		µs
t_{QVVL}	t_{VPH}	V_{PP} Hold from Valid SRD, RY/BY# High	2, 6	0		0		0		ns

NOTES:

1. Read timing characteristics during erase and byte write operations are the same as during read-only operations. Refer to *AC Characteristics—Read-Only Operations*.
2. Sampled, not 100% tested.
3. Refer to Table 3 for valid A_{IN} for byte write or block erasure.
4. Refer to Table 3 for valid D_{IN} for byte write or block erasure.
5. The on-chip Write State Machine incorporates all byte write and block erase system functions and overhead of standard Intel flash memory, including byte program and verify (byte write) and block precondition, precondition verify, erase and erase verify (block erase).
6. Byte write and block erase durations are measured to completion (SR.7 = 1, RY/BY# = V_{OH}). V_{PP} should be held at V_{PPH} until determination of byte write/block erase success (SR.3/4/5 = 0)
7. See High Speed AC Input/Output Reference Waveforms and High Speed AC Testing Load Circuits for testing characteristics.
8. See AC Input/Output Reference Waveforms and AC Testing Load Circuits for testing characteristics.

9.10 Block Erase and Byte Write Performance

Parameter	Notes	28F008SA-85		28F008SA-120		Unit
		Typ[1]	Max	Typ[1]	Max	
Block Erase Time	2	1.6	10	1.6	10	sec
Block Write Time	2	0.6	2.1	0.6	2.1	sec
Byte Write Time		8	(Note 3)	8	(Note 3)	µs

NOTES:

1. 25 °C, 12.0 V V_{PP}.
2. Excludes System-Level Overhead.
3. Contact your Intel representative for information on the maximum byte write specification.

9.11 AC Characteristics—Write Operations[1]— Extended Temperature Operation

Versions			$V_{CC} \pm 10\%$	28F008SA-100[8]		Unit
Symbol		Parameter	Notes	Min	Max	
t_{AVAV}	t_{WC}	Write Cycle Time		100		ns
t_{PHWL}	t_{PS}	RP# High Recovery to WE# Going Low	2	1		µs
t_{ELWL}	t_{CS}	CE# Setup to WE# Going Low		10		ns
t_{WLWH}	t_{WP}	WE# Pulse Width		40		ns
t_{VPWH}	t_{VPS}	V_{PP} Setup to WE# Going High	2	100		ns
t_{AVWH}	t_{AS}	Address Setup to WE# Going High	3	40		ns
t_{DVWH}	t_{DS}	Data Setup to WE# Going High	4	40		ns
t_{WHDX}	t_{DH}	Data Hold from WE# High		5		ns
t_{WHAX}	t_{AH}	Address Hold from WE# High		5		ns
t_{WHEH}	t_{CH}	CE# Hold from WE# High		10		ns
t_{WHWL}	t_{WPH}	WE# Pulse Width High		30		ns
t_{WHRL}		WE# High to RY/BY# Going Low			100	ns
t_{WHQV1}		Duration of Byte Write Operation	5, 6	6		µs
t_{WHQV2}		Duration of Block Erase Operation	5, 6	0.3		sec
t_{WHGL}		Write Recovery before Read		0		µs
t_{QVVL}	t_{VPH}	V_{PP} Hold from Valid SRD, RY/BY# High	2, 6	0		ns

NOTES:

1. Read timing characteristics during erase and byte write operations are the same as during read-only operations. Refer to *AC Characteristics—Read-Only Operations.*
2. Sampled, not 100% tested.
3. Refer to Table 3 for valid A_{IN} for byte write or block erasure.
4. Refer to Table 3 for valid D_{IN} for byte write or block erasure.
5. The on-chip WSM incorporates all byte write and block erase system functions and overhead of standard Intel flash memory, including byte program and verify (byte write) and block precondition, precondition verify, erase and erase verify (block erase).
6. Byte write and block erase durations are measured to completion (SR.7 = 1, RY/BY# = V_{OH}). V_{PP} should be held at V_{PPH} until determination of byte write/block erase success (SR.3/4/5 = 0)
7. See High Speed AC Input/Output Reference Waveforms and High Speed AC Testing Load Circuits for testing characteristics.
8. See AC Input/Output Reference Waveforms and AC Testing Load Circuits for testing characteristics.

9.12 Block Erase and Byte Write Performance—Extended Temperature Operation

Parameter	Notes	28F008SA-100		Unit
		Typ(1)	Max	
Block Erase Time	2	1.6	10	sec
Block Write Time	2	0.6	2.1	sec
Byte Write Time		8	(Note 3)	µs

NOTES:
1. 25 °C, 12.0 V V_{PP}.
2. Excludes System-Level Overhead.
3. Contact your Intel representative for information on the maximum byte write specification.

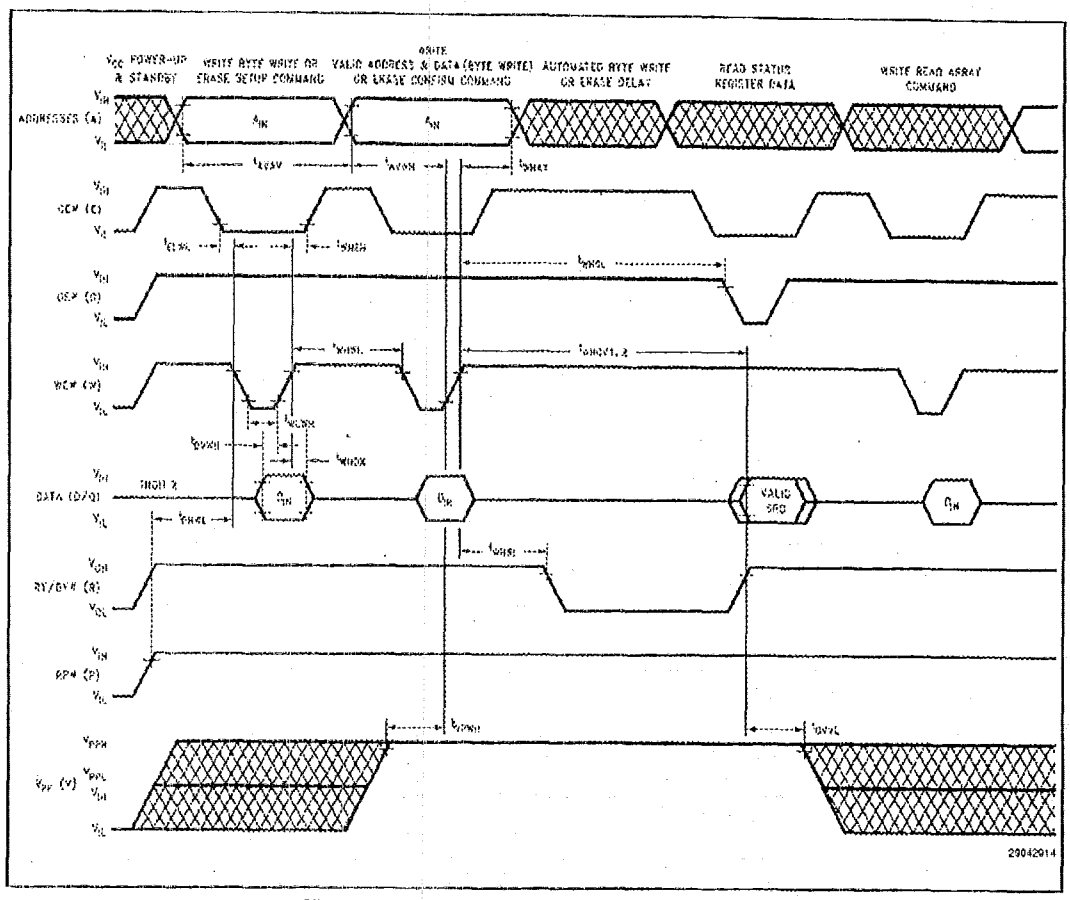


Figure 15. AC Waveform for Write Operations

9.13 Alternative CE#-Controlled Writes

Versions				V_{CC} ±5%		28F008SA-85[6]		—		—		Unit
				V_{CC} ±10%		—		28F008SA-85[7]		28F008SA-120[7]		
Sym		Parameter	Notes	Min	Max	Min	Max	Min	Max			
t_{AVAV}	t_{WC}	Write Cycle Time		85		90		120				ns
t_{PHEL}	t_{PS}	RP# High Recovery to CE# Going Low	2	1		1		1				µs
t_{WLEL}	t_{WS}	WE# Setup to CE# Going Low		0		0		0				ns
t_{ELEH}	t_{CP}	CE# Pulse Width		50		50		50				ns
t_{VPEH}	t_{VPS}	V_{PP} Setup to CE# Going High	2	100		100		100				ns
t_{AVEH}	t_{AS}	Address Setup to CE# Going High	3	40		40		40				ns
t_{DVEH}	t_{DS}	Data Setup to CE# Going High	4	40		40		40				ns
t_{EHDX}	t_{DH}	Data Hold from CE# High		5		5		5				ns
t_{EHAX}	t_{AH}	Address Hold from CE# High		5		5		5				ns
t_{EHWH}	t_{WH}	WE# Hold from CE# High		0		0		0				ns
t_{EHEL}	t_{EPH}	CE# Pulse Width High		25		25		25				ns
t_{EHRL}		CE# High to RY/BY# Going Low			100		100		100			ns
t_{EHQV1}		Duration of Byte Write Operation	5	6		6		6				µs
t_{EHQV2}		Duration of Block Erase Operation	5	0.3		0.3		0.3				sec
t_{EHGL}		Write Recovery before Read		0		0		0				µs
t_{QVVL}	t_{VPH}	V_{PP} Hold from Valid SRD, RY/BY# High	2, 5	0		0		0				ns

NOTES:

1. Chip-Enable Controlled Writes: Write operations are driven by the valid combination of CE# and WE#. In systems where CE# defines the write pulsewidth (within a longer WE# timing waveform), all setup, hold and inactive WE# times should be measured relative to the CE# waveform.
2. Sampled, not 100% tested.
3. Refer to Table 3 for valid A_{IN} for byte write or block erasure.
4. Refer to Table 3 for valid D_{IN} for byte write or block erasure.
5. Byte write and block erase durations are measured to completion (SR.7 = 1, RY/BY# = V_{OH}). V_{PP} should be held at V_{PPH} until determination of byte write/block erase success (SR.3/4/5 = 0)
6. See High Speed AC Input/Output Reference Waveforms and High Speed AC Testing Load Circuits for testing characteristics.
7. See AC Input/Output Reference Waveforms and AC Testing Load Circuits for testing characteristics.

9.14 Alternative CE#-Controlled Writes— Extended Temperature Operation

Versions			V_{CC}±10%	28F008SA-100[7]		
Symbol		Parameter	Notes	Min	Max	Unit
t_{AVAV}	t_{WC}	Write Cycle Time		100		ns
t_{PHEL}	t_{PS}	RP# High Recovery to CE# Going Low	2	1		µs
t_{WLEL}	t_{WS}	WE# Setup to CE# Going Low		0		ns
t_{ELEH}	t_{CP}	CE# Pulse Width		50		ns
t_{VPEH}	t_{VPS}	V_{PP} Setup to CE# Going High	2	100		ns
t_{AVEH}	t_{AS}	Address Setup to CE# Going High	3	40		ns
t_{DVEH}	t_{DS}	Data Setup to CE# Going High	4	40		ns
t_{EHDX}	t_{DH}	Data Hold from CE# High		5		ns
t_{EHAX}	t_{AH}	Address Hold from CE# High		5		ns
t_{EHWH}	t_{WH}	WE# Hold from CE# High		0		ns
t_{EHEL}	t_{EPH}	CE# Pulse Width High		25		ns
t_{EHRL}		CE# High to RY/BY# Going Low			100	ns
t_{EHQV1}		Duration of Byte Write Operation	5	6		µs
t_{EHQV2}		Duration of Block Erase Operation	5	0.3		sec
t_{EHGL}		Write Recovery before Read		0		µs
t_{QVVL}	t_{VPH}	V_{PP} Hold from Valid SRD, RY/BY# High	2, 5	0		ns

NOTES:

1. Chip-Enable Controlled Writes: Write operations are driven by the valid combination of CE# and WE#. In systems where CE# defines the write pulsewidth (within a longer WE# timing waveform), all setup, hold and inactive WE# times should be measured relative to the CE# waveform.
2. Sampled, not 100% tested.
3. Refer to Table 3 for valid A_{IN} for byte write or block erasure.
4. Refer to Table 3 for valid D_{IN} for byte write or block erasure.
5. Byte write and block erase durations are measured to completion (SR.7 = 1, RY/BY# = V_{OH}). V_{PP} should be held at V_{PPH} until determination of byte write/block erase success (SR.3/4/5 = 0)
6. See High Speed AC Input/Output Reference Waveforms and High Speed AC Testing Load Circuits for testing characteristics.
7. See AC Input/Output Reference Waveforms and AC Testing Load Circuits for testing characteristics.

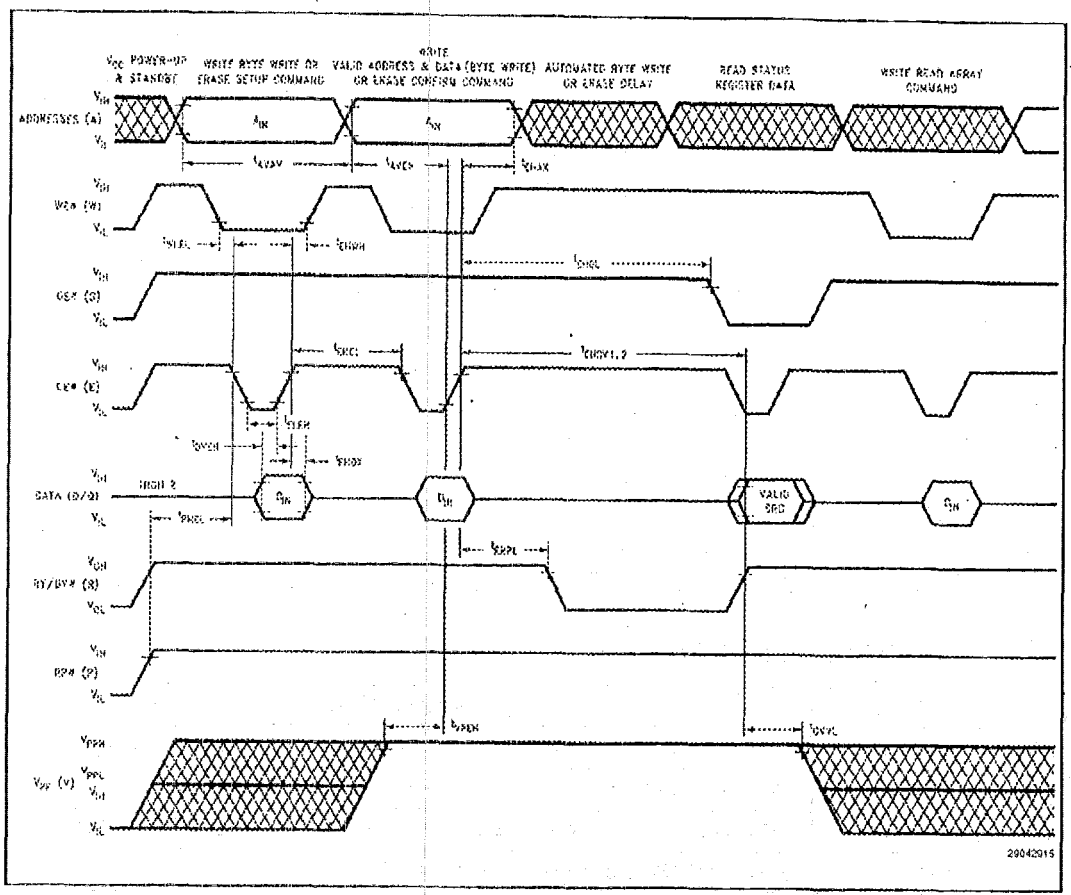
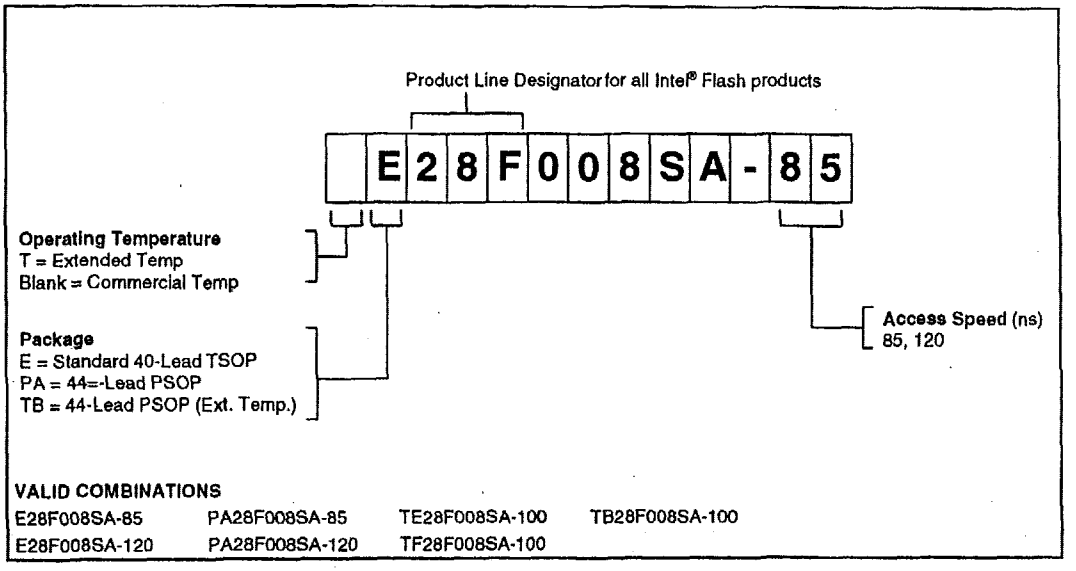


Figure 16. Alternate AC Waveform for Write Operations

10.0 ORDERING INFORMATION



VALID COMBINATIONS

E28F008SA-85	PA28F008SA-85	TE28F008SA-100	TB28F008SA-100
E28F008SA-120	PA28F008SA-120	TF28F008SA-100	

11.0 ADDITIONAL INFORMATION

Order Number	Document/Tool
290597	*5 Volt FlashFile™ Memory; 28F004S5, 28F008S5, 28F016S5* datasheet
290598	*3 Volt FlashFile™ Memory; 28F004S3, 28F008S3, 28F016S3* datasheet
271296	*28F008SA 8-Mbit (1-Mbit x 8) Flash Memory SmartDie™ Product Specification*
292180	*AP-625 28F008SC Compatibility with 28F008SA*
297183	*28F008 SA/SA-L Specification Update*
Note 3	*AP-359 28F008SA Hardware Interfacing*
Note 3	*AP-364 28F008SA Automation and Algorithms*

NOTES:

1. Please call the Intel Literature Center at (800) 548-4725 to request Intel documentation. International customers should contact their local Intel or distribution sales office.
2. Visit Intel's World Wide Web home page at http://www.intel.com for technical documentation and tools.
3. These documents can be located at the Intel World Wide Web support site, http://www.intel.com/support/flash/memory

2.5.2 Part Number Codes for the 28F008SA

The following is the vendor's product number codes for the Intel 28F008SA 1-Mbyte flash memory, which was downloaded from the Intel Internet site at: http://apps.intel.com/product_selector/chart1.asp.

Flash Memory Naming Conventions

	G	2	8	F	6	4	0	J	5	-L	150

Package Designator

Commercial Temperature:

G = µBGA
PA = PSOP
E = TSOP
F = TSOP, reverse pinout
DA = SSOP
DD = Dual Die
P = Plastic DIP
N = PLCC

Extended Temperature:

GT = µBGA
TB = PSOP
TE = TSOP
TF = TSOP, reverse pinout
DT = SSOP
TP = Plastic Dip
N = PLCC

Product Line Designator
for all Intel flash products

Density/Organization
00X = x8 only
X00 = x8/x16 Selectable

Temperature Ranges
Commercial (0°C to + 70°C)
Expanded (-20°C to +70°C)
Extended (-40°C to +85°C)
Automotive (-40°C to +125°C)

Architecture
J = Intel StrataFlash™ memory, 2 bits-per-cell
B = Boot Block High Integration x8/x16
C = Compact 48-Lead TSOP Boot Block
S = Symmetrical Block FlashFile™ Memory
Lowest Cost x8/x16 Selectable

Access Speed (ns
L = Low Voltage I/O (
T = Top Boot Positior
B = Bottom Boot Posi

Voltage Options (
3 = 3V/3, 12V
5 = 5V/5, 12V
C = 2.7, 3 or 5V/3, 5 c
E = 2.7, 5V/5, 12V
V = 3, 5V/5, 12V

Exception
28F640J5/28F320J5
28F320/28F160
3 = 2.7, 3.3V/2.7, 3.3
5 = 5V/5V

Memory Cards

Package Designator

PC = PC Card Standard, 68-pin compatible
Type 1 = 3.3 mm thickness
MC = Miniature Card Standard

Temperature Ranges:

Commercial
Value Series 100, Series 2 (0°C to +70°C)
Miniature Card (0°C to +60°C)

Extended
Series 2 (-4

to top of page

3.1 High-Level Description of Video Card Operation

The video card is used to drive a video display when the door of the AMS are open and the system is making a measurement of an unclassified object. The open-mode display is designed to aid both system development and authentication of the complete system. The display is used as the console output device.

3.1.1 Brief Overview Narrative of the Video Card

The Ampro SVG-II card is a compact, high-resolution graphics display controller, offering software selectable multimode operation. It is compatible with most popular VGA video standards (SVGA, VGA, EGA, CGA & MDA). It is a PC104 card, which requires only +5 VDC with less than 2 watts of power consumption.

3.1.2 Critical Specifications for the Video Card

The video card must work with the Ampro 3SXi processor card to provide video output.

3.2 Ampro, MiniModule SVG-II – Video Display Card

The following sections provide information regarding the Ampro MiniModule SVG-II card, which is used as the system video card.

3.2.1 Specifications for the Ampro MiniModule SVG-II Card

The specifications for the SGV-II video card are:

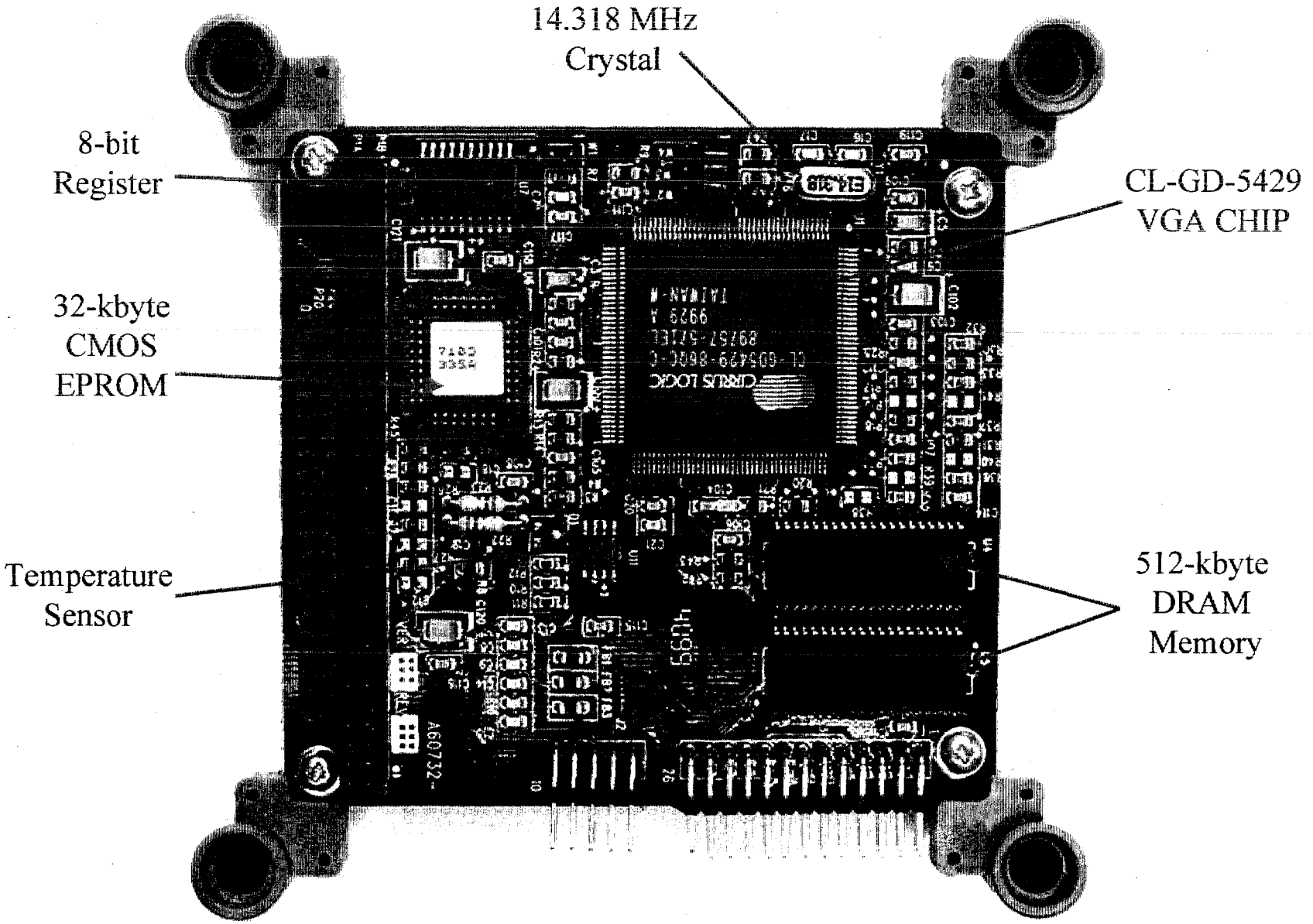
Video RAM	1 Mbyte of onboard RAM for MM2-SVG-Q-72 ← used 512 kbytes of onboard RAM for MM2-SVG-Q-71
Display address	A0000 to BFFFF hex
ROM-BIOS address	C0000 to C7FFF hex
I/O port addresses	3B4 to 3DA hex & 46E8 hex
Interrupt	IRQ2 mapped to IRQ9 on AT bus
Video Bandwidth	25 to 78.7 MHz
Horizontal Scan Rate	31.5 to 48.3 kHz
Vertical Scan Rate	56 to 75 kHz (87 Hz interlaced)
Voltage	+5 VDC
Power	1.7 watts maximum
Operating Humidity Range	5-95% (non-condensing)
Operating Temperature	0 to +70°C
Storage Temperature	-55 to +85°C

3.2.2 Photographs of the Ampro MiniModule SVG-II Card

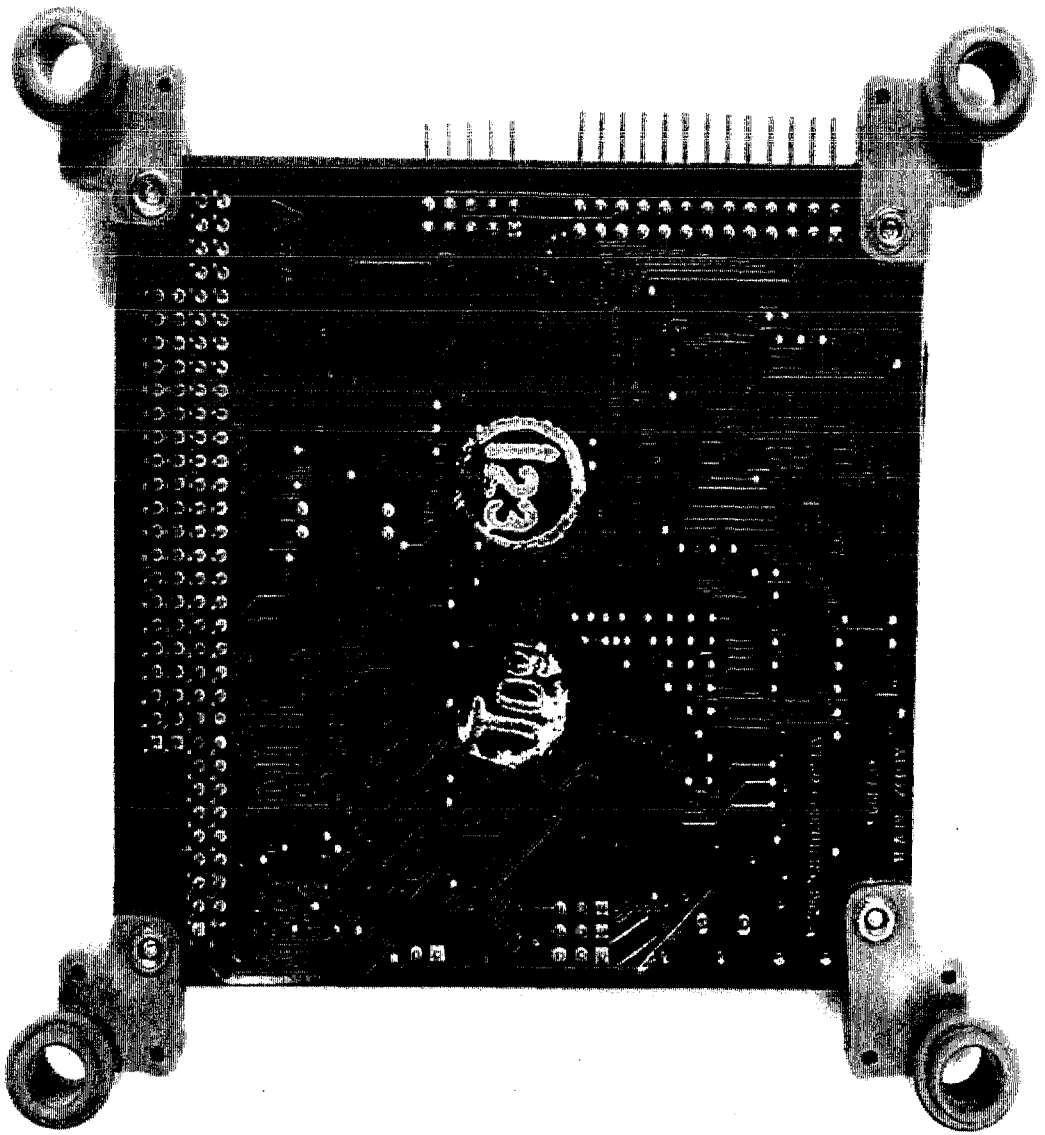
This section contains:

- Labeled top-view photograph of the Ampro MiniModule SVG-II Card
- Labeled bottom-view photograph of the Ampro MiniModule SVG-II Card.

AMPRO SVG-II – Video Card – Top-View Photograph



AMPRO SVG-II – Video Card – Bottom-View Photograph



3.2.3 X-Ray of the Ampro MiniModule SVG-II Card

This section contains:

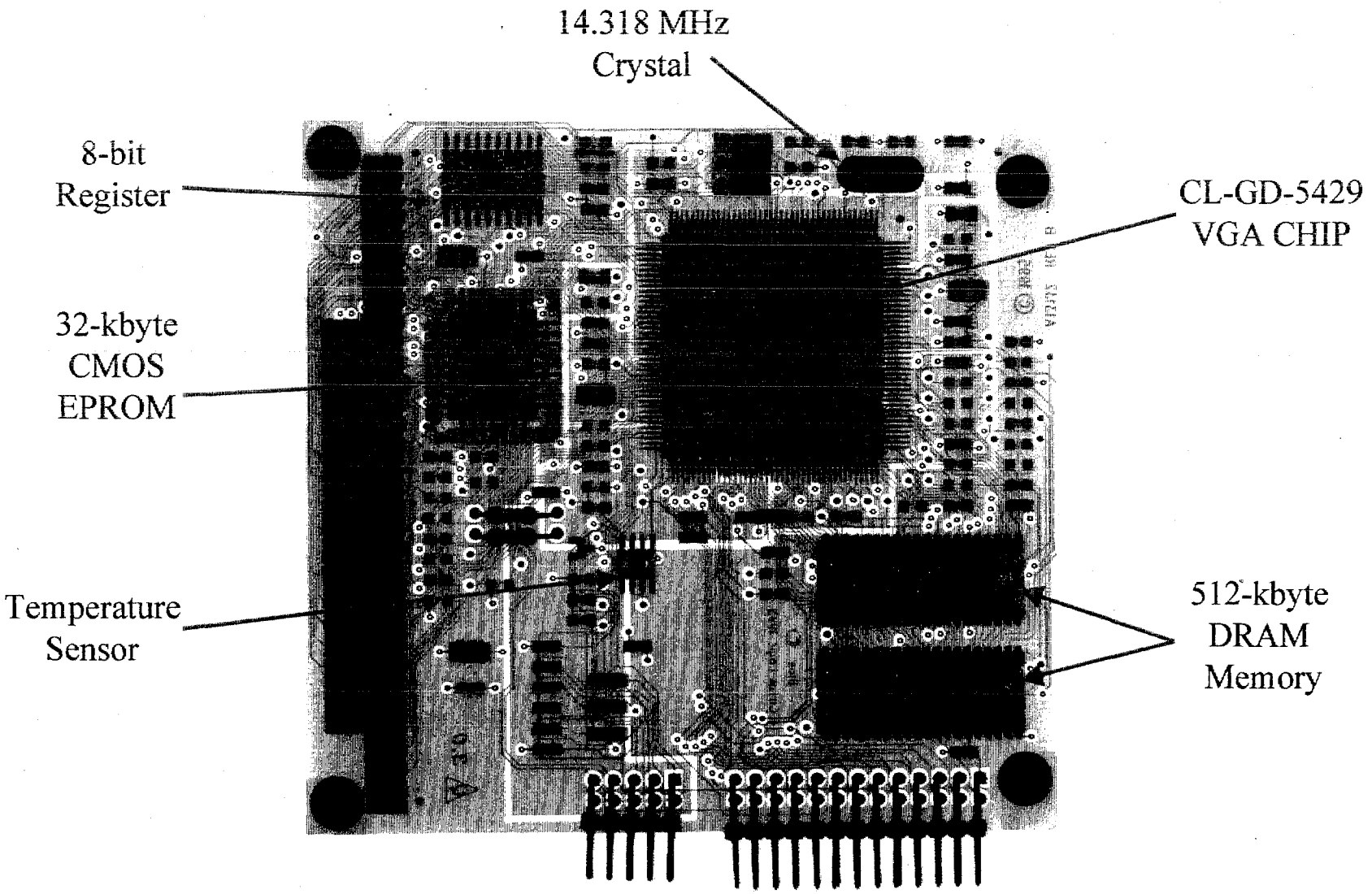
- Labeled top-view x-ray of the Ampro MiniModule SVG-II Card.

The x-ray can be compared to the top-view photograph in section 3.2.2. Note that the ICs are mounted on only one side of the printed circuit (PC) board. Thus, the large die in the center of the CL-GC5429 VGA IC can be seen without interfering images of ICs mounted on the other side of the PC board. The traces internal to the IC between the legs and the die can be seen. However, PC-feed-through holes can be seen under the VGA IC in the x-ray, which correspond to the holes observed in the bottom-view photograph. The PC traces and the internal IC traces can be distinguished most easily for the 32-kbyte CMOS EPROM. Traces from the top and bottom sides of the PC board can be distinguished to the.left of the DRAM memory ICs where the bottom-side "Cirrus Logic" label and traces can easily be seen to cross perpendicular topside traces.

The resolution of the actual x-ray negatives is considerably better than the scanning resolution used for these digital images. For example, the very fine wires internal to the IC between the die and the internal traces can be seen for many ICs.

The x-ray energy can be varied to become more penetrating. Presumably, the crystal in the can could be better viewed with more energetic x-rays. Ceramic ICs and plastic ICs warrant different x-ray energies for ideal viewing. The brightness, contrast, and gamma were optimized to maximize total detail for this print.

AMPRO SVG-II – Video Card – Top-View X-Ray



3.2.4 Jumper Settings for the Ampro MiniModule SVG-II Card

The SVG-II jumpers used select:

- No use of an interrupt
- Use of a 16-bit data bus.

Table of SVG-II Jumpers

Jumper	Function	Default	Used	Description
W1	IRQ selection	1=NC 2=NC	1=NC 2=NC	1=NC & 2=NC no IRQ used 1=2 IRQ used IRQ2 is used by the XT bus IRQ9 is used by the AT bus
W2	16-bit/8-bit bus selection	1=2 3=NC	1=2 3=NC	1=2 & 3=NC 16-bit bus used 1=NC & 2=3 8-bit bus used
W3	16-bit/8-bit bus selection	1=2 3=NC	1=2 3=NC	1=2 & 3=NC 16-bit bus used 1=NC & 2=3 8-bit bus used
W4	16-bit/8-bit bus selection	1=2 3=NC	1=2 3=NC	1=2 & 3=NC 16-bit bus used 1=NC & 2=3 8-bit bus used

The SVG-II manual only refers to IRQ2 usage, but the schematics refer to both IRQ2 and IRQ9. The AT bus is used for this application and the card would be using IRQ9.

The location of the jumpers on the SVG-II board is shown in Figure 2-1 on page 2-2 of the technical manual. The jumpers are well labeled on the board and easily found.

3.2.5 Parameter Settings Set by Software for the Ampro MiniModule SVG-II Card

The SVG-II comes up in standard VGA mode 03 with 25 lines of 80-character text and 720x400 resolution.

3.2.6 Schematics for the Ampro MiniModule SVG-II Card

The schematics for the Ampro SVG-II card require a non-disclosure agreement with Ampro.

3.2.7 Parts List for the Ampro MiniModule SVG-II Card

The following table list the integrated circuits used on the SVG-II video card.

Table of ICs Used in the SVG-II Video Card

Socket	Vendor	Part Number	Description
U1	CIRRUS	CL-GD5429	Memory-Mapped I/O VGA GUI Accelerator
U4	MT	4C16257	256kx16 -- 512-kbyte DRAM memory
U5	MT	4C16257	256kx16 -- 512-kbyte DRAM memory
U6	Unknown	27C256	32kx8 – 256 kbit CMOS EPROM Fairchild data sheet
U7	Motorola	74HCT244A	8-bit Register
U11	Unknown	L334-3A	Constant Source and Temperature Sensor Linear Technology data sheet
Y1	Unknown	E14.318	Clock crystal for CL-GD5429 -- 14.318 MHz

Note the 512-kbyte version of the video board (MM2-SVG-II-Q-71) has socket U5 empty and the 1-Mbyte version of the video board (MM2-SVG-II-Q-72) has socket U5 filled.

3.2.8 Voltage and Current Requirements for the Ampro MiniModule SVG-II Card

Power is provided to the SVG-II card via the standard PC104 AT bus extension. The following table provides the pins used and power consumption.

Table Providing Power Information for SVG-II Card

P1 Pin	**PC104 Standard**	**Comments**
B1	Ground	
B3	+5 VDC	Maximum: 1.7 watts 340 ma Typical: 1.0 watts 200 ma Minimum: 0.5 watts 100 ma
B5	-5 VDC	Not used
B7	-12 VDC	Not used
B9	+12 VDC	Not used
B31	Ground	
B29	+5 VDC	

3.2.9 Supplementary Narrative Describing Functional Blocks and Implementation for the Ampro MiniModule SVG-II Card

The primary component of the video card is the CIRRUS CL-GD5429 memory-mapped I/O VGA GUI accelerator IC.

The card uses a 27C256 256-kbit CMOS EPROM to convert the 16-bit input bus into an 8-bit bus in some undisclosed fashion using an 8-bit register (74HCT244A) to temporarily hold the output of the EPROM for the CIRRUS IC.

The card uses a 14.318-MHz crystal for the clock internal to the CIRRUS IC.

Details of the CIRRUS video IC's operation can be found in the data sheet in section 3.3.2 of this document.

3.2.10 Test Point Information for the Ampro MiniModule SVG-II Card

The following table contains illustrative examples of test points determined from the schematics and the individual IC data sheets. More test points can be readily added for a more thorough joint inspection.

Table of Recommended Test Points for the SVG-II Card

Location IC-Pin	**Test Object**	**Expected Result**	**Signal Name**	**Comments**
P1-B1	Input Power	Ground		Input Power Connector
P1-B3	Input Power	+5VDC ±5%		Input Power Connector
U1-69 or J2-7	Horizontal Synch	Pulses	HSYNC	Pulses consistent with normal Video horizontal synchronization
U1-68 J2-9	Vertical Synch	Pulses	VSYNC	Pulses consistent with normal Video vertical synchronization
U1-77 or J2-1	Red pixel Control line	Pulses	RED	700 mv full scale for pixel hard on in red
U1-76 or J2-3	Green pixel Control line	Pulses	GREEN	700 mv full scale for pixel hard on in green
U1-75 or J2-5	Blue pixel Control line	Pulses	BLUE	700 mv full scale for pixel hard on in blue
U1-48 or P18-B19	DRAM Refresh	Pulses	REFRESH*	L = Memory refresh is occurring
U1-158	Clock Freq.	14.318 MHz	XTAL	Clock crystal output

The width of the video pulses should be consistent with the video mode selected and the scan rates used.

3.2.11 Technical Manual for the Ampro MiniModule SVG-II Card

The technical manual for the SVG-II follows.

MiniModule™/SVG-II

Technical Manual

P/N: 5000921

Revision: A

Ampro Computers, Incorporated
4757 Hellyer Avenue ■ San Jose, CA 95138
Tel (408) 360-0200 ■ FAX (408) 360-0220
http://www.ampro.com

NOTICE

DISCLAIMER

Ampro Computers, Incorporated makes no representations or warranties with respect to the contents of this manual or of the associated Ampro software products, and specifically disclaims any implied warranties of merchantability or fitness for any particular purpose. Ampro shall under no circumstances be liable for incidental or consequential damages or related expenses resulting from the use of this product, even if it has been notified of the possibility of such damages. Ampro reserves the right to revise this publication from time to time without obligation to notify any person of such revisions. If errors are found, please contact Ampro at the address listed on the title page of this document.

TRADEMARKS

The Ampro logo is a registered trademark, and Ampro, Little Board, StackPlane, MiniModule, and CoreModule are trademarks of Ampro Computers, Inc. All other marks are the property of their respective companies.

TECHNICAL SUPPORT

Technical Support is available from 8:00 AM to 5:00 PM, Pacific time, Monday through Friday. Please have the product you wish to discuss at hand when calling. The telephone number is 800-966-5200.

Revision History

Revision	Reason for Change	Date
1	First Release	3/95
A	Production Release	7/95

PREFACE

This manual explains how to install a MiniModule™/SVG-II into a system based on an Ampro CPU module. This manual presumes that the reader has some knowledge of PC-based computer systems and, in particular, the PC VGA display architecture. This manual explains how to configure and install the controller, and how to set the modes for all the various resolutions, for text and graphics. It also provides information about the support software and drivers included with the MiniModule/SVG-II Development Kit.

Specifically, this manual contains three chapters, organized as follows:

- **Chapter 1—Introduction.** Provides an introduction to the features and specifications of the MiniModule/SVG-II. It provides information on which monitors are supported, and what resolutions are supported.

- **Chapter 2—Configuration and Installation.** Describes how to set jumpers, connect a monitor, and install the module in a system.

- **Chapter 3—Operation.** Shows how to install the support software, change video modes, and describes ways to program the controller using third-party software and BIOS calls. Provides a list of references to other vendor's software. Describes power management features. Outlines the functions of the feature connector.

TABLE OF CONTENTS

CHAPTER 1—INTRODUCTION

CHAPTER 2—CONFIGURATION AND INSTALLATION

CHAPTER 3—OPERATION

FIGURES

TABLES

CHAPTER 1

INTRODUCTION

1.1 GENERAL DESCRIPTION

The Ampro MiniModule/SVG-II expansion module is a compact, high-resolution graphics display controller, offering software selectable multimode operation. It is implemented in low power CMOS logic, and requires less than 2 watts of power. The module conforms to the PC/104™ Version 2.1 standard and attaches directly to any Ampro product having a PC/104-compatible header for MiniModule addition.

1.2 FEATURES

- Resolution up to 1024 x 768 with 256 colors, interlaced or non-interlaced.
- Text Density up to 132 columns x 43 rows
- Hardware and software compatible with the popular VGA video standards
- Software switching between modes
- Compatible with analog fixed frequency and multifrequency monitors
- Provides a feature connector for external video sources
- Conforms to the popular PC/104 Version 2.1 standard for embedded computers.
- 5 volt-only operation and low power consumption (<2 watts)
- Comprehensive power management features—Energy Star compliant

1.3 SOFTWARE COMPATIBILITY

The MiniModule/SVG-II is both register and BIOS-level compatible with all popular PC video standards. It supports the four most popular PC-compatible video formats, (VGA, EGA, CGA, and MDA), and also provides even higher Super VGA resolutions conforming to the VESA video display standard.

Specifically, the following extended-resolution Super VGA modes are available:

- 680 x 480 with up to 256 colors
- 800 x 600 with up to 256 colors
- 1024 x 768 with up to 256 colors, interlaced or non-interlaced
- Up to 132 columns x 43 rows text density

Software support for these Super VGA modes is widely available from a variety of sources, including C and other high-level language graphics libraries. A number of software drivers supporting the Super VGA modes is provided on the utility diskette that comes with the MiniModule/SVG-II Development Kit.

1.4 OUTPUT FLEXIBILITY

The MiniModule/SVG-II easily interfaces to standard VGA monitors. It supports various analog multifrequency monitors such as Sony MultiScan, NEC MultiSync, Packard Bell UniSync, Amdek

Smart Scan, Nanao FlexScan, and Mitsubishi Diamond Scan, and supports fixed frequency analog monitors such as the IBM 85XX family.

The MiniModule/SVG-II controller provides two display interface headers. One is for connecting a standard analog CRT display. The other is a PC-standard feature connector. Ampro provides a transition cable (DB15) with the MiniModule/SVG-II Development Kit to transfer signals to analog monitors.

1.5 POWER MANAGEMENT FEATURE

The MiniModule/SVG-II features comprehensive power management functions that support low-power operation. To take advantage of the features built into the MiniModule/SVG-II, the OEM can call video-BIOS extensions or set bits in registers in the video controller chip to place the controller and the attached display monitor in low-power modes. To take full advantage of the power saving features, the monitor must, of course, comply with the DPMS (Display Power Management Signaling) standard.

1.6 OPERATING MODES AND RAM REQUIREMENTS

The MiniModule/SVG-II controller supports many operating modes, offering a wide variety of resolution choices. The standard module comes with 512K bytes of video memory (DRAM), supporting all standard VGA compatible modes and a useful set of Super VGA modes. The module is also offered with 1 megabyte of video memory, useful when additional color support in high resolution modes are required.

Table 1-1 lists the characteristics of each text mode supported by the MiniModule/SVG-II, and Table 1-2 lists the characteristics for each graphics mode. Both tables show the amount of DRAM needed to support each mode. For the text modes, the table lists the number of rows and columns of text characters that can be displayed, and the cell size of the character. The table also tells you the number of colors that can be displayed in that mode. For the graphics modes, the table shows the total screen resolution in pixels and the number of colors that can be displayed simultaneously. Table 1-3 provides additional data useful in selecting monitors for each of the supported text and graphics modes, including horizontal and vertical frequency, buffer starting address, and clock rates.

As shown in Tables 1-1 and 1-2, with 512 KBytes of DRAM, the MiniModule/SVG-II controller supports extended text modes and graphics resolutions up to 1024 x 768 with 16 colors. With the full 1 megabyte of DRAM, resolutions up to 1024 x 768 with 256 colors are supported.

MODE	VESA MODE	PC STANDARD	ROWS/ COLUMNS	RESOLUTION	CELL SIZE	COLORS/ PALETTE	DRAM
0	-	CGA	40 x 25	320 x 200	8 x 8	16/256	256K
0	-	EGA	40 x 25	320 x 350	8 x 14	16/256	256K
0	-	VGA	40 x 25	360 x 400	9 x 16	16/256	256K
1	-	CGA	40 x 25	320 x 200	8 x 8	16/256	256K
1	-	EGA	40 x 25	320 x 350	8 x 14	16/256	256K
1	-	VGA	40 x 25	360 x 400	9 x 16	16/256	256K
2	-	CGA	80 x 25	640 x 200	8 x 8	16/256	256K
2	-	EGA	80 x 25	640 x 350	8 x 14	16/256	256K
2	-	VGA	80 x 25	720 x 400	9 x 16	16/256	256K
3	-	CGA	80 x 25	640 x 200	8 x 8	16/256	256K
3	-	EGA	80 x 25	640 x 350	8 x 14	16/256	256K
3	-	VGA	80 x 25	720 x 400	9 x 16	16/256	256K
7	-	MDA	80 x 25	720 x 350	9 x 14	mono	256K
7	-	VGA	80 x 25	720 x 400	9 x 16	mono	256K
14	-	Super VGA	132 x 25	1056 x 400	8 x 16	16/256K	512K
54	10A	Super VGA	132 x 43	1056 x 350	8 x 8	16/256K	512K
55	109	Super VGA	132 x 25	1056 x 350	8 x 14	16/256K	512K
NOTE: The MiniModule/SVG-II comes with a minimum of 512K DRAM							

Table 1-1 Display Characteristics of Text Modes

MODE	VESA MODE	PC VIDEO STANDARD	RESOLUTION	COLORS/ PALETTE	DRAM
4		CGA	320 x 200	4/256	256K
5		CGA	320 x 200	4/256	256K
6		CGA	640 x 200	2/256	256K
D		EGA	320 x 200	16/256	256K
E		EGA	640 x 200	16/256	256K
F		EGA	640 x 350	Mono	256K
10		EGA	640 x 350	16/256	256K
11		VGA	640 x 480	2/256	256K
12		VGA	640 x 480	16/256	256K
13		VGA	320 x 200	256/256	256K
58/6A	102	Super VGA	800 x 600	16/256K	256K
5C	103	Super VGA	800 x 600	256/256K	512K
5D	104	Super VGA	1024 x 768	16/256K	512K
5E	100	Super VGA	640 x 400	256/256K	512K
5F	101	Super VGA	640 x 480	256/256K	512K
60	105	Super VGA	1024 x 768	256/256K	1 M
NOTE: The MiniModule/SVG-II comes with a minimum of 512K DRAM					

Table 1-2 Display Characteristics of Graphics Modes

Mode	Horizontal (KHz)	Vertical (Hz)
0, 1, 2, 3, 4, 5, 6, 7, 0D, 0E, 0F, 10, 13, 14, 54, 55, 5E	31.5	70
11, 12, 5F	31.5, 37.9	60, 72
58, 6A, 5C	35.2, 37.8, 48.1, 46.9	56, 60, 72, 75
5D, 60	35.5, 48.3, 56, 58, 60	87, 60, 70, 72, 75

Table 1-3 Mode Timing Chart

Display Resolution	Horizontal (KHz)	Vertical (Hz)	Monitor type	CLMODE Monitor Type*
640 x 480	31.5	60	IBM 8512 IBM 8513 IBM 8503	0
640 x 480 1024 x 768	31.5 35.5	60 43.5 (i)	IBM 8514 IBM 8515	1
640 x 480 800 x 600	31.5 35.2	60 56	NEC 2A	2
640 x 480 800 x 600 1024 x 768	31.5 35.2 35.5	60 56 43.5 (i)	NEC II	3
640 x 480 800 x 600 1024 x 768	31.5 37.8 37.8	60 60 43.5 (i)	NEC 3D	4
640 x 480 800 x 600 1024 x 768	31.5 48.0 48.0	60 72 60	Sony CPD-1304 NEC 3FGX Nanao 9065S	5
640 x 480 800 x 600 1024 x 768	31.5 48.0 56.0	60 72 70	NEC 4D NEC 4FG Nanao T240i	6
640 x 480 800 x 600 1024 x 768	31.5 48.0 58.3	60 72 72	NEC 5D NEC 5FG, 6FG Nanao T560i	7

Table 1-4 Timing Parameters of Common Displays

* CLMODE Monitor Type is explained in Chapter 3, "Using the Ampro-Supplied Utility, CLMODE.EXE."

1.7 SPECIFICATIONS

The following sections list the specifications of the MiniModule/SVG-II.

1.7.1 Memory

- Onboard RAM — 512K bytes, 1 megabyte
- Display address — A0000h to BFFFFh
- ROM-BIOS address — C0000h to C7FFFh

1.7.2 I/O Address Space

- 03B4h to 03DAh, 46E8h

1.7.3 Interrupt Option

- IRQ2 (normally not used)

1.7.4 Video Characteristics

■ Bandwidth	25 MHz to 78.7 MHz
■ Horizontal Scan Rate	31.5 KHz to 48.3 KHz
■ Vertical Scan Rate	56 Hz to 75 Hz (87 Hz interlaced)

1.7.5 Physical

■ Power consumption	1.7 watts, worst case (Windows white screen) 1.0 watts, typical. 0.6 watts in Suspend Mode. 0.5 watts in Off Mode
■ Voltage	+5VDC ±5%
■ Size	3.6 x 3.8 x 0.6 inches (90 x 96 x 15 mm)
■ Operating Environment	
Temperature	0-70° C
Humidity	5-95% (non-condensing)
■ Storage temperature	-55° to +85° C
■ Weight	2.2 oz. (62.4 gms)

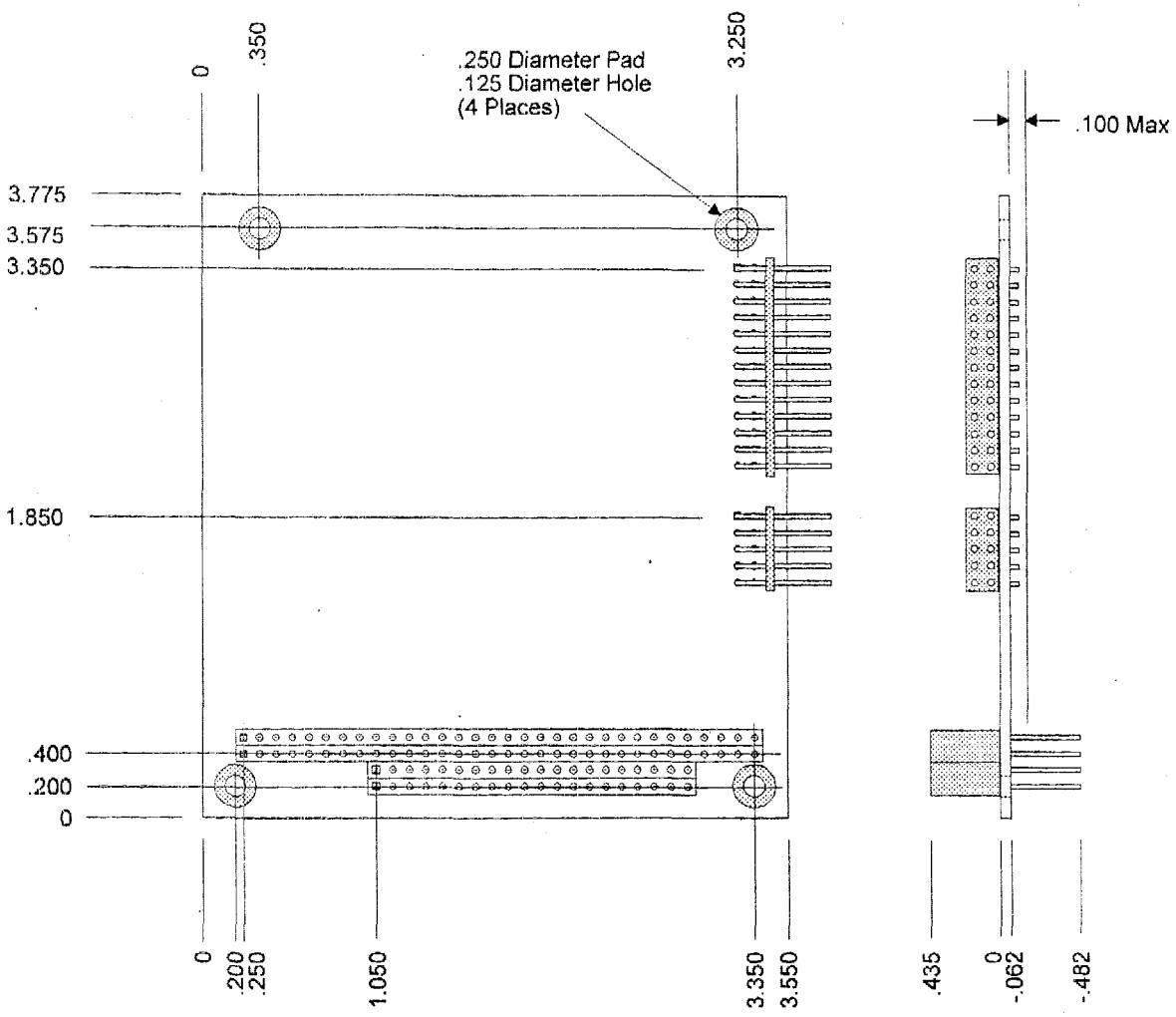


Figure 1-1 Mechanical Dimensions

CHAPTER 2

CONFIGURATION AND INSTALLATION

2.1 INTRODUCTION

This chapter provides information required to configure and install the MiniModule/SVG-II display controller in a system based on an Ampro CPU. This includes connecting cables and attaching the module to the Ampro CPU module. Since there are a minimal number of jumper settings and one interface cable, configuration is normally quite simple. When you complete these steps, your display controller is configured and ready to power up in the default VGA mode (730 X 400 Mode 3). If this is all you need to do, you may skip the rest of the manual. The information is organized as follows:

- **Setting the jumpers**—A discussion of how to set the configuration jumpers.
- **Selecting and cabling a monitor**—Information on interfacing Analog and TTL monitors.
- **Stacking the display controller with other Ampro modules**—Installation information for using the MiniModule/SVG-II in an Ampro CPU based system.

2.2 JUMPER SELECTIONS

There are four jumpers on the board, labeled W1, W2, W3, and W4. Table 2-1 summarizes how these jumpers are used.

Jumper	Description
W1	IRQ2 selection
W2, W3, and W4	16-bit/8-bit bus selection

Table 2-1 Jumper Summary

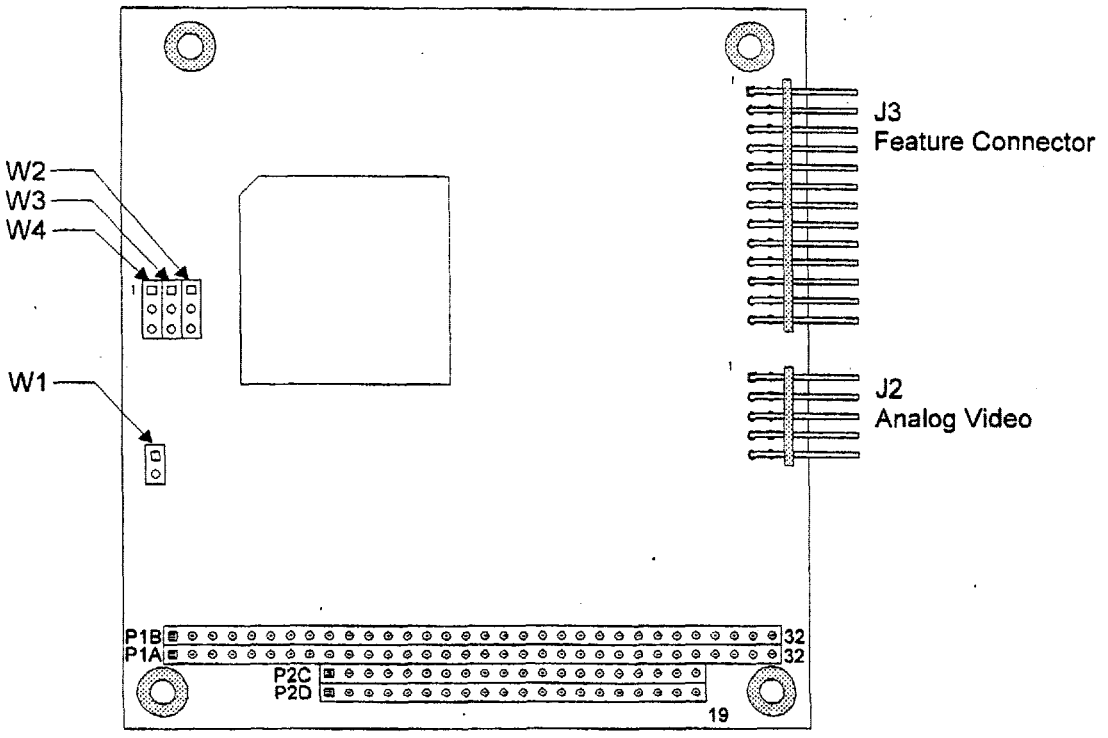


Figure 2-1 Connector and Jumper Locations

2.2.1 IRQ2 Jumpering (W1)

If your application needs interrupt IRQ2, short W1 with a jumper. Programs rarely require this interrupt enabled, so the default is disabled (no jumper installed), as shown in Figure 2-1. When it is not enabled, IRQ2 is available to other system devices.

2.2.2 16-bit/8-bit Bus Selection (W2-W4)

Jumpers W2, W3, and W4 are used to select whether the MiniModule/SVG-II will be used on an 8-bit bus or a 16-bit bus. W2, W3, and W4 are three-pin jumpers. The diagrams in Figure 2-2 illustrate how to set the jumpers for each configuration.

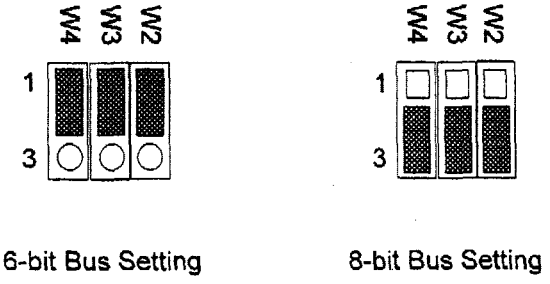


Figure 2-2 16-bit/8-bit Bus Selection Jumpering

2.3 ANALOG MONITOR INTERFACING (J2)

The MiniModule/SVG-II supports analog monitors for display. Table 1-3 and Table 1-4 in Chapter 1 provide cross references between video display modes and the timing specifications of popular monitors. Use these charts to select an appropriate monitor for your application. (Use of EL panels is discussed in Chapter 3, Using the Feature Connector.)

CAUTION

Because the MiniModule/SVG-II powers up in VGA mode, use *ONLY* a monitor that supports VGA mode as a minimum. During operation, never switch to a mode that the monitor cannot support or the monitor may be damaged.

Analog monitors use the 10-wire analog CRT transition cable, terminated in a female DB15 connector. This cable is included in the MiniModule/SVG-II Development Kit. If you are using this type of monitor, plug the cable's header connector into J2, and connect the monitor's cable to the DB15 connector. Table 2-2 shows the analog interface connector signal pinout and monitor cable wiring.

J2 Pins	Signal	DB15 Pin
1	Red	1
2	Ground	6
3	Green	2
4	Ground	7
5	Blue	3
6	Ground	8
7	Horizontal Sync	13
8	Ground	10
9	Vertical Sync	14
10	Ground	5

Table 2-2 Analog CRT Connector Signals (J2)

2.4 FEATURE CONNECTOR (J3)

The MiniModule/SVG-II provides a feature connector which presents all the standard IBM signals in the VESA standard pinout. You may use these signals with custom interfaces or with certain third-party products that use these signals. Chapter 3 describes more about the feature connector and about using it with third-party products (such as flat panel displays).

J3 Pin	Standard Connector Pin	Signal Description
2	Y1	Data Bit 0
4	Y2	Data Bit 1
6	Y3	Data Bit 2
8	Y4	Data Bit 3
10	Y5	Data Bit 4
12	Y6	Data Bit 5
14	Y7	Data Bit 6
16	Y8	Data Bit 7
18	Y9	DAC Clock
20	Y10	Blank
22	Y11	H Sync
24	Y12	V Sync
26	Y13	Ground
1	Z1	Ground
3	Z2	Ground
5	Z3	Ground
7	Z4	EVIDEO
9	Z5	ESYNC
11	Z6	EDCLK
13	Z7	n/c
15	Z8	Ground
17	Z9	Ground
19	Z10	Ground
21	Z11	Ground
23	Z12	n/c
25	Z13	n/c (key)

Table 2-3 Feature Connector Pinout (J3)

2.5 MODULE INSTALLATION

The MiniModule/SVG-II can be used in a variety of system configurations.

- It can be a part of a CoreModule™/MiniModule stack.
- It can plug onto an Ampro CoreModule or Little Board™ CPU.
- It can plug onto an Ampro accessory board.
- It can plug onto a custom carrier board.

The module is supplied with connectors compliant with the PC/104 specification. It is designed to plug to either a male or female header on any PC/104-compliant device.

To install the display controller:

- Either plug the pins of the module's bus header connector into the matching sockets of the bus header connector of the adjacent module or board, or plug the adjacent module's pins into the female bus header on the MiniModule/SVG-II. Be sure to mate the module's bus header correctly when you install it, or damage may occur when the unit is powered up.
- Use the spacers and associated hardware provided with the module to secure it to the parent board.
- Connect a monitor to the module, using an appropriate cable as discussed in the previous section.
- Once all attachments are secure, turn on the system power. The module will come up in standard VGA mode 03.

Refer to Chapter 3, Section 3-2 for information about initialization issues. These include the video type parameter that must be set on the CPU board, shadowing the video-BIOS for increased performance, and using the CLMODE utility to change the video mode and set the monitor timing.

CHAPTER 3

OPERATION

3.1 INTRODUCTION

This chapter discusses normal operation of the MiniModule/SVG-II, then describes using the display controller in resolution modes other than standard VGA. It also describes the various ways you can program the controller. It explains how to use the power management features. It concludes with a description of some of the uses for the feature connector.

3.2 INITIALIZATION

The MiniModule/SVG-II provides all standard VGA functions. Your software should work properly with all standard modes. The module also has enhancements to operate in "Super" VGA modes in the same manner as normal ones. All standard video-BIOS functions apply.

There is a "video type" parameter that must be set on the CPU. If it is not set to EGA/VGA, an error message will be displayed during the CPU's power-on self test (POST). If this is the case, invoke the CPU's SETUP function. (Consult the CPU's technical manual for details about the SETUP function.) Use SETUP to set the video type to "EGA/VGA." Use this setting even if you intend to use the display module in CGA or monochrome modes.

On most Ampro CPU modules, you can choose to use DRAM to "shadow" the video-BIOS. Shadowing the video-BIOS means that the contents of the video-BIOS PROM are copied into faster DRAM memory and executed from there, increasing the display speed. This increase of speed will be noticed in applications where the video-BIOS is used frequently. Consult your CPU technical manual to see if video-BIOS shadowing is available, and how to enable it in SETUP.

Upon power-up, the module will attempt to come up in standard VGA mode 03. You can change the display mode from the DOS command line using the CLMODE utility, provided on the utility diskette that comes with the MiniModule/SVG-II Development Kit. Its use is described in the following section.

3.3 SOFTWARE CONTROL OF THE DISPLAY

The MiniModule/SVG-II allows you to switch modes with your software application. This feature permits great flexibility, particularly in embedded applications.

The resolution you ultimately see on the screen is a function of the mode you select and the amount of RAM installed on the MiniModule/SVG-II.

There are a number ways to obtain various text and graphics modes and to operate or program the graphics capabilities of the MiniModule/SVG-II:

- Using the Ampro-supplied utility CLMODE.EXE.
- Using supplied drivers for 3rd party software.
- Using a graphics library or product.
- Using Video-BIOS functions.
- Programming the Cirrus CL-GD542x graphics chip directly.

3.3.1 Using the Ampro-Supplied Utility, CLMODE.EXE

The CLMODE.EXE utility allows the user to define the type of monitor attached to the MiniModule/SVG-II and to set video modes. CLMODE is a DOS command line-oriented utility. It is invoked from the DOS command line with the following command:

```
CLMODE
```

The main window presents a number of control "buttons". Each button represents a different option or menu. The underlined letter of a button name specifies the key combination for that item. To select a button item, hold down the ALT key and press the underlined letter key. Options require just the underlined letter. You may also use a mouse to select a control button. (If you will be using a mouse, be sure to install the mouse driver.)

The information in the main window displays the VGA controller chip type, the video-BIOS version number, and the amount of video memory present.

Choosing a Monitor Type

For the highest quality visual display for a given monitor, select the proper monitor type for the resolution you want. The monitor type also determines which video modes will be available to your system. It will also determine the vertical refresh rate timing sent to the display. Generally, the higher the refresh rate, the less flicker will be seen.

To choose a monitor type, select the **Monitor Type** button. The Monitor Type setup window will be displayed. The highlight will show the current monitor type. Use the cursor keys or the mouse to choose the type of monitor that you want.

Creating Custom Monitor Timings

Selecting **Advanced** from the Monitor Type setup window will enable you to enter custom monitor timings. Select the **Set Advanced** button. This brings up a dialog box with drop down menus for each available resolution. Select the desired refresh rate for each resolution you want to support. You may want to consult your monitor's technical manual to ascertain correct rates. When you have completed setting the refresh rates, select the **OK** button. To cancel the changes, select the **Cancel** button.

To test the new video timings, select the **Verify** button. The program will let you view each new video timing so that you can verify that they are compatible with your monitor. After each test screen is displayed, press **Enter** or click the left mouse button to see the next resolution. Press **ESC** or the right mouse button to exit to the monitor type setup menu. If a timing setting you made does not work, try a slower setting and then try verifying your choices again.

Once your are satisfied, save the changes. To save new monitor timing values, press the **Save** button. You will be asked if you wish to save the monitor type selection in your AUTOEXEC.BAT file. If you wish to have your system come up with the new settings, answer "yes".

Displaying the Available Video Modes

Select the **Video Mode** button. The Video Mode Preview window displays all the modes supported according to the monitor type selected and the amount of video memory present on the MiniModule/SVG-II. This list of video modes will tell you which are available in your current configuration for use with your application.

To see what different video modes look like on the monitor, select the **Preview** button. After each test screen is displayed, press **Enter** or the left mouse button to see the next video mode. Press **ESC** or the right mouse button to cancel previewing the video modes.

On-line Help

To view help, press the **Help** button. CLMODE provides help for the following items:

- Monitor type—Explains the different capabilities of each monitor type.
- Video modes—Defines the information given in the Video Mode window.
- Mouse—Explains how to use the mouse to make selections.
- Keyboard—Explains how to use the keyboard to make selections.
- About CLMODE—Displays the Cirrus Logic copyright message and the CLMODE version number.

Exiting CLMODE

To exit CLMODE, press **ALT** and **F4** simultaneously, or select the system button on the main window using the left mouse button. (The system button is the in the top left corner of the main window, shown as a dot.) You may also select the **Exit** button.

When you exit CLMODE, the current video mode, monitor type, and VGA refresh rate will be displayed.

Using CLMODE's Command Line Options

CLMODE can be commanded to select a mode number or resolution from the command line. This is especially useful when using CLMODE in an embedded system. You can invoke CLMODE with command line options from your AUTOEXEC.BAT file.

The command line options for CLMODE are:

```
CLMODE [[modenum][+ | * | -][m(type) | t6=x | t8=x | t1=x | t2=x][S]
```

where:

modenum	Video mode number
+	Selects 400 lines (default)
*	Selects 350 lines
-	Selects 200 line
type	Monitor type (Monitor types are given in Table 1-4.)
t6=x	t6 indicates 640 x 480 resolution x=0 means 60 Hz, x=1 means 72 Hz.
t8=x	t8 indicates 800 x 600 resolution x=0 means 56 Hz, x=1 means 60 Hz, X=2 means 72 Hz
t1=x	t1 indicates 1024 x 768 resolution x=0 means 87 Hz, interlaced, x=1 means 60 Hz, x=2 means 70 Hz, x=3 means 72 Hz.
S	Settings: entering S as a command line option will display the current CLMODE settings.

Entering an invalid parameter will display help text.

Examples:

- To select mode 3 for a Super VGA monitor (montype 2), type the following command at the DOS prompt:

```
CLMODE 3+ m2
```

- To select custom monitor timings with 640 x 480 at 60 Hz and 800 x 600 at 72 Hz, enter:

```
CLMODE t6=0 t8=2
```

3.3.2 Using Supplied Drivers for 3rd Party Software

Video drivers are available for a number of popular applications, including AutoCad, Ventura Publisher, Lotus 123, and MS Word for DOS. Embedded systems rarely use these 3rd party applications, so the drivers will not be covered in detail here, and are not included on the Utility Diskette that is supplied with the MiniModule/SVG-II Development Kit. The files that are included on the diskette change from time to time, so a detailed list is not given here. In general, however, the drivers for Microsoft Windows 3.1 are supplied. The OS/2 Warp operating system includes drivers for the CL-GD542X. Contact Ampro Technical Support for information about 3rd party drivers for the MiniModule/SVG-II.

Run INSTALL.EXE from the Utility Diskette to install the Windows drivers and the SetRES utility. You can also install drivers individually using the Windows Setup program. Once installed, use the SetRES utility to select the Windows driver for the resolution you want. It is easier to use than the standard Windows Setup program.

3.3.3 Using a Graphics Library or Product

Another way to change resolution modes is to use third-party products for VGA and Super VGA. Third party software often includes source code OEMs can modify and use in their own applications. These programs often perform many complex functions besides simple mode changes.

The following graphic libraries may be used with the MiniModule/SVG-II when using resolutions up to 1024 X 768 with 16 colors. They are compatible with Borland's *Turbo C* and *Turbo Pascal*, and with Microsoft's *Microsoft C* and *Microsoft Pascal*. These four popular languages also have substantial graphic support software included in their packages.

Essential Graphics, Version 4.0, South Mountain Software. 201-762-6965

GX Graphics, PCX Programmer's Toolkit, PCX Effect, GENUS Microprogramming. 713-870-0737

HALO Professional, Media Cybernetics. 800-992-4256

MetaWINDOW/PLUS, Metagraphics. 408-438-1550

You may find these two software product catalog/buyer's guides especially helpful in locating additional graphic libraries and products.

The Connection, published quarterly by Programmer's Connection Inc.,
7249 Whipple Ave., NW, North Canton, Ohio 44720-7143. 800-336-1166.

The Programmer's Shop, published quarterly by SDC Communications,
90 Industrial Park, Hingham, MA 02043. 800-447-8041.

3.3.4 Using Video-BIOS Functions

To program the graphics controller, you may use the built-in high level functions in the video BIOS. Using this programming interface, you can control resolution modes, and a number of other functions and characteristics. All calls are made through software interrupt 10h (INT 10H). When you make the call, the contents of the AH register in the graphics controller chip determine the primary function; the contents of the AL register determine the secondary function.

Table 3-1 is an overview of the interrupt 10 functions for register AH.

AH Content	Function
00	Mode Set
01	Set cursor type
02	Set cursor position
03	Read cursor position
04	Read light pen position (not supported)
05	Select active display page
06	Scroll active page up
07	Scroll active page down
08	Read character at current cursor position
09	Write characters at current cursor position
0A	Write characters only at current cursor position
0B	Set color palette
0C	Write dot
0D	Read dot
0E	Write teletypewriter to active page
0F	Return current video state
10	Set palette registers
11	Character generator routine
12	Alternate select
13	Write string
1A	Display combination code
1B	Return functionality/state information
1C	Save/restore
14-19	Reserved

Table 3-1 Video-BIOS Function Calls to INT 10, Register AH

3.4 PROGRAMMING THE CIRRUS CL-GD542X GRAPHICS CHIP DIRECTLY

Some applications may require the fine control gained by direct chip programming. This section contains a memory map of the MiniModule/SVG-II and the I/O addresses you will need to do this type of programming. You will need to consult the Cirrus Logic CL-GD542x Technical Reference Manual (available from Cirrus Logic) as well as one or more of the sources of information that are given in the next section.

3.4.1 Memory Map

Table 3-2 provides a typical system memory map for an embedded PC system that uses a VGA video controller.

Memory Address	Function
F0000-FFFFFh	Ampro ROM-BIOS
D0000-EFFFFh	SSD Sockets and expansion boards
C0000-C7FFFh	Video BIOS
A0000-BFFFFh	Video Screen RAM
00000-9FFFFh	640K bytes onboard DRAM for programs

Table 3-2 Typical System Memory Map

3.4.2 I/O Port Map

Table 3-3 lists the I/O port addresses used on the MiniModule/SVG-II display controller. These are the same I/O addresses used by all PC-compatible standard VGA adapters. The table lists the addresses of the six sets of register types. If you need detailed information about emulation modes dependency, DAC converters, VGA chip registers, accessing registers for programming, or other topics, consult the Cirrus Logic CL-GD542x Technical Reference Manual.

Function	I/O Port Address (hex)
General Registers	3BA or 3DA, 3CA, 3C2, 3CC
Reserved Registers	3C6 to 3C9
Sequencer Registers	3C4 and 3C5
CRTC Registers	3B4 to 3B5 or 3D4 to 3D5
Graphics Registers	3CE to 3CF
Attribute Registers	3C0 to 3C1
VGA Master Control	46E8/2E8*
* See note in text.	

Table 3-3 I/O Port Addresses

NOTE

Peripheral adapters that use address 2E8h will be in conflict with the address of the MiniModule/SVG-II display controller. This includes a popular address for COM4 serial ports, as well as a commonly used address for Arcnet LAN adapters. This is because the IBM VGA standard uses a 16-bit field for this particular I/O address, whereas most I/O devices only decode 10 bits. When the 16-bit address (46E8h) is truncated to 10 bits, it is decoded as 2E8h.

3.5 USING THE POWER MANAGEMENT FEATURES

The MiniModule/SVG-II's graphics controller chip, the Cirrus Logic CL-GD542x provides features that implement a comprehensive set of power management controls that can be used to sharply reduce power consumption of the board and monitor during periods of inactivity. These features can be used to meet the U.S. EPA's Energy Star power management standards.

This section describes methods that can be used with the MiniModule/SVG-II to reduce power consumption. Some of the methods involve programming registers in the CL-GD542x controller chip. Note that if your application program does reprogram controller chip registers, the following considerations apply:

- It must first save the register contents so that it can subsequently restore them.
- It may be necessary to unlock the extended registers.

The greatest power savings can be obtained by placing the attached display monitor into a low-power mode. A monitor compliant with the Display Power Management Signaling (DPMS) standard is required to take advantage of this type of power reduction. The VESA (Video Electronics Standards Association) DPMS standard defines four levels of display power, as shown in the following table:

Name	Definition	HSYNC	VSYNC	Note
On	Full operation	Active	Active	
Stand by	Minimal power reduction	Inactive	Active	
Suspend	Significant power reduction	Active	Inactive	
Off	Maximum power reduction	Inactive	Inactive	

3.5.1 Video-BIOS Functions for Power Management

Power management modes are set by calls to video-BIOS extensions. The MiniModule/SVG-II video-BIOS is compliant with the VESA Display Power Management BIOS Extensions, VBE/PM. These video-BIOS calls are described in this section.

There are 3 video-BIOS calls that can be used to control the power-saving modes:

- Report VBE/PM Capabilities
- Set Display Power State
- Get Display Power State

Report VBE/PM Capabilities

Input:	AH = 4Fh	VESA extension
	AL = 10h	VBE/PM services
	BL = 00h	Report VBE/PM capabilities
	ES:DI	NULL pointer, must be 0000:0000h. (Reserved for future use.)
Output:	AX	Status
	BH	Power saving state signals supported by the controller (Note 1)
		1 = supported, 0 = not supported
		Bit 0 Stand by (not supported)
		Bit 1 Suspend
		Bit 2 Off
		Bit 3 Reduced on (intended for flat panel displays)

Note 1: The attached monitor may not support all the power states that can be signaled by the controller. It is the responsibility of the power-management program to determine which power-saving states are available. The MiniModule/SVG-II supports **Suspend** and **Off**.

Set Display Power State

Input:	AH = 4Fh	VESA extension
	AL = 10h	VBE/PM services
	BL = 01h	Set display power state
	BH =	Requested power state:
		00h On
		01h Stand by (not supported)
		02h Suspend
		04h Off
		08h Reduced on (intended for flat panel displays)
Output:	AX	Status: If the requested state is not available, this function will return AX = 14h to indicate that the call failed.
	BH	Unchanged

Get Display Power State

Input:	AH = 4Fh	VESA extension
	AL = 10h	VBE/PM services
	BL = 02h	Get display power state
Output:	AX	Status: If this function is not supported by the controller hardware, AL = 01 is returned.
	BH	Returns the current power state:
		00h On
		01h Stand by (not supported)
		02h Suspend
		04h Off
		08h Reduced on (intended for flat panel displays)

3.5.2 Power Management Hardware Bits

This section documents hardware bits in the video controller chip that can be set directly from application programs to control the power consumption of the MiniModule/SVG-II. Implementation details for using these internal hardware register bits are beyond the scope of this manual. Details about how to use these features can be found in the Cirrus Logic CL-GD542x Technical Reference Manual.

DAC Power-Down

The DAC can be put into a power-down state by programming the Hidden DAC register to C7h.

Video Clock/Display Memory Refresh

The video clock can be reduced to as low as 3.46 MHz and the DRAM refresh can be reduced to 5 cycles every 4.6 uS without losing the contents of display memory. This reduces the power in the display memory array and also reduces the power in the controller chip.

Memory Clock

The memory clock, MCLK, can be reduced to as low as 7.14 MHz. This further reduces the power of the module.

3.6 USING THE FEATURE CONNECTOR

The MiniModule/SVG-II provides the standard IBM VGA feature connector signals at J3. You can use J3 to route an external TTL video source to an analog monitor connected at J2, or to drive certain flat panels. For more information on using the Feature Connector and its use for external video input, see the Cirrus Logic CL-GD542x Technical Reference Manual.

Table 2-3 gives the pinout for the feature connector. The signals are in the same order as a standard VESA feature connector found on some desktop PC expansion cards. Ampro does not offer a cable to support the feature connector.

To control the monitor with an external video source via the feature connector, you may need to block certain signals. Except for EVIDEO, EDCLK, and ESYNC, the signals are bi-directional.

EVIDEO, EDCLK, and ESYNC are input-only. When pulled low, these three signals tri-state (that is, set to a high impedance state) video data, pixel clock, and sync and blank signals generated on the MiniModule/SVG-II, allowing them to be driven by signals connected to the feature connector. This allows corresponding signals from the external source to drive an analog monitor connected at J2.

Various vendors' products use the feature connector signals as inputs or outputs. In particular, Planar Systems, Inc., of Beaverton, Oregon, makes electroluminescent (EL) flat panel displays that are compatible with the MiniModule/SVG-II feature connector. Such panels have VGA resolution of 640 X 400 or higher, up to 16 shades. A panel such as this can replace a monitor, or run in parallel with a monitor.

For further information, contact Planar Systems, Inc., 1400 Compton Dr., Beaverton, OR 97006. Phone (503) 690-1100; FAX (503) 645-7074

INDEX

3.3 Cirrus Logic, CL-GD5429 – Memory-Mapped I/O VGA GUI Accelerator IC

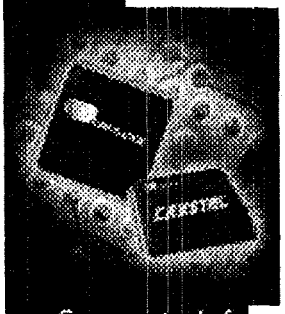
The CL-GD5429 IC is the primary component on the Ampro MiniModule SVG-II card. It determines most of the video functions and outputs. The specifications and data sheet for this IC are included for extra clarity and to supplement the SVG-II technical manual.

3.3.1 Specifications for the CL-GD5429

The following CL-GD5429 specifications were printed off the CIRRUS Internet site at:

http://www.cirrus.com/products/overviews/gd5429.html

Home
Site Map
Search

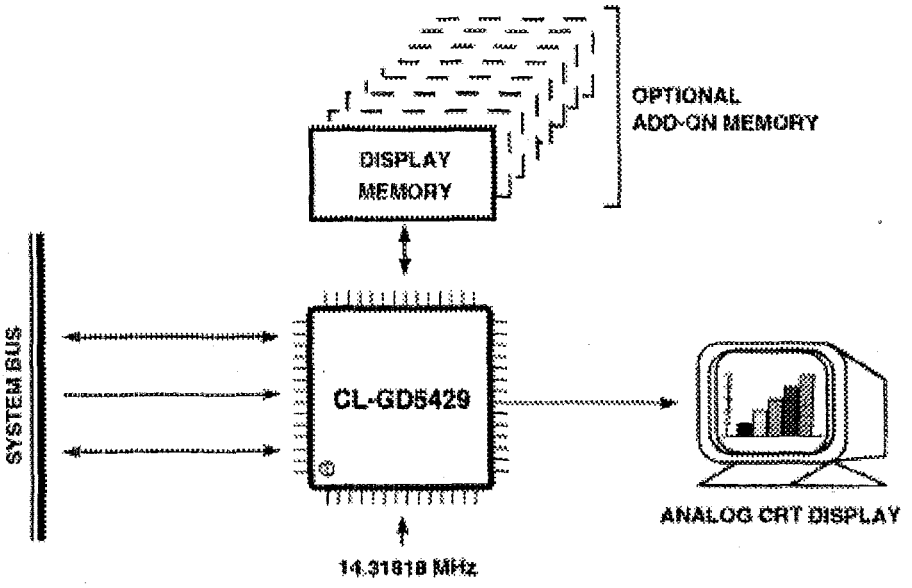


Corporate Info
News
Products
View by Category
View by Number
Sales Contacts
Publications
Year 2000
OEM Support
Drivers

Products

CIRRUS LOGIC

CL-GD5429: Memory-Mapped I/O VGA GUI Accelerator



- Pin- and software-compatible with CL-GD5420, CL-GD5422, CL-GD5424, CL-GD5426, and CL-GD5428
- Windows® acceleration features
  - Improved BitBLT engine with memory-mapped I/O for GUI acceleration
  - 64 x 64 hardware cursor
  - 8- or 16-bit/pixel color expansion
- VESA® VL-Bus™ 32-bit '486 and 16-bit '386 interface
- Integrated ISA bus interface
- Resolutions up to 1280 x 1024
  - 1280 x 1024 x 256 colors interlaced
  - 1024 x 768 x 32K and 64K colors
  - 800 x 600 x 32K and 64K colors
  - 640 x 480 x 16.8 million colors
- Integrated 24-bit true-color RAMDAC
- Integrated clock synthesizer, dot clock programmable up to 86 MHz
- Supports 2 Mbytes of display memory (16- or 32-bit bus)
- x4-, x8-, x16-wide DRAMs with multiple CAS* or WE*
- Video overlay with color key
- VGA solution with one or more DRAMs
- 100% hardware- and BIOS-compatible with IBM® VGA display standards
- Low-power CMOS technology in a 160-pin MQFP package

The CL-GD5429 supports all the CL-GD5428 features, has an improved BitBLT engine with memory-mapped register access, 60-MHz MCLK for high-speed DRAM, supports the VAFC (VESA advanced feature connector) for video overlay, and is fully 'Green PC' compliant.

3.3.2 Data Sheet for the CL-GD5429

The following CL-GD5429 data sheet was printed off the CIRRUS Internet site at:

http://216.35.18.143:80/servlet/SiteDriver/Content/656/gd542xtrm.pdf.

This data sheet is provided for clarity. This data sheet valuable provides details of the video card operation not available in the SVG-II manual.

CIRRUS LOGIC®

CL-GD542X

Preliminary Data Book

FEATURES

- **32-bit GUI acceleration (CL-GD5426/'28/'29)**
  - BitBLT (Bit block transfer) engine
  - Color expansion for 8- or 16-bit pixels
- **16/32-bit CPU interface**
  - VESA® VL-Bus™ (up to 50 MHz)
  - ISA bus (12.5 MHz)
  - Zero-wait-state write cycles
- **Resolutions up to 1280 x 1024**
  - 1024 x 768 x 256 colors, non-interlaced
  - 800 x 600 x 64K colors, non-interlaced
  - 640 x 480 x 16M colors, non-interlaced
  - 1280 x 1024 x 256 colors, interlaced
  - 1024 x 768 x 64K colors, interlaced
- **Programmable dual-clock synthesizer**
  - Pixel clock programmable up to 86 MHz
  - Memory clock programmable up to 60 MHz
- **Integrated 24-bit true-color RAMDAC**
- **'Green PC' power-saving features**
  - VESA® DPMS (Display Power Management Signal)
  - Internal DAC with programmable power-down mode
  - Static monitor sync signals
- **Support for multimedia applications**
  - 3-3-2 RGB DAC modes for video playback (CL-GD5425/'28/'29)
  - Support of VAFC (VESA® advanced feature connector) baseline for video overlay (CL-GD5425/'29)
- **100% hardware- and BIOS-compatible with IBM® VGA display standards**

(cont.)

True Color VGA Family

CL-GD5429	– Memory-Mapped I/O VGA GUI Accelerator with Local Bus
CL-GD5428	– Enhanced VGA GUI Accelerator with Local Bus
CL-GD5426	– VGA GUI Accelerator with Local Bus
CL-GD5425	– True Color VGA Controller with TV Output
CL-GD5424	– True Color VGA with Local Bus
CL-GD5422	– True Color VGA
CL-GD5420	– Super VGA

OVERVIEW

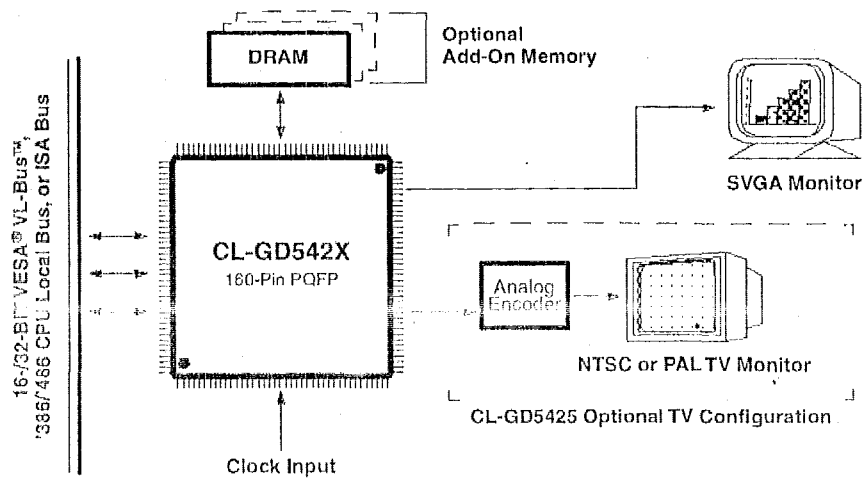
The CL-GD542X family of true-color VGA controllers offers an extensive range of industry-leading features and functionality for IBM®-compatible personal computers.

Ideally suited to highly integrated systems, CL-GD542X devices require no external support other than display memory and a crystal frequency reference. CL-GD542X devices are 100% hardware- and BIOS-compatible with IBM VGA standards, and connect directly to an ISA or local bus, allowing a minimum adapter solution.

Operating at dot clock rates programmable up to 86 MHz, CL-GD542X devices support standard and VESA® high-resolution and extended modes. The internal palette DAC may be configured as an industry-standard RAMDAC to provide a palette of 256K colors, or true-color displays of 32K, 64K, and 16.8 million colors.

(cont.)

System Block Diagram



OVERVIEW (cont.)

The internal dual-frequency synthesizer requires a single crystal or reference for all supported screen resolutions, as well as all standard display memory speeds and formats. The CL-GD542X devices implement all control and data registers according to current VGA standards. They also implement all standard data path and manipulation functions, providing complete hardware compatibility.

In addition, the CL-GD542X devices support extended registers and capabilities to provide functional and performance enhancements beyond standard VGA.

CL-GD542X devices support ISA or 32-bit VESA VL-Bus interfaces in all operations, including I/O and memory operations in planar modes. The write cycles to memory are optimized with zero-wait-state capability. Sixteen-/thirty-two-bit local bus interfacing can be achieved for '386SX, '386DX, and '486 microprocessors as well as VESA VL-Bus. The CL-GD5426/'28/'29 also offer BitBLT operation for GUI acceleration.

The CL-GD542X family also includes many power-saving ('Green PC') features, including an internal DAC with programmable power-down mode, sync signals that can be individually disabled (static levels), and internal clocks programmable to low frequencies for nearly static operation.

Software Support

Software Drivers	Resolution Supported[a]	No. of Colors
Microsoft® Windows® 95 *CL-GD5425*	640 × 480, 800 × 600, 1024 × 768	256
	640 × 480, 800 × 600	32,768
	640 × 480	16.8 million
Microsoft® Intel® DCI *CL-GD5425*	640 × 480, 800 × 600, 102 4 × 768	256
	640 × 480, 800 × 600	65,536
	640 × 480	16.8 million
Microsoft® Windows® v3.X *CL-GD5425*	640 × 480, 800 × 600, 1024 × 768, 1280 × 1024	16
	640 × 480, 800 × 600,1024 × 768, 1280 × 1024	256
	640 × 480, 800 × 600, 1024 × 768	65,536
	640 × 480	16.8 million
Microsoft® Windows NT™ v1.X	640 × 480, 800 × 600, 1024 × 768	16 and 256
OS/2® v2.0, v2.1	800 × 600, 1024, × 768	16 [b]
	640 × 480, 800 × 600, 1024, × 768	256 [b]
AutoCAD® v11, v12, Autoshade® v2.0, w/ Renderman, 3D Studio™ v1,v2	640 × 480, 800 × 600, 1024 × 768, 1280 × 1024	16
	640 × 480, 800 × 600,1024 × 768, 1280 × 1024	256
	640 × 480, 800 × 600, 1024 × 768	65,536
	640 × 480	16.8 million
GEM™ v3.X	800 × 600, 1024 × 768	16
Ventura Publisher® v2, v3	800 × 600, 1024 × 768	16
Lotus® 1-2-3® v2.X,	132 × 25, 132 × 43 (text)	16
	800 × 600	16
Lotus® 1-2-3® v3.X	132 × 25, 132 × 43 (text)	16
	800 × 600, 1024 × 768	16
Microsoft® Word v5.X	132 × 25, 132 × 43 (text)	16
	800 × 600, 1024 × 768	16
WordPerfect® v5.0	800 × 600	16
WordPerfect® v5.1	132 × 25, 132 × 43 (text)	16
	800 × 600, 1024 × 768	16
WordStar® v5.5–v7.0	800 × 600, 1024 × 768	16

[a] Not all monitors support all resolutions; 640 × 480 drivers will run on PS/2®,-type monitors. Extended resolutions are dependent upon monitor type and VGA system implementation.

[b] OS/2® v2.0 requires a v2.0 Corrective Service Pack for 256 .

CIRRUS LOGIC

CL-GD5425 Highlights
True Color VGA Controller with TV Output

FEATURES

- **True color VGA controller with TV output**
  - — Scaling fits full VGA display into TV viewing area while maintaining proper aspect ratio
  - — Flicker-filter reduces interlaced artifacts associated with computer-generated graphic images
- **Glueless interface to popular TV encoders**
- **Multimedia support**
  - — Video overlay of 16-bit RGB, 16-bit YCrCb
  - — 8-bit feature connector
  - — 16-bit VAFC (VESA® advanced feature connector)
  - — GENLOCK support
- **Graphics acceleration features:**
  - — Color expansion reduces host bus traffic
  - — 64 x 64 hardware cursor
  - — Display memory linear addressing
- **Flexible 16-bit host interface**
  - — VESA® VL-Bus™ (up to 50 MHz)
  - — ISA bus
- **Flexible 32-bit display memory interface**
  - — Supports 256K x4, x8, x16 DRAMs
  - — 512-Kbyte or 1-Mbyte memory capacity
- **Integrated 24-bit DAC**
  - — VGA resolution up to 1024 x 768, 256 colors
  - — NTSC resolution up to 640 x 480, 64K colors, scaled with flicker filter
  - — PAL resolution up to 640 x 480, 64K colors with flicker filter

OVERVIEW

The CL-GD5425 integrates a Super VGA controller, dual-frequency synthesizer, true-color palette DAC, and TV processing support into a single device.

A member of the industry-standard CL-GD542X family of true color VGA controllers, the CL-GD5425 is fully backed by software and design support.

The CL-GD5425 provides NTSC/PAL timing for standard VGA display modes, as well as the following extended resolutions:

Extended Resolutions for TV Output

Resolution	No. of Colors	Memory
640 x 480	256	512 Kbyte
640 x 480	64K	1 Mbyte
640 x 400	64K	512 Kbyte

The CL-GD5425 provides integrated scaling, flicker reduction, and a glueless encoder interface that delivers high-quality TV display at the lowest possible cost without the need for additional frame or line stores.

The programmable flicker-reduction function reduces interlaced artifacts inherent in computer-generated images displayed on interlaced TV monitors. The degree of filtering is selectable by the end-user.

The CL-GD5425 is 100% hardware- and BIOS-compatible with VGA standards, and connects directly to the VESA® VL-Bus™ or ISA bus. A single DRAM, two frequency references, and an economical analog encoder are added to make a complete set-top graphics system.

CL-GD542X ADVANTAGES

Unique Features	Benefits
Cost Effectiveness —	
■ Glueless interface to as few as one DRAM, built-in true-color palette DAC and dual-frequency synthesizer	❒ Minimizes chip count and board space; enables a cost-effective solution.
■ Interface to x4, x8, x16 DRAMs	❒ Allows design flexibility for use of appropriate type and amount of memory.
High Performance —	
■ 16-bit VESA® VL-Bus™ and local bus interface	❒ Increases system throughput.
■ Hardware BitBLT for Windows® (CL-GD5425/'26/'28/'29)	❒ Accelerates GUI applications such as Microsoft® Windows® and similar applications.
■ 32-bit-wide DRAM interface	❒ Eliminates display-memory bottleneck.
■ Maximizes fast-page mode access to display-memory DRAMs	❒ Improves CPU performance by accessing maximum bandwidth available from DRAM display memory.
■ Host access to DRAMs through advanced write buffers	❒ Provides faster host access for writes to display memory.
■ 15-, 16-, or 24-bit true-color palette DAC	❒ Provides high- and true-color display for photo-realistic images; 32K, 64K, or 16.8 million colors displayed simultaneously on screen for lifelike images.
Multimedia —	
■ 3-3-2 RGB DAC modes for video playback (CL-GD5425/'28/'29)	❒ Enables high-resolution playback for live video applications.
■ NTSC or PAL output (CL-GD5425)	❒ Allows TV viewing of PC games and applications.
■ Overlay and 'color key', and GENLOCK support	❒ Allows 16-bit per pixel interfacing through the VESA® connector for multimedia applications.
Compatibility —	
■ Compatible with VGA and VESA® standards	❒ Ensures compatibility with installed base of systems and software.
■ Drivers supplied at various resolutions for Windows® 3.1, Windows® 95™, and other key applications	❒ Provides a 'ready-to-go' solution that minimizes the need for additional driver development.
■ Connects directly to multifrequency analog monitors	❒ Drives all PC-industry-standard, high-resolution monitors to ensure compatibility.

BIOS SUPPORT

- Fully IBM® VGA-compatible BIOS
- Relocatable, 32 Kbytes with VESA® VL-Bus™ local bus support
- VBE (VESA® BIOS extensions) support in ROM
- Support for DPMS (display power management signaling) in ROM

UTILITIES

- Graphics and video diagnostics test
- Windows® and DOS utilities
- Video mode configuration utility — CLMODE
- Set resolution in Windows® utility — WINMODE
- Configurable system integration for OEMs — OEMSI

CL-GD542X Family Features

Features	'GD5420	'GD5422	'GD5424	'GD5425	'GD5426	'GD5428	'GD5429
Performance							
VESA® VL-Bus™ and Direct 80386 or 80486 CPU interface			✓	✓	✓	✓	✓
BitBLT engine					✓	Enhanced	MM I/O
Zero-wait-state operation	✓	✓	✓	✓	✓	✓	✓
Maximum display memory	1 Mbyte	1 Mbyte	1 Mbyte	1 Mbyte	2 Mbytes	2 Mbytes	2 Mbytes
Display memory interface	16-bit	32-bit	32-bit	32-bit	32-bit	32-bit	32-bit
Hardware cursor (in pixels)	up to 32 × 32	up to 64 × 64	up to 64 × 64	up to 64 × 64	up to 64 × 64	up to 64 × 64	up to 64 × 64
Maximum dot clock frequency	75 MHz	80 MHz	80 MHz	80 MHz	80 MHz	80 MHz	86 MHz
Maximum MCLK frequency	50 MHz	50 MHz	50 MHz	60 MHz	50 MHz	50 MHz	60 MHz
High integration							
Integrated palette DAC and dual-frequency synthesizer	✓	✓	✓	✓	✓	✓	✓
Motherboard VGA solution with only two ICs	✓	✓	✓	Plus Encoder	✓	✓	✓
Built-in port for VESA® feature connector	✓	✓	✓	✓	✓	✓	✓
Built-in ISA (up to 12.5 MHz) bus support	✓	✓	✓	✓	✓	✓	✓
Built-in TV output support				✓			
Flexibility							
Support for x4-, x8-, and x16-bit-wide DRAMs	✓	✓	✓	✓	✓	✓	✓
8- or 16-bit host bus I/O and memory interface	✓	✓	✓	✓	✓	✓	✓
8-bit gray and 3-3-2 RGB DAC modes				✓		✓	✓
CCIR 601 YCrCb mode				✓			
General							
100% hardware- and BIOS-compatible with IBM® VGA display standards	✓	✓	✓	✓	✓	✓	✓
'Green PC' compliant	✓	✓	✓	✓	✓	✓	✓
132-column text mode support	✓	✓	✓	✓	✓	✓	✓
46E8 or 3C3 sleep mechanism		✓	✓	✓	✓	✓	✓
Video overlay and 'color key' support		✓	✓	✓	✓	✓	✓
VESA® VAFC™ base support (for video overlay)				✓			✓
Low-power CMOS, 160-pin package	✓	✓	✓	✓	✓	✓	✓
Screen resolution and colors							
640 × 480	up to 256	up to 16M	up to 16M	up to 16M	up to 16M	up to 16M	up to 16M
800 × 600	up to 256	up to 64K	up to 64K	up to 64K	up to 64K	up to 64K	up to 64K
1024 × 768 (interlaced)	up to 256	up to 256	up to 256	up to 256	up to 256	up to 64K	up to 64K
1024 × 768 (non-interlaced)	up to 256	up to 256	up to 256	up to 256	up to 256	up to 256	up to 256
1280 × 1024 (interlaced)		up to 16	up to 16	up to 16	up to 256	up to 256	up to 256

Table of Contents

List of Figures

List of Tables

Revision History

The following are the differences between the July 1994 and May 1995 versions of this data book:

- The CL-GD5425 device and all pertinent information regarding it has been added.
- The BIOS timing diagrams have been updated.

CONVENTIONS

This section lists conventions used in this data book. 'CL-GD542X' represents CL-GD5420, CL-GD5422, CL-GD5424, CL-GD5425, CL-GD5426, CL-GD5428, and CL-GD5429, the six members of the True Color VGA controller family.

Abbreviations

Units of measure	Symbol
degree Celsius	°C
hertz (cycle per second)	Hz
kilobyte (1,024 bytes)	Kbyte
kilohertz	kHz
kilohm	kΩ
megabyte (1,048,576 bytes)	Mbyte
megahertz (1,000 kilohertz)	MHz
microfarad	µF
microsecond (1,000 nanoseconds)	µs
milliampere	mA
millisecond (1,000 microseconds)	ms
nanosecond	ns
picovolt	pV

The use of 'tbd' indicates values that are 'to be determined', 'n/a' designates 'not available', and 'n/c' indicates a pin that is a 'no connect'.

Acronyms

The following table lists acronyms used in this data book.

Acronym	Definition
AC	alternating current
BIOS	basic input/output system
BitBLT	bit boundary block transfer
CAD	computer-aided design
CAS	column address strobe
CLUT	color lookup table
CMOS	complementary metal-oxide semiconductor
CRT	cathode ray tube
DAC	digital-to-analog converter
DC	direct current

Acronym	Definition *(cont.)*
DPMS	display power management signaling
DRAM	dynamic random-access memory
EEPROM	electrically erasable/programmable read-only memory
EISA	extended industry standard architecture
EPROM	electrically programmable read-only memory
FIFO	first in/first out
HI-Z	high-impedance
HSYNC/VSYNC	horizontal/vertical synchronization
ISA	industry standard architecture
LSB	least-significant bit
LUT	lookup table
MD	memory data
MSB	most-significant bit
PCI	peripheral component interconnect
PQFP	plastic quad-flat pack
RAM	random-access memory
RAS	row address strobe
RGB	red, green, blue
ROPs	raster operations
R/W	read/write
SC	serial clock
TSR	terminate and stay resident
TTL	transistor-transistor logic
VAFC	VESA® advance feature connector
VESA®	Video Electronics Standards Association
VGA	video graphics array
VRAM	video random-access memory

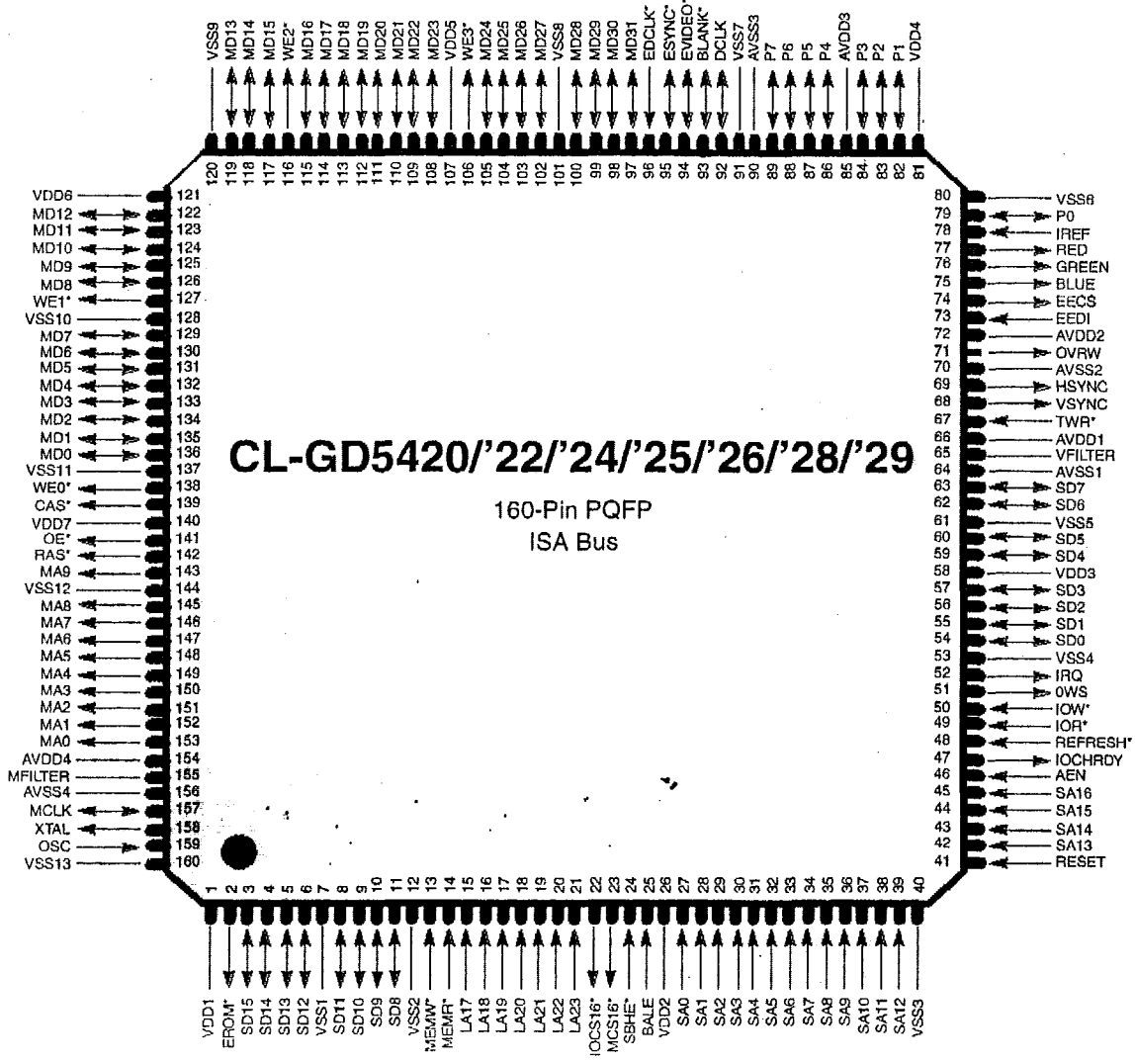
Numeric Naming

Hexadecimal numbers are represented with all letters in uppercase and a lowercase 'h' is appended to them (for example, '14h', '3A7h', and 'C000h' are hexadecimal numbers). Numbers not indicated by an 'h' are decimal.

1. PIN INFORMATION

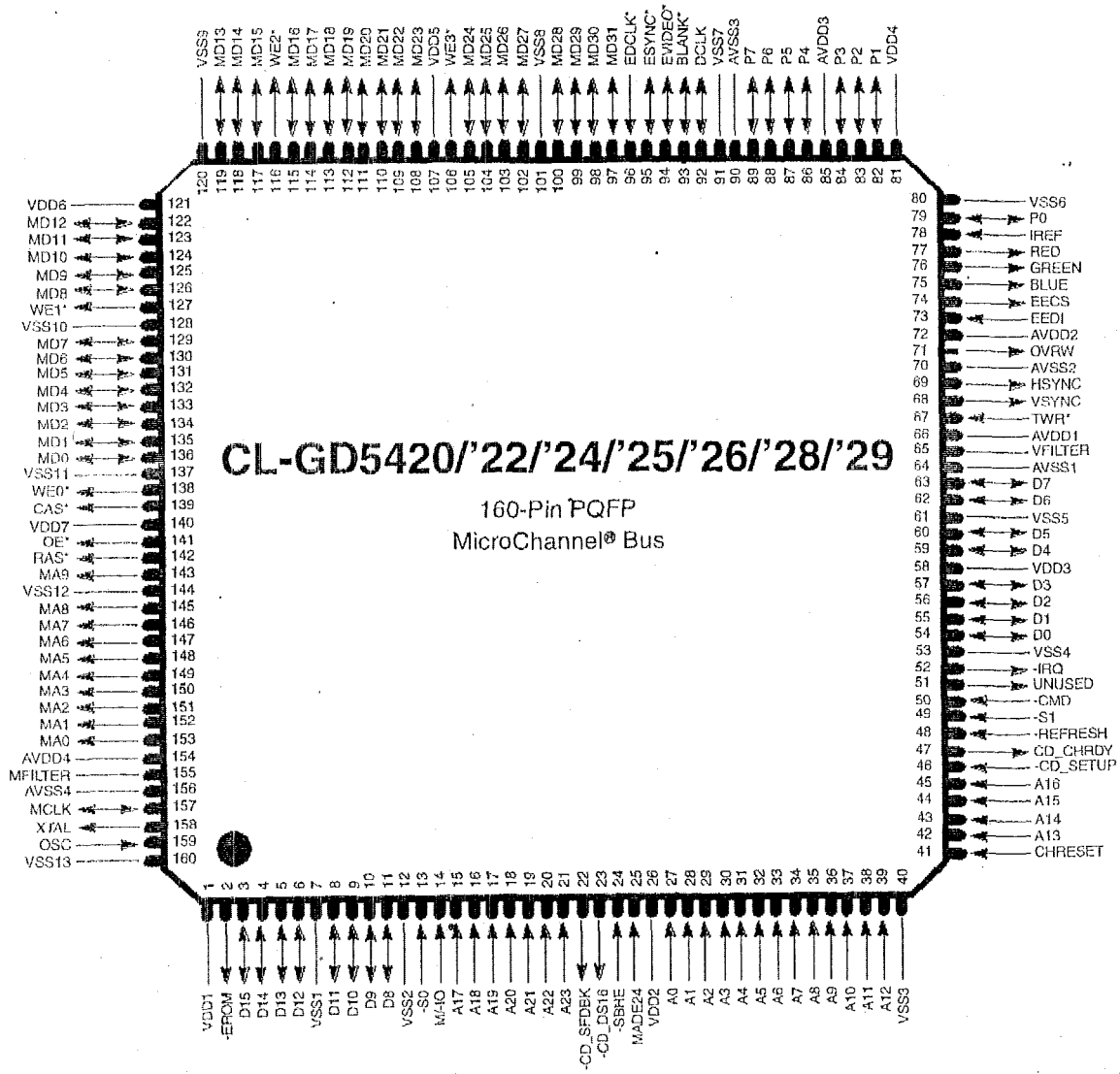
The CL-GD542X family of VGA controllers is available in a 160-pin quad flat pack device configuration, shown below.

1.1 Pin Diagram (ISA Bus)



NOTE: WE1*, WE0*, MD[15:0], and OVRW are reserved on CL-GD5420.

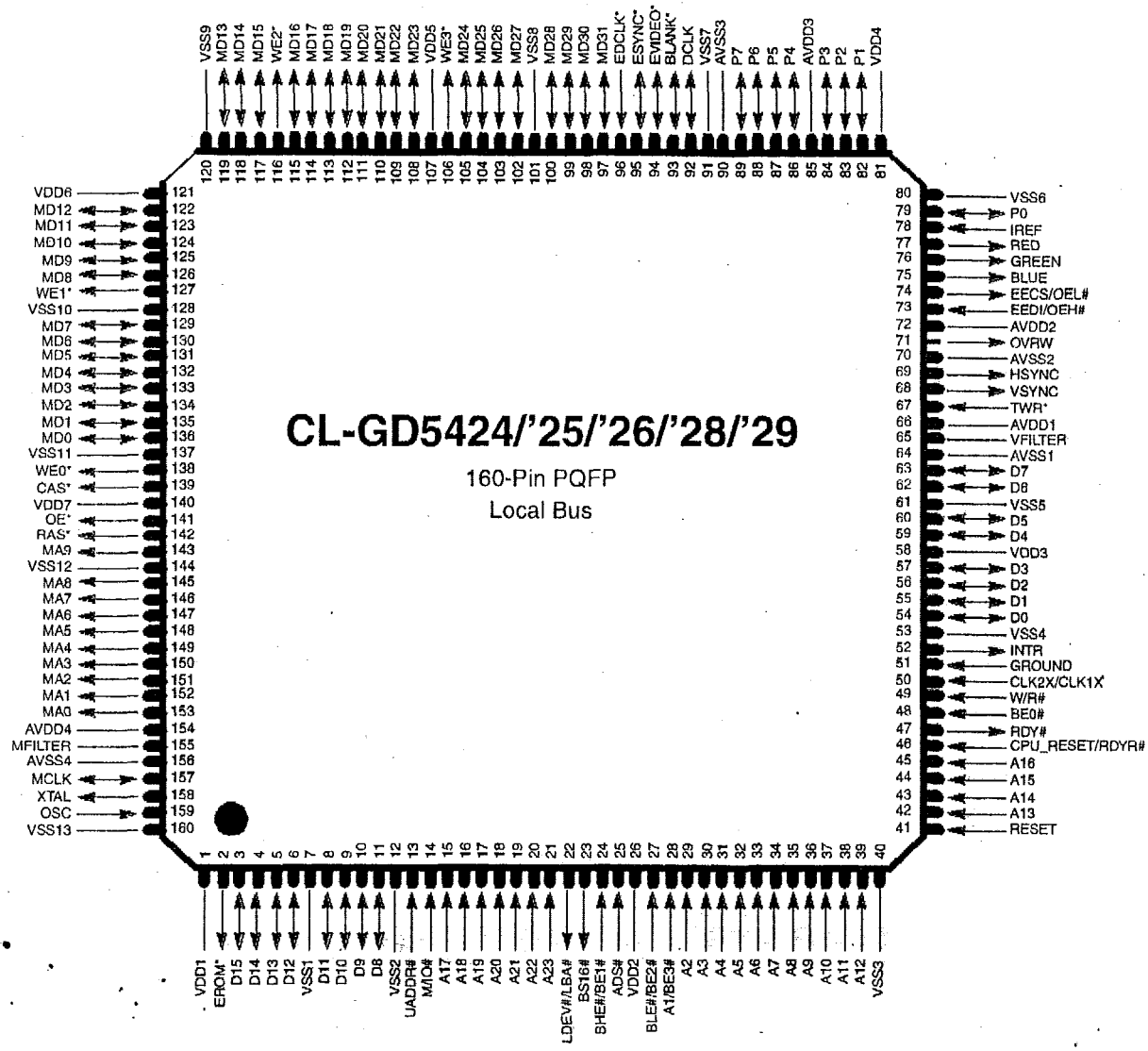
1.2 Pin Diagram (MicroChannel® Bus)



NOTES:

1) WE1*, WE0*, MD[15:0], and OVRW are reserved on CL-GD5420.
2) '-' indicates active-low on the MicroChannel bus.

1.3 Pin Diagram (Local Bus)



1.4 Pin Summary

The following abbreviations are used for pin types in the following tables: (I) indicates input; (O) indicates output; (I/O) indicates input or output depending on how the device is configured and programmed

Table 1-1. Host Interface — ISA/MicroChannel®

Pin Number	Pin Type	Pull-up[a]	I_{OH}[b] (mA)	I_{OL}(mA)	Load (pF)	ISA	MicroChannel®
21	I	•				LA23	A23
20	I	•				LA22	A22
19	I	•				LA21	A21
18	I	•				LA20	A20
17	I	•				LA19	A19
16	I	•				LA18	A18
15	I	•				LA17	A17
45	I					SA16	A16
44	I					SA15	A15
43	I					SA14	A14
42	I					SA13	A13
39	I					SA12	A12
38	I					SA11	A11
37	I					SA10	A10
36	I					SA9	A9
35	I					SA8	A8
34	I					SA7	A7
33	I					SA6	A6
32	I					SA5	A5
31	I					SA4	A4
30	I					SA3	A3
29	I					SA2	A2
28	I					SA1	A1
27	I					SA0	A0
3	I/O	•	–3	12	240	SD15	D15
4	I/O	•	–3	12	240	SD14	D14
5	I/O	•	–3	12	240	SD13	D13
6	I/O	•	–3	12	240	SD12	D12
8	I/O	•	–3	12	240	SD11	D11
9	I/O	•	–3	12	240	SD10	D10
10	I/O	•	–3	12	240	SD9	D9
11	I/O	•	–3	12	240	SD8	D8
63	I/O		–3	12	240	SD7	D7
62	I/O		–3	12	240	SD6	D6
60	I/O		–3	12	240	SD5	D5
59	I/O		–3	12	240	SD4	D4
57	I/O		–3	12	240	SD3	D3
56	I/O		–3	12	240	SD2	D2

Table 1-1. Host Interface — ISA/MicroChannel® *(cont.)*

Pin Number	Pin Type	Pull-up[a]	I_{OH}[b] (mA)	I_{OL}(mA)	Load (pF)	ISA	MicroChannel®
55	I/O		–3	12	240	SD1	D1
54	I/O		–3	12	240	SD0	D0
24	I	•				SBHE*	–SBHE
25	I					BALE	MADE24
46	I					AEN	–CD_SETUP
49	I					IOR*	–S1
50	I					IOW*	–CMD
14	I					MEMR*	M/–IO
13	I					MEMW*	–S0
41	I	•				RESET	CHRESET
48	I					REFRESH*	–REFRESH
47	O		–3	20	200	IOCHRDY	CD_CHRDY
22	O		–3	20	200	IOCS16*	–CD_SFDBK
23	O		–3	20	200	MCS16*	–CD_DS16
51	O		(OC)	20	200	0WS	(unused)
52	O		–3	20	200	IRQ	–IRQ

[a] • indicates the presence of a 250 kΩ, ± 50 % pull-up resistor.

[b] Data pads nominally rated at –3 mA I_{OH} will sink –15 mA at V_{OH} = 2.0 V.

Table 1-2. Host Interface — Local Bus (CL-GD5424/'25/'26/'28/'29 only)

Pin Number	Pin Type	Pull-up[a]	I_{OH}[b] (mA)	I_{OL} (mA)	Load (pF)	'386SX	'386DX	'486	VESA® VL-Bus™
21	I	•				A23	A23	A23	A23
20	I	•				A22	A22	A22	A22
19	I	•				A21	A21	A21	A21
18	I	•				A20	A20	A20	A20
17	I	•				A19	A19	A19	A19
16	I	•				A18	A18	A18	A18
15	I	•				A17	A17	A17	A17
45	I					A16	A16	A16	A16
44	I					A15	A15	A15	A15
43	I					A14	A14	A14	A14
42	I					A13	A13	A13	A13
39	I					A12	A12	A12	A12
38	I					A11	A11	A11	A11
37	I					A10	A10	A10	A10
36	I					A9	A9	A9	A9
35	I					A8	A8	A8	A8
34	I					A7	A7	A7	A7
33	I					A6	A6	A6	A6

Table 1-2. Host Interface — Local Bus (CL-GD5424/'25/'26/'28/'29 only) *(cont.)*

Pin Number	Pin Type	Pull-up[a]	I_{OH}[b] (mA)	I_{OL} (mA)	Load (pF)	'386SX	'386DX	'486	VESA® VL-Bus™
32	I					A5	A5	A5	A5
31	I					A4	A4	A4	A4
30	I					A3	A3	A3	A3
29	I					A2	A2	A2	A2
28	I					A1	BE3#	BE3#	BE3#
27	I					BLE#	BE2#	BE2#	BE2#
3	I/O	•	–3	12	240	D15	D15	D15	D15
4	I/O	•	–3	12	240	D14	D14	D14	D14
5	I/O	•	–3	12	240	D13	D13	D13	D13
6	I/O	•	–3	12	240	D12	D12	D12	D12
8	I/O	•	–3	12	240	D11	D11	D11	D11
9	I/O	•	–3	12	240	D10	D10	D10	D10
10	I/O	•	–3	12	240	D9	D9	D9	D9
11	I/O	•	–3	12	240	D8	D8	D8	D8
63	I/O		–3	12	240	D7	D7	D7	D7
62	I/O		–3	12	240	D6	D6	D6	D6
60	I/O		–3	12	240	D5	D5	D5	D5
59	I/O		–3	12	240	D4	D4	D4	D4
57	I/O		–3	12	240	D3	D3	D3	D3
56	I/O		–3	12	240	D2	D2	D2	D2
55	I/O		–3	12	240	D1	D1	D1	D1
54	I/O		–3	12	240	D0	D0	D0	D0
24	I	•				BHE#	BE1#	BE1#	BE1#
25	I					ADS#	ADS#	ADS#	LADS#
46	I					CPU–Reset	CPU–Reset	CPU–Reset	RDYRTN#
49	I					W/R#	W/R#	W/R#	W/R#
50	I					CLK2X	CLK2X	CLK1X	LCLK
14	I					M/IO#	M/IO#	M/IO#	M/IO#
13	I					(unused)	UADDR#	UADDR#	UADDR#
41	I	•				RESET	RESET	RESET	RESET
48	I					(unused)	BE0#	BE0#	BE0#
47	O		–3	20	200	READY#	READY#	RDY#	RDY#
22	O		–3	20	200	LBA#	LBA#	LBA#	LDEV#
23	O		–3	20	200	(unused)	BS16#	BS16#	LDS16#
51	I		(OC)	20	200	GROUND	GROUND	GROUND	GROUND
52	O		–3	20	200	INTR	INTR	INTR	INTR

[a] • indicates the presence of a 250 kΩ, ± 50 % pull up resistor.

[b] Data pads nominally rated at –3 mA I_{OH} will sink –15 mA at V_{OH} = 2.0 V.

Table 1-3. Synthesizer Interface

Pin Number	Pin Type	Pull-up	I_{OH} (mA)	I_{OL} (mA)	Load (pF)	Name
159	I					OSC
158	Analog Out/TTL In *(CL-GD5425 only)*					XTAL
155	Analog					MFILTER
65	Analog					VFILTER
157	I/O		–12	12	20	MCLK

Table 1-4. Video Interface

Pin Number	Pin Type	Pull-up[a]	I_{OH} (mA)	I_{OL} (mA)	Load (pF)	Name
68	I/O		–12	–12	50	VSYNC
69	I/O		–12	–12	50	HSYNC
93	I/O		–12	12	50	BLANK*
89	I/O		–12	12	50	P7
88	I/O		–12	12	50	P6
87	I/O		–12	12	50	P5
86	I/O		–12	12	50	P4
84	I/O		–12	12	50	P3
83	I/O		–12	12	50	P2
82	I/O		–12	12	50	P1
79	I/O		–12	12	50	P0
92	I/O		–12	12	50	DCLK
95	I/O	•				ESYNC*
94	I/O	•				EVIDEO*
96	I	•				EDCLK*
77	Analog Out					RED
76	Analog Out					GREEN
75	Analog Out					BLUE
78	Analog In					IREF

[a] • indicates the presence of a 250 kΩ, ± 50% pull-up resistor.

Table 1-5. Display Memory Interface

Pin Number	Pin Type	Pull-up[a]	I_{OH} (mA)	I_{OL} (mA)	Load (pF)	Name
142	O		–8	12	50	RAS*
139	O		–12	12	50	CAS*[b]
141	O		–12	12	50	OE*[c]
106	O		–12	12	50	WE3*[d]
116	O		–12	12	50	WE2*
127	O		–12	12	50	WE1*[e]
138	O		–12	12	50	WE0*[e]
143	O		–12	12	50	MA9
145	O		–12	12	50	MA8
146	O		–12	12	50	MA7
147	O		–12	12	50	MA6
148	O		–12	12	50	MA5
149	O		–12	12	50	MA4
150	O		–12	12	50	MA3
151	O		–12	12	50	MA2
152	O		–12	12	50	MA1
153	O		–12	12	50	MA0
97	I/O	•	–12	12	50	MD31
98	I/O	•	–12	12	50	MD30
99	I/O	•	–12	12	50	MD29
100	I/O	•	–12	12	50	MD28
102	I/O	•	–12	12	50	MD27
103	I/O	•	–12	12	50	MD26
104	I/O	•	–12	12	50	MD25
105	I/O	•	–12	12	50	MD24
108	I/O	•	–12	12	50	MD23
109	I/O	•	–12	12	50	MD22
110	I/O	•	–12	12	50	MD21

Table 1-5. Display Memory Interface *(cont.)*

Pin Number	Pin Type	Pull-up[a]	I_{OH} (mA)	I_{OL} (mA)	Load (pF)	Name
111	I/O	•	–12	12	50	MD20
112	I/O	•	–12	12	50	MD19
113	I/O	•	–12	12	50	MD18
114	I/O	•	–12	12	50	MD17
115	I/O	•	–12	12	50	MD16
117	I/O	•	–12	12	50	MD15[e]
118	I/O	•	–12	12	50	MD14[e]
119	I/O	•	–12	12	50	MD13[e]
122	I/O	•	–12	12	50	MD12[e]
123	I/O	•	–12	12	50	MD11[e]
124	I/O	•	–12	12	50	MD10[e]
125	I/O	•	–12	12	50	MD9[e]
126	I/O	•	–12	12	50	MD8[e]
129	I/O	•	–12	12	50	MD7[e]
130	I/O	•	–12	12	50	MD6[e]
131	I/O	•	–12	12	50	MD5[e]
132	I/O	•	–12	12	50	MD4[e]
133	I/O	•	–12	12	50	MD3[e]
134	I/O	•	–12	12	50	MD2[e]
135	I/O	•	–12	12	50	MD1[e]
136	I/O	•	–12	12	50	MD0[e]

[a] • indicates the presence of a 250 kΩ, ± 50 % pull-up resistor.

[b] CAS* is redefined as WE* for multiple-CAS* 256K x 16 DRAMs for the CL-GD5422/'24/'25/'26/'28/'29.

[c] OE* is redefined as RAS1* for 2-Mbyte display memory configurations for the CL-GD5426/'28/'29 only.

[d] WE*[3:0] are redefined as CAS*[3:0] for multiple-CAS* 256K x 16 DRAMs for the CL-GD5422/'24/'25/'26/'28/'29.

[e] WE1*, WE0, MD[15:0] are reserved on the CL-GD5420.

Table 1-6. Miscellaneous Pins

Pin Number	Pin Type	Pull-up[a]	I_{OH} (mA)	I_{OL} (mA)	Load (pF)	Name
74	Out		–12	12	35	EECS[b]
73	In					EEDI[c]
2	Out		–12	12	35	EROM*
71	Out		–12	12	35	OVRW[d]
67	In					TWR*

[a] • indicates the presence of a 250 kΩ, ± 50 % pull–up resistor.

[b] EECS is redefined as OEL# when the CL-GD5424/'25/'26/'28/'29 (only) is configured for '486, VESA VL-Bus, or local bus operation.

[c] EEDI is redefined as OEH# when the CL-GD5424/'25/'26/'28/'29 (only) is configured for '486, VESA VL-Bus, or local bus operation.

[d] OVRW is reserved on the CL-GD5420.

Table 1-7. Power and Ground

Pin Number	Pin Type	Pull-up	I_{OH}	I_{OL}	Load (pF)	Name	Note
140	Power					VDD7	Digital
121	Power					VDD6	Digital
107	Power					VDD5	Digital
81	Power					VDD4	Digital
58	Power					VDD3	Digital
26	Power					VDD2	Digital
1	Power					VDD1	Digital
160	Ground					VSS13	Digital
144	Ground					VSS12	Digital
137	Ground					VSS11	Digital
128	Ground					VSS10	Digital
120	Ground					VSS9	Digital
101	Ground					VSS8	Digital
91	Ground					VSS7	Digital
80	Ground					VSS6	Digital
61	Ground					VSS5	Digital
53	Ground					VSS4	Digital
40	Ground					VSS3	Digital
12	Ground					VSS2	Digital
7	Ground					VSS1	Digital
66	Power					AVDD1	VCLK
64	Ground					AVSS1	VCLK
154	Power					AVDD4	MCLK
156	Ground					AVSS4	MCLK
85	Power					AVDD3	DAC
72	Power					AVDD2	DAC
90	Ground					AVSS3	DAC
70	Ground					AVSS2	DAC

2. DETAILED PIN DESCRIPTIONS

The following abbreviations are used for pin types in the following sections: (I) indicates input; (O) indicates output; (I/O) indicates a bidirectional signal; (TS) indicates three-state; (OC) indicates open collector.

2.1 Host Interface — ISA Bus Mode

Name	Type	Description
LA[23:17]	I	**ADDRESS [23:17]:** These inputs, in conjunction with SA[16:0], are used to select the resource to be accessed during memory operations. These address bits are latched with the falling edge of BALE.
SA[16:0]	I	**ADDRESS [16:0]:** These inputs, in conjunction with LA[23:17]; are used to select the resource to be accessed during any memory or I/O operation. These address bits must remain valid throughout the cycle.
SD[15:8]	TS	**SYSTEM DATA [15:8]:** These bidirectional pins are used to transfer data during 16-bit memory or I/O operations. These pins can be directly connected to the corresponding ISA bus pins. These pads have pull-up resistors to guarantee a valid input level when not connected.
SD[7:0]	TS	**SYSTEM DATA [7:0]:** These bidirectional pins are used to transfer data during any memory or I/O operation. These pins can be directly connected to the corresponding ISA bus pins.
SBHE*	I	**SYSTEM BYTE HIGH ENABLE:** This input is used in conjunction with A[0] to determine the width and alignment of a data transfer. SBHE* and A[0] are decoded as shown in Table 2-1:
BALE	I	**BUS ADDRESS LATCH ENABLE:** This active-high input is used to latch LA[23:17] on the high-to-low transition.
AEN	I	**ADDRESS ENABLE:** If this input is high, it indicates that the current cycle is a DMA cycle. In this case, the CL-GD542X will not respond to I/O cycles. There is no effect on memory cycles.

Table 2-1. SBHE/A0 Decoding

SBHE*	A0	Function
0	0	16-bit Transfer
0	1	Upper-byte Transfer
1	0	Lower-byte Transfer
1	1	Lower-byte Transfer *(on odd address)*

2.1 Host Interface — ISA Bus Mode *(cont.)*

Name	Type	Description
IOR*	I	**I/O READ:** This active-low input is used to indicate that an I/O read is occurring. If the address on SA[15:0] is within the range of the CL-GD542X, it will respond by placing the contents of the appropriate register on the System Data bus.
IOW*	I	**I/O WRITE:** This active-low input is used to indicate that an I/O write is occurring. If the address on SA[15:0] is within the range of the CL-GD542X, it will respond by transferring the contents of the System Data bus into the appropriate register. The transfer will occur on the trailing (rising) edge of this signal. A list of I/O addresses that the CL-GD542X will respond appears in Section 5 on page 47. When a 16-bit I/O write is done, the address specified is typically the Index register for one of the VGA groups. The index should appear on SD[7:0] and the data should appear on SD[15:8].
MEMR*	I	**MEMORY READ:** This active-low input is used to indicate that a memory read is occurring. If linear addressing is being used, this pin must be connected to ISA signal MEMR*. If linear addressing is not being used, this pin must be connected to ISA signal SMEMR*. The CL-GD542X decodes A[23:15] to determine if a display memory read is occurring. If so, data is placed on the System Data pins according to the read mode and the contents of display memory. The CL-GD542X decodes A[23:15] to determine if a BIOS read is occurring. If so, the CL-GD542X makes EROM* active for the duration of MEMR*.
MEMW*	I	**MEMORY WRITE:** This active-low input is used to indicate that a memory write is occurring. If linear addressing is being used, this pin must be connected to ISA signal MEMW*. If linear addressing is not being used, this pin must be connected to ISA signal SMEMW*. The CL-GD542X decodes A[23:15] to determine if a display memory write is occurring. If so, data is written into display memory according to the write mode and the data on SD[15:0]. The data are latched in the CL-GD542X on the rising edge of this signal, and are actually transferred to display memory later.
RESET	I	**RESET:** This active-high signal is used to initialize the CL-GD542X to a known state. The trailing (falling) edge of this input loads the Configuration register CF[14:0] with the data on MD[30:16], determined by internal pull-up resistors and (optional) external pull-down resistors.
REFRESH*	I	**REFRESH*:** This active-low signal indicates that a DRAM refresh is occurring. The CL-GD542X ignores memory read operations occurring when REFRESH* is active since it controls the refresh of display memory.

2.1 Host Interface — ISA Bus Mode *(cont.)*

Name	Type	Description
IOCHRDY	TS	**I/O CHANNEL READY:** When driven low, this output indicates that additional wait states are to be inserted into the current display memory read or write cycle. This output is never driven low during I/O cycles or BIOS reads. During a display memory read cycle, this signal is always driven low as soon as MEMR* goes active. When the data are ready to be placed on the System Data bus, this signal is driven high. It remains high until MEMR* goes inactive; it then goes high impedance. During a display memory write cycle, this signal is driven high as soon as MEMW* goes active if there is space in the Write Buffer. If there is no space in the Write Buffer, this signal is driven low as soon as MEMW* goes active and remains low until there is space. Once there is space in the Write Buffer, this signal is driven high. It will remain high until MEMW* goes inactive; it then goes high-impedance.
IOCS16*	OC	**I/O CHIP SELECT 16*:** This open-collector output is driven low to indicate that the CL-GD542X can execute an I/O operation at the address currently on the bus in 16-bit mode. This signal is generated from a decode of A[15:0] and AEN. Table 2-2 indicates the range of addresses that the CL-GD542X will generate IOCS16*:

Table 2-2. IOCS16* Addresses

Address	Function
3C4, 3C5	Sequencer
3CE, 3CF	Graphics controller
3B4/3D4, 3B5/3D5	CRT controller
3BA/3DA	Input Status register 1

Name	Type	Description
MCS16*	OC	**MEMORY CHIP SELECT 16*:** This open-collector output is driven low to indicate that the CL-GD542X can execute a memory operation at the address currently on the bus in 16-bit mode. Table 2-3 summarizes the conditions where MCS16* is made active.

Table 2-3. MSC16* Addresses

Resource	Address Bits	Address Range	Qualifier
Display memory	A[23:17]	A000:0-BFFF:F	SR8[6] = 1 (No other VGA card)
Display memory	A[23:17]	1 Mbyte	SR7[7:4] ≠ 0 Linear Addressing
BIOS	A[23:15]	C000:0-C7FF:F	CF[6] = 0 (16-bit BIOS)

NOTE: The SA bits are generated late enough to typically make them unusable for generating MCS16*. The CL-GD542X uses a fast path from SA[16:15] to MCS16*.

2.1 Host Interface — ISA Bus Mode *(cont.)*

Name	Type	Description
0WS*	OC	**ZERO WAIT STATE*:** This open-collector output is driven low to indicate that the current cycle can be completed without any additional wait states. The circumstances under which 0WS* will be made active are summarized in Table 2-4.
IRQ	TS	**INTERRUPT REQUEST:** This active-high output indicates the CL-GD542X has reached the end of an active field. Specifically, the transition occurs at the beginning of the bottom border. This pin is typically unused in PC/AT add-in cards, but can be connected to IRQ2/IRQ9 via a jumper block. See register CR11 for a description of the controls for this pin.

Table 2-4. Zero Wait State* Cycles

Cycle Type	Qualifier
Display memory write	Write buffer not full
BIOS Read	CF[1] = 0 (not on CL-GD5429)

2.2 Host Interface — MicroChannel® Bus Mode

Name	Type	Description
A[23:0]	I	**ADDRESS [23:0]:** These inputs are used to select the resource to be accessed during a memory or I/O operation. These address bits are latched with the falling edge of –CMD. A[23:17] have internal pull-ups, whereas A[16:0] do not.
D[15:0]	TS	**DATA [15:0]:** These bidirectional pins are used to transfer data during memory or I/O operation. These pins can be directly connected to the corresponding MicroChannel bus pins.
–SBHE	I	**–SYSTEM BYTE HIGH ENABLE:** This input is used in conjunction with A[0] to determine the width and alignment of a data transfer. This signal is latched with –CMD low. This pad has a pull-up resistor. –SBHE and A[0] are decoded as shown in Table 2-5.
MADE24	I	**MEMORY ADDRESS ENABLE 24:** This active-high input is latched the falling edge of –CMD. It indicates that the address is in the lower 16 Mbytes of address space. MADE24 must be high for the CL-GD542X to participate in a memory cycle.
–CD_SETUP	I	**–CARD SETUP:** When this active-low input is active, the CL-GD542X is placed in Setup mode. In Setup mode, the CL-GD542X will respond only to POS102 accesses. It will not respond to any other I/O accesses or to display memory accesses. It will respond to BIOS reads. This signal is latched with the falling edge of –CMD.

Table 2-5. –SBHE/A0 Decoding

–SBHE	A0	Function
0	0	16-bit Transfer
0	1	Upper-byte Transfer
1	0	Lower-byte Transfer
1	1	Lower-byte Transfer *(on odd address)*

2.2 Host Interface — MicroChannel® Bus Mode *(cont.)*

Name	Type	Description
-S1	I	**-STATUS 1:** This signal, in conjunction with -S0 and M/-IO, is used to determine the cycle type that occurs. The encoding is shown in Table 2-6:
-CMD	I	**-COMMAND:** The falling edge of this input is used to latch the address bus, MADE24, -SBHE, -REFRESH, M/-IO, -CD_SETUP, -S0, and -S1. It is also used to time the actual data transfer. During I/O or memory-read cycles, the CL-GD542X drives valid data onto the bus prior to the trailing edge of this signal. During write cycles, the CL-GD542X expects valid data while this input is active and latches the data at the trailing edge.
M/-IO	I	**MEMORY/-IO:** This signal, in conjunction with -S0 and -S1, is decoded to determine the cycle type. See the description of -S1.
-S0	I	**-STATUS 0:** This signal, in conjunction with -S1 and M/-IO, is decoded to determine the cycle type. See the description of -S1.
RESET	I	**RESET:** This active-high signal is used to initialize the CL-GD542X to a known state. The trailing (falling) edge of this input loads the Configuration register CF[14:0] with the data on MD[30:16], determined by internal pull-up resistors and (optional) external pull-down resistors.
-REFRESH	I	**-REFRESH:** This active-low signal indicates that a DRAM refresh is occurring. This signal is latched with -CMD low. The CL-GD542X ignores memory-read operations occurring when -REFRESH is active since it controls the refresh of display memory.

Table 2-6. MicroChannel® Cycle Type Encoding

M/-IO	-S0	-S1	Cycle
0	0	0	Reserved
0	0	1	I/O write
0	1	0	I/O read
0	1	1	Reserved
1	0	0	Reserved
1	0	1	Memory write
1	1	0	Memory read
1	1	1	Reserved

2.2 Host Interface — MicroChannel® Bus Mode *(cont.)*

Name	Type	Description
CD_CHRDY	O	**CARD CHANNEL READY:** This output is driven low to request that additional wait states be inserted into the current display memory read or write cycle. This output is never driven low during I/O cycles or BIOS reads. During a display memory read cycle, this signal is always driven low as soon as -S1 goes low. When the data are ready to be placed on the System Data bus, this signal is driven high. During a display memory write cycle, this signal is driven high as soon as -S0 goes low if there is space in the Write Buffer. If there is no space in the Write Buffer, this signal is driven low as soon as -S0 goes low, and remains low until there is space. Once there is space in the Write Buffer, this signal is driven high.
-CD_SFDBK	OC	**-CARD SELECTED FEEDBACK:** This open-collector output is driven low to indicate that the CL-GD542X can respond to the addresses currently on the bus. This signal is generated from a decode of -REFRESH, MADE24, A[23:0], and M/-IO. This signal is made active for the Address Range C000:0–C7FF:F only if CF[6] = 0 (indicating a 16-bit BIOS). If CF[6] = 1, this signal will not be made active for Address Range C000:0–C7FF:F. Also, this signal will not be made active for Addresses 102 or 103 if M/-IO is low, indicating I/O.
-CD_DS16	OC	**-CARD SIZE 16:** This open-collector output is driven low to indicate that the CL-GD542X can execute a memory or I/O operation at the address currently on the bus in 16-bit mode. This output is generated from a decode of A[23:0], MADE24, -REFRESH, and M/-IO. Table 2-7 summarizes the conditions under which -CD_DS16 is made active.

Table 2-7. -CD_DS16 Addresses

Resource	Address Bits	Address Range
Display memory	A[23:15]	A000:0–BFFF:F
Display memory	A[23:15]	1 Mbyte
BIOS (CF[6] = 0 only)	A[23:11]	C000:0–C7FF:F
I/O	A[15:1]	3C4, 3C5 3CE, 3CF 3B/D4, 3B/D5 3B/DA

-IRQ	OC	**-INTERRUPT REQUEST:** This open-collector output indicates the CL-GD542X has reached the end of an active field. Specifically, the transition occurs at the beginning of the bottom border. This pin is typically connected to IRQ9 via a jumper block. See register CR11 for a description of the controls for this pin. This pin is never driven high.

2.3 Host Interface — Local Bus (CL-GD5424/'25/'26/'28/'29 only)

A number of bus interface pins are redefined according to the local bus type connecting to the CL-GD5424/'25/'26/'28/'29. The host interface pins are listed in Table 2-8 by CL-GD5424/'25/'26/'28/'29 pin number.

Table 2-8. Redefined Host Interface Pins

Pin	'386SX	'386DX	'486	VESA® VL-Bus™
13	(unused)	UADDR#	UADDR#	UADDR#
23	(unused)	BS16#	BS16#	LBS16#
24	BHE#	BE1#	BE1#	BE1#
27	BLE#	BE2#	BE2#	BE2#
28	A1	BE3#	BE3#	BE3#
46	CPU-RESET	CPU-RESET	GND	RDYRTN#
47	READY#	READY#	BRDY#	BRDY#
48	(unused)	BE0#	BE0#	BE0#
50	CLK2X	CLK2X	CLK1X	LCLK
51	GROUND	GROUND	GROUND	GROUND
73	(unused)	(unused)	OEH#	OEH#
74	(unused)	(unused)	OEL#	OEL#

2.3 Host Interface — Local Bus (CL-GD5424/'25/'26/'28/'29 only) *(cont.)*

Name	Type	Description
A[23:2]	I	**ADDRESS [23:2]:** These inputs are used to select the resource to be accessed during memory or I/O operations. A[23:17] have internal pull-up resistors; A[16:2] do not. A[3:2] are burst address bits for the '486.
D[15:0]	TS	**DATA [15:0]:** These bidirectional pins are used to transfer data during any memory or I/O operation. These pins are directly connected to D[15:0] of the '386SX or '386DX bus. These pins are connected via four bidirectional data transceivers to the 32 data pins of the '486 or VESA VL-Bus. The transceivers are controlled with OEH#, OEL#, and W/R#. These pads have pull-up resistors.
BE[3:0]#	I	**BYTE ENABLE [3:0]#:** These active-low inputs are directly connected to the '386DX/'486 or VESA VL-Bus byte enable outputs. In the case of the '386SX, BE0# is unused and can be left unconnected. BE1#, 2, and 3 are redefined as BHE#, BLE#, and A1, respectively. They must be directly connected to the corresponding '386 outputs.
ADS#	I	**ADDRESS STROBE:** This active-low input indicates that a new cycle has begun. It must be directly connected to the ADS# pin on the CPU. For VESA VL-Bus, this pin is connected to LADS#.
CPU-RESET	I	**CPU RESET:** When this active-high input is active, the CL-GD542X is forced into an initial condition. It is used to synchronize the CL-GD542X to CLK1X or CLK2X. This pin *must* be connected to the RESET pin of the CPU in a '386 system; it *must* be connected to ground in a '486 system; it *must* be connected to RDYRTN# in a VESA VL-Bus system.
W/R#	I	**WRITE/READ:** This input indicates whether a write or read operation is to occur. It must be directly connected to the W/R# pin on the CPU. If W/R# is high, a write will occur. If it is low, a read will occur.
CLK2X	I	**CLOCK 2X:** This is the timing reference for the CL-GD542X when connected to a '386SX or '386DX local bus. This is redefined as CLK1X for the '486 local bus. In either case, it must be directly connected to the corresponding CPU pin. For VESA VL-Bus, this pin is connected to LCLK.
M/IO#	I	**MEMORY/IO#:** This input indicates whether a memory or I/O operation is to occur. It must be directly connected to the M/IO# pin on the CPU. If M/IO# is high, a memory operation will occur. If it is low, an I/O operation will occur.
UADDR#	I	**UPPER ADDRESS:** This active-low input is a decode of the upper-CPU Address bits A[32:24]. This input is unused in the case of a '386SX local bus and can be left unconnected. Refer to appendixes in the *CL-GD542X Technical Reference Manual* for information on the generation of this signal.

2.3 Host Interface — Local Bus (CL-GD5424/'25/'26/'28/'29 only) *(cont.)*

Name	Type	Description
RESET	I	**RESET:** This active-high input initializes the CL-GD542X to a known state. The trailing (falling) edge of this input loads the Configuration register CF[14:0] with the data on MD[30:16], determined by internal pull-up resistors and (optional) external pull-down resistors.
RDY#	TS	**RDY #:** This active-low signal is used as an output to terminate a CL-GD542X bus cycle.
LBA#	TS	**LOCAL BUS ACKNOWLEDGE #:** This open-collector output is driven low to indicate that the CL-GD542X will respond to the current cycle. This signal is generated from a decode of A[23:2], UADDR#, and M/IO#. This output will be active before the middle of the first T2 after an active ADS#. For VESA VL-Bus, this pin is connected to LDEV#.
BS16#	OC	**BUS SIZE 16 #:** This active-low output is driven by the CL-GD542X to indicate that the current cycle addresses a 16-bit resource. The '386DX/'486 will convert the cycle to an appropriate number of 16-bit transfers. This pin is not used for a '386SX local bus and can be left unconnected. For VESA VL-Bus, this pin is connected to LBS16#.
INTR	TS	**INTERRUPT REQUEST:** This active-high output indicates the CL-GD542X has reached the end of an active field. Specifically, the transition occurs at the beginning of the bottom border. See register CR11 for a description of the controls for this pin.
OEH#	O	**OUTPUT ENABLE HIGH#:** This active-low output controls the output enables for the data transceivers that connect the CL-GD542X SD[15:0] pins to the '486 or VESA VL-Bus D[31:16] pins.
OEL#	O	**OUTPUT ENABLE LOW#:** This active-low output controls the output enables for the data transceivers that connect the CL-GD542X SD[15:0] pins to the '486 or VESA VL-Bus D[15:0] pins.

2.4 Dual-Frequency Synthesizer Interface

Name	Type	Description
OSC	I	**OSCILLATOR INPUT:** This TTL input pin supplies the reference frequency for the dual-frequency synthesizer. It requires an input frequency of 14.31818 ± 0.01% MHz with a duty cycle of 50 ± 10%. This input can be supplied from the appropriate pin on the ISA or MicroChannel bus, from an oscillator, or with a series-resonant crystal connected between this pin and the XTAL pin. **NOTE:** When the CL-GD5425 (only) is configured for PAL/NTSC operation (pull-down installed on MD16), this pin should be driven with 17.734475 MHz. This is the reference frequency for PAL.
XTAL	I/O	**CRYSTAL:** This output pin allows the use of a crystal to supply the reference frequency for the synthesizer. A series-resonant crystal can be connected between this pin and the OSC pin. If this pin is not used for a crystal connection, it *must* be left unconnected (except for CL-GD5425). When the CL-GD5425 (only) is configured for PAL/NTSC operation (pull-down installed on MD16), this is an input pin and should be driven with 14.31818 MHz. This is the reference frequency for NTSC, the MCLK synthesizer, and VGA modes.
MFILTER	O	**MEMORY CLOCK FILTER:** This pin must be connected to a π RC filter returned to AVSS4. The values of the two capacitors and the resistor are shown in Appendix B17 in the *CL-GD542X Technical Reference Manual*. The filter components, especially the input capacitor and the resistor, must be placed as closely as possible to this pin.
VFILTER	O	**VIDEO CLOCK FILTER:** This pin *must* be connected to a π RC filter returned to AVSS1 or AVDD1, depending on processing. The values of the two capacitors and the resistor are shown in Appendix B17 in the *CL-GD542X Technical Reference Manual*. The filter components, especially the input capacitor and the resistor, *must* be placed as closely as possible to this pin.
MCLK	I/O	**MEMORY CLOCK:** This pin is normally an output and can be used to monitor the internal MCLK. Typically, it would not be connected. If CF[4] is a zero, MCLK will be an input and the internal MCLK oscillator will be disabled. This configuration is intended for testing only. **NOTE:** For the CL-GD5425 and CL-GD5429 only, this pin can be configured to output the internal VCLK VCO. If a pull-down is installed on MD31 (and no pull-down on MD20), the internal VCLK VCO (prior to the post-scalar) will be driven onto this pin.

2.5 Video Interface

Name	Type	Description
VSYNC	I/O	**VERTICAL SYNC:** This output supplies the vertical synchronization pulse to the monitor. The polarity of this output is programmable. This pin is put into high impedance when ESYNC* is low. This pin can be directly connected to the corresponding pin on the feature connector. **NOTE:** When the CL-GD5425 (only) is configured for VSYNC GENLOCK, by programming CR1C[7] to '1', VSYNC becomes an input.
HSYNC	I/O	**HORIZONTAL SYNC:** This output supplies the horizontal synchronization pulse to the monitor. The polarity of this output is programmable. This pin is put into high impedance when ESYNC* is low. This pin can be directly connected to the corresponding pin on the feature connector. **NOTE:** When the CL-GD5425 (only) is configured for HSYNC GENLOCK, by programming CR1C[6] to '1', HSYNC becomes an input.
BLANK*	I/O	**BLANK*:** This is a bidirectional pin. If ESYNC* is high, BLANK* is an output. As an output, it supplies a blanking signal to the feature connector. If ESYNC* is low, BLANK* is an input. As an active-low input, it forces the RED, GREEN, and BLUE outputs to zero current. This pin can be directly connected to the corresponding pin on the feature connector. **NOTE:** When the CL-GD5425 (only) is configured for TV mode by programming CR30[3] to '1', this pin becomes the Color Carrier Reference (Nx fsc).
P[7:0]	I/O	**PIXEL BUS [7:0]:** These are bidirectional pins. If EVIDEO* is high, these pins are outputs and reflect the address into the palette DAC. If EVIDEO* is low, these pins are inputs and can be used to drive pixel values into the palette DAC. These pins can be directly connected to the corresponding pins on the feature connector.
DCLK	I/O	**VIDEO DOT CLOCK:** This is a bidirectional pin. If EDCLK* is high, this is an output and can be used to externally latch the data on the Pixel Bus. If EDCLK* is low, this is an input and can be used to clock data on the Pixel bus into the CL-GD542X. This pin can be directly connected to the corresponding pin on the feature connector.
ESYNC*	I/O	**ENABLE SYNC AND BLANK:** This input is used to control the buffers on HSYNC, VSYNC, and BLANK*. If ESYNC* is high, the controlled pins are outputs. If ESYNC* is low, BLANK* is an input. HSYNC and VSYNC are not driven by the CL-GD542X and must be driven externally to valid input levels. This pin can be directly connected to the corresponding pin on the feature connector. **NOTE:** For the CL-GD5425, the ESYNC* pin will be an output and will reflect SR8[2] whenever Overlay mode is selected by programming CR1A[3:2] to any value other than '0,0' or whenever TV mode is selected by programming CR30[3] to'1'.
EVIDEO*	I/O	**ENABLE VIDEO:** This input controls the buffers on P[7:0]. If EVIDEO* is high, P[7:0] are outputs; if EVIDEO* is low, P[7:0] are inputs. This pin can be directly connected to the corresponding pin on the feature connector. This pin is not limited to static operation; it can switch at the DCLK rate.

2.5 Video Interface *(cont.)*

Name	Type	Description
EDCLK*	I	**ENABLE DOT CLOCK:** This input is used to control the buffer on DCLK. If EDCLK* is high, DCLK is an output; if EDCLK* is low, DCLK is an input. This pin can be directly connected to the corresponding pin on the feature connector.
RED	O	**RED VIDEO:** This analog output supplies current corresponding to the red value of the pixel being displayed. Each of the three DACs consists of 255 summed current sources. For each pixel, either the 6-bit value from the LUT or a 5-, 6-, or 8-bit true-color value is applied to each DAC input to determine the number of current sources to be summed. Full-scale current on the RED, GREEN, and BLUE outputs is related to IREF as follows: $If = (63/30) \times IREF$ To maintain IBM VGA compatibility, each DAC output is typically terminated to monitor ground with a 75-Ω 2-percent resistor. This resistor, in parallel with the 75-Ω resistor in the monitor, will yield a 37.5-Ω impedance to ground. For a full-scale voltage of 700 mV, full-scale current output should be 18.7 mA.
GREEN	O	**GREEN VIDEO:** This analog output supplies current corresponding to the green value of the pixel being displayed. See the description of RED for information regarding the termination of this pin.
BLUE	O	**BLUE VIDEO:** This analog output supplies current corresponding to the blue value of the pixel being displayed. See the description of RED for information regarding the termination of this pin.
IREF	I	**DAC CURRENT REFERENCE:** The current drawn from AV_{DD} through this pin determines the full-scale output of each DAC. Connect this pin to a constant current source. A recommended circuit is provided in appendixes of the *CL-GD542X Technical Reference Manual.*

2.6 Display Memory Interface

Name	Type	Description
RAS*	O	**ROW ADDRESS STROBE *:** This active-low output is used to latch the row address from MA[9:0] into the DRAMs. This pin must be connected to the RAS* pins of all the DRAMs in the display memory array. These pads, and those for the other DRAM controls, are matched for one-to-four loads. If eight DRAMs are used, damping resistors may be required to control edge rates and undershoot on these, and other, control pins.
CAS*	O	**COLUMN ADDRESS STROBE *:** This active-low output is used to latch the Column Address from MA[9:0] into the DRAMs. This pin must be connected to the CAS* pins of all the DRAMs in the display memory array. **NOTE:** If CF[12] = 0 (dual-CAS* DRAMs), this pin becomes WE*.
OE*	O	**OUTPUT ENABLE *:** This active-low output is used to control the output enables of the DRAMs. For 256K x 4 DRAMs and 256K x 16 DRAMs with Dual-write Enables, this pin *must* be connected to the OE* pins of all the DRAMs in the display memory array. For 256K x 16 DRAMs with dual-CAS*, this pin is a no-connect. For the CL-GD5426/'28/'29 with 2 Mbytes of display memory, this pin becomes RAS1*. See the DRAM configuration tables in Section 4 on page 41. **NOTE:** For the CL-GD5425 (only), this pin is ODD/EVEN when the chip is configured for TV-out and indicates which field is being displayed.
WE[3:0]*	O	**WRITE ENABLE [3:0]*:** These active-low outputs are used to control the write enable inputs of the DRAMs. These pins *must* be connected to the WE* pins of the DRAMs as indicated in the DRAM configuration tables in Section 4 on page 41. **NOTE:** If CF[12] = 0 (dual-CAS* DRAMs) these pins become CAS[3:0]*. These pins can be connected to the CAS* pins of the DRAMs. WE[1:0]* are reserved on the CL-GD5420 (Revision 'A').
MA[9]	O	**MEMORY ADDRESS [9]:** This pin controls one address input of the DRAMs. See the DRAM configuration tables in Section 4 on page 41. When the CL-GD5425 (only) is configured for TV mode, by programming CR30[3] to '1', this pin becomes CSYNC out. CSYNC includes all horizontal and vertical timing and serration pulses.
MA[8:0]	O	**MEMORY ADDRESS [8:0]:** These pins control the address inputs of the DRAMs. These pins must be connected to the address pins of the DRAMs. See the DRAM configuration tables in Section 4 on page 41.
MD[31:0]	TS	**MEMORY DATA [31:0]:** These pins are used to transfer data between the CL-GD542X and the display memory. These pins must be connected to the data pins of the DRAMs. See the DRAM configuration tables in Section 4 on page 41. These pins are forced into high impedance when RESET is active. This allows the configuration pull-down resistors to override the weak pull-ups and be loaded into the Configuration register CF. MD[15:0] are reserved on the CL-GD5420 (Revision 'A').

2.7 Miscellaneous Pins

Name	Type	Description
EECS	O	**EEPROM CHIP SELECT:** This pin is used to control the Chip Select of the optional configuration EEPROM, and should be directly connected to that pin (ISA and Micro-channel only). **NOTE:** This pin is redefined as OEL* for the '486 or VESA VL-Bus (CL-GD5424/'26/'28/'29 only).
EEDI	I	**EEPROM DATA IN:** This pin is used to read the data from the optional configuration EEPROM, and should be directly connected to the Data Out pin (ISA and Micro-channel only). **NOTE:** This pin is redefined as OEH* for the '486 or VESA VL-Bus (CL-GD5424/'26/'28/'29 only).
EROM*	O	**ENABLE ROM BUFFERS*:** This active-low output is used to control the Output Enable pins of up to two 8-bit bus drivers. These buffers are used to connect the data pins of the BIOS EPROMs to the System Data bus. This output is forced high when RESET is active. This output goes active only for memory read cycles to the address range C000:0–C7FF:F. It is gated with MEMR* in ISA mode, and with –CMD in MicroChannel mode. It is un-latched address decode in Local Bus modes.
OVRW	O	**OVERLAY WINDOW:** This output signal is active-high. It is intended to be used in applications involving video overlays. For additional connectivity information, see Appendix B14 in the CL-GD542X technical reference manual. OVRW is reserved on the CL-GD5420.
TWR*	I	**TEST LATCH LOAD ENABLE*:** This pin is intended for factory testing and must be pulled-up for normal operation. It can be used in board-level testing to disable most of the CL-GD542X output pins. For additional information, see Appendix B14 in the *True Color VGA Family — CL-GD542X Technical Reference Manual.*

2.8 Power Pins

Name	Type	Description
VDD[7:1]	Power	**PLUS FIVE (LOGIC):** These seven pins are used to supply +5 volts to the core logic of the CL-GD542X. Each pin *must* be connected to the VCC rail as described in Appendixes B1–B3 of the *True Color VGA Family — CL-GD542X Technical Reference Manual*. Each pin *must* be bypassed with a 0.1-µF capacitor with proper high-frequency characteristics, placed as closely to the pin as possible. If a multi-layer board is used, each VDD pin *must* be connected to the power plane as outlined in Appendixes B1–B3 in the *True Color VGA Family — CL-GD542X Technical Reference Manual*.
VSS[13:1]	Ground	**GROUND (LOGIC):** These 13 pins are used to supply ground reference to the core logic of the CL-GD542X. Each pin *must* be directly connected to the GND rail. If a multi-layer board is used, each VSS pin *must* be connected to the ground plane.
AVDD[1]	Power	**PLUS FIVE (VCLK):** This pin is used to supply +5 volts to the video clock synthesizer of the CL-GD542X. This pin must be connected to the VCC rail via a 33-Ω resistor, and bypassed to AVSS4 with a 10-µF capacitor.
AVSS[1]	Ground	**GROUND (VCLK):** This pin is used to supply ground reference to the video clock synthesizer of the CL-GD542X. This pin *must* be connected to the GND rail.
AVDD[4]	Power	**PLUS FIVE (MCLK):** This pin is used to supply +5 volts to the memory clock synthesizer of the CL-GD542X. This pin must be connected to the VCC rail through a 33-Ω resistor and bypassed to AVSS4 with a 10-µF capacitor.
AVSS[4]	Ground	**GROUND (MCLK):** This pin is used to supply ground reference to the video clock synthesizer of the CL-GD542X. This pin *must* be connected to the GND rail.
AVDD[3:2]	Power	**PLUS FIVE (DAC):** These two pins are used to supply +5 volts to the palette DAC of the CL-GD542X. Each pin must be directly connected to the VCC rail. Each pin must be bypassed, as closely to the pin as possible, with a 0.1-µF capacitor with proper high-frequency characteristics. If a multi-layer board is used, each VDD pin *must* be connected to the power plane.
AVSS[3:2]	Ground	**GROUND (DAC):** These two pins are used to supply the ground reference to the palette DAC of the CL-GD542X. Each pin must be connected to the GND rail. For various adapter board and motherboard solutions, see the appendixes in the *True Color VGA Family — CL-GD542X Technical Reference Manual*.

3. FUNCTIONAL DESCRIPTION

3.1 General

The CL-GD542X family of VGA controllers offers a complete VGA-standards-compatible solution. All of the hardware necessary for CPU updates to memory, screen refresh, and DRAM refresh is included in the CL-GD542X. A complete VGA motherboard solution can be implemented with one 256K x 16 DRAM with any CL-GD542X chip.

The chip block diagram in Figure 3-1 shows the CL-GD542X connection to the host, display memory, and monitor. Each member of the CL-GD542X family of VGA controllers is pin-to-pin compatible on the system bus.

3.2 Functional Blocks

The following functional blocks have been integrated into the CL-GD542X.

CPU Interface

The CL-GD542X connects directly to the ISA bus, E-ISA bus, MicroChannel bus, or '386 and '486 bus (CL-GD5424/'26/'28/'29 only). No glue logic is required. The CL-GD542X internally decodes a 16- or 24-bit address, and responds to the applicable control lines. It executes both I/O accesses and memory accesses as either an 8- or 16-bit device.

CPU Write Buffer

The CPU Write Buffer contains a queue of CPU write accesses to display memory that have not been executed because of memory arbitration. Maintaining a queue allows the CL-GD542X to release the CPU as soon as it has recorded the address and data, and to execute the operation when display memory is available, increasing CPU performance.

Graphics Controller

The Graphics Controller is located between the CPU interface and the Memory Sequencer. It performs text manipulation, data rotation, color mapping, and other miscellaneous operations.

BitBLT

This is a unique GUI acceleration feature in the CL-GD5426/'28/'29. The BitBLT function moves data with ROPs (raster operations). This operation occurs in Packed-pixel modes with 8-, 16-, or 24-bit-per-pixel transfers. Color expansion can be used to translate monochrome images to 8- or 16-bit color. The source or destination of a BitBLT operation can be system memory.

Memory Arbitrator

The Memory Arbitrator allocates bandwidth to the four functions that compete for the limited bandwidth of display memory. These are CPU access, screen refresh, DRAM refresh, and BitBLT operations. DRAM refresh is handled invisibly by allocating a selectable number of CAS*-before-RAS* refresh cycles at the beginning of each scanline. Screen refresh and CPU/BitBLT accesses are allocated cycles according to the FIFO-control parameters, with priority given to screen refreshes.

Memory Sequencer

The Memory Sequencer generates timing for display memory. This includes RAS*, CAS* and multiplexed-address timing, as well as WE* and OE* timing. The Sequencer generates CAS*-before-RAS* refresh cycles, Random Read and Random Early Write cycles, and Fast-page mode Read and Early Write cycles. The Memory Sequencer generates multiple-CAS* or multiple-WE* signals according to the memory type used.

CRT Controller

The CRT controller generates the HSYNC and VSYNC Signals required for the monitor, as well as the BLANK* signals required by the palette DAC.

Video FIFO

The Video FIFO allows the Memory Sequencer to execute the display memory accesses needed for screen refresh at maximum memory speed rather than at the screen refresh rate. This makes it possible to collect the accesses for screen refresh near the beginning of the scanline, and to execute them in Fast-page mode rather than Random Read mode.

Attribute Controller

The Attribute Controller formats the display for the screen. Display color selection, text blinking and underlining are performed by the Attribute Controller. Alternate font selection also occurs in the Attribute Controller.

Palette DAC

The palette DAC block contains the color palette and three 8-bit digital-to-analog converters. The color palette, with 256 18-bit entries, converts a color code that specifies the color of a pixel into three 6-bit values, one each for red, green, and blue. The CL-GD5425 (only) supports YCrCb and AccuPak-to-RGB conversion.

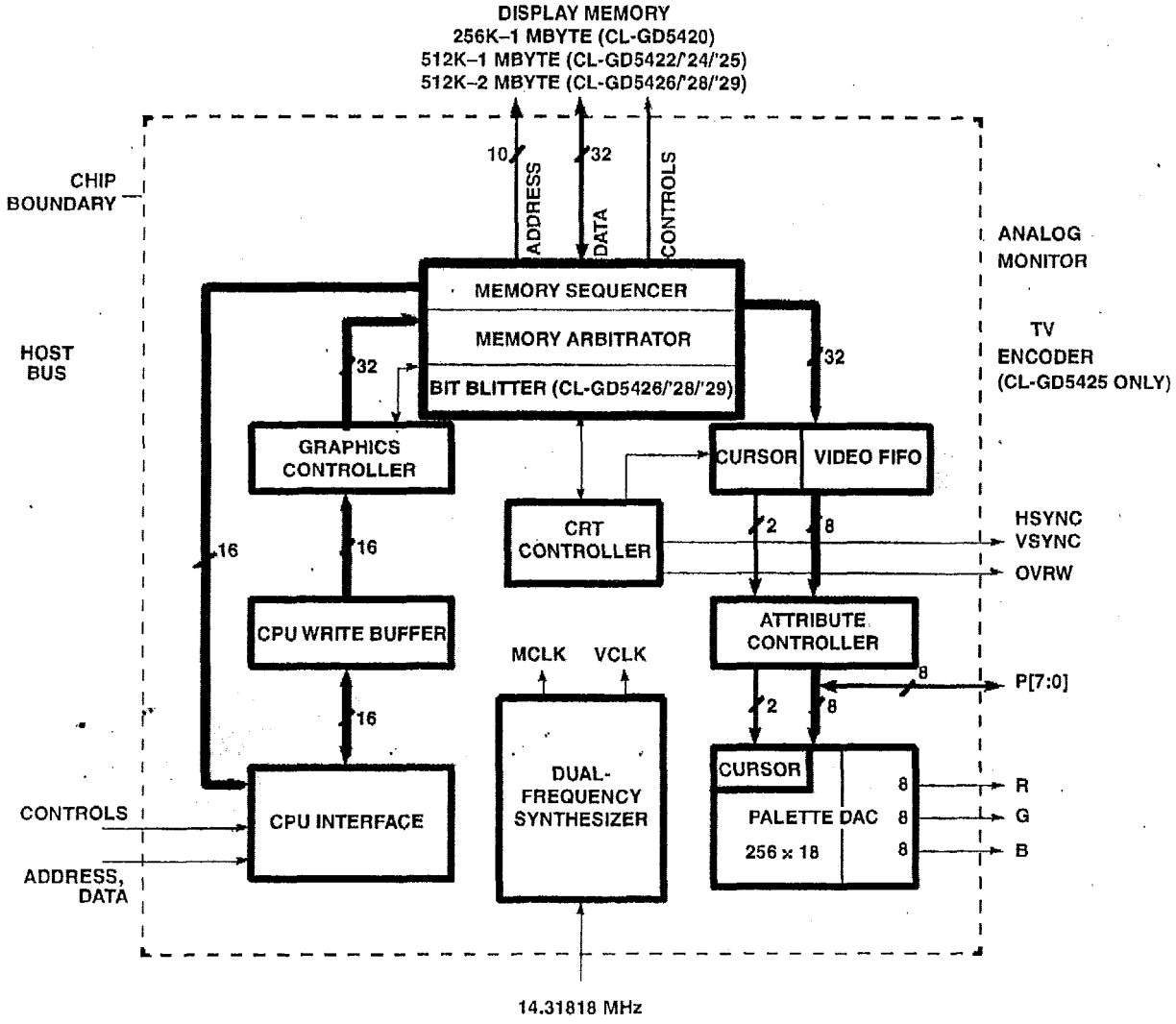


Figure 3-1. CL-GD542X Chip Block Diagram

Alternatively, the CL-GD542X (excluding the CL-GD5420) can be configured for 15-, 16-, or 24-bit pixels. This allows 32K, 65K, or 16 million simultaneous colors to be displayed on the screen. The bits are allocated as 8-8-8 for the 16 million colors, 5-6-5 for the 64K Color mode, or five to each (red, green, and blue) DAC for the 32K Color mode.

Dual-Frequency Synthesizer

The dual-frequency synthesizer generates the Memory Sequencer Clock and the Video Display Clock from a single reference frequency. The frequency of each clock is programmable. The reference frequency can be generated with an internal crystal-controlled oscillator. Alternatively, it can be supplied from an external TTL source.

VESA® Connector/VGA Pass-through Connector

The CL-GD542X is designed to connect directly to a VESA connector. It supports the three enable/disable inputs, and the Pixel bus can directly drive the connector. Through this connector, the overlay feature could be used in multimedia applications. This allows for internal DAC utilization in 16-bit-per-pixel mode. The CL-GD5425/'29 supports the VAFC (VESA Advanced Feature Connector) Baseline for Video Overlay.

TV Encoder (CL-GD5425 only)

The CL-GD5425 provides integrated scaling, flicker reduction, and a glueless encoder interface that delivers high-quality TV display at the lowest possible cost without the need for additional frame or line stores.

The programmable flicker-reduction function reduces interlaced artifacts inherent in computer-generated images displayed on interlaced TV monitors. The degree of filtering is selectable by the end-user.

3.3 Functional Operation

The four major operations handled by the CL-GD542X are discussed below.

CPU Access to Registers

The host can be any processor controlling an ISA, E-ISA, MicroChannel, or '386 and '486 local bus. It accesses CL-GD542X registers by setting up 16- or 24-bit addresses and making controls such as IORD* or IOWR* active. The CL-GD542X can respond either as an 8- or 16-bit peripheral, depending on how the chip has been designed into the system.

DRAM and screen refresh occur concurrently with, and independently of, register access (unless the host is changing display parameters or has suppressed refresh). Registers are described in detail in the *True Color VGA Family — CL-GD542X Technical Reference Manual.*

CPU Access to Display Memory

All host accesses to display memory are handled by the CL-GD542X. The host first sets up certain parameters, such as color and write masks, then generates a memory access in the range where the CL-GD542X is programmed to respond.

Display Memory Refresh

The CL-GD542X automatically generates a selectable number of CAS*-before-RAS* refresh cycles during each horizontal timing period.

Screen Refresh

The CRT monitor requires a near-constant rewriting since its only memory is the phosphor persistence. This persistence is typically only a few milliseconds. The CL-GD542X fetches information from the display memory for each scanline as quickly as possible, using Fast-page mode cycles to fill the Video FIFO. This allows the maximum possible time for the host to access the display memory.

3.4 Performance

The CL-GD542X is designed with the following performance-enhancing features:

- Accelerated Microsoft Windows with BitBLT (CL-GD5426/'28/'29 only)
- 16-bit CPU interface to I/O registers for faster host access
- 16-bit CPU interface to display memory for faster host access in all modes, including Planar mode
- 32-bit display memory data bus for faster access to display memory (CL-GD5422/'24/'25/'26/'28/'29)
- DRAM Fast-page mode operations for faster access to display memory
- Zero-wait-state performance and a CPU write buffer allows faster CPU access for writes to display memory
- Video FIFO to minimize memory contention
- 32 x 32 and 64 x 64 hardware cursor to improve Microsoft Windows performance
- Increased throughput with '386 and '486 local bus interface (CL-GD5424/'25/'26/'28/'29)

3.5 Compatibility

The CL-GD542X includes all registers and data paths required for VGA controllers.

The CL-GD542X supports extensions to VGA, including 1024 x 768 x 256 interlaced and non-interlaced, and 1280 x 1024 x 256 interlaced modes. Additionally, various 132-column text modes are supported.

3.6 Board Testability

The CL-GD542X chip is testable, even when installed on a PC board. By using pin scan testing, any IC signal pins not connected to the board or shorted to a neighboring pin or trace, will be detected. The Signature Generator allows the entire system, including the display memory, to be tested at speed. For further information on pin scan testing and the Signal Generator, refer to Appendixes B11 and B13 in the *True Color VGA Family — CL-GD542X Technical Reference Manual.*

4. CL-GD542X CONFIGURATION TABLES

4.1 Video Modes

Table 4-1. Standard VGA Modes

Mode No.	VESA® Mode No.	No. of Colors	Char. × Row	Char. Cell	Screen Format	Display Mode	Horiz. Freq. kHz	Vert. Freq. Hz
00/01	–	16/256	40 × 25	8 × 8	320 × 200	Text	31.5	70
00*/01*	–	16/256	40 × 25	8 × 14	320 × 350	Text	31.5	70
00+/01+	–	16/256	40 × 25	9 × 16	360 × 400	Text	31.5	70
02/03	–	16/256	80 × 25	8 × 8	640 × 200	Text	31.5	70
02*/03*	–	16/256	80 × 25	8 × 14	640 × 350	Text	31.5	70
02+/03+	–	16/256	80 × 25	9 × 16	720 × 400	Text	31.5	70
04/05	–	4/256	40 × 25	8 × 8	320 × 200	Graphics	31.5	70
6	–	2/256	80 × 25	8 × 8	640 × 200	Graphics	31.5	70
07*	–	Monochrome	80 × 25	9 × 14	720 × 350	Text	31.5	70
07+	–	Monochrome	80 × 25	9 × 16	720 × 400	Text	31.5	70
0D	–	16/256	40 × 25	8 × 8	320 × 200	Graphics	31.5	70
0E	–	16/256	80 × 25	8 × 8	640 × 200	Graphics	31.5	70
0F	–	Monochrome	80 × 25	8 × 14	640 × 350	Graphics	31.5	70
10	–	16/256	80 × 25	8 × 14	640 × 350	Graphics	31.5	70
11	–	2/256	80 × 30	8 × 16	640 × 480	Graphics	31.5	60
11+	–	2/256	80 × 30	8 × 16	640 × 480	Graphics	37.9	72
12	–	16/256	80 × 30	8 × 16	640 × 480	Graphics	31.5	60
12+	–	16/256	80 × 30	8 × 16	640 × 480	Graphics	37.9	72
13	–	256/256	40 × 25	8 × 8	320 × 200	Graphics	31.5	70

Table 4-2. Cirrus Logic Extended Video Modes

Mode No.	VESA® No.	No. of Colors	Char. x Row	Char. Cell	Screen Format	Display Mode	Pixel Freq. MHz	Horiz. Freq. kHz	Vert. Freq. Hz
14	–	16/256K	132 x 25	8 x 16	1056 x 400	Text	41.5	31.5	70
54	10A	16/256K	132 x 43	8 x 8	1056 x 350	Text	41.5	31.5	70
55	109	16/256K	132 x 25	8 x 14	1056 x 350	Text	41.5	31.5	70
58, 6A	102	16/256K	100 x 37	8 x 16	800 x 600	Graphics	36	35.2	56
58, 6A	102	16/256K	100 x 37	8 x 16	800 x 600	Graphics	40	37.8	60
58, 6A	102	16/256K	100 x 37	8 x 16	800 x 600	Graphics	50	48.1	72
58, 6A	102	16/256K	100 x 37	8 x 16	800 x 600	Graphics	49.5	46.9	75
5C	103	256/256K	100 x 37	8 x 16	800 x 600	Graphics	36	35.2	56
5C	103	256/256K	100 x 37	8 x 16	800 x 600	Graphics	40	37.9	60
5C	103	256/256K	100 x 37	8 x 16	800 x 600	Graphics	50	48.1	72
5C	103	256/256K	100 x 37	8 x 16	800 x 600	Graphics	49.5	46.9	75
5D†	104	16/256K	128 x 48	8 x 16	1024 x 768	Graphics	44.9	35.5	87†
5D	104	16/256K	128 x 48	8 x 16	1024 x 768	Graphics	65	48.3	60
5D	104	16/256K	128 x 48	8 x 16	1024 x 768	Graphics	75	56	70
5D	104	16/256K	128 x 48	8 x 16	1024 x 768	Graphics	77	58	72
5D	104	16/256K	128 x 48	8 x 16	1024 x 768	Graphics	78.7	60	75
5E	100	256/256K	80 x 25	8 x 16	640 x 400	Graphics	25	31.5	70
5F	101	256/256K	80 x 30	8 x 16	640 x 480	Graphics	25	31.5	60
5F	101	256/256K	80 x 30	8 x 16	640 x 480	Graphics	31.5	37.9	72
60†	105	256/256K	128 x 48	8 x 16	1024 x 768	Graphics	44.9	35.5	87†
60	105	256/256K	128 x 48	8 x 16	1024 x 768	Graphics	65	48.3	60
60	105	256/256K	128 x 48	8 x 16	1024 x 768	Graphics	75	56	70
60	105	256/256K	128 x 48	8 x 16	1024 x 768	Graphics	77	58	72
60	105	256/256K	128 x 48	8 x 16	1024 x 768	Graphics	78.7	60	75
64	111	64K	–	–	640 x 480	Graphics	25	31.5	60
64	111	64K	–	–	640 x 480	Graphics	31.5	37.9	72
65	114	64K	–	–	800 x 600	Graphics	36	35.2	56
65	114	64K	–	–	800 x 600	Graphics	40	37.8	60
65	114	64K	–	–	800 x 600	Graphics	50	48.1	72

Table 4-2. Cirrus Logic Extended Video Modes *(cont.)*

Mode No.	VESA® No.	No. of Colors	Char. x Row	Char. Cell	Screen Format	Display Mode	Pixel Freq. MHz	Horiz. Freq. kHz	Vert. Freq. Hz
66	110	32K‡	–	–	640 x 480	Graphics	25	31.5	60
66	110	32K‡	–	–	640 x 480	Graphics	31.5	37.9	72
67	113	32K‡	–	–	800 x 600	Graphics	36	35.2	56
67	113	32K‡	–	–	800 x 600	Graphics	40	37.8	60
67	113	32K‡	–	–	800 x 600	Graphics	50	48.1	72
68†	116	32K‡	–	–	1024 x 768	Graphics	44.9	35.5	87†
6C†	106	16/256K	160 x 64	8 x 16	1280 x 1024	Graphics	75	48	87†
6D†	107	256/256K	160 x 64	8 x 16	1280 x 1024	Graphics	75	48	87†
71	112	16M	–	–	640 x 480	Graphics	25	31.5	60
74†	117	64K	–	–	1024 x 768	Graphics	44.9	35.5	87†

NOTES:

1) Some modes are not supported by all CL-GD542X controllers. Refer to the CL-GD542X data book and software release kit for the list of video modes supported by the CL-GD542X BIOS.
2) Not all monitors support all modes. The fastest vertical refresh rate for the monitor type selected will be used automatically.
3) ‡ indicates 32K Direct-Color/256-color Mixed mode.
4) '†' indicates Interlaced mode.

Table 4-3. NTSC TV Video Modes (CL-GD5425 only)

Mode No.	Colors	Screen Format	Char. x Row	Char. Cell	Display Mode	Note
00/01	16/256K	320 x 200	40 x 25	8 x 8	Text	–
02/03	16/256K	640 x 200	80 x 25	8 x 8	Text	–
04/05	4/256K	320 x 200	40 x 25	8 x 8	Graphics	Double scanned
06	2/256K	640 x 200	80 x 25	8 x 8	Graphics	Double scanned
0D	16/256K	320 x 200	40 x 25	8 x 8	Graphics	Double scanned
0E	16/256K	640 x 200	80 x 25	8 x 8	Graphics	Double scanned
10	16/256K	640 x 350	80 x 25	8 x 14	Graphics	7:8 expansion
11/12	16/256K	640 x 480	80 x 30	8 x 16	Graphics	6:5 scale
13	256/256K	320 x 200	40 x 25	8 x 8	Graphics	Double scanned
5E	256/256K	640 x 400	80 x 25	8 x 16	Graphics	–
5F	256/256K	640 x 480	80 x 30	8 x 16	Graphics	6:5 scale
64	64K	640 x 480	–	–	Graphics	6:5 scale
7A	64K	640 x 400	–	–	Graphics	–

Table 4-4. PAL TV Video Modes (CL-GD5425 only)

Mode No.	Colors	Screen Format	Char. x Row	Char. Cell	Display Mode	Note
00/01	16/256K	320 x 200	40 x 25	8 x 8	Text	5:6 expansion
02/03	16/256K	640 x 200	80 x 25	8 x 8	Text	5:6 expansion
04/05	4/256K	320 x 200	40 x 25	8 x 8	Graphics	Double Scanned 5:6 expansion
06	2/256K	640 x 200	80 x 25	8 x 8	Graphics	Double Scanned 5:6 expansion
0D	16/256K	320 x 200	40 x 25	8 x 8	Graphics	Double scanned 5:6 expansion
0E	16/256K	640 x 200	80 x 25	8 x 8	Graphics	Double scanned 5:6 expansion
10	16/256K	640 x 350	80 x 25	8 x 14	Graphics	5:7 expansion 490 scanlines
11/12	16/256K	640 x 480	80 x 30	8 x 16	Graphics	–
13	256/256K	320 x 200	40 x 25	8 x 8	Graphics	5:6 expansion
5E	256/256K	640 x 400	80 x 25	8 x 16	Graphics	5:6 Expansion
5F	256/256K	640 x 480	80 x 30	8 x 16	Graphics	–
64	64K	640 x 480	–	–	Graphics	–
7A	64K	640 x 400	–	–	Graphics	5:6 expansion

4.2 Configuration Register, CF1

When RESET (system power-on reset) goes active, the CL-GD542X samples the levels on several of the Display Memory Data MD[x] pins. These levels are latched into a write-only configuration register (CF1). The data bits in this register are not accessible to the host CPU. The levels on the Memory Data bus are, by default, a logic '1' during power-on reset due to internal 250-kΩ pull-up resistors. A logic '0' is achieved by installing an external 6.8-kΩ pull-down resistor on the memory data line corresponding to the appropriate bit in the Configuration register. The following table identifies the Configuration register bits and the particular VGA function enabled by the latched level on the Memory Data bus during power-on reset.

Table 4-5. Configuration Register Bits

CF Bits	Level	Description	Memory Data Bit	Pin Number
15	0 1	Source VCLK on MCLK pin (CL-GD5425/'29) Source MCLK on MCLK pin (CL-GD5425/'29)	MD31	97
14; 7, 5	000 001 010 011 100 101 110 111	Reserved '386DX local bus (CL-GD5424/'25/'26/'28/'29) '386SX local bus (CL-GD5424/'25/'26/'28/'29) '486SX/DX local bus (CL-GD5424/'25/'26/'28) VESA® VL-Bus™ > 33 MHz (CL-GD5425/'29) MicroChannel® bus VESA® VL-Bus™ (CL-GD5424/'25/'26/'28/'29) ISA bus	MD30, 23, 21	98, 108, 110
13	0 1	Asymmetric DRAM (Not CL-GD5425) Symmetric DRAM IREF Adjust (CL-GD5425)	MD29	99
12	0 1	CAS[3:0]*, single-WE* (Reserved in CL-GD5420/'25) WE[3:0]*, single-CAS*	MD28	100
11	0 1	7-MCLK RAS* cycle 6-MCLK RAS* cycle	MD27	102
10,9	00 01 10 11	50.11363-MHz MCLK (Reserved in CL-GD5425) 44.74431-MHz MCLK 41.16477-MHz MCLK 37.58523-MHz MCLK	MD26, 25	103, 104
8	0 1	64K ROM BIOS @ C0000–CFFFF 32K ROM BIOS @ C0000–C7FFF	MD24	105
6	0 1	16-bit BIOS ROM (ISA bus only) (MCS16* generated for 64K or 32K) 8-bit BIOS ROM (CL-GD5424 does not generate MCS16*)	MD22	109
4	0 1	External MCLK (pin 157 is an input) (Test) Internal MCLK (pin 157 is an output)	MD20	111
3	0 1	Port 3C3h is Video System Sleep register Port 46E8h is Video System Sleep register	MD19	112
2		Reserved	MD18	113
1	0 1 0 1	Zero wait enabled (except CL-GD5425/'29) Zero wait disabled (except CL-GD5425/'29) Disable NTSC Black-Level offset (CL-GD5425) Enable NTSC Black-Level offset (CL-GD5425)	MD17	114
0	0 1	XTAL, OSC configured for two ref (CL-GD5425) XTAL, OSC configured for one ref (CL-GD5425)	MD16	115

4.3 Host Interface Signals

With the pin connections listed below, the CL-GD542X will interface directly to an ISA, MicroChannel, or local bus.

Table 4-6. Bus Connections

CL-GD542X Pin	ISA Bus	MicroChannel® Bus	'386SX	'386DX	'486	VESA® VL-Bus™
[45..42], [39..29]	SA[16:2]	A[16:2]	A[16:2]	A[16:2]	A[16:2]	A[16:2]
28	SA1	A1	A1	BE3#	BE3#	BE3#
27	SA0	A0	BLE#	BE2#	BE2#	BE2#
[21..15]	LA[23:17]	A[23:17]	A[23:17]	A[23:17]	A[23:17]	A[23:17]
[11..8], [6..3]	SD[15:8]	D[15:8]	D[15:8]	D[15:8]	D[15:8]†	D[15:8]†
[63..62], [60..59], [57..54]	SD[7:0]	D[7:0]	D[7:0]	D[7:0]	D[7:0]†	D[7:0]†
24	SBHE*	–SBHE	BHE#	BE1#	BE1#	BE1#
25	BALE	MADE24	ADS#	ADS#	ADS#	ADS#
46	AEN	–CD_SETUP	CPU-Reset	CPU-Reset	Ground	RDYRTN#
49	IOR*	–S1	W/R#	W/R#	W/R#	W/R#
50	IOW*	–CMD	CLK2X	CLK2X	CLK1X	CLK1X
14	MEMR*	M/-IO	M/IO#	M/IO#	M/IO#	M/IO#
13	MEMW*	–S0	(unused)	UADDR#	UADDR#	UADDR#
41	RESET	CHRESET	RESET	RESET	RESET	RESET
48	REFRESH	–REFRESH	(unused)	BE0#	BE0#	BE0#
23	MCS16*	–CD_DS16	(unused)	BS16#	BS16#	BS16#
51	0WS*	(unused)	GROUND	GROUND	GROUND	GROUND
52	IRQ	–IRQ	INTR	INTR	INTR	INTR
47	IOCHRDY	CD_CHRDY	READY#	READY#	RDY#	RDY#
22	IOCS16*	–CD_SFDBK	LBA#	LBA#	LBA#	LBA#
159	OSC	OSC	OSC	OSC	OSC	OSC
2	EROM*	EROM*	EROM*	EROM*	EROM*	EROM*

NOTES:

1) For ISA-bus applications, note that SA[19..17] are not found on the CL-GD542X; this means that an adapter board will only function in a 16-bit slot.
2) The OSC and EROM* pins are common in all configurations.
3) The OSC pin is an input for 14.31818 MHz.
4) † Data lines D[15:0] connect to external, data-steering transceiver.

5. VGA REGISTER PORT MAP

Table 5-1. VGA Register Port Map

Address	Port
94	POS 102 Access Control (3C3 sleep)
102	POS102 register
3B4	CRT Controller Index (R/W — monochrome)
3B5	CRT Controller Data (R/W — monochrome)
3BA	Feature Control (W), Input Status register 1 (R — monochrome)
3C0	Attribute Controller Index/Data (Write)
3C1	Attribute Controller Index/Data (Read)
3C2	Miscellaneous Output (W), Input Status register 0 (R)
3C3	MotherBoard Sleep
3C4	Sequencer Index (R/W)
3C5	Sequencer Data (R/W)
3C6	Video DAC Pixel Mask (R/W), Hidden DAC register (R/W)
3C7	Pixel Address Read mode (W), DAC State (R)
3C8	Pixel Mask Write mode (R/W)
3C9	Pixel Data (R/W)
3CA	Feature Control Readback (R)
3CC	Miscellaneous Output Readback (R)
3CE	Graphics Controller Index (R/W)
3CF	Graphics Controller Data (R/W)
3D4	CRT Controller Index (R/W — color)
3D5	CRT Controller Data (R/W — color)
3DA	Feature Control (W), Input Status register 1 (R — color)
46E8	Adapter Sleep

6. CL-GD542X REGISTERS

Table 6-1. External/General Registers

Abbreviation	Register Name	Index	Port
102 Access	POS 94: 102 Access Control	–	94
POS102	POS102	–	102
VSSM	Motherboard Sleep Address *(CL-GD5424/'25/'26/'28/'29 only)*	–	3C3
VSSM	Adapter Sleep	–	46E8
MISC	Miscellaneous Output	–	3C2 (Write)
MISC	Miscellaneous Output	–	3CC (Read)
FC	Feature Control	–	3?A (Write)[a]
FC	Feature Control	–	3CA (Read)
FEAT	Input Status Register 0	–	3C2
STAT	Input Status Register 1	–	3?A
3C6	Pixel Mask	–	3C6
3C7	Pixel Address Read Mode	–	3C7 (Write)
3C7	DAC State	–	3C7 (Read)
3C8	Pixel Address Write Mode	–	3C8
3C9	Pixel Data	–	3C9

[a] '?' in the above register addresses is 'B' in Monochrome mode and 'D' in Color mode.

Table 6-2. VGA Sequencer Registers

Abbreviation	Register Name	Index	Port
SRX	Sequencer Index	–	3C4
SR0	Reset	0	3C5
SR1	Clocking Mode	1	3C5
SR2	Plane Mask	2	3C5
SR3	Character Map Select	3	3C5
SR4	Memory Mode	4	3C5

Table 6-3. CRT Controller Registers

Abbreviation	Register Name	Index	Port
CRX	CRTC Index	–	3?4[a]
CR0	Horizontal Total	0	3?5
CR1	Horizontal Display End	1	3?5
CR2	Horizontal Blanking Start	2	3?5
CR3	Horizontal Blanking End	3	3?5
CR4	Horizontal Sync Start	4	3?5
CR5	Horizontal Sync End	5	3?5
CR6	Vertical Total	6	3?5
CR7	Overflow	7	3?5
CR8	Screen A Preset Row Scan	8	3?5
CR9	Character Cell Height	9	3?5
CRA	Text Cursor Start	A	3?5
CRB	Text Cursor End	B	3?5
CRC	Screen Start Address High	C	3?5
CRD	Screen Start Address Low	D	3?5
CRE	Text Cursor Location High	E	3?5
CRF	Text Cursor Location Low	F	3?5
CR10	Vertical Sync Start	10	3?5
CR11	Vertical Sync End	11	3?5
CR12	Vertical Display End	12	3?5
CR13	Offset	13	3?5
CR14	Underline Row Scanline	14	3?5
CR15	Vertical Blanking Start	15	3?5
CR16	Vertical Blanking End	16	3?5
CR17	Mode Control	17	3?5
CR18	Line Compare	18	3?5
CR22	Graphics Data Latches Readback	22	3?5
CR24	Attribute Controller Toggle Readback	24	3?5
CR26	Attribute Controller Index Readback	26	3?5

[a] '?' in the above register addresses is 'B' in Monochrome mode and 'D' in Color mode.

Table 6-4. VGA Graphics Controller Registers

Abbreviation	Register Name	Index	Port
GRX	Graphics Controller Index	–	3CE
GR0	Set/Reset	0	3CF
GR1	Set/Reset Enable	1	3CF
GR2	Color Compare	2	3CF
GR3	Data Rotate	3	3CF
GR4	Read Map Select	4	3CF
GR5	Mode	5	3CF
GR6	Miscellaneous	6	3CF
GR7	Color Don't Care	7	3CF
GR8	Bit Mask	8	3CF

Table 6-5. VGA Attribute Controller Registers

Abbreviation	Register Name	Index	Port
ARX	Attribute Controller Index	–	3C0/3C1
AR0-ARF	Attribute Controller Palette	0:F	3C0/3C1
AR10	Attribute Controller Mode	10	3C0/3C1
AR11	Overscan (Border) Color	11	3C0/3C1
AR12	Color Plane Enable	12	3C0/3C1
AR13	Pixel Panning	13	3C0/3C1
AR14	Color Select	14	3C0/3C1

Table 6-6. Extension Registers

Abbreviation	Register Name	Index	Port
SR2	Enable Writing Pixel Extension	2	3C5
SR6	Unlock ALL Extensions	6	3C5
SR7	Extended Sequencer Mode	7	3C5
SR8	EEPROM Control	8	3C5
SR9	Scratch-Pad 0	9	3C5
SRA	Scratch-Pad 1	A	3C5
SRB	VCLK0 Numerator	B	3C5
SRC	VCLK1 Numerator	C	3C5
SRD	VCLK2 Numerator	D	3C5
SRE	VCLK3 Numerator	E	3C5
SRF	DRAM Control	F	3C5
SR10	Graphics Cursor Y Position	10	3C5
SR11	Graphics Cursor X Position	11	3C5
SR12	Graphics Cursor Attributes	12	3C5
SR13	Graphics Cursor Pattern Address Offset	13	3C5
SR14	Scratch-Pad 2 *(CL-GD5425/'26/'28/'29 only)*	14	3C5
SR15	Scratch-Pad 3 *(CL-GD5425/'26/'28/'29 only)*	15	3C5
SR16	Performance Tuning *(CL-GD5424/'25/'26/'28/'29 only)*	16	3C5
SR17	Configuration Readback and Extended Control *(except CL-GD5420)*	17	3C5
SR18	Signature Generator Control *(except CL-GD5420)*	18	3C5
SR19	Signature Generator Result Low Byte *(except CL-GD5420)*	19	3C5
SR1A	Signature Generator Result High Byte *(except CL-GD5420)*	1A	3C5
SR1B	VCLK0 Denominator and Post-Scalar Value	1B	3C5
SR1C	VCLK1 Denominator and Post-Scalar Value	1C	3C5
SR1D	VCLK2 Denominator and Post-Scalar Value	1D	3C5
SR1E	VCLK3 Denominator and Post-Scalar Value	1E	3C5
SR1F	BIOS ROM Write Enable and MCLK Select	1F	3C5
GR0	Write Mode 5 Background Extension	0	3CF
GR1	Write Mode 4, 5 Foreground Extension	1	3CF
GR9	Offset Register 0	9	3CF
GRA	Offset Register 1	A	3CF

Table 6-6. Extension Registers *(cont.)*

Abbreviation	Register Name	Index	Port
GRB	Graphics Controller Mode Extensions	B	3CF
GRC	Color Key *(CL-GD5424/'25/'26/'28/'29 only)*	C	3CF
GRD	Color Key Mask *(CL-GD5424/'25/'26/'28/'29 only)*	D	3CF
GRE	Miscellaneous Control *(CL-GD5425/'28/'29 only)*	E	3CF
GR10	16-bit Pixel BG Color High Byte *(except CL-GD5420)*	10	3CF
GR11	16-bit Pixel FG Color High Byte *(except CL-GD5420)*	11	3CF
GR18	Extended DRAM Controls *(CL-GD5429 only)*	18	3CF
CR19	Interlace End	19	3?5[a]
CR1A	Miscellaneous Control	1A	3?5
CR1B	Extended Display Controls	1B	3?5
CR1C	Sync Adjust and GENLOCK *(CL-GD5425 only)*	1C	3?5
CR1D	Overlay Mode Controls *(CL-GD5425/'29 only)*	1D	3?5
CR25	Part Status	25	3?5
CR27	ID	27	3?5
CR30	TV-Out Mode Control *(CL-GD5425 only)*	30	3?5
HDR	Hidden DAC *(except CL-GD5420)*	–	3C6

[a] '?' in the above register addresses is 'B' in Monochrome mode and 'D' in Color mode.

Table 6-7. CL-GD5426/'28/'29 BitBLT Registers

Abbreviation	Register Name	Index	Port
GR20	BLT Width Low	20	3CF
GR21	BLT Width High	21	3CF
GR22	BLT Height Low	22	3CF
GR23	BLT Height High	23	3CF
GR24	BLT Destination Pitch Low	24	3CF
GR25	BLT Destination Pitch High	25	3CF
GR26	BLT Source Pitch Low	26	3CF
GR27	BLT Source Pitch High	27	3CF
GR28	BLT Destination Start Low	28	3CF
GR29	BLT Destination Start Mid	29	3CF
GR2A	BLT Destination Start High	2A	3CF

Table 6-7. CL-GD5426/'28/'29 BitBLT Registers *(cont.)*

Abbreviation	Register Name	Index	Port
GR2C	BLT Source Start Low	2C	3CF
GR2D	BLT Source Start Mid	2D	3CF
GR2E	BLT Source Start High	2E	3CF
GR2F	BLT Write Mask Destination *(CL-GD5429 only)*	2F	3CF
GR30	BLT Mode	30	3CF
GR31	BLT Start/Status	31	3CF
GR32	BLT Raster Operation	32	3CF
GR34	Transparent Color Select Low *(except CL-GD5429)*	34	3CF
GR35	Transparent Color Select High *(except CL-GD5429)*	35	3CF
GR38	Transparent Color Mask Low *(except CL-GD5429)*	38	3CF
GR39	Transparent Color Mask High *(except CL-GD5429)*	39	3CF

7. ELECTRICAL SPECIFICATIONS

7.1 Absolute Maximum Ratings

Ambient temperature under bias ..... 0° to 70° C

Storage temperature ..... -65° to 150° C

Voltage on any pin ..... V_{SS} -0.5 V to V_{CC} + 0.5 V

Operating power dissipation ..... 1.5 Watts

Power supply voltage ..... 7 Volts

Injection current (latch-up testing) ..... 100 mA

NOTE: Stresses above those listed may cause permanent damage to system components. These are stress ratings only. Functional operation at these or any conditions above those indicated in the operational ratings of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect system reliability.

7.2 DC Specifications (Digital)

(V_{CC} = 5V ± 5%, T_A = 0° to 70° C, unless otherwise specified)

Symbol	Parameter	MIN	MAX	Units	Test Conditions	Note
V_{CC}	Power Supply Voltage	4.75	5.25	Volts	Normal Operation	
V_{IL}	Input Low Voltage	0	0.8	Volts		
V_{IH}	Input High Voltage	2.0	V_{CC} + 0.5	Volts		
V_{OL}	Output Low Voltage		0.5	Volts	I_{OL} = 4 mA	1
V_{OH}	Output High Voltage	2.4		Volts	I_{OH} = 400 µA	2
I_{CC}	Supply Current				V_{CC} Nominal	3
I_{IH}	Input High Current		10	µA	$V_{IL} = V_{DD}$	
I_{IL}	Input Low Current	–10		µA	V_{DD} = 5.25, V_{IL} = 0	
I_{IHP}	Input High Current (pull-up)	–10	10	µA	$V_{IL} = V_{DD}$	
I_{ILP}	Input Low Current (pull-up)	–45	–12	µA	V_{DD} = 5.25, V_{IL} = 0	
I_{OZ}	Input Leakage	–10	10	µA	$0 < V_{IN} < V_{CC}$	
C_{IN}	Input Capacitance		10	pF		4
C_{OUT}	Output Capacitance		10	pF		4

NOTES:

1) I_{OL} is specified for a standard buffer. See the pin summary for further information.
2) I_{OH} is specified for a standard buffer. See the pin summary for further information.
3) I_{CC} is measured with VCLK and MCLK as indicated in the table below:

Part Number	VCLK	MCLK	I_{CC}
CL-GD5420	75 MHz	50 MHz	250 mA
CL-GD5422	80 MHz	50 MHz	260 mA
CL-GD5424/'25	80 MHz	50 MHz	260 mA
CL-GD5426/'28	80 MHz	50 MHz	260 mA
CL-GD5429	86 MHz	70 MHz	310 mA

4) This is not 100% tested, but is periodically sampled.

7.3 DC Specifications (Palette DAC)

(V_{CC} = 5V ± 5%, T_A = 0° to 70° C, unless otherwise specified)

Symbol	Parameter	MIN	MAX	Units	Test Conditions	Note
A_{VDD}	DAC Supply Voltage	4.75	5.25	Volts	Normal Operation	
I_{REF}	DAC Reference Current	-3	-10	mA		1

NOTE: See the Detailed Pin Description for information regarding nominal I_{REF}.

7.4 DC Specifications (Frequency Synthesizer)

(V_{CC} = 5 V ± 5%, T_A = 0° to 70° C, unless otherwise specified)

Symbol	Parameter	MIN	MAX	Units	Test Conditions	Note
A_{VDD}	Synthesizer Supply Voltage	4.75	5.25	Volts		

7.5 DAC Characteristics

(V_{CC} = 5 V ± 5%, T_A = 0° to 70° C, unless otherwise specified)

Symbol	Parameter	MIN	MAX	Units	Test Conditions	Note
R	Resolution	8	Max	Bits		
IO	Output Current	30	Max	mA	VO < 1V	
TR	Analog Output Rise/Fall Time	8	Max	ns		2, 3, 4
TS	Analog Output Settling Time	15	Max	ns		2, 3, 5
TSK	Analog Output Skew	tbd	Max	ns		2, 3, 6
DT	DAC-to-DAC Correlation	2.5	Max	%		6, 7
GI	Glitch Impulse		Typical	pV-sec.		2, 3, 6
IL	Integral Linearity	1.5	Max	LSB		
DL	Differential Lineaity	1.5	Max	LSB		

NOTES:

1) TD is measured from the 50% point of VCLK to the 50% point of full-scale transition.
2) Load is 50 Ω and 30 pF per analog output.
3) I_{REF} = -6.67 mA.
4) TR is measured from 10% to 90% full-scale.
5) TS is measured from 50% of full-scale transition to the output remaining within 2% of final value.
6) Outputs loaded identically.
7) About the mid-point of the distribution of the three DACs measured at full-scale output.

7.6 List of Waveforms

Table 7-1. I/O Write Timing (ISA Bus) [a]

Symbol	Parameter	MIN	MAX	Units
t_1	Address, SBHE* setup to IOW* active	5	–	ns
t_2	IOW* pulse width	40	–	ns
t_3	Data setup to IOW* inactive	5	–	ns
t_4	Data hold from IOW* inactive	10	–	ns
t_5	Address, SBHE* hold from IOW* inactive	5	–	ns
t_6	IOW* inactive to any command active	80	–	ns

[a] AEN must be inactive.

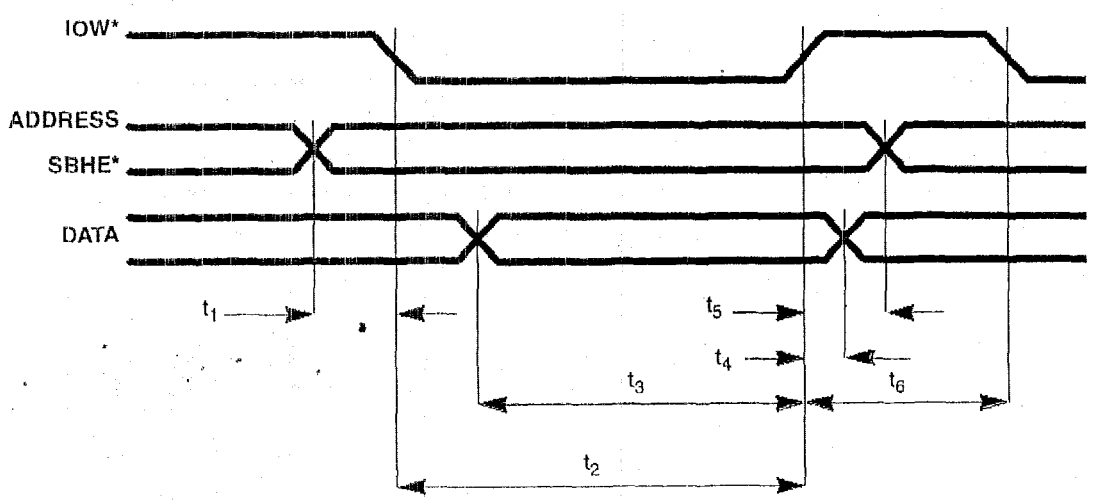


Figure 7-1. I/O Write Timing (ISA Bus)

Table 7-2. I/O Read Timing (ISA Bus)[a]

Symbol	Parameter	MIN	MAX	Units
t_1	Address, SBHE* setup to IOR* active	5	–	ns
t_2	IOR* active to low-impedance delay	0	–	ns
t_3	Data delay from IOR* active (IOR* access time)	–	60	ns
t_4	IOR* pulse width	70	–	ns
t_5	Data hold from IOR* inactive	0	20	ns
t_6	Address, SBHE* hold from IOR* inactive	0	–	ns
t_7	IOR* inactive to high-impedance delay	0	20	ns

[a] AEN must be inactive.

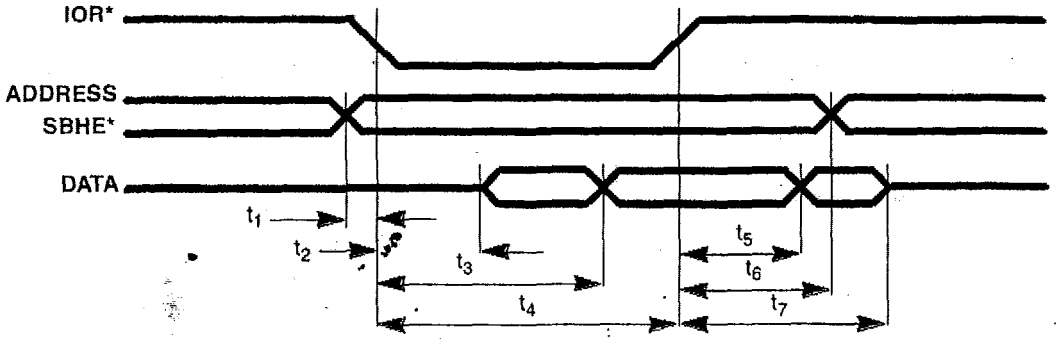


Figure 7-2. I/O Read Timing (ISA Bus)

Table 7-3. Memory Write Timing (ISA Bus)

Symbol	Parameter	MIN	MAX	Units
t_1	Address, SBHE* to SMEMW* active setup	5	–	ns
t_2	SMEMW* pulse width	3	–	m[a]
t_3	Data valid from SMEMW* active	–	3	m
t_4	Data hold from SMEMW* inactive	10	–	ns
t_5	Address, SBHE* hold from SMEMW* inactive	0	–	ns
t_6	SMEMW* inactive to next SMEMW*	3	–	m

[a] m = MCLK period.

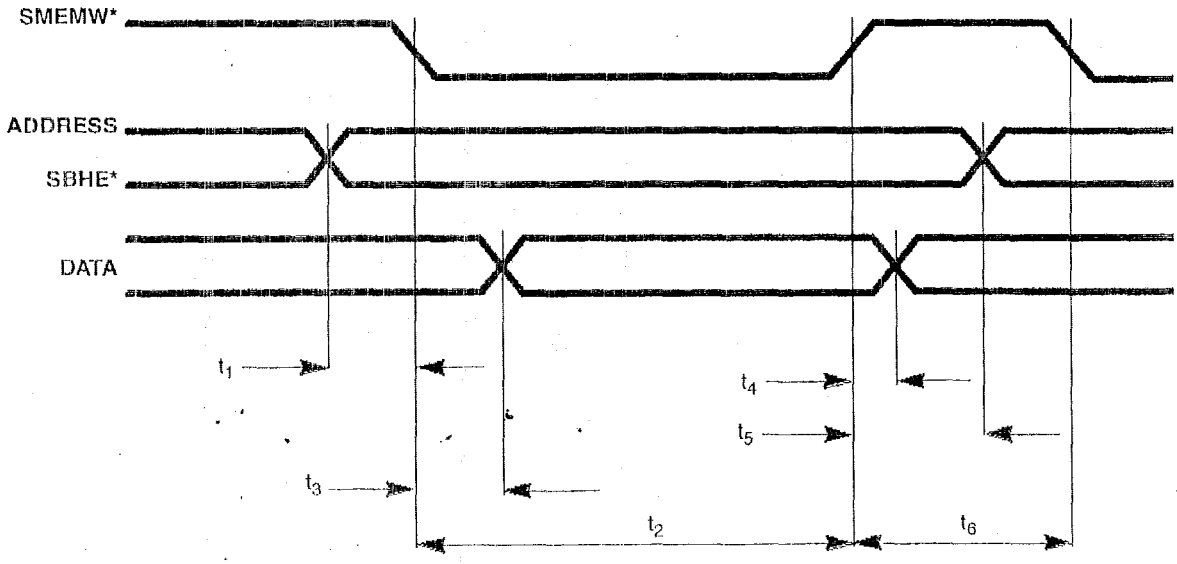


Figure 7-3. Memory Write Timing (ISA Bus)

Table 7-4. Memory Read Timing (ISA Bus)

Symbol	Parameter	MIN	MAX	Units
t_1	Address, SBHE* to SMEMR* active	5	–	ns
t_2	SMEMR* active to low-impedance delay	0	–	ns
t_3	Data delay from IOCHRDY active	–	15	ns
t_4	SMEMR* pulse width	–	[a]	ns
t_5	Data hold from SMEMR* inactive	0	20	ns
t_6	Address, SBHE* hold from SMEMR* inactive	0	–	ns
t_7	SMEMR* inactive to high-impedance delay	–	20	ns

[a] SMEMR* active-pulse width is determined by IOCHRDY.

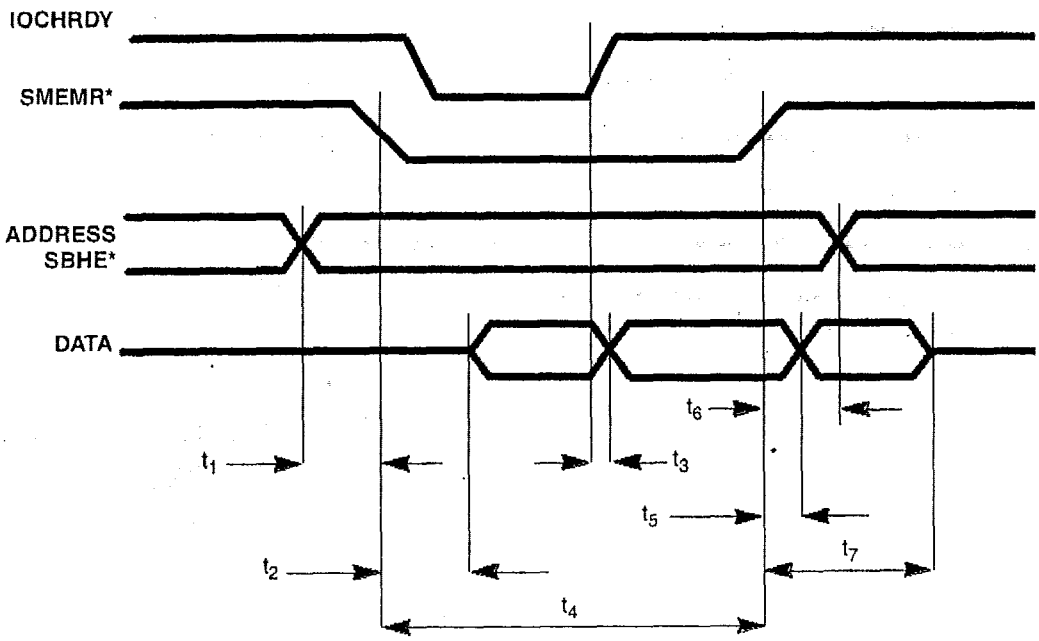


Figure 7-4. Memory Read Timing (ISA Bus)

Table 7-5. MCS16* Timing (ISA Bus)

Symbol	Parameter	MIN	MAX	Units
t_{1a}	MCS16* active delay from LA[23:17] valid	–	20	ns
t_{1b}	MCS16* active delay from SA[16:15] valid	–	14	ns
t_2	MCS16* inactive delay from address invalid	–	25	ns

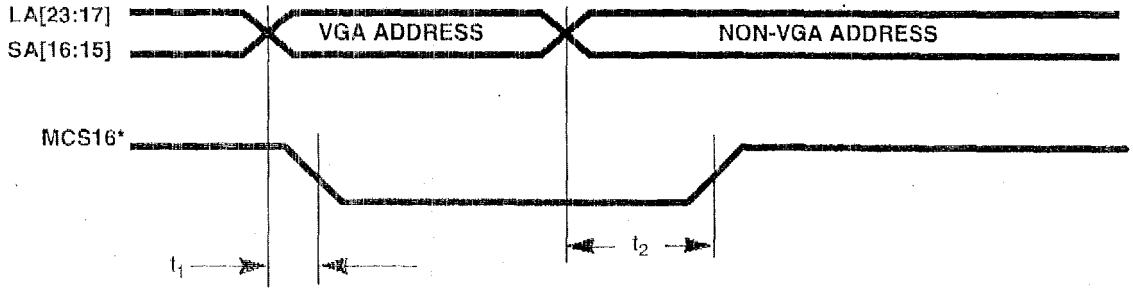


Figure 7-5. MCS16* Timing (ISA Bus)

Table 7-6. IOCS16* Timing (ISA Bus)

Symbol	Parameter	MIN	MAX	Units
t_1	IOCS16* active delay from address	–	25	ns
t_2	IOCS16* inactive delay from address	–	30	ns

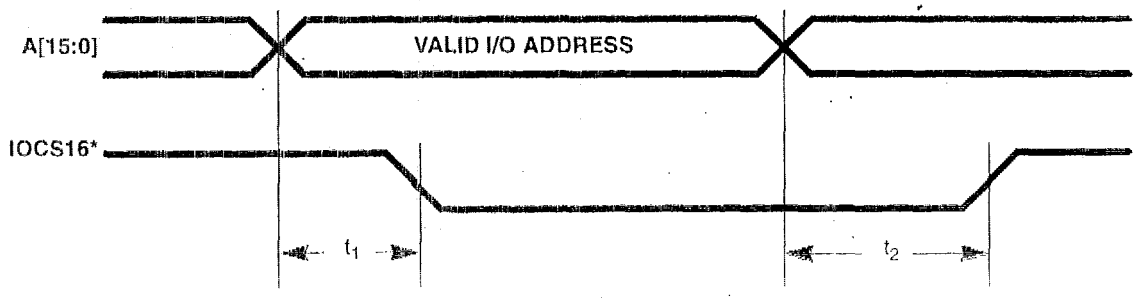


Figure 7-6. IOCS16* Timing (ISA Bus)

Table 7-7. BALE Timing (ISA Bus)

Symbol	Parameter	MIN	MAX	Units
t_1	LA[23:17] setup to BALE negative transition	20	–	ns
t_2	SBHE* setup to BALE negative transition	20	–	ns
t_3	LA[23:17] hold from BALE negative transition	20	–	ns
t_4	SBHE* hold from BALE negative transition	20	–	ns
t_5	BALE pulse width	20	–	ns

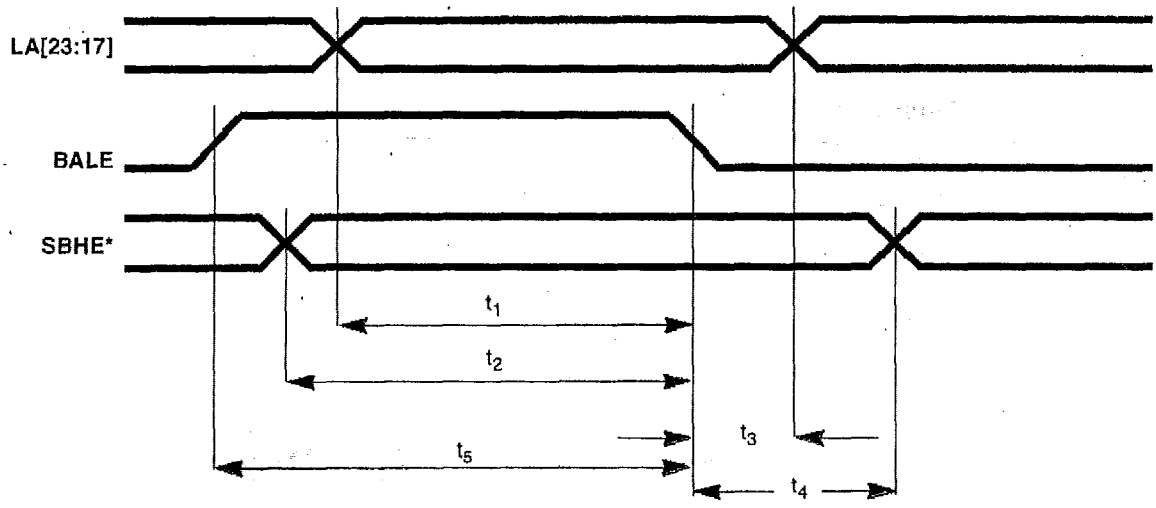


Figure 7-7. BALE Timing (ISA Bus)

Table 7-8. IOCHRDY for Memory Access Timing (ISA Bus)

Symbol	Parameter	MIN	MAX	Units
t_1	SMEMR* or SMEMW* active to IOCHRDY inactive low	–	28	ns
t_2	IOCHRDY inactive low pulse width	10	[a]	ns

[a] Video mode dependent.

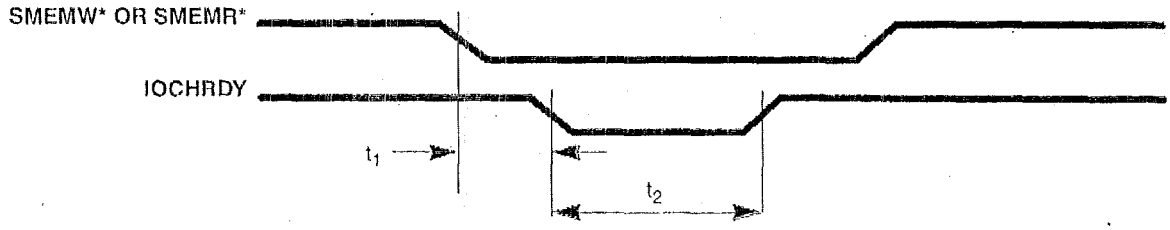


Figure 7-8. IOCHRDY for Memory Access Timing (ISA Bus)

Table 7-9. 0WS* Timing (ISA Bus)

Symbol	Parameter	MIN	MAX	Units
t_{1a}	0WS* active delay from SMEMR* (BIOS ACCESS)	–	22	ns
t_{1b}	0WS* active delay from SMEMW* (display memory write)	–	18	ns
t_{2a}	0WS* high-impedance delay from SMEMR*	–	18	ns
t_{2b}	0WS* high-impedance delay from SMEMW*	–	19	ns

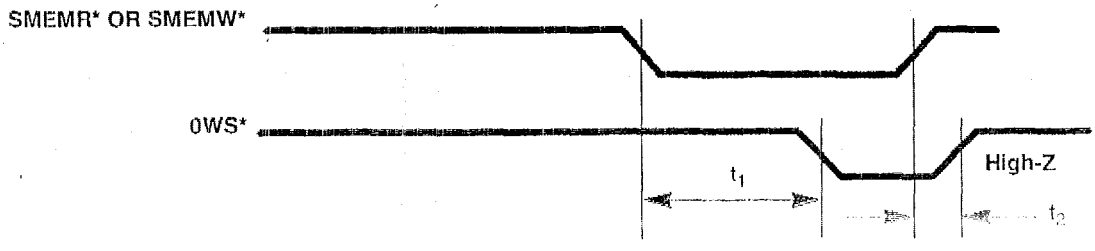


Figure 7-9. 0WS* Timing (ISA Bus)

Table 7-10. Refresh Timing (ISA Bus)

Symbol	Parameter	MIN	MAX	Units
t_1	REFRESH* active setup to SMEMR* active	20	–	ns
t_2	REFRESH* active hold from SMEMR* active	0	–	ns

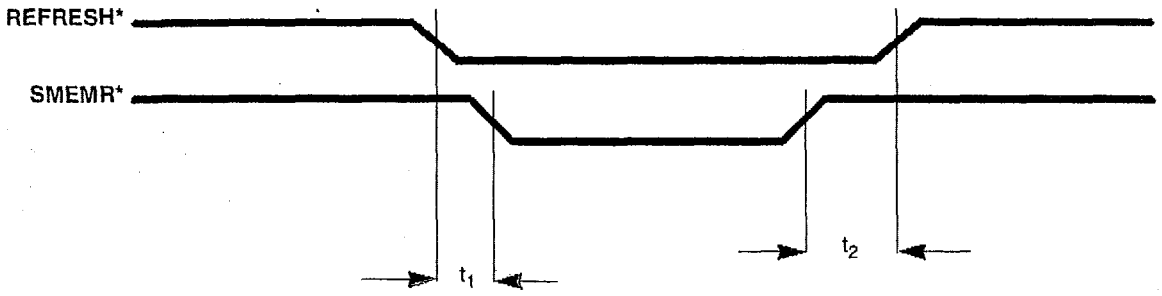


Figure 7-10. Refresh Timing (ISA Bus)

Table 7-11. EROM* Timing (ISA Bus)

Symbol	Parameter	MIN	MAX	Units
t_1	EROM* active delay from SMEMR* active	–	30	ns
t_2	EROM* inactive delay from SMEMR* inactive	–	20	ns

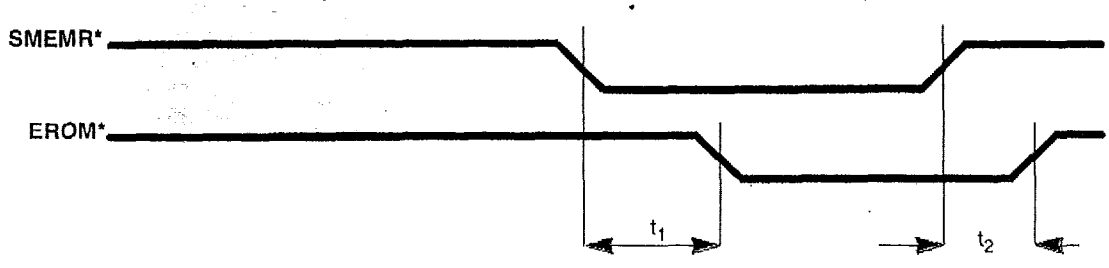


Figure 7-11. EROM* Timing (ISA Bus)

Table 7-12. AEN Timing (ISA Bus)[a]

Symbol	Parameter	MIN	MAX	Units
t_1	AEN setup to IOR* or IOW* active	5	–	ns
t_2	AEN hold from IOR* or IOW* inactive	5	–	ns

[a] AEN high, as shown below, will cause the CL-GD542X to **ignore** the I/O cycle.

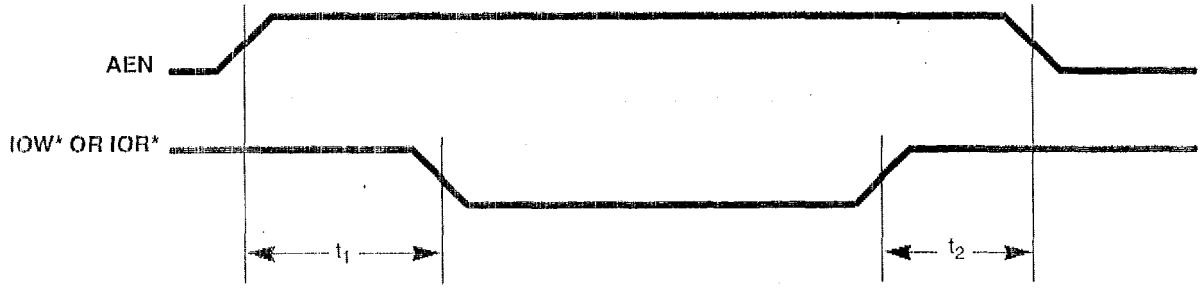


Figure 7-12. AEN Timing (ISA Bus)

Table 7-13. Write Timing (MicroChannel® Bus)

Symbol	Parameter	MIN	MAX	Units
t_1	Data to –CMD active setup	–15	–	ns
t_{2a}	–CMD active pulse width (default cycle)	90	–	ns
t_{2b}	–CMD active pulse width (synchronous extended cycle)	190	–	ns
t_{2c}	–CMD active pulse width (asynchronous extended cycle)	a	a	ns
t_3	Data hold from –CMD inactive	0	–	ns
t_4	CD_CHRDY active to –CMD inactive delay	0	–	ns

[a] The maximum t_{2c} depends on display memory activity.

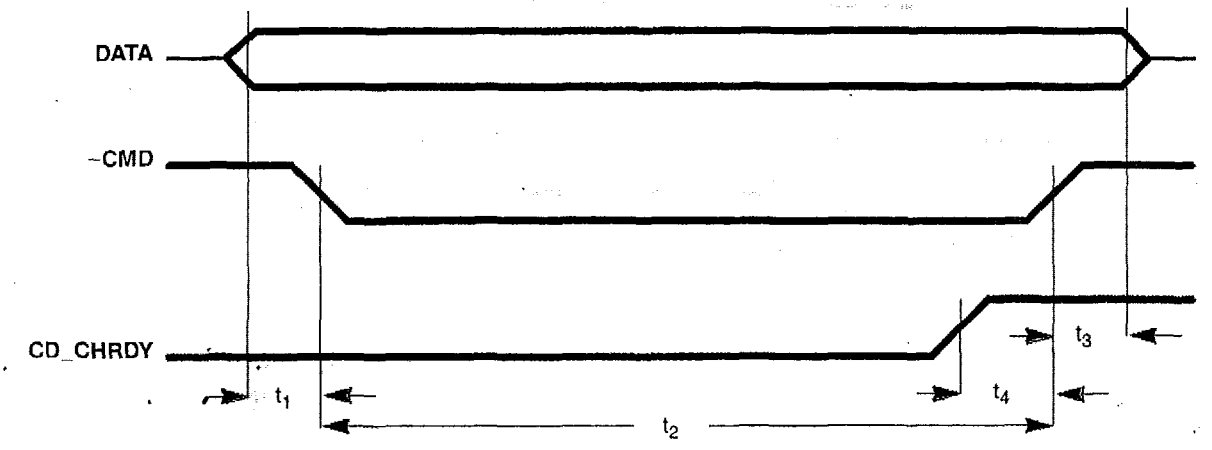


Figure 7-13. Write Timing (MicroChannel® Bus)

Table 7-14. Read Timing (MicroChannel® Bus)

Symbol	Parameter	MIN	MAX	Units
t_1	–CMD active to DATA valid delay (default cycle) (I/O read)	–	45	ns
t_2	CD_CHRDY active to DATA valid delay (memory read cycle)	–	45	ns
t_3	READ DATA hold from –CMD inactive	0	–	ns
t_4	READ DATA high-impedance from –CMD inactive	–	30	ns

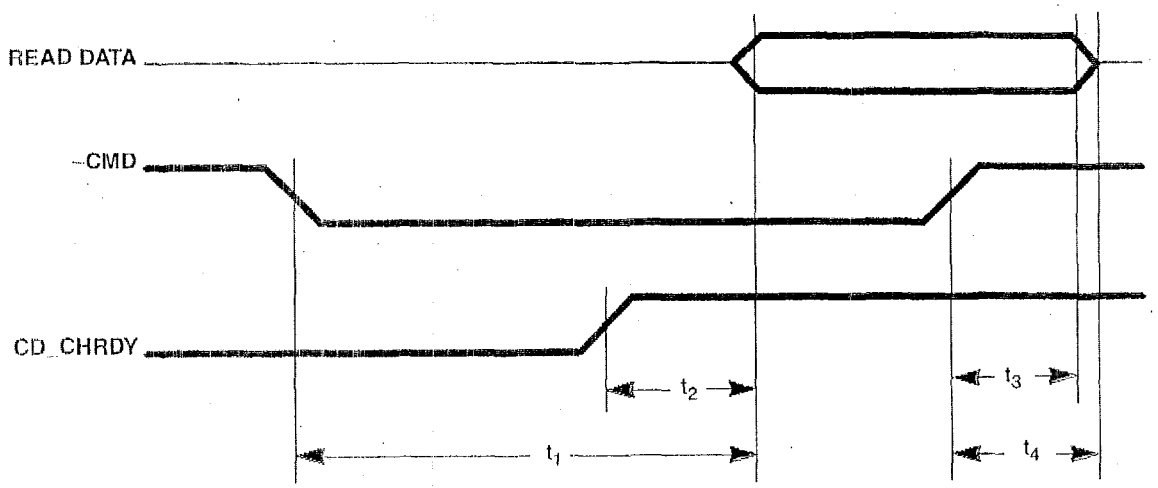


Figure 7-14. Read Timing (MicroChannel® Bus)

Table 7-15. –CD_DS16 Timing (MicroChannel® Bus)

Symbol	Parameter	MIN	MAX	Units
t_1	–CD_DS16 active from address, M/-IO, MADE24 valid	0	50	ns

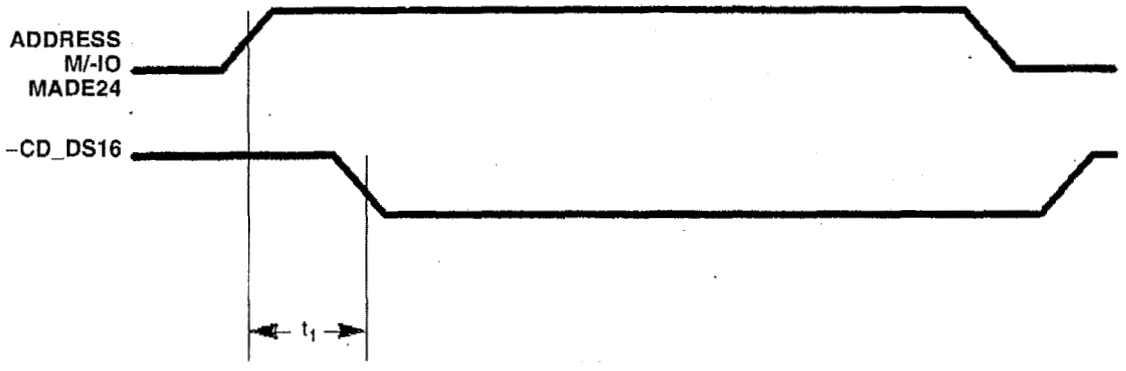


Figure 7-15. –CD_DS16 Timing (MicroChannel® Bus)

Table 7-16. –CMD Timing (MicroChannel® Bus)

Symbol	Parameter	MIN	MAX	Units
t_1	Address valid setup to –CMD active	80	–	ns
t_2	Status active setup to –CMD active	50	–	ns
t_3	–SBHE valid setup to –CMD active	35	–	ns
t_4	–CMD pulse width	90	–	ns
t_5	Address, –SBHE hold from –CMD active	25	–	ns
t_6	Status hold from –CMD active	25	–	ns

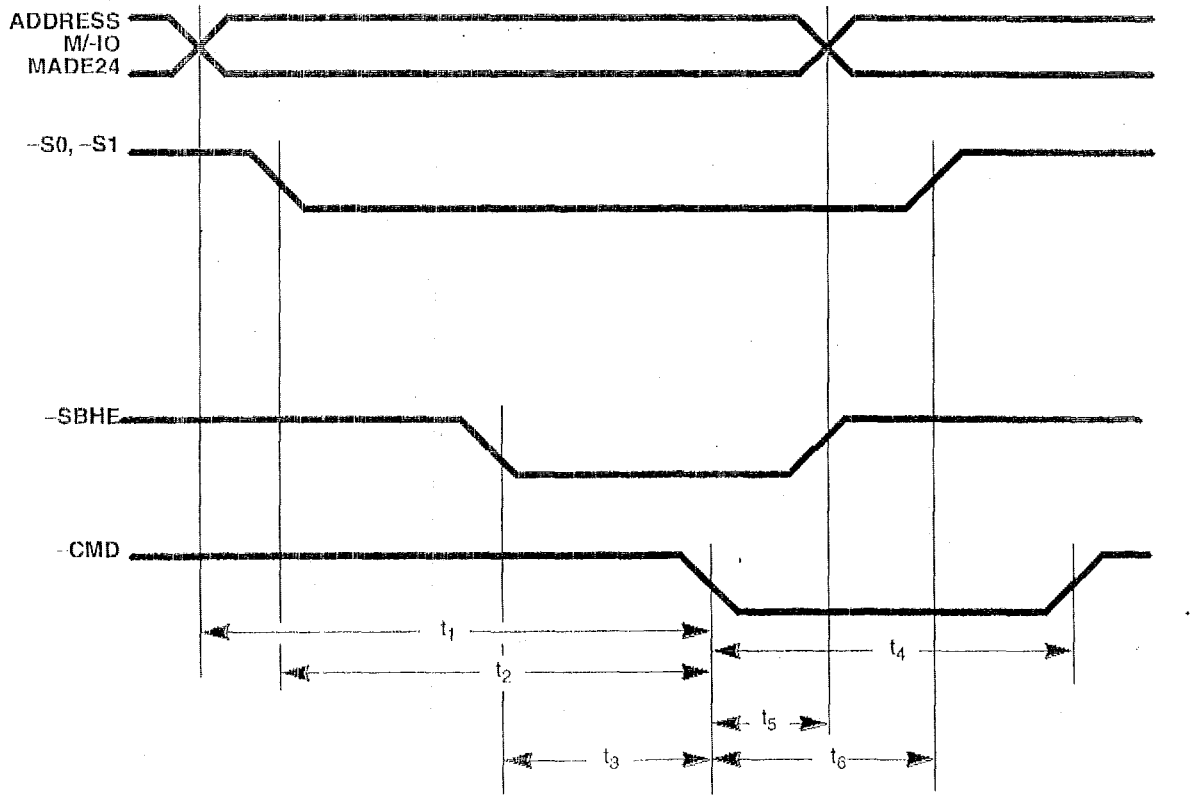


Figure 7-16. –CMD Timing (MicroChannel® Bus)

Table 7-17. CD_CHRDY Timing (MicroChannel® Bus)

Symbol	Parameter	MIN	MAX	Units
t_1	CD_CHRDY inactive (low) from Address valid	–	50	ns
t_2	CD_CHRDY inactive (low) from Status active	–	25	ns
t_3	CD_CHRDY inactive (low) pulse width	–	3.5	ms

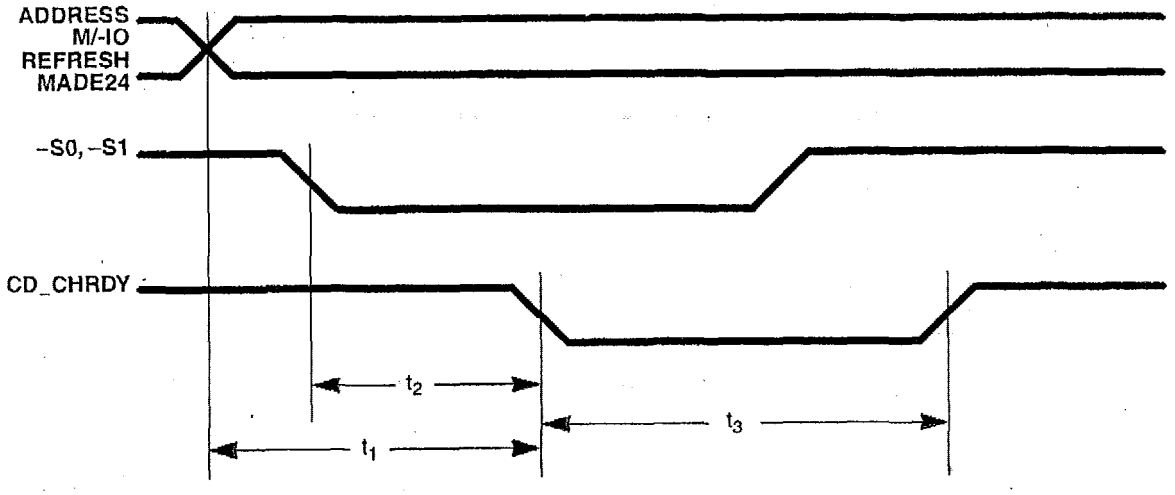


Figure 7-17. CD_CHRDY Timing (MicroChannel® Bus)

Table 7-18. –REFRESH Timing (MicroChannel® Bus)

Symbol	Parameter	MIN	MAX	Units
t_1	–REFRESH setup to –Status active	5	–	ns
t_2	–REFRESH setup to –CMD active	40	–	ns
t_3	–REFRESH hold from –CMD active	25	–	ns

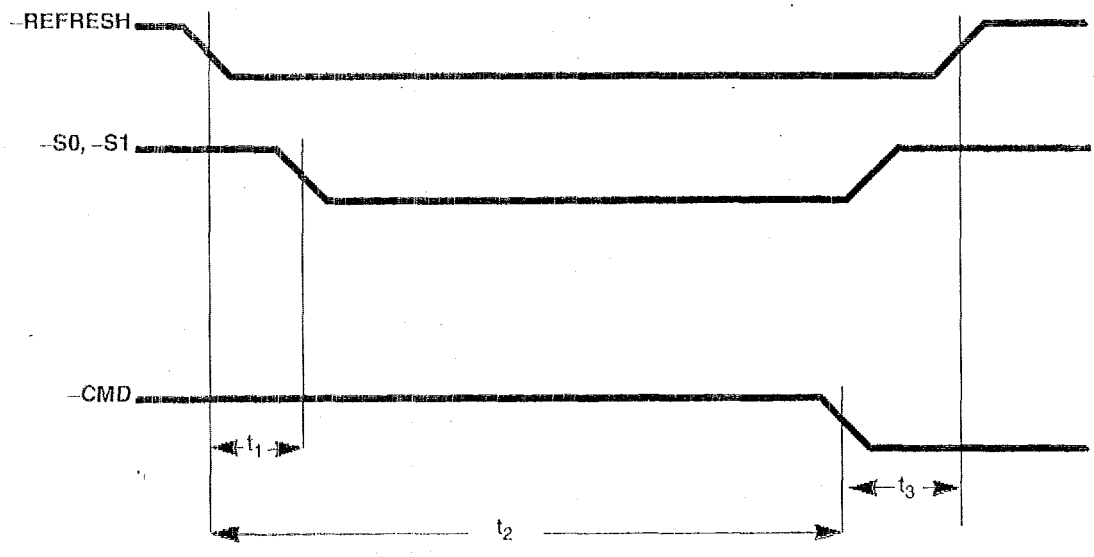


Figure 7-18. –Refresh Timing (MicroChannel® Bus)

Table 7-19. –EROM Timing (MicroChannel® Bus)

Symbol	Parameter	MIN	MAX	Units
t_1	–CMD active to –EROM active	–	30	ns
t_2	–CMD inactive to –EROM inactive	0	30	ns

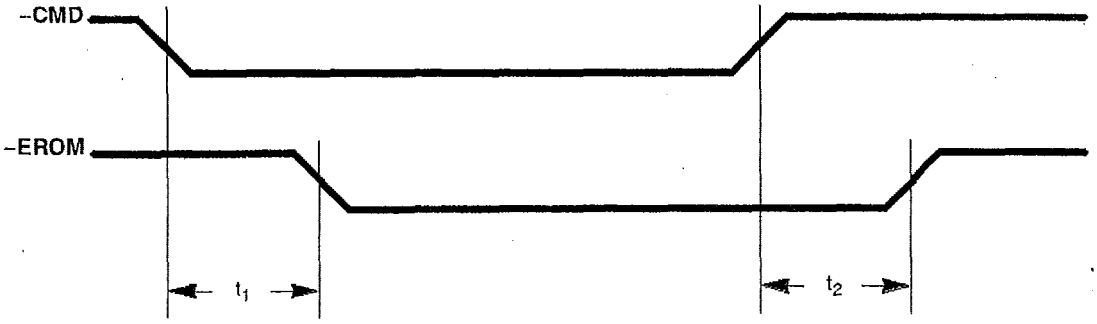


Figure 7-19. –EROM Timing (MicroChannel® Bus)

Table 7-20. –CD_SFDBK Timing (MicroChannel® Bus)

Symbol	Parameter	MIN	MAX	Units
t_1	Address, M/-IO, MADE24 valid to –CD_SFDBK active delay	–	55	ns
t_2	Address, M/-IO, MADE24 invalid to –CD_SFDBK inactive delay	0	–	ns

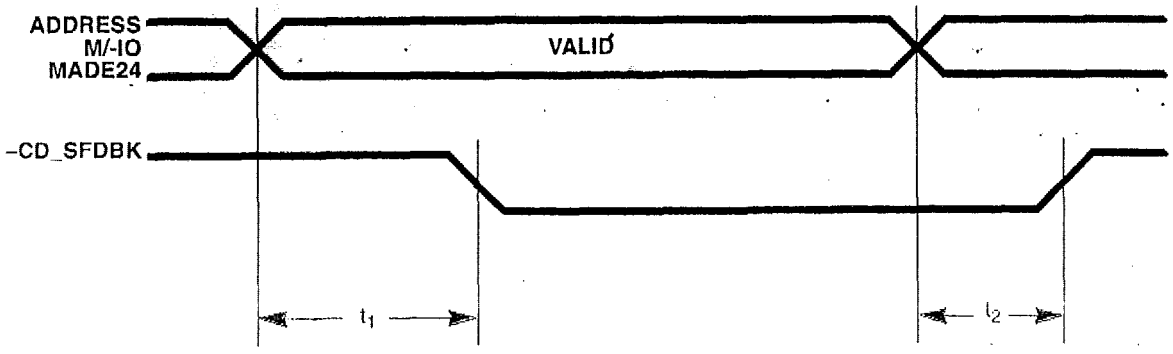


Figure 7-20. –CD_SFDBK Timing (MicroChannel® Bus)

Table 7-21. –CD_SETUP Timing (MicroChannel® Bus)

Symbol	Parameter	MIN	MAX	Units
t_1	–CD_SETUP active setup to –CMD active	10	–	ns
t_2	–CD_SETUP delay to CD_CHRDY inactive	–	95	ns
t_3	–CMD active to –CD_SETUP inactive hold	25	–	ns

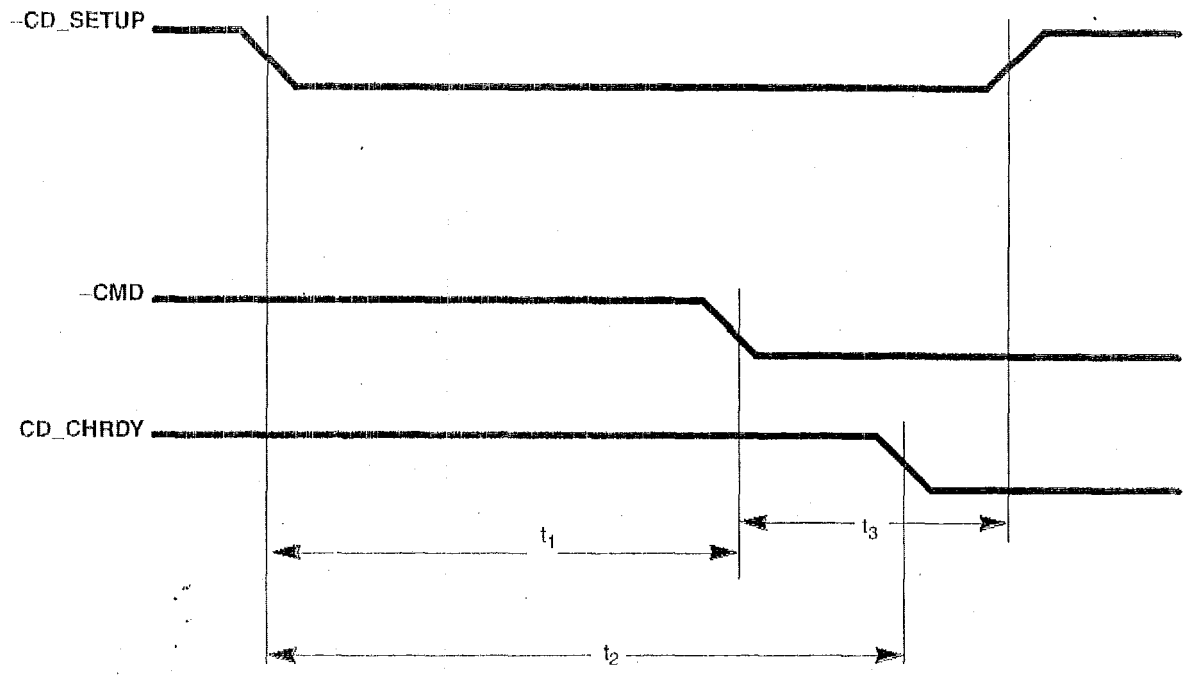


Figure 7-21. –CD_SETUP Timing (MicroChannel® Bus)

Table 7-22. CLK1X, CLK2X Timing (Local Bus)

Symbol	Parameter	MIN	MAX	Units
t_1	Rise time (CLK1X) '486 (V_{IL} - V_{IH})	–	4	ns
t_2	Fall time (CLK1X) '486 (V_{IH} - V_{IL})	–	4	ns
t_3	High period (CLK1X) '486 (V_{IH} - V_{IH})	40	60	% t_5
t_4	Low period (CLK1X) '486 (V_{IL} - V_{IL})	40	60	% t_5
t_5	Period (CLK1X) '486	25	–	ns
t_5	Period (CLK1X) '486 (CL-GD5429)	20	–	ns
t_1	Rise time (CLK2X) '386 (V_{IL} - V_{IH})	–	4	ns
t_2	Fall time (CLK2X) '386 (V_{IH} - V_{IL})	–	4	ns
t_3	High period (CLK2X) '386 (V_{IH} - V_{IH})	40	60	% t_5
t_4	Low period (CLK2X) '386 (V_{IL} - V_{IL})	40	60	% t_5
t_5	Period (CLK2X) '386	12.5	–	ns

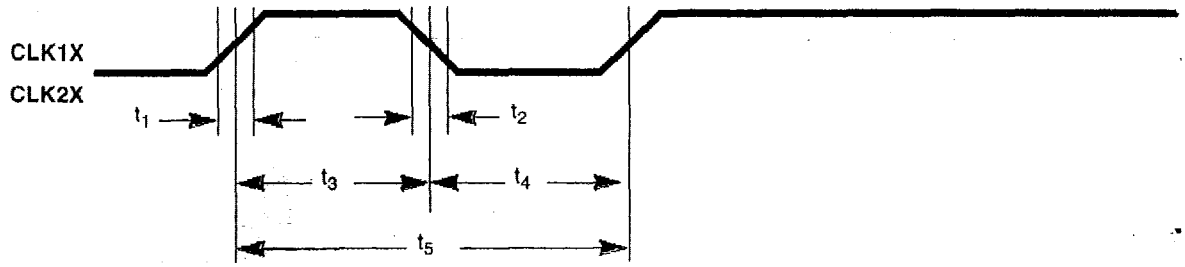


Figure 7-22. CLK1X, CLK2X Timing (Local Bus)

Table 7-23. RESET Timing (Local Bus)[a]

Symbol	Parameter	MIN	MAX	Units
t_1	CPU_RESET hold time from CLK2X	10	–	ns
t_2	CPU_RESET setup time to CLK2X	2	–	ns

[a] Applies to '386 only. For '486, pin 46 must be tied to ground. For VESA VL-Bus, pin 46 must be tied to RDYRTN#.

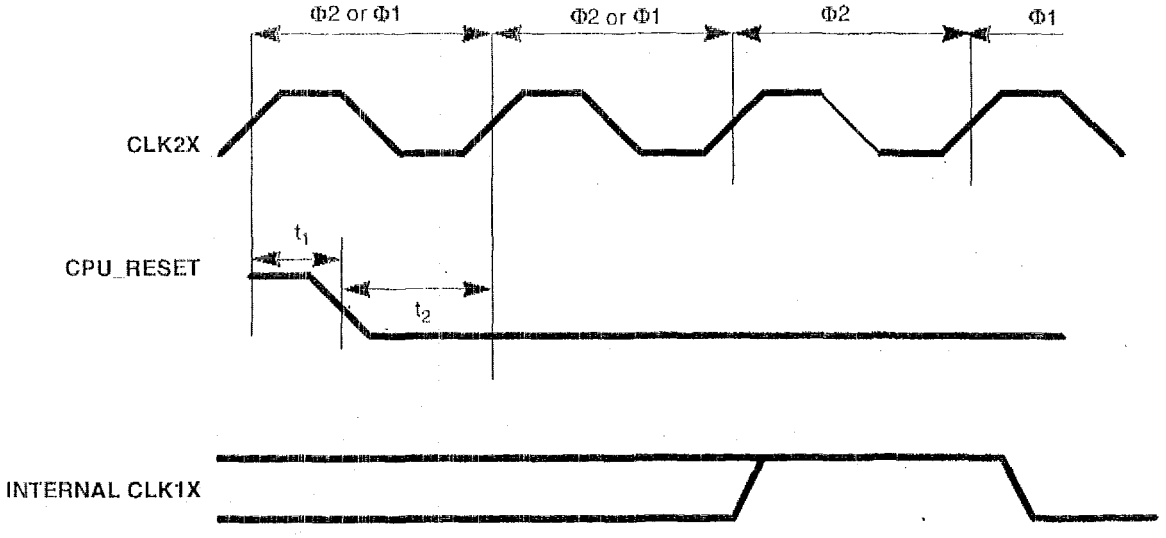


Figure 7-23. RESET Timing (Local Bus)

Table 7-24. ADS#, LBA#, BS16# Timing (Local Bus)

Symbol	Parameter	MIN '24/'26/'28	MAX '24/'26/'28	MIN '29	MAX '29	Units
t_1	Address, Status, ADS# setup to CLK1X	8	–	4	–	ns
t_2	LBA#/LDEV# active delay from Address, Status	–	15	–	15	ns
t_3	BS16# active delay CLK1X	–	15	–	3	ns
t_4	LBA# inactive delay from Address, Status	–	18	–	18	ns

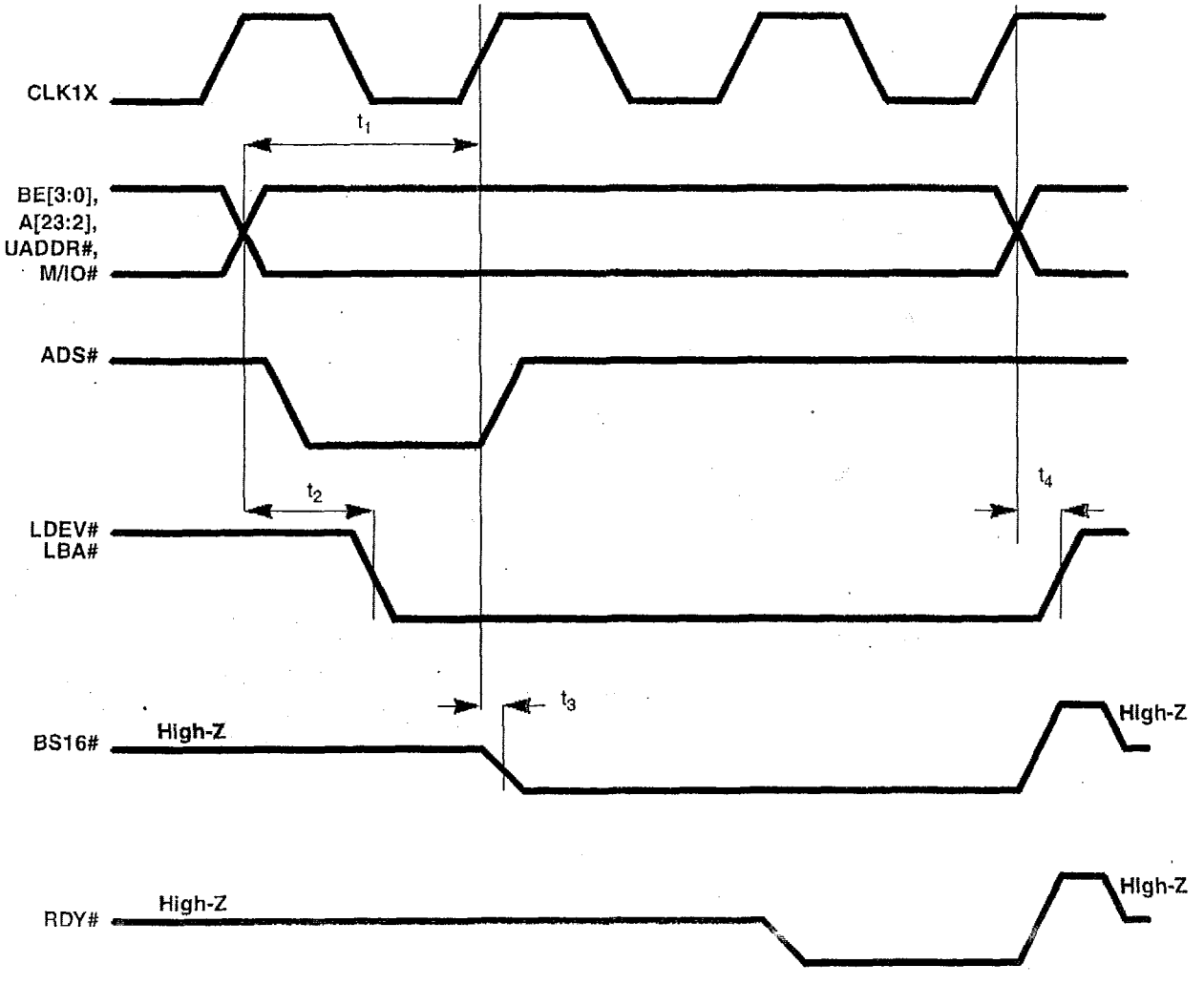


Figure 7-24. ADS#, LBA#, BS16# Timing (Local Bus)

Table 7-25. RDY# Delay (Local Bus)

Symbol	Parameter	MIN	MAX	Units
t_1	RDY# active delay from CLK1X	3	13	ns
t_2	RDY# inactive delay from CLK1X	–	23	ns
t_3	RDY# high before High-Z	–	1/2	CLK1X

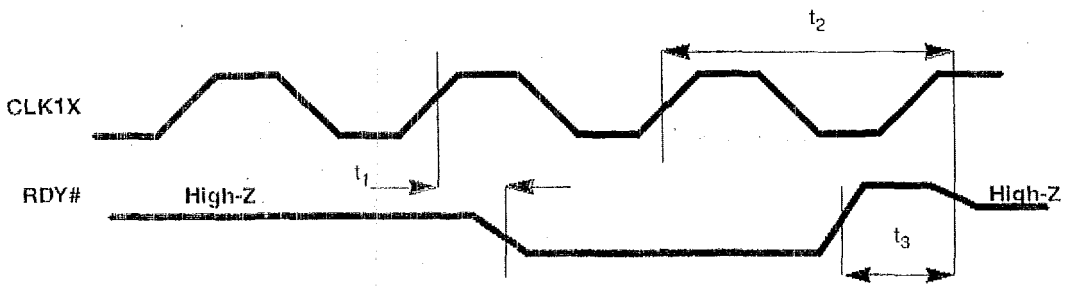


Figure 7-25. RDY# Delay (Local Bus)

Table 7-26. Read Data Timing (Local Bus)

Symbol	Parameter	MIN '24/'26/'28	MAX '24/'26/'28	MIN '5429	MAX '5429	Units
t_1	Read data setup to RDY# active	0	–	0	–	ns
t_2	Read data hold from RDY# inactive	12	–	12	–	ns
t_3	RDYRTN# setup to CLK1X	8	–	5	–	ns
t_4	RDYRTN# hold from CLK1X	5	–	2	–	ns

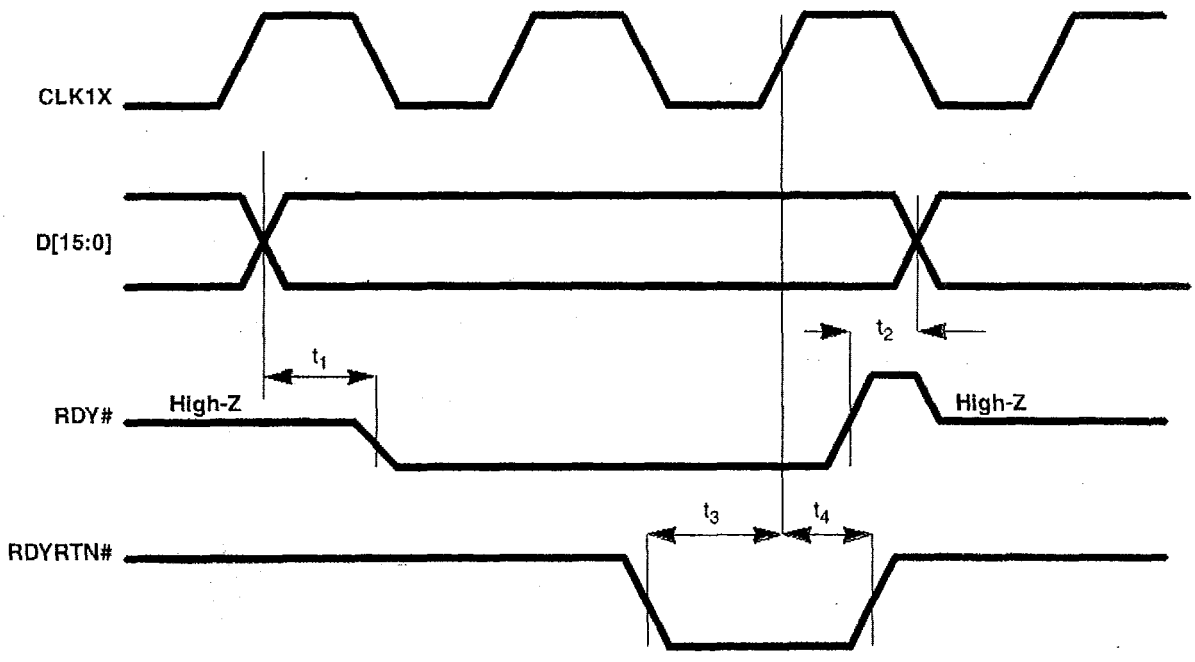
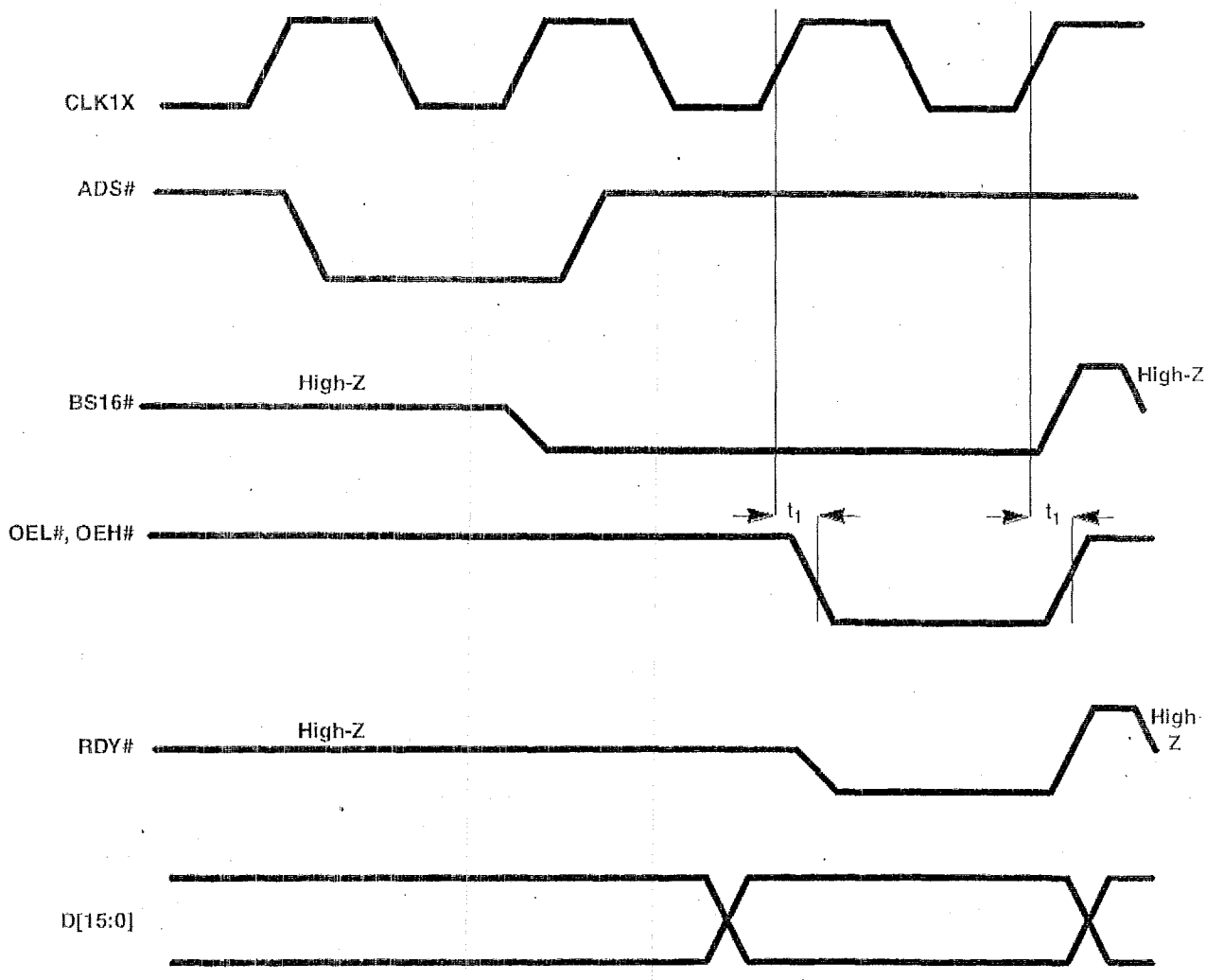


Figure 7-26. Read Data Timing (Local Bus)

Table 7-27. Buffer Control Timing – Read Cycle (Local Bus)

Symbol	Parameter	MIN	MAX	Units
t_1	OEL#, OEH# delay	0	14	ns

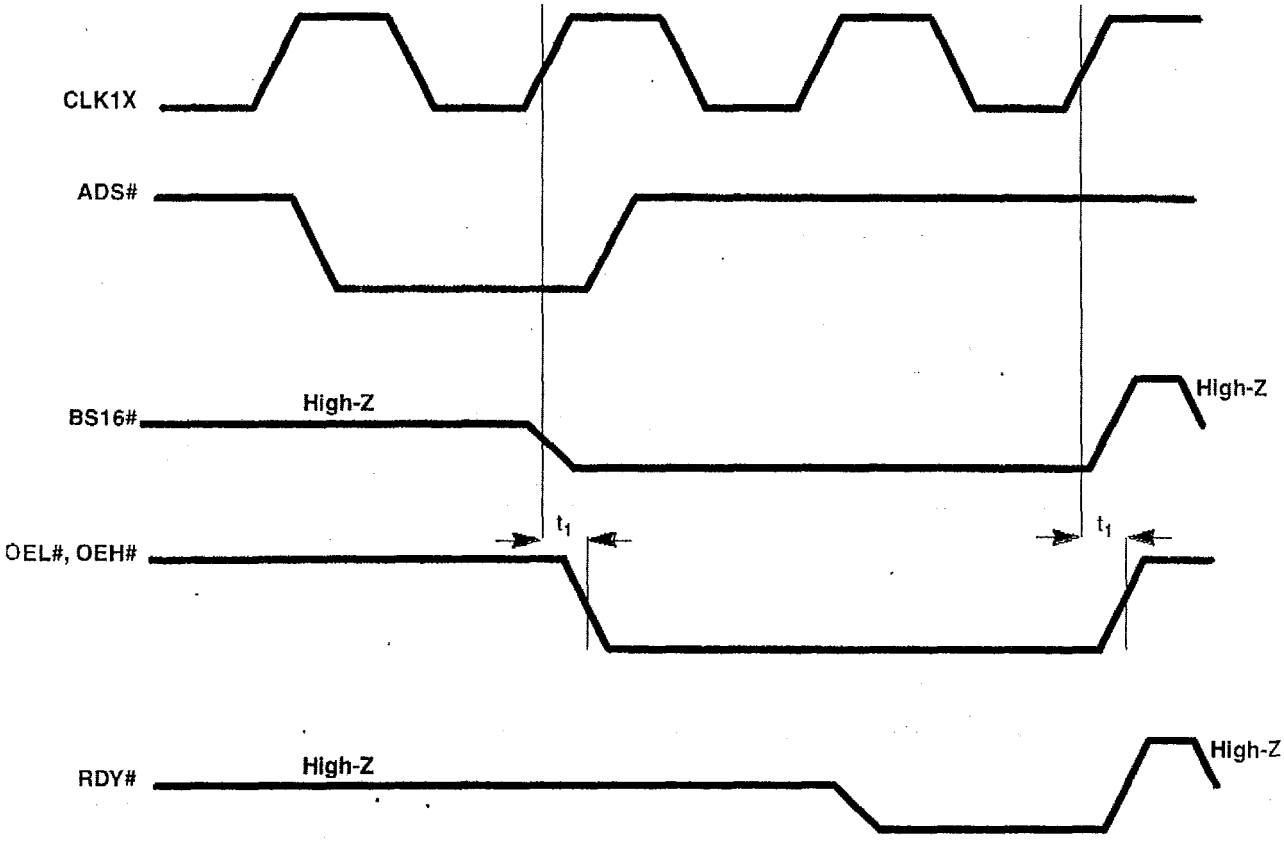


NOTE: Both OEL# and OEH# are active for read cycles.

Figure 7-27. Buffer Control Timing – Read Cycle (Local Bus)

Table 7-28. Buffer Control Timing – Write Cycle (Local Bus)

Symbol	Parameter	MIN	MAX	Units
t_1	OEL#, OEH# delay	0	14	ns



NOTE: Only one of OEL# and OEH# is active for write cycles.

Figure 7-28. Buffer Control Timing – Write Cycle (Local Bus)

Table 7-29. Display Memory Bus – Common Parameters

Symbol	Parameter	MIN	MAX
t_1	t_{ASR}: address setup to RAS* active	1.5m [a] – 2 ns	–
t_2	t_{RAH}: row address hold from RAS* active	1.5m	–
t_3	t_{ASC}: address setup to CAS* active	1m – 1 ns	–
t_4	t_{CAH}: column address hold from CAS* active	1m	–
t_5	t_{RCD}: RAS* active to CAS* active delay (standard RAS)	2.5m – 2 ns	–
t_5	t_{RCD}: RAS* active to CAS* active delay (extended RAS)	3m – 2 ns	–
t_6	t_{RAS}: RAS* pulse width low (standard RAS)	3.5m	–
t_6	t_{RAS}: RAS* pulse width low (extended RAS)	4m – 2 ns	–
t_7	t_{RP}: RAS* precharge (RAS* pulse width high — standard RAS)	2.5 m – 2 ns	–
t_7	t_{RP}: RAS* precharge (RAS* pulse width high — extended RAS)	3m	–
t_8	t_{CAS}: CAS* pulse width low	1m + 3 ns	1m + 6 ns
t_9	t_{CP}: CAS* precharge (CAS* pulse width high)	1m – 6 ns	1m – 3 ns
t_{10}	t_{RC}: Random cycle (standard RAS)	6m	–
t_{10}	t_{RC}: Random cycle (extended RAS)	7m	–
t_{11}	t_{PC}: Page mode cycle	2m	–

[a] m = MCLK

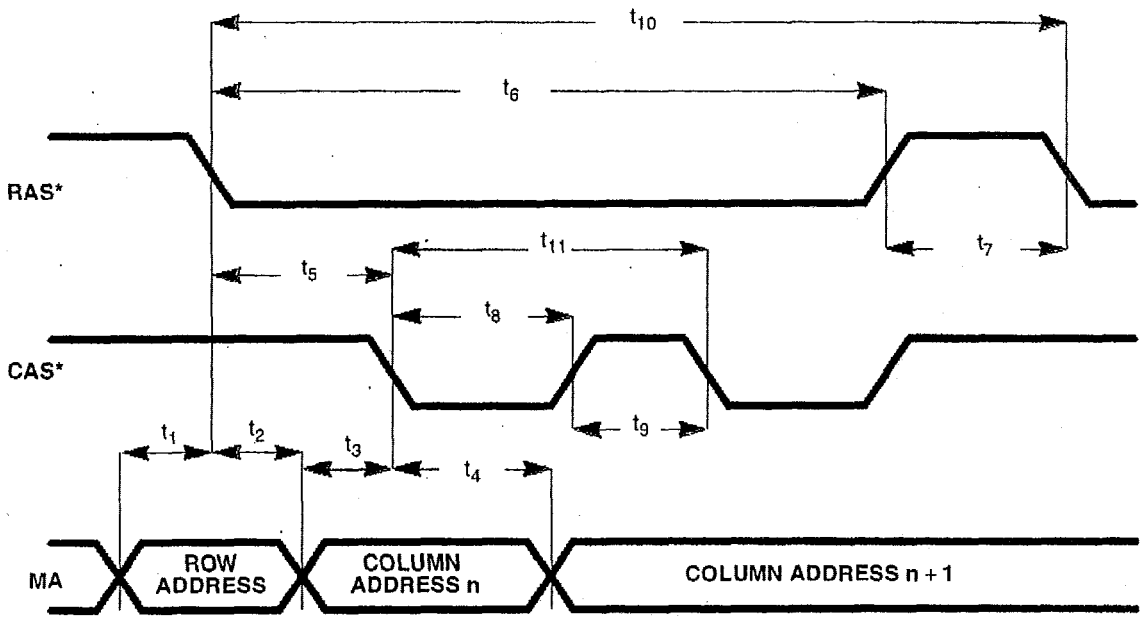


Figure 7-29. Display Memory Bus – Common Parameters

Table 7-30. Display Memory Bus – Read Cycles[a]

Symbol	Parameter	MIN	MAX
t_1	Read data setup to CAS* rising edge	0	–
t_2	Read data hold from CAS* high	10 ns	–
t_3 [b]	RAS* active to first CAS* rising edge delay (standard RAS)	–	4m – 1 ns
t_3	RAS* active to first CAS* rising edge delay (extended RAS)	–	4.5m – 1 ns
t_4 [c]	Column address valid to CAS* rising edge delay	–	2m
t_5 [d]	CAS* active pulse width	–	1m + 3 ns
t_6 [e]	CAS* period	–	2m

[a] Only parameters t_1 and t_2 are defined for the device. The remaining parameters in this table are calculated from parameters in the Table 7-29. They are provided so that system designers can determine DRAM requirements.

[b] Parameter t_3 corresponds to DRAM parameter t_{RAC} (access time from RAS*).

[c] Parameter t_4 corresponds to DRAM parameter t_{AA} (access time from column address).

[d] Parameter t_5 corresponds to DRAM parameter t_{CAC} (access time from CAS*).

[e] Parameter t_6 corresponds to DRAM parameter t_{CAP} (access time from CAS* precharge).

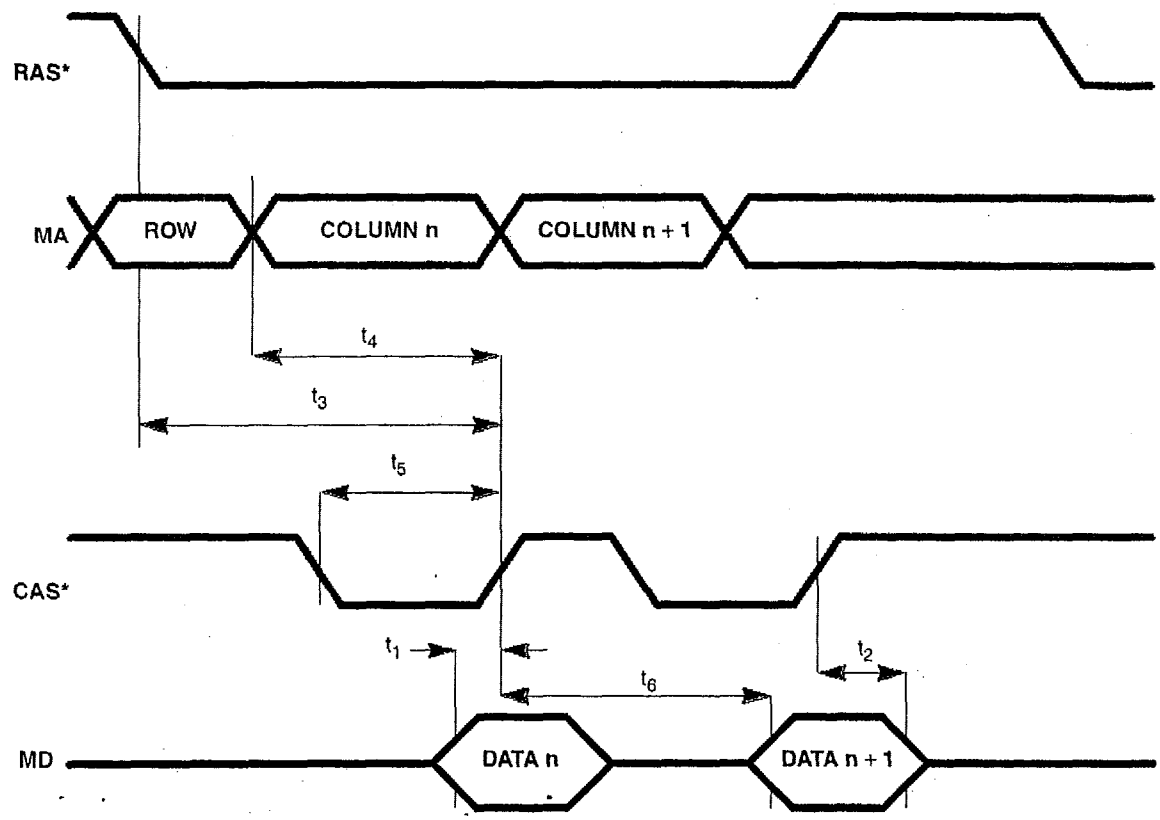


Figure 7-30. Display Memory Bus – Read Cycles

Table 7-31. Display Memory Bus – Write Cycles

Symbol	Parameter	MIN	MAX
t_1	t_{CWL}: WE* active setup to CAS* active	1m[a] + 2 ns	–
t_2	t_{DS}: Write data setup to CAS* active	1m – 2 ns	1m + 2 ns
t_3	t_{DH}: write data hold from CAS* active	1m + 1 ns	–
t4	t_{WCH}: WE active hold from CAS* active	1.5m	–

[a] m = MCLK

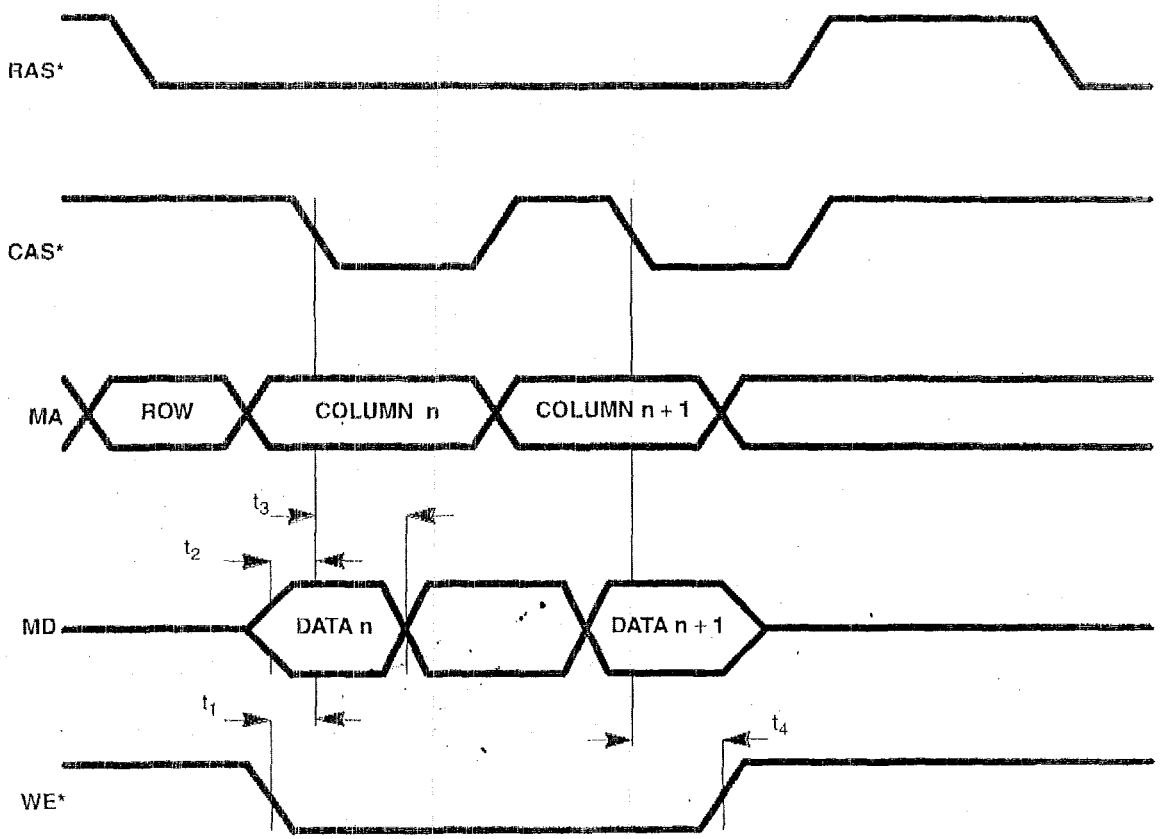


Figure 7-31. Display Memory Bus – Write Cycles

Table 7-32. CAS*-before-RAS* Refresh Timing (Display Memory Bus)[a]

Symbol	Parameter	MIN	MAX
t_1	t_{CSR}: CAS* active setup to RAS* active1	1m[b]	–
t_2	t_{RAS}: RAS* low pulse width	4m	–
t_3	t_{RP}: RAS* high pulse width	3m	–

[a] There will be either three or five RAS* pulses while CAS* remains low.

[b] m = MCLK

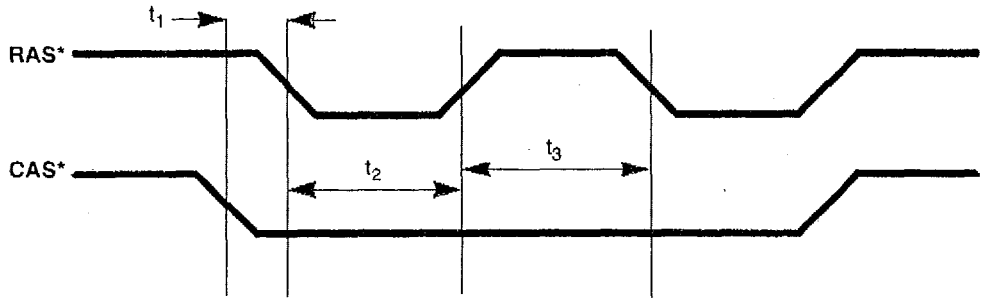


Figure 7-32. CAS*-before-RAS* Refresh Timing (Display Memory Bus)

Table 7-33. P-Bus as Inputs, 8-Bit Mode (DCLK input as reference)

Symbol	Parameter	MIN	MAX	Units
t_1	P[7:0], BLANK* setup to DCLK	–1	–	ns
t_2	P[7:0], BLANK* hold from DCLK	7	–	ns

NOTE: CL-GD542X RAMDAC driven externally

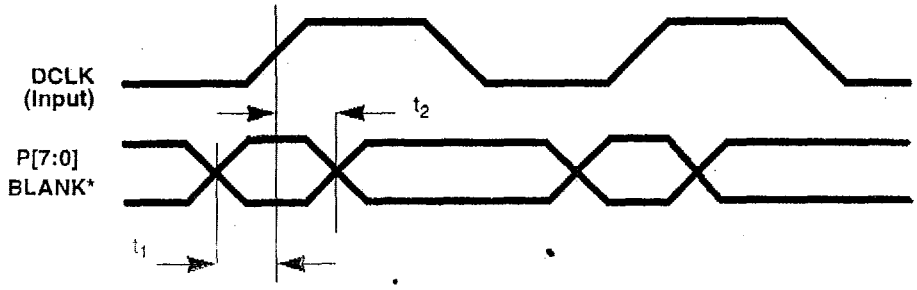


Figure 7-33. P-Bus as Inputs, 8-bit Mode (External DCLK)

Table 7-34. Feature Bus Timing, 8-Bit Mode, Outputs (DCLK output as reference)

Symbol	Parameter	MIN	MAX	Units
t_1	DCLK to BLANK* delay	–1	1	ns
t_2	DCLK to HSYNC, VSYNC delay	1	3	ns
t_3	DCLK to P[7:0] delay	–2	0	ns
t_4	DCLK to OVRW delay	–1	1	ns

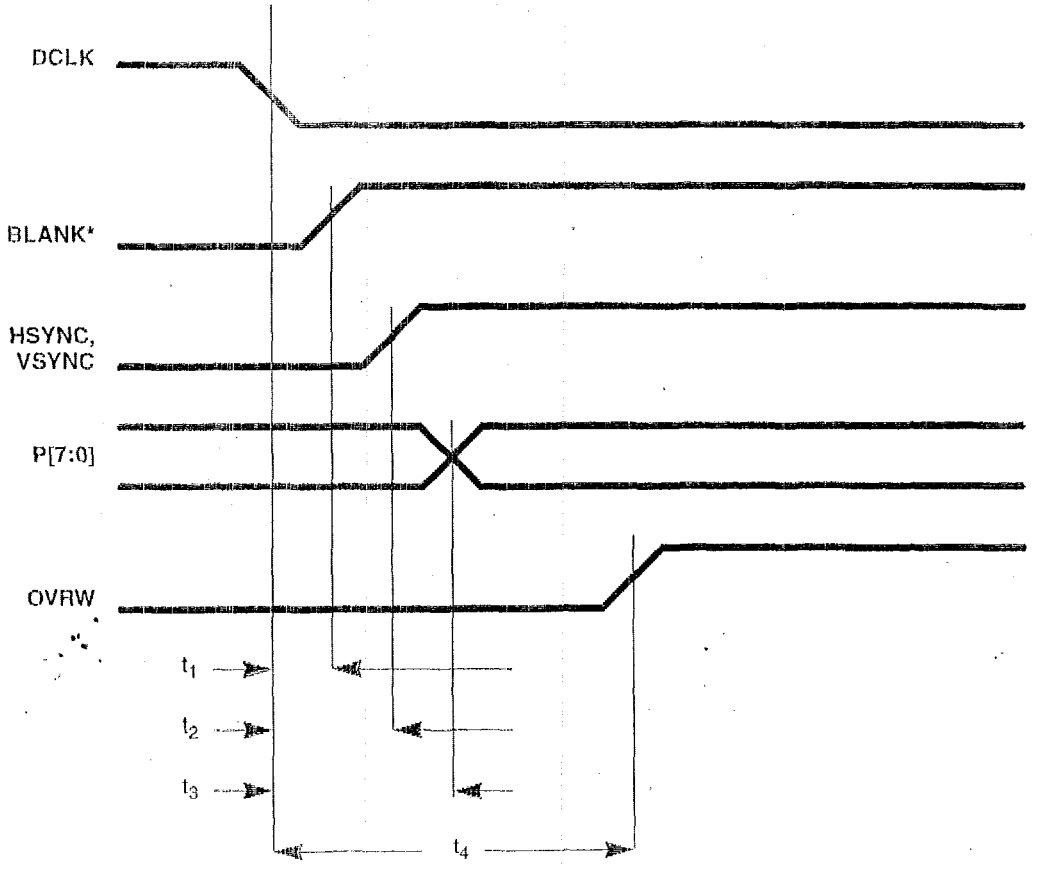


Figure 7-34. Feature Bus Timing, 8-Bit Mode, Outputs (Internal DCLK)

Table 7-35. P-Bus as Outputs, 16-bit Mode (DCLK output as reference) CL-GD5425/'28/'29 only[a]

Symbol	Parameter	MIN	MAX	Units
t_1	DCLK (rising edge) to P[7:0] delay	–2	0	ns
t_2	DCLK (falling edge) to P[7:0] delay	–1	1	ns

[a] SR7[2:1] is programmed to '0,1' and GRE[0] is programmed to '1'.

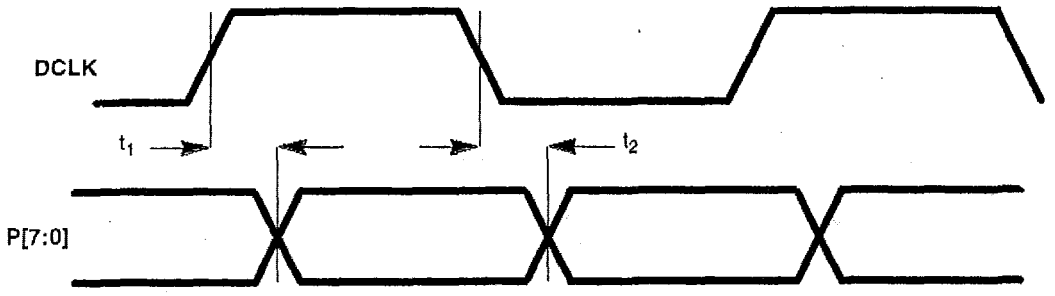


Figure 7-35. P-Bus as Outputs, 16-bit Mode (Internal DCLK) CL-GD5425/'28/'29 only

Table 7-36. P-Bus as Inputs, 16-Bit Mode, Clock Mode 1 (DCLK input as reference) [a]

Symbol	Parameter	MIN '24/'26	MIN '28	MIN '25/'29	Units
t_1	P[7:0] setup to DCLK (rising edge — external DCLK)	–1	–1	–3	ns
t_2	P[7:0] hold from DCLK (rising edge — external DCLK)	5	5	7	ns
t_3	P[7:0] setup to DCLK (falling edge — external DCLK)	–1	–1	–3	ns
t_4	P[7:0] hold from DCLK (rising edge — external DCLK)	5	5	7	ns

[a] Clock mode 1 selected in Hidden DAC register (D5 programmed to '0').

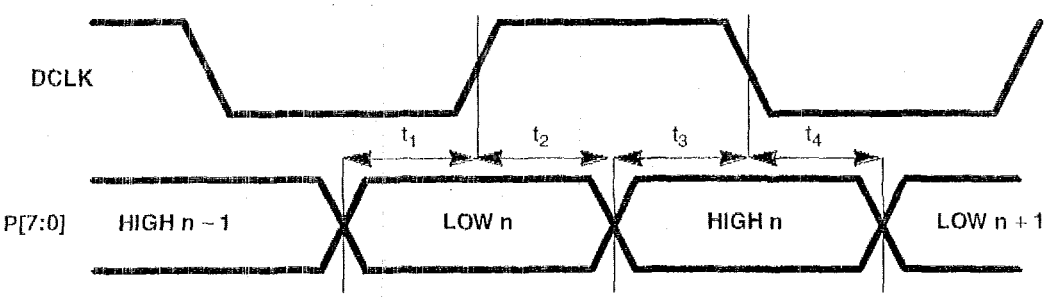


Figure 7-36. P-Bus as Inputs, 16-Bit Mode, Clock Mode 1 (External DCLK)

Table 7-37. P-Bus as Inputs, 16-Bit Mode, Clock Mode 2[a] (DCLK input as reference)

Symbol	Parameter	MIN '24/'26	MIN '28	MIN '25/'29	Units
t_1	P[7:0], BLANK*[b] setup to DCLK	-1	-2	-2	ns
t_2	P[7:0], BLANK* hold from DCLK	5	6	4	ns

[a] Clock mode 2 selected in HIdden DAC register (D5 = '1').

[b] The first low byte of 16-bit data input must be synchronized with BLANK* or the start of overlay window, whichever is later. The first high byte will be clocked on the next rising edge of DCLK.

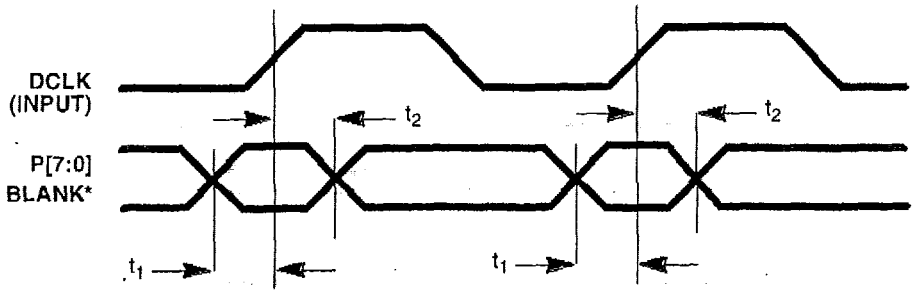


Figure 7-37. P-Bus as Inputs, 16-Bit Mode, Clock Mode 2 (External DCLK)

Table 7-38. P-Bus as Inputs, 16-Bit Mode (DCLK output as reference)[a]

Symbol	Parameter	MIN '24/'26	MIN '28	MIN '25/'29	Units
t_1	P[7:0] setup to DCLK (rising edge — internal DCLK)	3	2	3	ns
t_2	P[7:0] hold from DCLK (rising edge — internal DCLK)	1	1	1	ns
t_3	P[7:0] setup to DCLK (falling edge — internal DCLK)	3	2	3	ns
t_4	P[7:0] hold from DCLK (rising edge — internal DCLK)	1	1	1	ns

[a] Clock mode 1 selected in Hidden DAC register (D5 = '0').

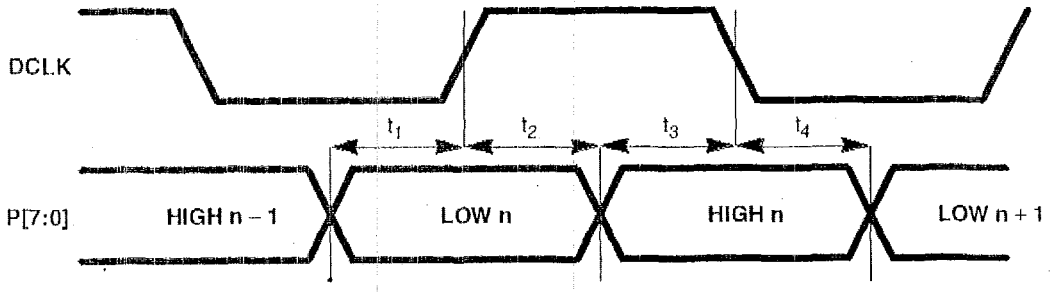


Figure 7-38. P-Bus as Inputs, 16-Bit Mode (External DCLK)

Table 7-39. DCLK as Input

Symbol	Parameter: CL-GD5420/'22/'24/'26/'28	MIN	MAX	Units
t_1	Rise time	–	3	ns
t_2	Fall time	–	3	ns
t_3	High period	40	60	% of t_5
t_4	Low period	40	60	% of t_5
t_5	Period	17	–	ns
	Parameter: CL-GD5425/'29			
t_1	Rise time	–	3	ns
t_2	Fall time	–	3	ns
t_3	High period: Clock mode 1	45	55	% of t_5
t_3	High period: Clock mode 2	30	70	% of t_5
t_4	Low period: Clock mode 1	45	55	% of t_5
t_4	Low period: Clock mode 2	30	70	% of t_5
t_5	Period (DCLK)	12.5	–	ns

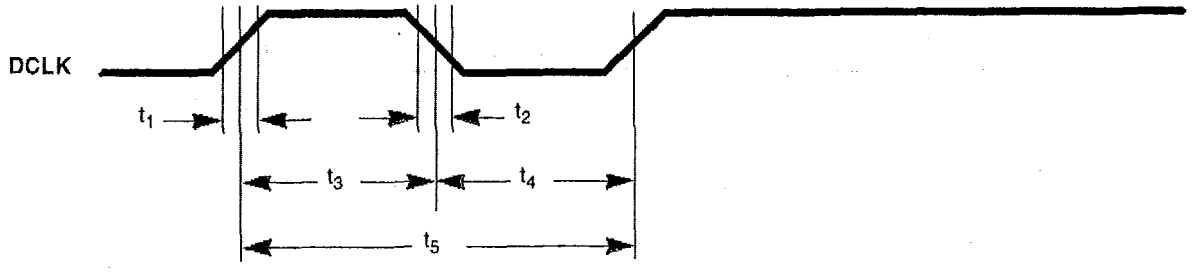


Figure 7-39. DCLK As Input

Table 7-40. RESET Timing

Symbol	Parameter	MIN	MAX	Units
t_1	RESET pulse width	12	–	MCLK
t_2	MD[31:16] setup to RESET falling edge	2	–	ns
t_3	MD[31:16] hold from RESET falling edge	25	–	ns
t_4	RESET low to first IOW*	12	–	MCLK

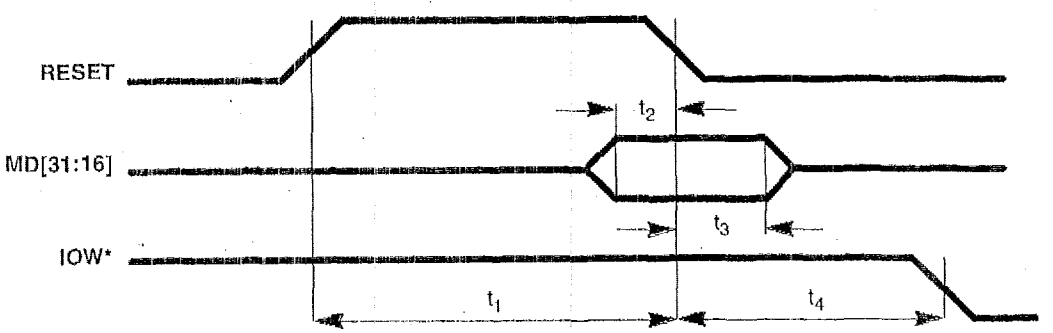


Figure 7-40. RESET Timing

Table 7-41. Horizontal Period (NTSC — CL-GD5425 only)

Symbol	Parameter	Nominal	Units
t_1	Horizontal period	63.56	µsec.
t_2	HSYNC pulse width (low)	4.65	µsec.

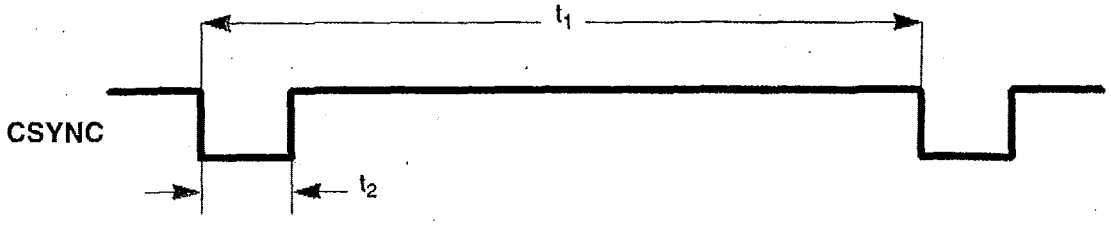


Figure 7-41. Horizontal Period (NTSC — CL-GD5425 only)

Table 7-42. NTSC Vertical Retrace (CL-GD5425 only)

Symbol	Parameter	Nominal	Units
t_1	Vertical blanking period	20	H-period

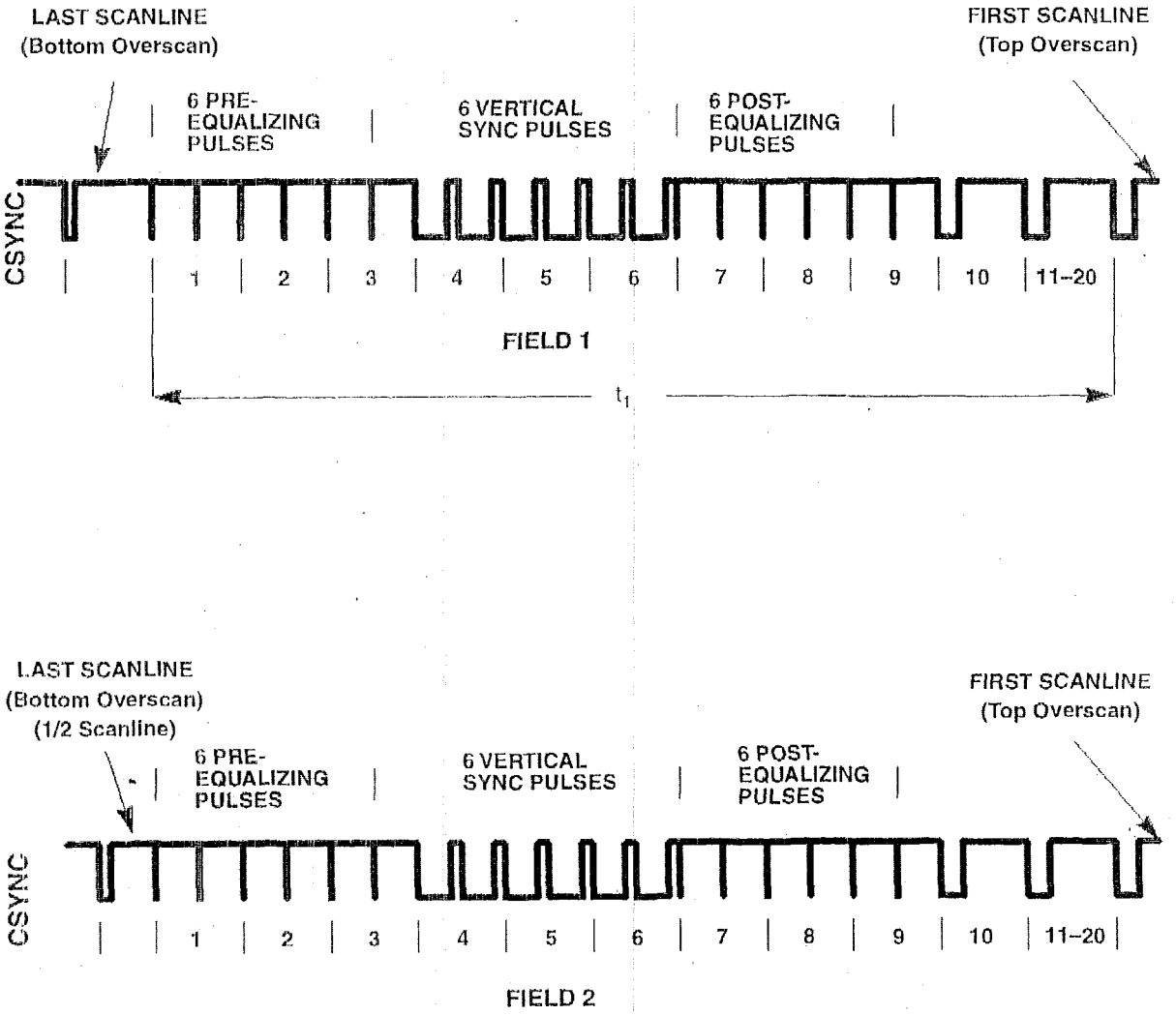


Figure 7-42. NTSC Vertical Retrace (CL-GD5425 only)

Table 7-43. NTSC Vertical Blanking Detail (CL-GD5425 only)

Symbol	Parameter	Nominal	Units
t_1	Equalizing pulse width (low)	2.32	µsec.
t_2	Serrations pulse width (high)	4.65	µsec.
t_3	Equalization to serration	1	H-period
t_4	First serration	1/2	H-period

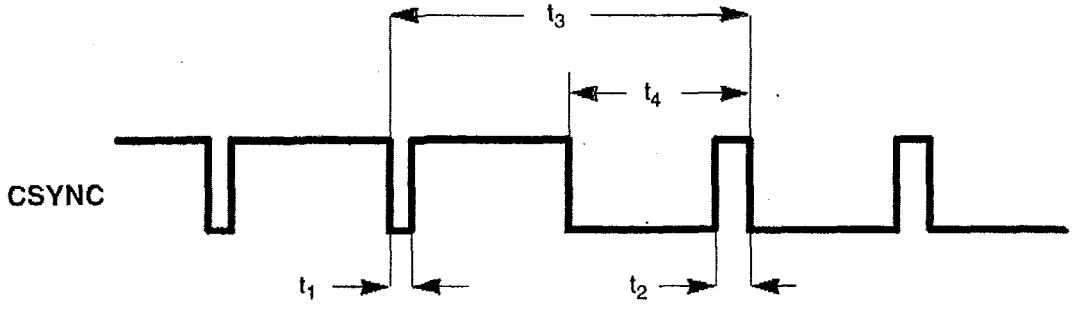


Figure 7-43. NTSC Vertical Blanking Detail (CL-GD5425 only)

Table 7-44. Horizontal Period (PAL — CL-GD5425 only)

Symbol	Parameter	Nominal	Units
t_1	Horizontal period	64.00	µsec.
t_2	HSYNC pulse width (low)	4.65	µsec.

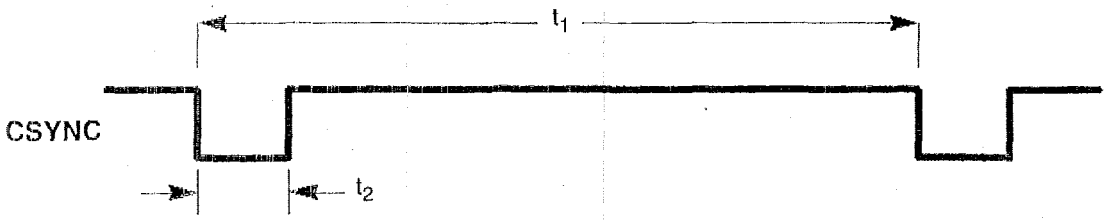


Figure 7-44. Horizontal Period (PAL — CL-GD5425 only)

Table 7-45. PAL Vertical Retrace (CL-GD5425 only)

Symbol	Parameter	Nominal	Units
t_1	Vertical blanking period	25	H-period

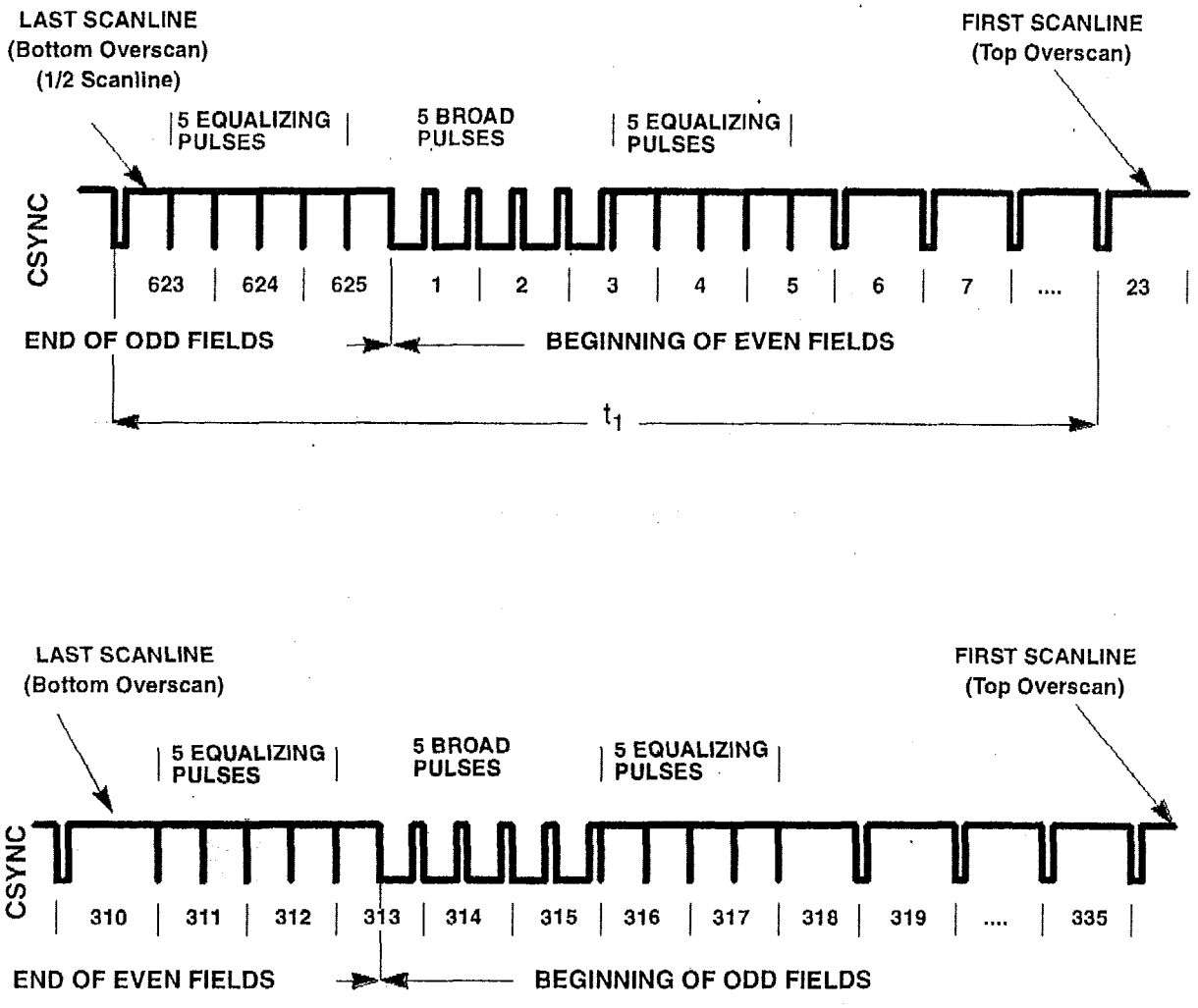
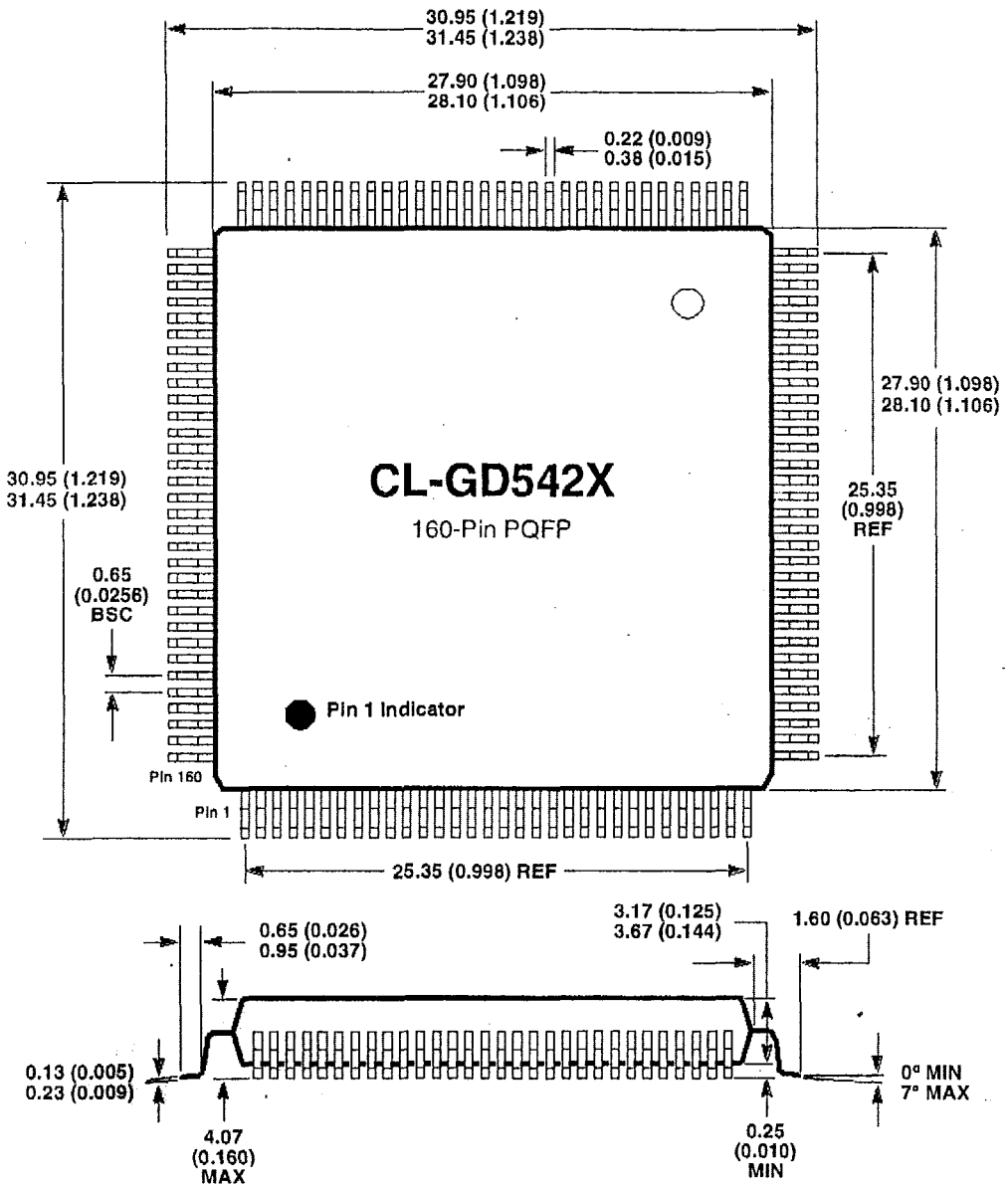


Figure 7-45. PAL Vertical Retrace (CL-GD5425 only)

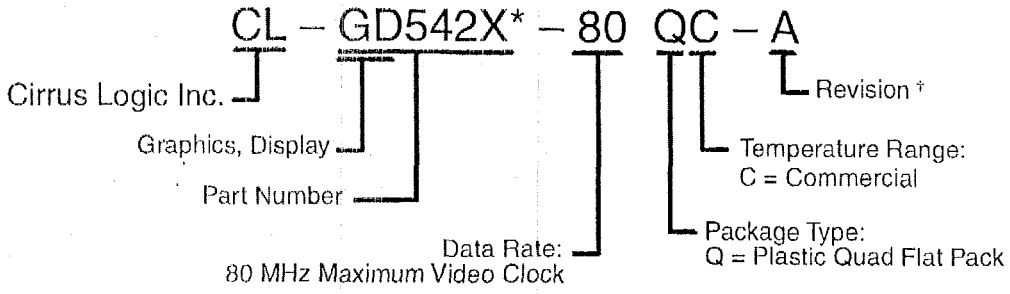
8. PACKAGE DIMENSIONS



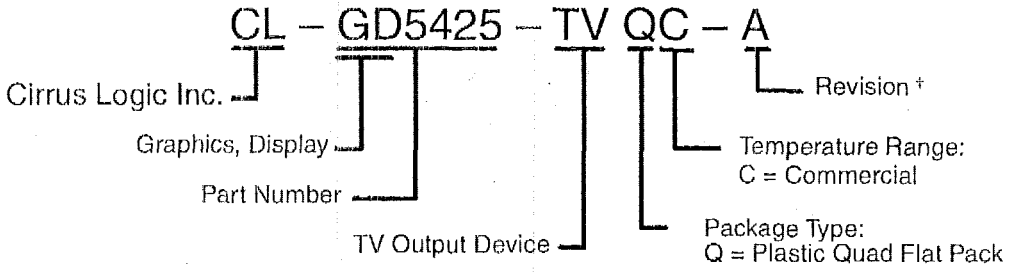
NOTES:

1) Dimensions are in millimeters (inches), and controlling dimension is millimeter.
2) Drawing above does not reflect exact package pin count.
3) Before beginning any new design with this device, please contact Cirrus Logic for the latest package information..

9. ORDERING INFORMATION EXAMPLES



† Contact Cirrus Logic Inc. for up-to-date information on revisions.
* '2X' represents CL-GD5420/'22/'24/'26/'28/'29, respectively.



† Contact Cirrus Logic Inc. for up-to-date information on revisions.

Direct Sales Offices

Domestic

N. CALIFORNIA
Fremont
TEL: 510/623-8300
FAX: 510/252-6020

Sacramento
TEL: 916/933-4200
FAX: 916/933-4211

S. CALIFORNIA
Tustin
TEL: 714/573-9911
FAX: 714/573-4665

Thousand Oaks
TEL: 805/371-5381
FAX: 805/371-5382

NORTHWESTERN AREA
Portland, OR
TEL: 503/620-5547
FAX: 503/624-5665

ROCKY MOUNTAIN AREA
Denver, CO
TEL: 303/786-9696
FAX: 303/786-9695

SOUTH CENTRAL AREA
Austin, TX
TEL: 512/255-0080
FAX: 512/255-0733

Dallas, TX
TEL: 214/252-6698
FAX: 214/252-5681

Houston, TX
TEL: 713/379-5772
FAX: 713/379-4341

CENTRAL AREA
Chicago, IL
TEL: 708/981-6950
FAX: 708/981-6846

NORTHEASTERN AREA
Andover, MA
TEL: 508/474-9300
FAX: 508/474-9149

Boston, MA
TEL: 617/721-1439
FAX: 617/721-4509

Iselin, NJ
TEL: 908/632-2771
FAX: 908/632-2914

Philadelphia, PA
TEL: 215/625-0781
FAX: 215/625-0731

SOUTHEASTERN AREA
Atlanta, GA
TEL: 404/623-4653
FAX: 404/497-0414

Boca Raton, FL
TEL: 407/241-5777
FAX: 407/241-7990

Raleigh, NC
TEL: 919/481-9610
FAX: 919/481-9640

International

GERMANY
Herrsching
TEL: 49/8152-40084
FAX: 49/8152-40077

FRANCE
Rosny sous bois
TEL: 33/1-48-122812
FAX: 33/1-48-122810

HONG KONG
Tsimshatsui
TEL: 852/376-0801
FAX: 852/375-1202

JAPAN
Tokyo
TEL: 81/3-3340-9111
FAX: 81/3-3340-9120

KOREA
Seoul
TEL: 82/2-565-8561
FAX: 82/2-565-8565

SINGAPORE
TEL: 65/353-2122
FAX: 65/353-2166

TAIWAN
Taipei
TEL: 886/2-718-4533
FAX: 886/2-718-4526

UNITED KINGDOM
Hertfordshire, England
TEL: 44/1727-872424
FAX: 44/1727-875919

The Company

Headquartered in Fremont, California, Cirrus Logic Inc. develops innovative architectures for analog and digital system functions. The Company implements those architectures in proprietary integrated circuits and related software for applications that include user interface and multimedia (graphics, audio, and video), mass storage, communications, and data acquisition.

Key markets for Cirrus Logic's products include desktop and portable computing, workstations, telecommunications, and consumer electronics.

The Cirrus Logic formula combines innovative architectures in silicon with system design expertise. We deliver complete solutions — chips, software, evaluation boards, and manufacturing kits — on-time, to help you win in the marketplace.

Cirrus Logic's manufacturing strategy, unique in the semiconductor industry, employs a full manufacturing infrastructure to ensure maximum product quality, availability, and value for our customers.

Talk to our systems and applications specialists; see how you can benefit from a new kind of semiconductor company.

Cirrus Logic Inc. 3100 West Warren Ave. Fremont, CA 94538
TEL: 510/623-8300
FAX: 510/252-6020

recommended operating conditions

		MIN	NOM	MAX	UNIT
Supply voltage, V_{CC}		4.75	5	5.25	V
Clock high-level input voltage at XTAL1, $V_{IH(CLK)}$		2		V_{CC}	V
Clock low-level input voltage at XTAL1, $V_{IL(CLK)}$		−0.5		0.8	V
High-level input voltage, V_{IH}		2		V_{CC}	V
Low-level input voltage, V_{IL}		−0.5		0.8	V
Clock frequency, f_{clock}				16	MHz
Operating free-air temperature, T_A	TL16C554	0		70	°C
	TL16C554I	−40		85	°C

electrical characteristics over recommended ranges of operating free-air temperature and supply voltage (unless otherwise noted)

PARAMETER		TEST CONDITIONS	MIN	TYP†	MAX	UNIT
V_{OH}‡	High-level output voltage	I_{OH} = −1 mA	2.4			V
V_{OL}‡	Low-level output voltage	I_{OL} = 1.6 mA			0.4	V
I_{lkg}	Input leakage current	V_{CC} = 5.25 V, GND = 0, V_I = 0 to 5.25 V, All other terminals floating			±10	µA
I_{OZ}	High-impedance output current	V_{CC} = 5.25 V, GND = 0, V_O = 0 to 5.25 V, Chip selected in write mode or chip deselected			±20	µA
I_{CC}	Supply current	V_{CC} = 5.25 V, T_A = 25°C, RX, DSR, DCD, CTS, and RI at 2 V, All other inputs at 0.8 V, XTAL1 at 4 MHz, No load on outputs, Baud rate = 50 kilobits per second			50	mA
$C_{i(XTAL1)}$	Clock input capacitance	V_{CC} = 0, V_{SS} = 0, All other terminals grounded, f = 1 MHz, T_A = 25°C		15	20	pF
$C_{o(XTAL2)}$	Clock output capacitance			20	30	pF
C_i	Input capacitance			6	10	pF
C_o	Output capacitance			10	20	pF

† All typical values are at V_{CC} = 5 V, T_A = 25°C.
‡ *These parameters apply for all outputs except XTAL2.*

clock timing requirements over recommended ranges of operating free-air temperature and supply voltage (see Figure 1)

		MIN	MAX	UNIT
t_{w1}	Pulse duration, clock high (external clock)	31		ns
t_{w2}	Pulse duration, clock low (external clock)	31		ns
t_{w3}	Pulse duration, RESET	1000		ns

read cycle timing requirements over recommended ranges of operating free-air temperature an supply voltage (see Figure 4)

		MIN	MAX	UNIT
t_{w4}	Pulse duration, $\overline{IOR}$ low	75		ns
t_{su1}	Setup time, $\overline{CSx}$ valid before $\overline{IOR}$ low (see Note 2)	10		ns
t_{su2}	Setup time, A2–A0 valid before $\overline{IOR}$ low (see Note 2)	15		ns
t_{h1}	Hold time, A2–A0 valid after $\overline{IOR}$ high (see Note 2)	0		ns
t_{h2}	Hold time, $\overline{CSx}$ valid after $\overline{IOR}$ high (see Note 2)	0		ns
t_{d1}	Delay time, $t_{su2} + t_{w4} + t_{d2}$ (see Note 3)	140		ns
t_{d2}	Delay time, $\overline{IOR}$ high to $\overline{IOR}$ or $\overline{IOW}$ low	50		ns

NOTES: 2. The internal address strobe is always active.
3. In the FIFO mode, t_{d1} = 425 ns (min) between reads of the receiver FIFO and the status registers (interrupt identification regis and line status register).

write cycle timing requirements over recommended ranges of operating free-air temperature ar supply voltage (see Figure 5)

		MIN	MAX	UNIT
t_{w5}	Pulse duration, $\overline{IOW}\downarrow$	50		ns
t_{su3}	Setup time, $\overline{CSx}$ valid before $\overline{IOW}\downarrow$ (see Note 2)	10		ns
t_{su4}	Setup time, A2–A0 valid before $\overline{IOW}\downarrow$ (see Note 2)	15		ns
t_{su5}	Setup time, D7–D0 valid before $\overline{IOW}\uparrow$	10		ns
t_{h3}	Hold time, A2–A0 valid after $\overline{IOW}\uparrow$ (see Note 2)	5		ns
t_{h4}	Hold time, $\overline{CSx}$ valid after $\overline{IOW}\uparrow$ (see Note 2)	5		ns
t_{h5}	Hold time, D7–D0 valid after $\overline{IOW}\uparrow$	25		ns
t_{d3}	Delay time, $t_{su4} + t_{w5} + t_{d4}$	120		ns
t_{d4}	Delay time, $\overline{IOW}\uparrow$ to $\overline{IOW}$ or $\overline{IOR}\downarrow$	55		ns

NOTE 2: The internal address strobe is always active.

read cycle switching characteristics over recommended ranges of operating free-air temperatu and supply voltage, C_L = 100 pF (see Note 4 and Figure 4)

PARAMETER		MIN	MAX	UNIT
t_{en}	Enable time, $\overline{IOR}\downarrow$ to D7–D0 valid		30	ns
t_{dis}	Disable time, $\overline{IOR}\uparrow$ to D7–D0 released	0	20	ns

NOTE 4: V_{OL} and V_{OH} (and the external loading) determine the charge and discharge time.

transmitter switching characteristics over recommended ranges of operating free-air temperature and supply voltage (see Figures 6, 7, and 8)

PARAMETER		TEST CONDITIONS	MIN	MAX	UNIT
t_{d5}	Delay time, INTx↓ to TXx↓ at start		8	24	RCLK cycles
t_{d6}	Delay time, TXx↓ at start to INTx↑	See Note 5	8	8	RCLK cycles
t_{d7}	Delay time, $\overline{IOW}$ high or low (WR THR) to INTx↑	See Note 5	16	32	RCLK cycles
t_{d8}	Delay time, TXx↓ at start to $\overline{TXRDY}$↓	C_L = 100 pF		8	RCLK cycles
t_{pd1}	Propagation delay time, $\overline{IOW}$ (WR THR)↓ to INTx↓	C_L = 100 pF		35	ns
t_{pd2}	Propagation delay time, $\overline{IOR}$ (RD IIR)↑ to INTx↓	C_L = 100 pF		30	ns
t_{pd3}	Propagation delay time, $\overline{IOW}$ (WR THR)↑ to $\overline{TXRDY}$↑	C_L = 100 pF		50	ns

NOTE 5: If the transmitter interrupt delay is active, this delay is lengthened by one character time minus the last stop bit time.

receiver switching characteristics over recommended ranges of operating free-air temperature and supply voltage (see Figures 9 through 13)

PARAMETER		TEST CONDITIONS	MIN	MAX	UNIT
t_{d9}	Delay time, stop bit to INTx↑ or stop bit to $\overline{RXRDY}$↓ or read RBR to set interrupt	See Note 6		1	RCLK cycle
t_{pd4}	Propagation delay time, Read RBR/LSR to INTx↓/LSR interrupt↓	C_L = 100 pF, See Note 7		40	ns
t_{pd5}	Propagation delay time, $\overline{IOR}$ RCLK↓ to $\overline{RXRDY}$↑	See Note 7		30	ns

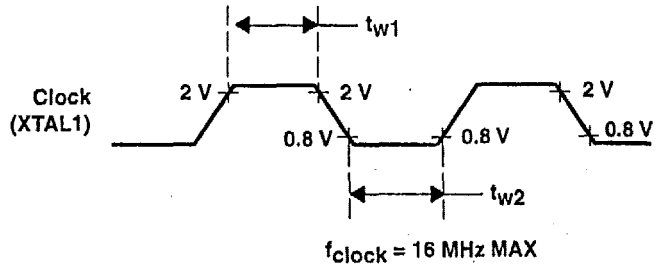
NOTES: 6. The receiver data available indicator, the overrun error indicator, the trigger level interrupts, and the active $\overline{RXRDY}$ indicator are delayed three RCLK (internal receiver timing clock) cycles in the FIFO mode (FCR0 = 1). After the first byte has been received, status indicators (PE, FE, BI) are delayed three RCLK cycles. These indicators are updated immediately for any further bytes received after $\overline{IOR}$ goes active for a read from the RBR register. There are eight RCLK cycle delays for trigger change level interrupts.

7. RCLK is an internal signal derived from divisor latch LSB (DLL) and divisor latch MSB (DLM) divisor latches.

modem control switching characteristics over recommended ranges of operating free-air temperature and supply voltage, C_L = 100 pF (see Figure 14)

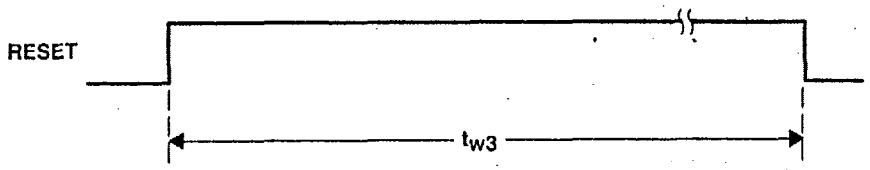
PARAMETER		MIN	MAX	UNIT
t_{pd6}	Propagation delay time, $\overline{IOW}$ (WR MCR)↑ to $\overline{RTSx}$, $\overline{DTRx}$↑		50	ns
t_{pd7}	Propagation delay time, modem input $\overline{CTSx}$, $\overline{DSRx}$, and $\overline{DCDx}$ ↓↑ to INTx↑		30	ns
t_{pd8}	Propagation delay time, $\overline{IOR}$ (RD MSR)↑ to interrupt↓		35	ns
t_{pd9}	Propagation delay time, $\overline{RIx}$↑ to $\overline{INTx}$↑		30	ns

PARAMETER MEASUREMENT INFORMATION



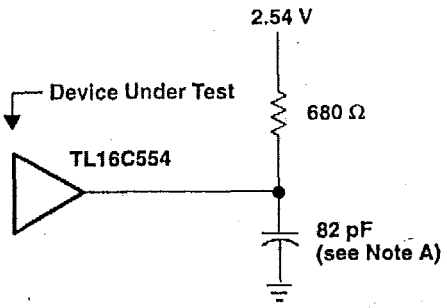
f_{clock} = 16 MHz MAX

(a) CLOCK INPUT VOLTAGE WAVEFORM



(b) RESET VOLTAGE WAVEFORM

Figure 1. Clock Input and RESET Voltage Waveforms



NOTE A: This includes scope and jig capacitance.

Figure 2. Output Load Circuit

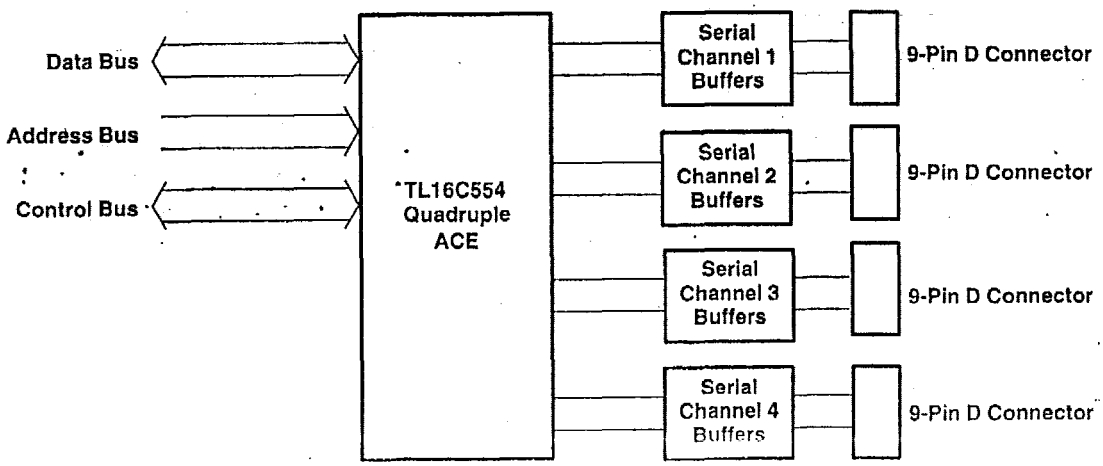


Figure 3. Basic Test Configuration

PARAMETER MEASUREMENT INFORMATION

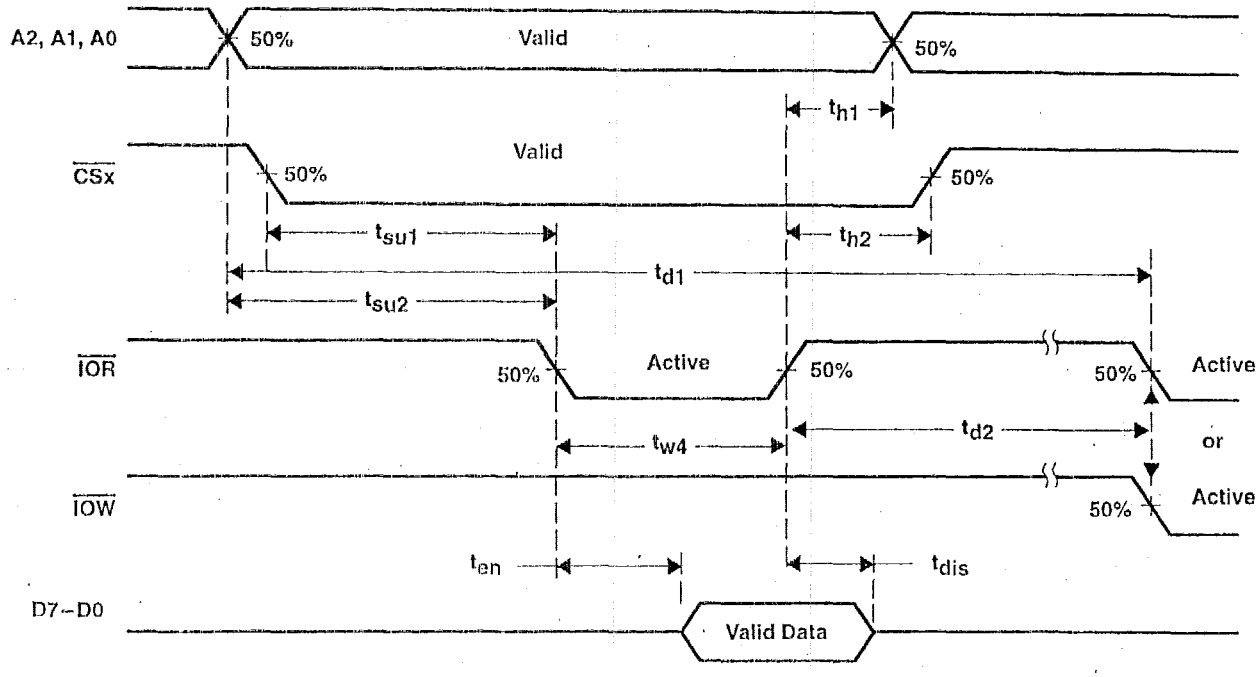


Figure 4. Read Cycle Timing Waveforms

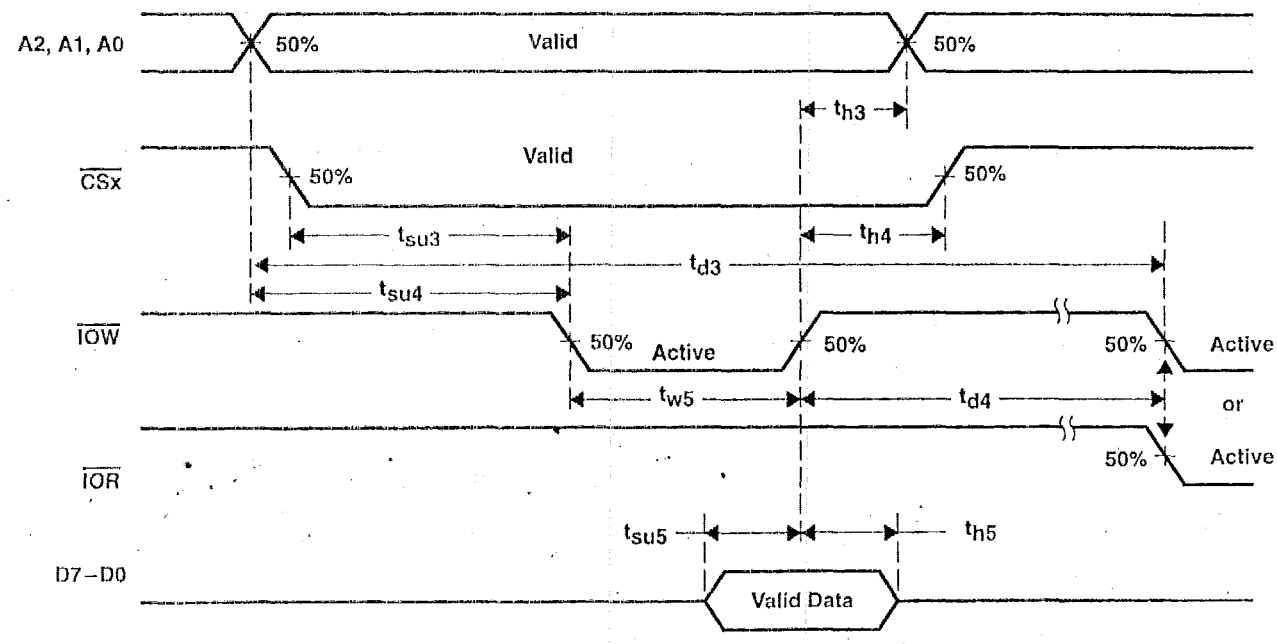


Figure 5. Write Cycle Timing Waveforms

PARAMETER MEASUREMENT INFORMATION

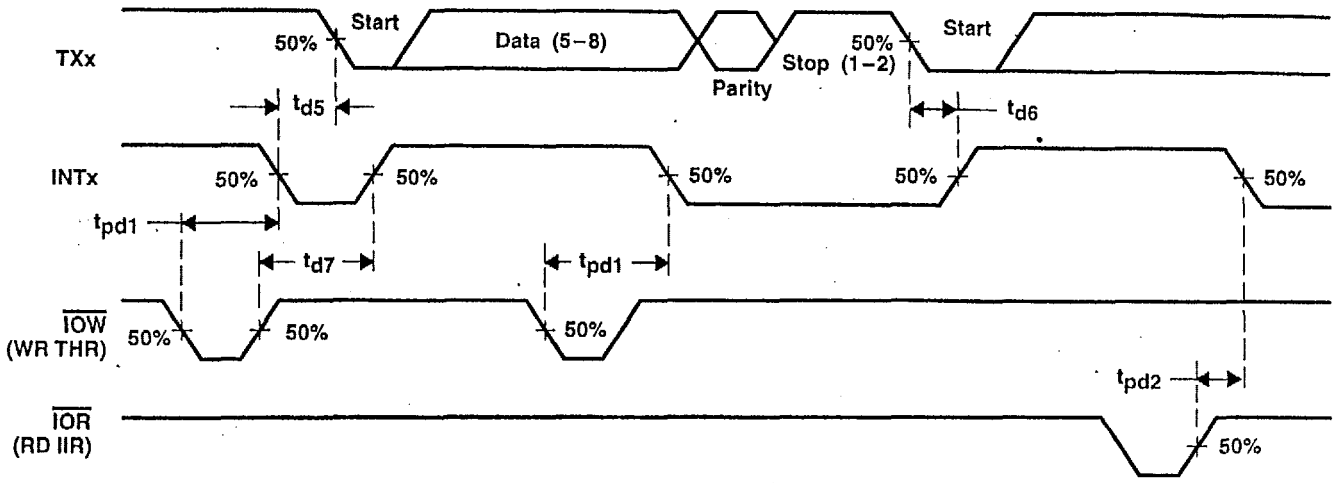


Figure 6. Transmitter Timing Waveforms

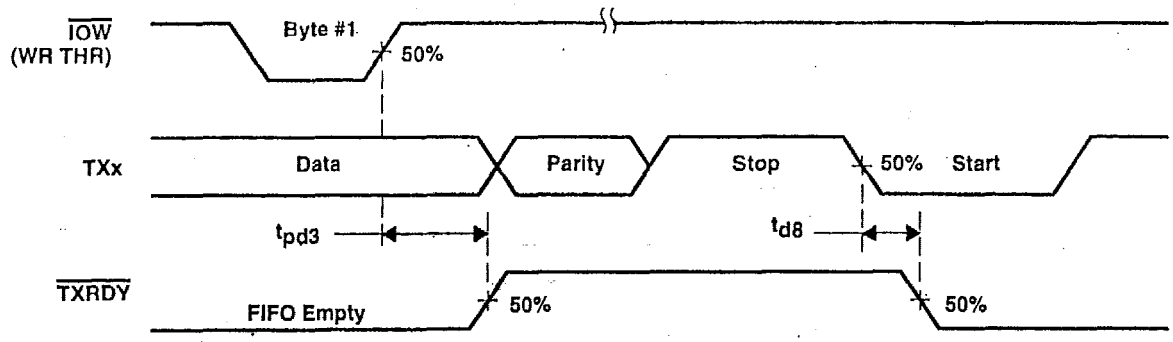


Figure 7. Transmitter Ready Mode 0 Timing Waveforms

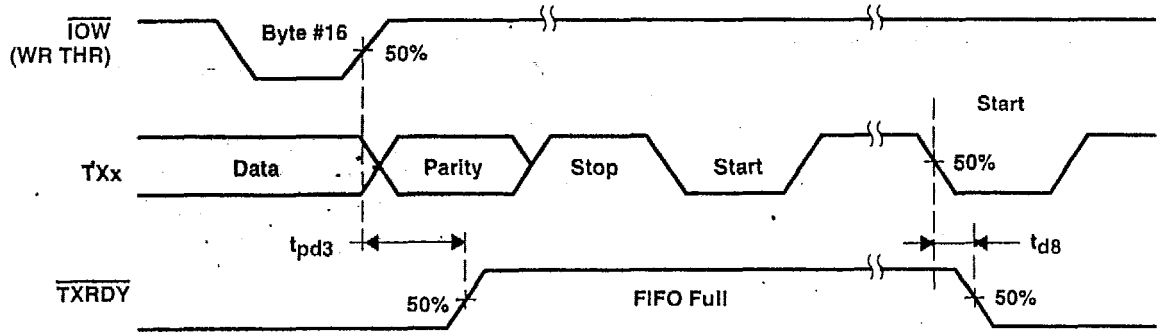


Figure 8. Transmitter Ready Mode 1 Timing Waveforms

PARAMETER MEASUREMENT INFORMATION

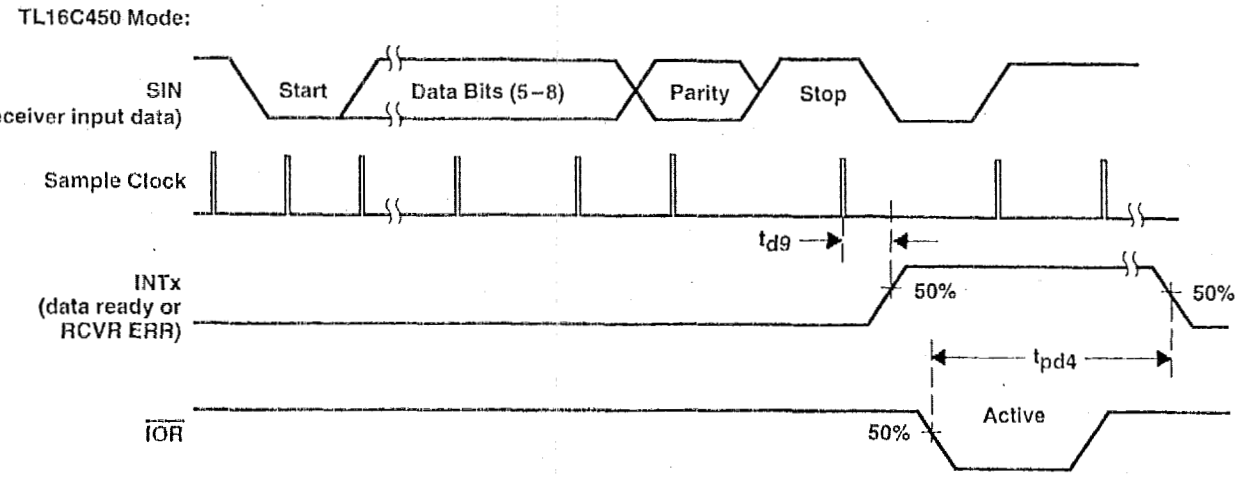


Figure 9. Receiver Timing Waveforms

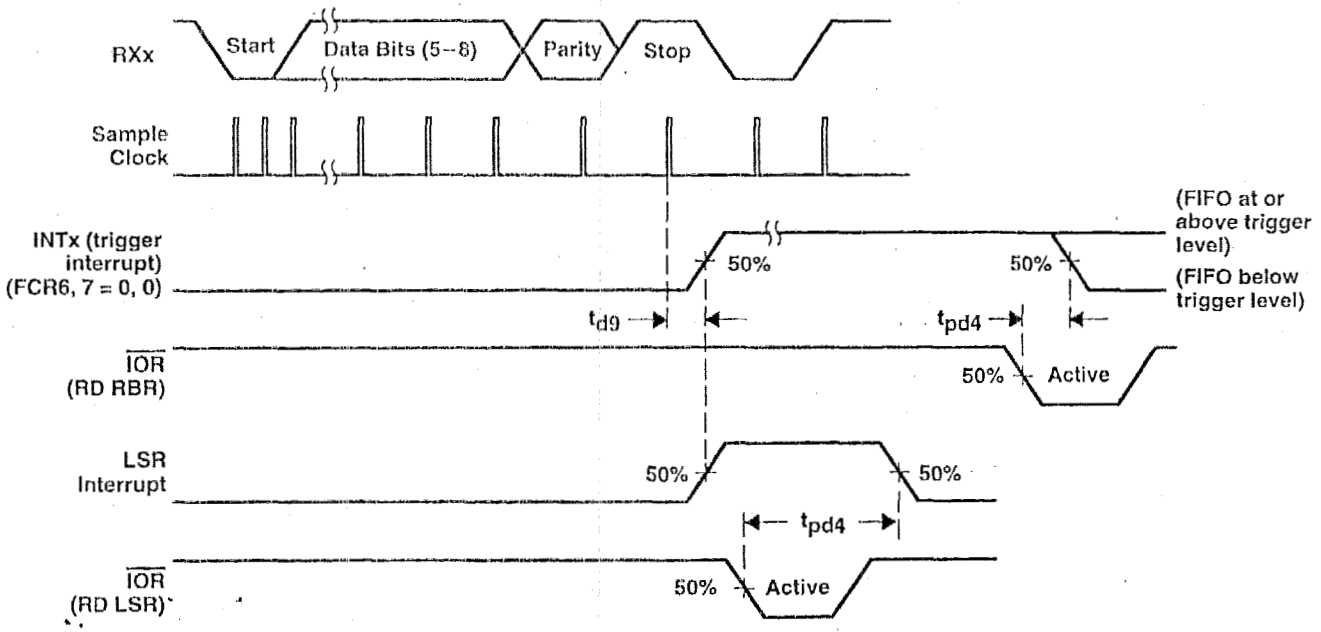
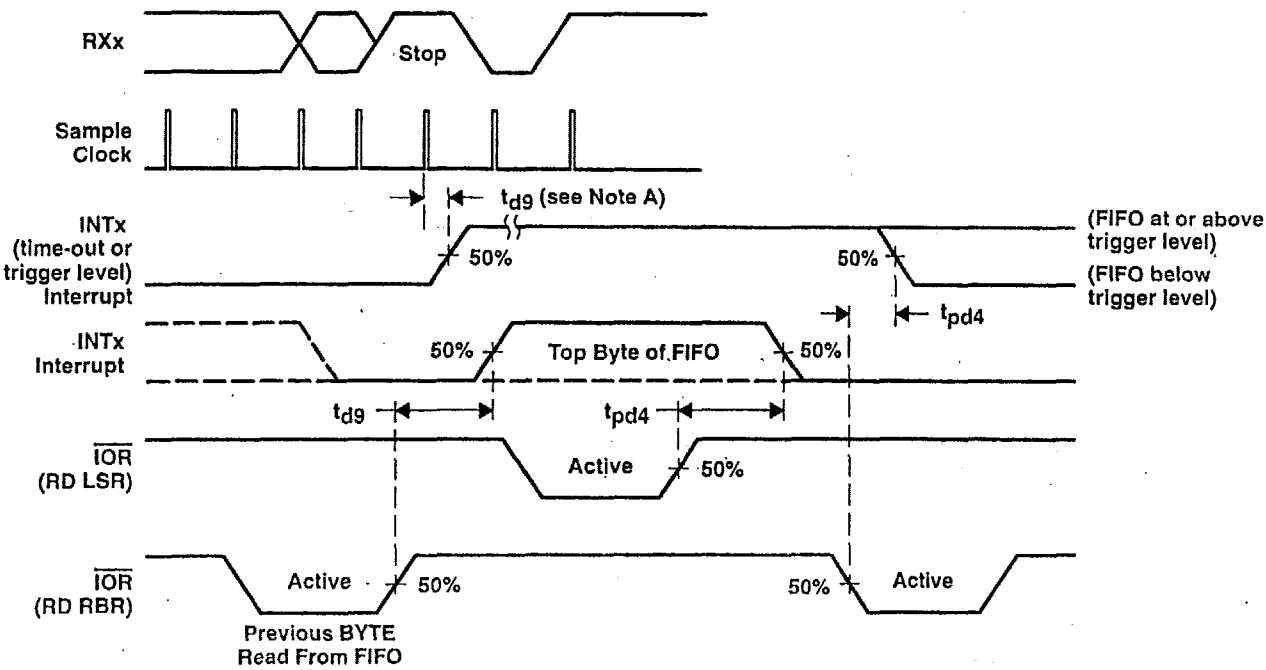


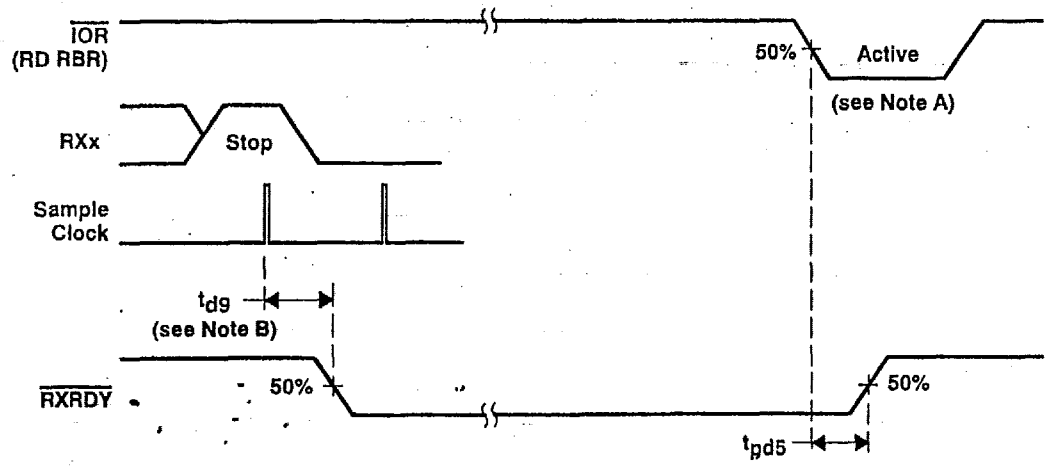
Figure 10. Receiver FIFO First Byte (Sets RDR) Waveforms

PARAMETER MEASUREMENT INFORMATION



NOTE A: This is the reading of the last byte in the FIFO.

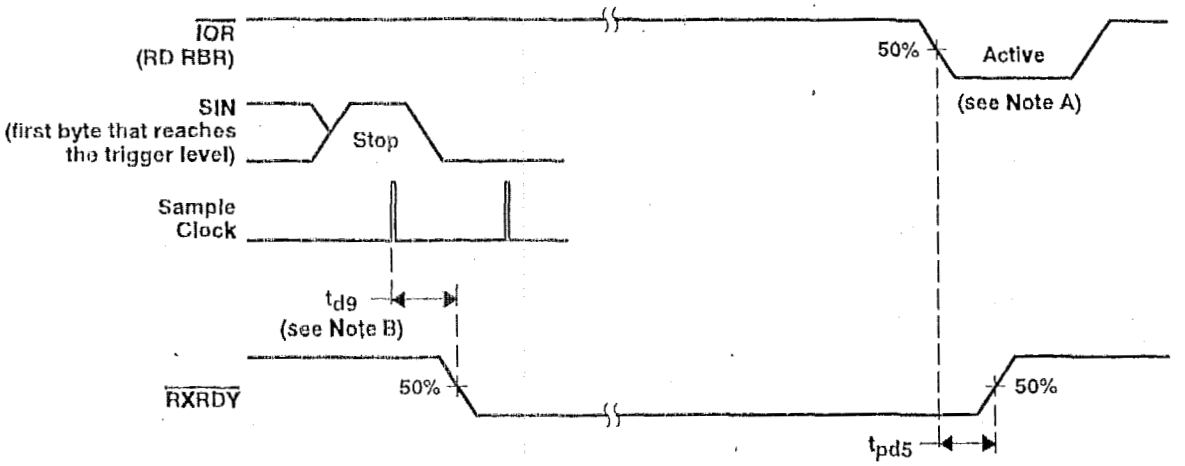
Figure 11. Receiver FIFO After First Byte (After RDR Set) Waveforms



NOTES: A. This is the reading of the last byte in the FIFO.
B. If FCR0 = 1, then t_{d9} = 3 RCLK cycles. For a time-out interrupt, t_{d9} = 8 RCLK cycles.

Figure 12. Receiver Ready Mode 0 Timing Waveforms

PARAMETER MEASUREMENT INFORMATION



NOTES: A. This is the reading of the last byte in the FIFO.
B. If FCR0 = 1, t_{d9} = 3 RCLK cycles. For a trigger change level interrupt, t_{d9} = 8 RCLK.

Figure 13. Receiver Ready Mode 1 Timing Waveforms

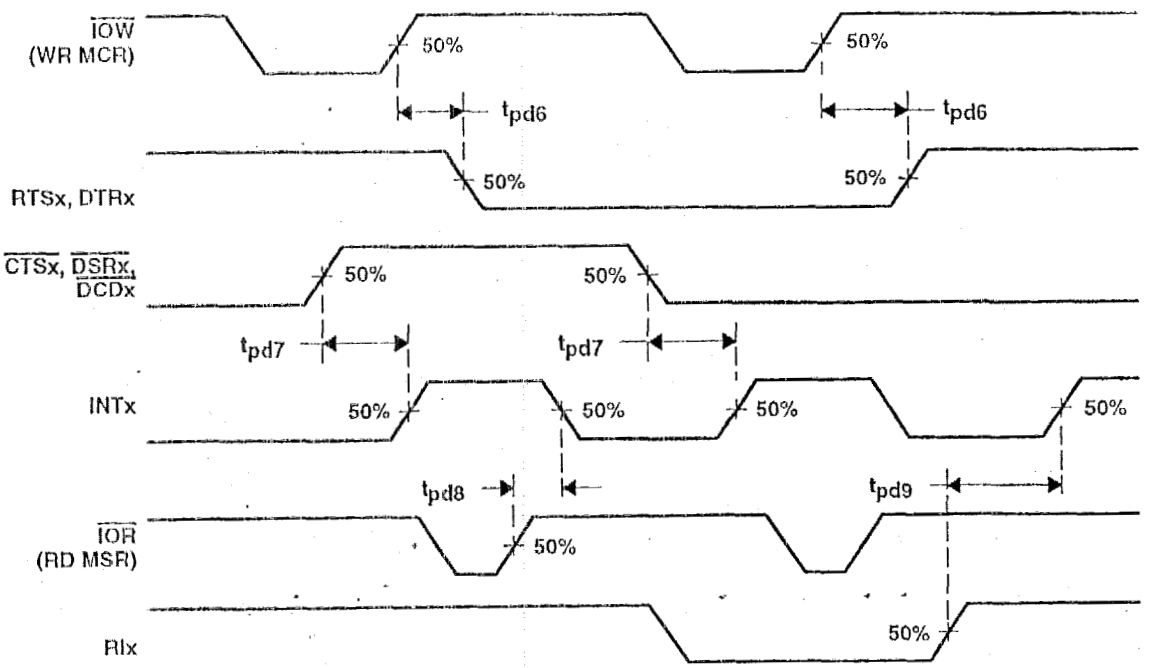


Figure 14. Modem Control Timing Waveforms

PRINCIPLES OF OPERATION

Three types of information are stored in the internal registers used in the ACE: control, status, and data. Mnemonic abbreviations for the registers are shown in Table 1. Table 2 defines the address location of each register and whether it is read only, write only, or read writable.

Table 1. Internal Register Mnemonic Abbreviations

CONTROL	MNEMONIC	STATUS	MNEMONIC	DATA	MNEMONIC
Line control register	LCR	Line status register	LSR	Receiver buffer register	RBR
FIFO control register	FCR	Modem status register	MSR	Transmitter holding register	THR
Modem control register	MCR				
Divisor latch LSB	DLL				
Divisor latch MSB	DLM				
Interrupt enable register	IER				

Table 2. Register Selection†

DLAB‡	A2§	A1§	A0§	READ MODE	WRITE MODE
0	0	0	0	Receiver buffer register	Transmitter holding register
0	0	0	1		Interrupt enable register
X	0	1	0	Interrupt identification register	FIFO control register
X	0	1	1		Line control register
X	1	0	0		Modem control register
X	1	0	1	Line status register	
X	1	1	0	Modem status register	
X	1	1	1	Scratchpad register	Scratchpad register
1	0	0	0		LSB divisor latch
1	0	0	1		MSB divisor latch

X = irrelevant, 0 = low level, 1 = high level
† The serial channel is accessed when either $\overline{CSA}$ or $\overline{CSD}$ is low.
‡ DLAB is the divisor latch access bit and bit 7 in the LCR.
§ A2–A0 are device terminals.

Individual bits within the registers with the bit number in parenthesis are referred to by the register mnemonic. For example, LCR7 refers to line control register bit 7. The transmitter buffer register and receiver buffer register are data registers that hold from five to eight bits of data. If less than eight data bits are transmitted, data is right justified to the LSB. Bit 0 of a data word is always the first serial data bit received and transmitted. The ACE data registers are double buffered (TL16450 mode) or FIFO buffered (FIFO mode) so that read and write operations can be performed when the ACE is performing the parallel-to-serial or serial-to-parallel conversion.

PRINCIPLES OF OPERATION

accessible registers

The system programmer, using the CPU, has access to and control over any of the ACE registers that are summarized in Table 1. These registers control ACE operations, receive data, and transmit data. Descriptions of these registers follow Table 3.

Table 3. Summary of Accessible Registers

ADDRESS	REGISTER MNEMONIC	REGISTER ADDRESS							
		BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
0	RBR (read only)	Data Bit 7 (MSB)	Data Bit 6	Data Bit 5	Data Bit 4	Data Bit 3	Data Bit 2	Data Bit 1	Data Bit 0 (LSB)
0	THR (write only)	Data Bit 7	Data Bit 6	Data Bit 5	Data Bit 4	Data Bit 3	Data Bit 2	Data Bit 1	Data Bit 0
0†	DLL	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
1†	DLM	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
1	IER	0	0	0	0	(EDSSI) Enable modem status interrupt	(ERLSI) Enable receiver line status interrupt	(ETBEI) Enable transmitter holding register empty interrupt	(ERBI) Enable received data available interrupt
2	FCR (write only)	Receiver Trigger (MSB)	Receiver Trigger (LSB)	Reserved	Reserved	DMA mode select	Transmit FIFO reset	Receiver FIFO reset	FIFO Enable
2	IIR (read only)	FIFOs Enabled‡	FIFOs Enabled‡	0	0	Interrupt ID Bit (3)‡	Interrupt ID Bit (2)	Interrupt ID Bit (1)	0 If interrupt pending
3	LCR	(DLAB) Divisor latch access bit	Set break	Stick parity	(EPS) Even parity select	(PEN) Parity enable	(STB) Number of stop bits	(WLSB1) Word length select bit 1	(WLSB0) Word length select bit 0
4	MCR	0	0	0	Loop	OUT2 Enable external interrupt (INT)	Reserved	(RTS) Request to send	(DTR) Data terminal ready
5	LSR	Error in receiver FIFO‡	(TEMT) Transmitter registers empty	(THRE) Transmitter holding register empty	(BI) Break interrupt	(FE) Framing error	(PE) Parity error	(OE) Overrun error	(DR) Data ready
6	MSR	(DCD) Data carrier detect	(RI) Ring indicator	(DSR) Data set ready	(CTS) Clear to send	(ΔDCD) Delta data carrier detect	(TERI) Trailing edge ring indicator	(ΔDSR) Delta data set ready	(ΔCTS) Delta clear to send
7	SCR	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

† DLAB = 1

‡ These bits are always 0 when FIFOs are disabled.

PRINCIPLES OF OPERATION

FIFO control register (FCR)

The FCR is a write-only register at the same location as the IIR. It enables the FIFOs, sets the trigger level of the receiver FIFO, and selects the type of DMA signalling.

- Bit 0: FCR0 enables the transmit and receiver FIFOs. All bytes in both FIFOs can be cleared by clearing FCR0. Data is cleared automatically from the FIFOs when changing from the FIFO mode to the TL16C450 mode (see FCR bit 0) and vice versa. Programming of other FCR bits is enabled by setting FCR0.
- Bit 1: When set, FCR1 clears all bytes in the receiver FIFO and resets its counter. This does not clear the shift register.
- Bit 2: When set, FCR2 clears all bytes in the transmit FIFO and resets the counter. This does not clear the shift register.
- Bit 3: When set, FCR3 changes $\overline{\text{RXRDY}}$ and $\overline{\text{TXRDY}}$ from mode 0 to mode 1 if FCR0 is set.
- Bits 4 and 5: FCR4 and FCR5 are reserved for future use.
- Bits 6 and 7: FCR6 and FCR7 set the trigger level for the receiver FIFO interrupt (see Table 4).

Table 4. Receiver FIFO Trigger Level

BIT		RECEIVER FIFO TRIGGER LEVEL (BYTES)
7	6	
0	0	01
0	1	04
1	0	08
1	1	14

FIFO interrupt mode operation

The following receiver status occurs when the receiver FIFO and receiver interrupts are enabled.

1. LSR0 is set when a character is transferred from the shift register to the receiver FIFO. When the FIFO is empty, it is reset.
2. IIR = 06 receiver line status interrupt has higher priority than the receive data available interrupt IIR = 04.
3. Receive data available interrupt is issued to the CPU when the programmed trigger level is reached by the FIFO. As soon as the FIFO drops below its programmed trigger level, it is cleared.
4. IIR = 04 (receive data available indicator) also occurs when the FIFO reaches its trigger level. It is cleared when the FIFO drops below the programmed trigger level.

The following receiver FIFO character time-out status occurs when receiver FIFO and receiver interrupts are enabled.

PRINCIPLES OF OPERATION

FIFO interrupt mode operation (continued)

1. When the following conditions exist, a FIFO character time-out interrupt occurs:
   a. Minimum of one character in FIFO
   b. Last received serial character is longer than four continuous previous character times ago. (If two stop bits are programmed, the second one is included in the time delay.)
   c. The last CPU of the FIFO read is more than four continuous character times earlier. At 300 baud and 12-bit characters, the FIFO time-out interrupt causes a latency of 160 ms maximum from received character to interrupt issued.
2. By using the XTAL1 input for a clock signal, the character times can be calculated. The delay is proportional to the baud rate.
3. The time-out timer is reset after the CPU reads the receiver FIFO or after a new character is received. This occurs when there has been no time-out interrupt.
4. A time-out interrupt is cleared and the timer is reset when the CPU reads a character from the receiver FIFO.

Transmit interrupts occurs as follows when the transmitter and transmit FIFO interrupts are enabled (FCR0 = 1, IER = 1).

1. When the transmitter FIFO is empty, the transmitter holding register interrupt (IIR = 02) occurs. The interrupt is cleared when the transmitter holding register is written to or the IIR is read. One to sixteen characters can be written to the transmit FIFO when servicing this interrupt.
2. The transmitter FIFO empty indicators are delayed one character time minus the last stop bit time whenever the following occurs:

   THRE = 1, and there has not been a minimum of two bytes at the same time in transmit FIFO since the last THRE = 1. The first transmitter interrupt after changing FCR0 is immediate, however, assuming it is enabled.

Receiver FIFO trigger level and character time-out interrupts have the same priority as the receive data available interrupt. The transmitter holding register empty interrupt has the same priority as the transmitter FIFO empty interrupt.

FIFO polled mode operation

Clearing IER0, IER1, IER2, IER3, or all to zero with FCR0 = 1 puts the ACE into the FIFO polled mode. receiver and transmitter are controlled separately. Either or both can be in the polled mode.

In the FIFO polled mode, there is no time-out condition indicated or trigger level reached. However, the Receiver and transmit FIFOs still have the capability of holding characters. The LSR must be read to determine the ACE status.

interrupt enable register (IER)

The IER independently enables the four serial channel interrupt sources that activate the interrupt (INTA, B, C, D) output. All interrupts are disabled by clearing IER0 – IER3 of the IER. Interrupts are enabled by setting the appropriate bits of the IER. Disabling the interrupt system inhibits the IIR and the active (high) interrupt output. All other system functions operate in their normal manner, including the setting of the LSR and MSR. The contents of the IER are shown in Table 3 and described in the following bulleted list:

PRINCIPLES OF OPERATION

interrupt enable register (IER) (continued)

- Bit 0: When IER0 is set, IER0 enables the received data available interrupt and the timeout interrupts in the FIFO mode.
- Bit 1: When IER1 is set, the transmitter holding register empty interrupt is enabled.
- Bit 2: When IER2 is set, the receiver line status interrupt is enabled.
- Bit 3: When IER3 is set, the modem status interrupt is enabled.
- Bits 4 – 7: IER4 – IER7. These four bits of the IER are cleared.

interrupt identification register (IIR)

In order to minimize software overhead during data character transfers, the serial channel prioritizes interrupts into four levels. The four levels of interrupt conditions are as follows:

- Priority 1 – Receiver line status (highest priority)
- Priority 2 – Receiver data ready or receiver character timeout
- Priority 3 – Transmitter holding register empty
- Priority 4 – Modem status (lowest priority)

Information indicating that a prioritized interrupt is pending and the type of interrupt that is stored in the IIR. The IIR indicates the highest priority interrupt pending. The contents of the IIR are indicated in Table 5.

Table 5. Interrupt Control Functions

INTERRUPT IDENTIFICATION REGISTER				INTERRUPT SET AND RESET FUNCTIONS			
BIT 3	BIT 2	BIT 1	BIT 0	PRIORITY LEVEL	INTERRUPT TYPE	INTERRUPT SOURCE	INTERRUPT RESET CONTROL
0	0	0	1	—	None	None	—
0	1	1	0	First	Receiver line status	OE, PE, FE, or BI	LSR read
0	1	0	0	Second	Received data available	Receiver data available or trigger level reached	RBR read until FIFO drops below the trigger level
1	1	0	0	Second	Character time-out indicator	No characters have been removed from or input to the receiver FIFO during the last four character times, and there is at least one character in it during this time.	RBR read
0	0	1	0	Third	THRE	THRE	IIR read if THRE is the interrupt source or THR write
0	0	0	0	Fourth	Modem status	$\overline{CTS}$, $\overline{DSR}$, $\overline{RI}$, or $\overline{DCD}$	

PRINCIPLES OF OPERATION

interrupt identification register (IIR) (continued)

- Bit 0: IIR0 indicates whether an interrupt is pending. When IIR0 is cleared, an interrupt is pending.
- Bits 1 and 2: IIR1 and IIR2 identify the highest priority interrupt pending as indicated in Table 5.
- Bit 3: IIR3 is always cleared when in the TL16C450 mode. This bit is set along with bit 2 when in the FIFO mode and a trigger change level interrupt is pending.
- Bits 4 and 5: IIR4 and IIR5 are always cleared.
- Bits 6 and 7: IIR6 and IIR7 are set when FCR0 = 1.

line control register (LCR)

The format of the data character is controlled by the LCR. The LCR may be read. Its contents are described in the following bulleted list and shown in Figure 15.

- Bits 0 and 1: LCR0 and LCR1 are word length select bits. These bits program the number of bits in each serial character and are shown in Figure 15.
- Bit 2: LCR2 is the stop bit select bit. This bit specifies the number of stop bits in each transmitted character. The receiver always checks for one stop bit.
- Bit 3: LCR3 is the parity enable bit. When LCR3 is set, a parity bit between the last data word bit and stop bit is generated and checked.
- Bit 4: LCR4 is the even parity select bit. When this bit is set and parity is enabled (LCR3 is set), even parity is selected. When this bit is cleared and parity is enabled, odd parity is selected.
- Bit 5: LCR5 is the stick parity bit. When parity is enabled (LCR3 is set) and this bit is set, the transmission and reception of a parity bit is placed in the opposite state from the value of LCR4. This forces parity to a known state and allows the receiver to check the parity bit in a known state.
- Bit 6: LCR6 is a break control bit. When this bit is set, the serial outputs TXx are forced to the spacing state (low). The break control bit acts only on the serial output and does not affect the transmitter logic. If the following sequence is used, no invalid characters are transmitted because of the break.

  Step 1. Load a zero byte in response to the transmitter holding register empty (THRE) status indicator.

  Step 2. Set the break in response to the next THRE status indicator.

  Step 3. Wait for the transmitter to be idle when transmitter empty status signal is set (TEMT = 1); then clear the break when the normal transmission has to be restored.
- Bit 7: LCR7 is the divisor latch access bit (DLAB) bit. This bit must be set to access the divisor latches DLL and DLM of the baud rate generator during a read or write operation. LCR7 must be cleared to access the receiver buffer register, the transmitter holding register, or the interrupt enable register.

PRINCIPLES OF OPERATION

line control register (LCR) (continued)

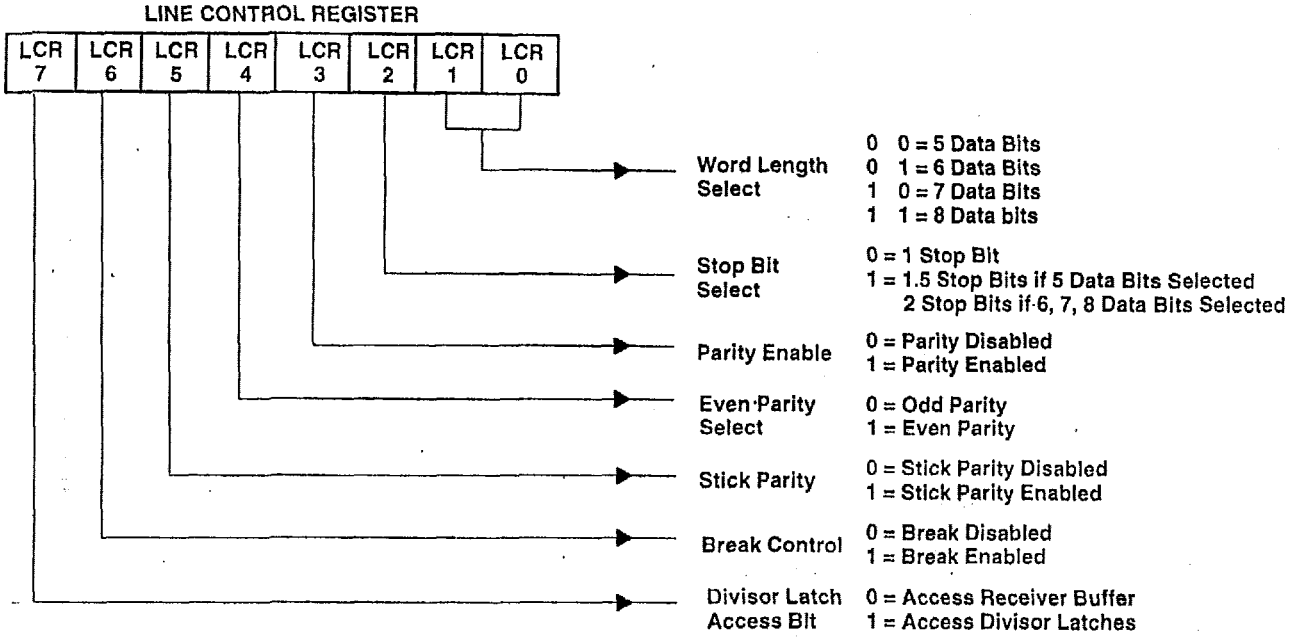


Figure 15. Line Control Register Contents

line status register (LSR)

The LSR is a single register that provides status indicators. The LSR shown in Table 6 is described in the following bulleted list:

- Bit 0: LSR0 is the data ready (DR) bit. Data ready is set when an incoming character is received and transferred into the receiver buffer register or the FIFO. LSR0 is cleared by a CPU read of the data in the receiver buffer register or the FIFO.
- Bit 1: LSR1 is the overrun error (OE) bit. An overrun error indicates that data in the receiver buffer register is not read by the CPU before the next character is transferred into the receiver buffer register overwriting the previous character. The OE indicator is cleared whenever the CPU reads the contents of the LSR. An overrun error occurs in the FIFO mode after the FIFO is full and the next character is completely received. The overrun error is detected by the CPU on the first LSR read after it happens. The character in the shift register is not transferred to the FIFO, but it is overwritten.
- Bit 2: LSR2 is the parity error (PE) bit. A parity error indicates that the received data character does not have the correct parity as selected by LCR3 and LCR4. The PE bit is set upon detection of a parity error and is cleared when the CPU reads the contents of the LSR. In the FIFO mode, the parity error is associated with a particular character in the FIFO. LSR2 reflects the error when the character is at the top of the FIFO.
- Bit 3: LSR3 is the framing error (FE) bit. A framing error indicates that the received character does not have a valid stop bit. LSR3 is set when the stop bit following the last data bit or parity bit is detected as a zero bit (spacing level). The FE indicator is cleared when the CPU reads the contents of the LSR. In the FIFO mode, the framing error is associated with a particular character in the FIFO. LSR3 reflects the error when the character is at the top of the FIFO.

PRINCIPLES OF OPERATION

line status register (LSR) (continued)

- Bit 4: LSR4 is the break interrupt (BI) bit. Break interrupt is set when the received data input is held in the spacing (low) state for longer than a full word transmission time (start bit + data bits + parity + stop bits). The BI indicator is cleared when the CPU reads the contents of the LSR. In the FIFO mode, this is associated with a particular character in the FIFO. LSR2 reflects the BI when the break character is at the top of the FIFO. The error is detected by the CPU when its associated character is at the top of the FIFO during the first LSR read. Only one zero character is loaded into the FIFO when BI occurs.

LSR1 – LSR4 are the error conditions that produce a receiver line status interrupt (priority 1 interrupt in the interrupt identification register) when any of the conditions are detected. This interrupt is enabled by setting IER2 in the interrupt enable register.

- Bit 5: LSR5 is the transmitter holding register empty (THRE) bit. THRE indicates that the ACE is ready to accept a new character for transmission. The THRE bit is set when a character is transferred from the transmitter holding register (THR) into the transmitter shift register (TSR). LSR5 is cleared by the loading of the THR by the CPU. LSR5 is not cleared by a CPU read of the LSR. In the FIFO mode, when the transmit FIFO is empty, this bit is set. It is cleared when one byte is written to the transmit FIFO. When the THRE interrupt is enabled by IER1, THRE causes a priority 3 interrupt in the IIR. If THRE is the interrupt source indicated in IIR, INTRPT is cleared by a read of the IIR.
- Bit 6: LSR6 is the transmitter register empty (TEMT) bit. TEMT is set when the THR and the TSR are both empty. LSR6 is cleared when a character is loaded into THR and remains low until the character is transferred out of TXx. TEMT is not cleared by a CPU read of the LSR. In the FIFO mode, when both the transmitter FIFO and shift register are empty, this bit is set.
- Bit 7: LSR7 is the receiver FIFO error bit. The LSR7 bit is cleared in the TL16C450 mode (see FCR bit 0). In the FIFO mode, it is set when at least one of the following data errors is in the FIFO: parity error, framing error, or break interrupt indicator. It is cleared when the CPU reads the LSR if there are no subsequent errors in the FIFO.

NOTE

The LSR may be written. However, this function is intended only for factory test. It should be considered as read only by applications software.

Table 6. Line Status Register Bits

LSR BITS	1	0
LSR0 data ready (DR)	Ready	Not ready
LSR1 overrun error (OE)	Error	No error
LSR2 parity error (PE)	Error	No error
LSR3 framing error (FE)	Error	No error
LSR4 break interrupt (BI)	Break	No break
LSR5 transmitter holding register empty (THRE)	Empty	Not empty
LSR6 transmitter register empty (TEMT)	Empty	Not empty
LSR7 receiver FIFO error	Error in FIFO	No error in FIFO

modem control register (MCR)

The MCR controls the interface with the modem or data set as described in Figure 16. MCR can be written and read. The $\overline{RTS}$ and $\overline{DTR}$ outputs are directly controlled by their control bits in this register. A high input asserts a low signal (active) at the output terminals. MCR bits 0, 1, 2, 3, and 4 are shown as follows:

- Bit 0: When MCR0 is set, the $\overline{DTR}$ output is forced low. When MCR0 is cleared, the $\overline{DTR}$ output is forced high. The $\overline{DTR}$ output of the serial channel may be input into an inverting line driver in order to obtain the proper polarity input at the modem or data set.
- Bit1: When MCR1 is set, the $\overline{RTS}$ output is forced low. When MCR1 is cleared, the $\overline{RTS}$ output is forced high. The $\overline{RTS}$ output of the serial channel may be input into an inverting line driver to obtain the proper polarity input at the modem or data set.
- Bit 2: MCR2 has no affect on operation.
- Bit 3: When MCR3 is set, the external serial channel interrupt is enabled.
- Bit 4: MCR4 provides a local loopback feature for diagnostic testing of the channel. When MCR4 is set, serial output TXx is set to the marking (high) state and SIN is disconnected. The output of the TSR is looped back into the RSR input. The four modem control inputs ($\overline{CTS}$, $\overline{DSR}$, $\overline{DCD}$, and $\overline{RI}$) are disconnected. The modem control outputs ($\overline{DTR}$ and $\overline{RTS}$) are internally connected to the four modem control inputs. The modem control output terminals are forced to their inactive (high) state on the TL16C554. In the diagnostic mode, data transmitted is immediately received. This allows the processor to verify the transmit and receive data paths of the selected serial channel. Interrupt control is fully operational; however, interrupts are generated by controlling the lower four MCR bits internally. Interrupts are not generated by activity on the external terminals represented by those four bits.
- Bit 5 – Bit 7: MCR5, MCR6, and MCR7 are permanently cleared.

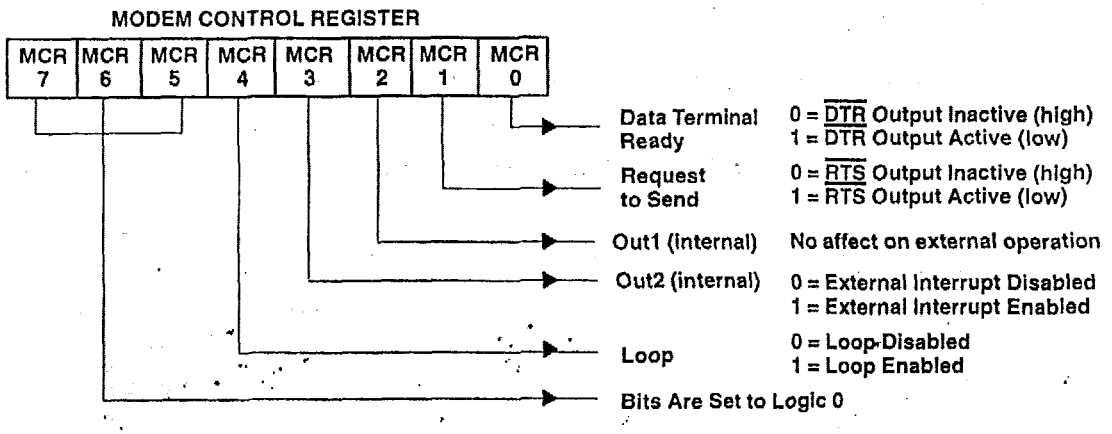


Figure 16. Modem Control Register Contents

PRINCIPLES OF OPERATION

modem status register (MSR)

The MSR provides the CPU with status of the modem input lines for the modem or peripheral devices. The MSR allows the CPU to read the serial channel modem signal inputs by accessing the data bus interface of the ACE. It also reads the current status of four bits of the MSR that indicate whether the modem inputs have changed since the last reading of the MSR. The delta status bits are set when a control input from the modem changes states and are cleared when the CPU reads the MSR.

The modem input lines are $\overline{CTS}$, DSR, and $\overline{DCD}$. MSR4 – MSR7 are status indicators of these lines. A status bit = 1 indicates the input is low. When the status bit is cleared, the input is high. When the modem status interrupt in the IER is enabled (IIR3 is set), an interrupt is generated whenever MSR0 – MSR3 is set. The MSR is a priority 4 interrupt. The contents of the MSR are described in Table 7.

- Bit 0: MSR0 is the delta clear-to-send (ΔCTS) bit. DCTS indicates that the $\overline{CTS}$ input to the serial channel has changed state since it was last read by the CPU.
- Bit 1: MSR1 is the delta data set ready (ΔDSR) bit. ΔDSR indicates that the $\overline{DSR}$ input to the serial channel has changed states since the last time it was read by the CPU.
- Bit 2: MSR2 is the trailing edge of ring indicator (TERI) bit. TERI indicates that the $\overline{RIx}$ input to the serial channel has changed states from low to high since the last time it was read by the CPU. High-to-low transitions on RI do not activate TERI.
- Bit 3: MSR3 is the delta data carrier detect (ΔDCD) bit. ΔDCD indicates that the $\overline{DCD}$ input to the serial channel has changed states since the last time it was read by the CPU.
- Bit 4: MSR4 is the clear-to-send (CTS) bit. CTS is the complement of the $\overline{CTS}$ input from the modem indicating to the serial channel that the modem is ready to receive data from SOUT. When the serial channel is in the loop mode (MCR4 = 1), MSR4 reflects the value of RTS in the MCR.
- Bit 5: MSR5 is the data set ready DSR bit. DSR is the complement of the $\overline{DSR}$ input from the modem to the serial channel that indicates that the modem is ready to provide received data from the serial channel receiver circuitry. When the channel is in the loop mode (MCR4 is set), MSR5 reflects the value of DTR in the MCR.
- Bit 6: MSR6 is the ring indicator (RI) bit. RI is the complement of the $\overline{RIx}$ inputs. When the channel is in the loop mode (MCR4 is set), MSR6 reflects the value of $\overline{OUT1}$ in the MCR.
- Bit 7: MSR7 is the data carrier detect (DCD) bit. Data carrier detect indicates the status of the data carrier detect ($\overline{DCD}$) input. When the channel is in the loop mode (MCR4 is set), MSR7 reflects the value of OUT2 in the MCR.

Reading the MSR clears the delta modem status indicators but has no affect on the other status bits. For LSR and MSR, the setting of status bits is inhibited during status register read operations. If a status condition is generated during a read $\overline{IOR}$ operation, the status bit is not set until the trailing edge of the read. When a status bit is set during a read operation and the same status condition occurs, that status bit is cleared at the trailing edge of the read instead of being set again. In the loopback mode when modem status interrupts are enabled, $\overline{CTS}$, $\overline{DSR}$, $\overline{RI}$, and $\overline{DCD}$ inputs are ignored; however, a modem status interrupt can still be generated by writing to MCR3–MCR0. Applications software should not write to the MSR.

PRINCIPLES OF OPERATION

modem status register (MSR) (continued)

Table 7. Modem Status Register Bits

MSR BIT	MNEMONIC	DESCRIPTION
MSR0	ΔCTS	Delta clear to send
MSR1	ΔDSR	Delta data set ready
MSR2	TERI	Trailing edge of ring indicator
MSR3	ΔDCD	Delta data carrier detect
MSR4	CTS	Clear to send
MSR5	DSR	Data set ready
MSR6	RI	Ring indicator
MSR7	DCD	Data carrier detect

programming

The serial channel of the ACE is programmed by the control registers LCR, IER, DLL, DLM, MCR, and FCR. These control words define the character length, number of stop bits, parity, baud rate, and modem interface.

While the control registers can be written in any order, the IER should be written last because it controls the interrupt enables. Once the serial channel is programmed and operational, these registers can be updated any time the ACE serial channel is not transmitting or receiving data.

programmable baud rate generator

The ACE serial channel contains a programmable baud rate generator (BRG) that divides the clock (dc to 8 MHz) by any divisor from 1 to ($2^{16}-1$). Two 8-bit divisor latch registers store the divisor in a 16-bit binary format. These divisor latch registers must be loaded during initialization. Upon loading either of the divisor latches, a 16-bit baud counter is immediately loaded. This prevents long counts on initial load. The BRG can use any of three different popular frequencies to provide standard baud rates. These frequencies are 1.8432 MHz, 3.072 MHz, and 8 MHz. With these frequencies, standard bit rates from 50 kbps to 512 kbps are available. Tables 8, 9, 10, and 11 illustrate the divisors needed to obtain standard rates using these three frequencies. The output frequency of the baud rate generator is 16× the data rate [divisor # = clock ÷ (baud rate × 16)] referred to in this document as RCLK.

PRINCIPLES OF OPERATION

programmable baud rate generator (continued)

Table 8. Baud Rates Using an 1.8432-MHz Crystal

BAUD RATE DESIRED	DIVISOR (N) USED TO GENERATE 16× CLOCK	PERCENT ERROR DIFFERENCE BETWEEN DESIRED AND ACTUAL
50	2304	—
75	1536	—
110	1047	0.026
134.5	857	0.058
150	768	—
300	384	—
600	192	—
1200	96	—
1800	64	—
2000	58	0.690
2400	48	—
3600	32	—
4800	24	—
7200	16	—
9600	12	—
19200	6	—
38400	3	—
56000	2	2.860

Table 9. Baud Rates Using an 3.072-MHz Crystal

BAUD RATE DESIRED	DIVISOR (N) USED TO GENERATE 16× CLOCK	PERCENT ERROR DIFFERENCE BETWEEN DESIRED AND ACTUAL
50	3840	—
75	2560	—
110	1745	0.026
134.5	1428	0.034
150	1280	—
300	640	—
600	320	—
1200	160	—
1800	107	0.312
2000	96	—
2400	80	—
3600	53	0.628
4800	40	—
7200	27	1.230
9600	20	—
19200	10	—
38400	5	—

PRINCIPLES OF OPERATION

programmable baud rate generator (continued)

Table 10. Baud Rates Using an 8-MHz Clock

BAUD RATE DESIRED	DIVISOR (N) USED TO GENERATE 16× CLOCK	PERCENT ERROR DIFFERENCE BETWEEN DESIRED AND ACTUAL
50	10000	—
75	6667	0.005
110	4545	0.010
134.5	3717	0.013
150	333	0.010
300	1667	0.020
600	883	0.040
1200	417	0.080
1800	277	0.080
2000	250	—
2400	208	0.160
3600	139	0.080
4800	104	0.160
7200	69	0.644
9600	52	0.160
19200	26	0.160
38400	13	0.160
56000	9	0.790
128000	4	2.344
256000	2	2.344
512000	1	2.400

PRINCIPLES OF OPERATION

programmable baud rate generator (continued)

Table 11. Baud Rates Using an 16-MHz Clock

BAUD RATE DESIRED	DIVISOR (N) USED TO GENERATE 16× CLOCK	PERCENT ERROR DIFFERENCE BETWEEN DESIRED AND ACTUAL
50	20000	0
75	13334	0.00
110	9090	0.01
134.5	7434	0.01
150	6666	0.01
300	3334	−0.02
600	1666	0.04
1200	834	−0.08
1800	554	0.28
2000	500	0.00
2400	416	0.16
3600	278	−0.08
4800	208	0.16
7200	138	0.64
9600	104	0.16
19200	52	0.16
38400	26	0.16
56000	18	−0.79
128000	8	−2.34
256000	4	−2.34
512000	2	−2.34
1000000	1	0.00

receiver

Serial asynchronous data is input into the RXx terminal. The ACE continually searches for a high-to-low transition from the idle state. When the transition is detected, a counter is reset and counts the 16× clock to 7 1/2, which is the center of the start bit. The start bit is valid when the RXx is still low. Verifying the start bits prevents the receiver from assembling a false data character due to a low going noise spike on the RXx input.

The LCR determines the number of data bits in a character (LCR0, LCR1). When parity is enabled, LCR3 and the polarity of parity LCR4 are needed. Status for the receiver is provided in the LSR. When a full character is received including parity and stop bits, the data received indicator in LSR0 is set. The CPU reads the RBR, which clears LSR0. If the character is not read prior to a new character transfer from the RSR to the RBR, the overrun error status indicator is set in LSR1. If there is a parity error, the parity error is set in LSR2. If a stop bit is not detected, a framing error indicator is set in LSR3.

In the FIFO mode operation, the data character and the associated error bits are stored in the receiver FIFO. If the data into RXx is a symmetrical square wave, the center of the data cells occurs within ±3.125% of the actual center, providing an error margin of 46.875%. The start bit can begin as much as one 16× clock cycle prior to being detected.

PRINCIPLES OF OPERATION

reset

After power up, the ACE RESET input should be held high for one microsecond to reset the ACE circuits to an idle mode until initialization. A high on RESET causes the following:

1. It initializes the transmitter and receiver internal clock counters.
2. It clears the LSR, except for transmitter register empty (TEMT) and transmit holding register empty (THRE), which are set. The MCR is also cleared. All of the discrete lines, memory elements, and miscellaneous logic associated with these register bits are also cleared or turned off. The LCR, divisor latches, RBR, and transmitter buffer register are not affected.

$\overline{\text{RXRDY}}$ operation

In mode 0, $\overline{\text{RXRDY}}$ is asserted (low) when the receive FIFO is not empty; it is released (high) when the FIFO is empty. In this way, the receiver FIFO is read when $\overline{\text{RXRDY}}$ is asserted (low).

In mode 1, $\overline{\text{RXRDY}}$ is asserted (low) when the receive FIFO has filled to the trigger level or a character time-out has occurred (four character times with no transmission of characters); it is released (high) when the FIFO is empty. In this mode, multiple received characters are read by the DMA device, reducing the number of times it is interrupted.

$\overline{\text{RXRDY}}$ and $\overline{\text{TXRDY}}$ outputs from each of the four internal ACEs of the TL16C554 are ANDed together internally. This combined signal is brought out externally to $\overline{\text{RXRDY}}$ and $\overline{\text{TXRDY}}$.

Following the removal of the reset condition (RESET low), the ACE remains in the idle mode until programmed. A hardware reset of the ACE sets the THRE and TEMT status bits in the LSR. When interrupts are subsequently enabled, an interrupt occurs due to THRE. A summary of the effect of a reset on the ACE is given in Table 12.

Table 12. RESET Affects on Registers and Signals

REGISTER/SIGNAL	RESET CONTROL	RESET STATE
Interrupt enable register	Reset	All bits cleared (0–3 forced and 4–7 permanent)
Interrupt identification register	Reset	Bit 0 is set, bits 1, 2, 3, 6, and 7 are cleared, Bits 4–5 are permanently cleared
Line control register	Reset	All bits cleared
Modem control register	Reset	All bits cleared (5–7 permanent)
FIFO control register	Reset	All bits cleared
Line status register	Reset	All bits cleared, except bits 5 and 6 are set
Modem status register	Reset	Bits 0–3 cleared, bits 4–7 input signals
TXx	Reset	High
Interrupt (RCVR ERRS)	Read LSR/Reset	Low
Interrupt (receiver data ready)	Read RBR/Reset	Low
Interrupt (THRE)	Read IIR/Write THR/Reset	Low
Interrupt (modem status changes)	Read MSR/Reset	Low
$\overline{\text{RTS}}$	Reset	High
$\overline{\text{DTR}}$	Reset	High

PRINCIPLES OF OPERATION

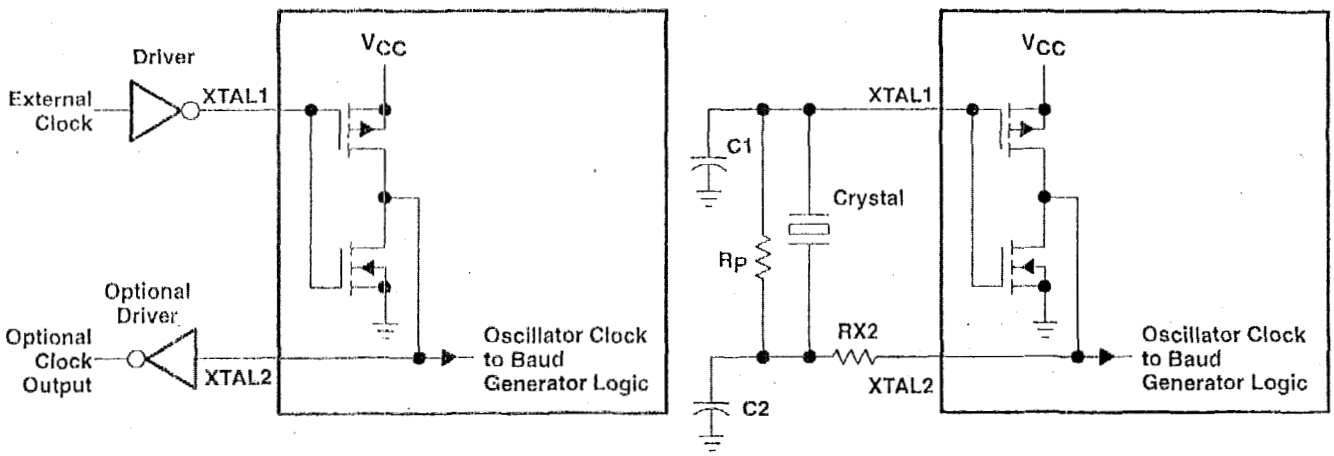
scratchpad register

The scratch register is an 8-bit read/write register that has no affect on either channel in the ACE. It is intended to be used by the programmer to hold data temporarily.

$\overline{\text{TXRDY}}$ operation

In mode 0, $\overline{\text{TXRDY}}$ is asserted (low) when the transmit FIFO is empty; it is released (high) when the FIFO contains at least one byte. In this way, the FIFO is written with 16 bytes when $\overline{\text{TXRDY}}$ is asserted (low).

In mode 1, $\overline{\text{TXRDY}}$ is asserted (low) when the transmit FIFO is not full; in this mode, the transmit FIFO is written with another byte when $\overline{\text{TXRDY}}$ is asserted (low).



TYPICAL CRYSTAL OSCILLATOR NETWORK

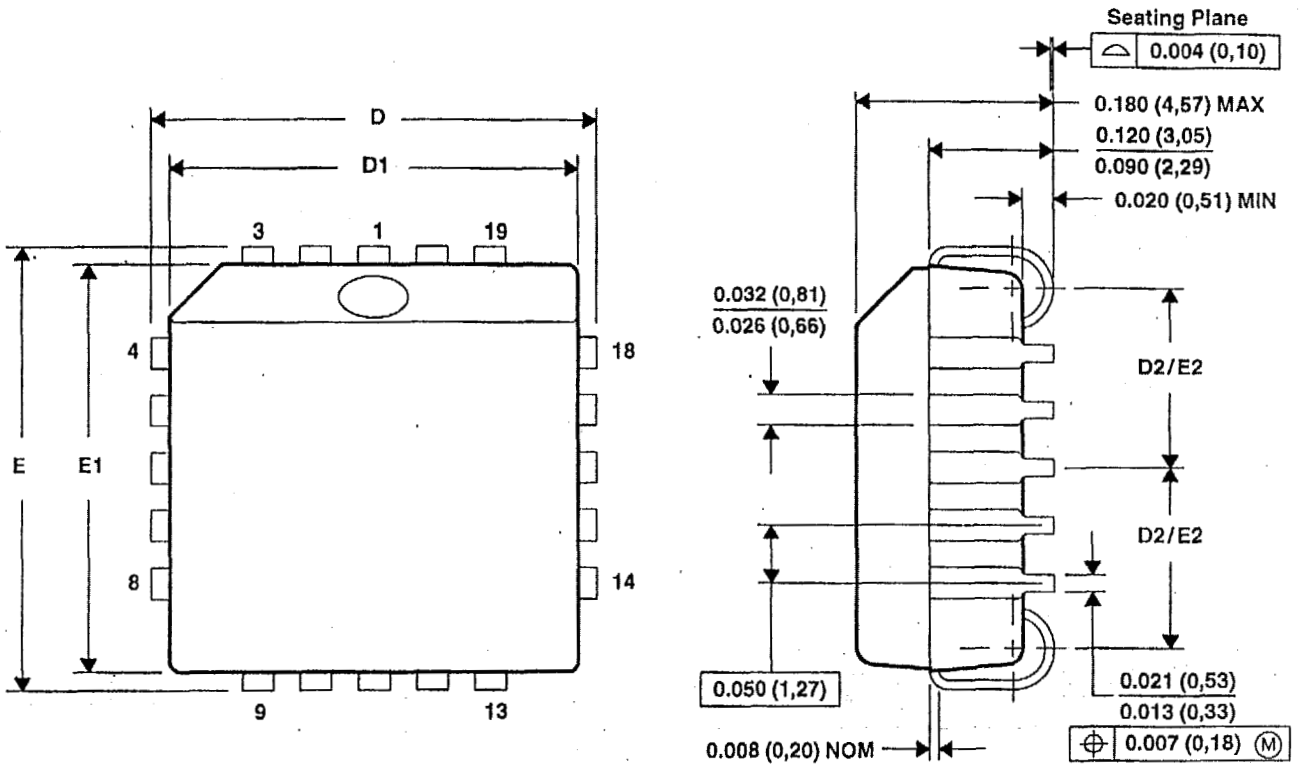
CRYSTAL	RP	RX2	C1	C2
3.1 MHz	1 MΩ	1.5 kΩ	10–30 pF	40–60 pF
1.8 MHz	1 MΩ	1.5 kΩ	10–30 pF	40–60 pF

Figure 17. Typical Clock Circuits

MECHANICAL DATA

FN (S-PQCC-J)** — **PLASTIC J-LEADED CHIP CARRIER**

20 PIN SHOWN



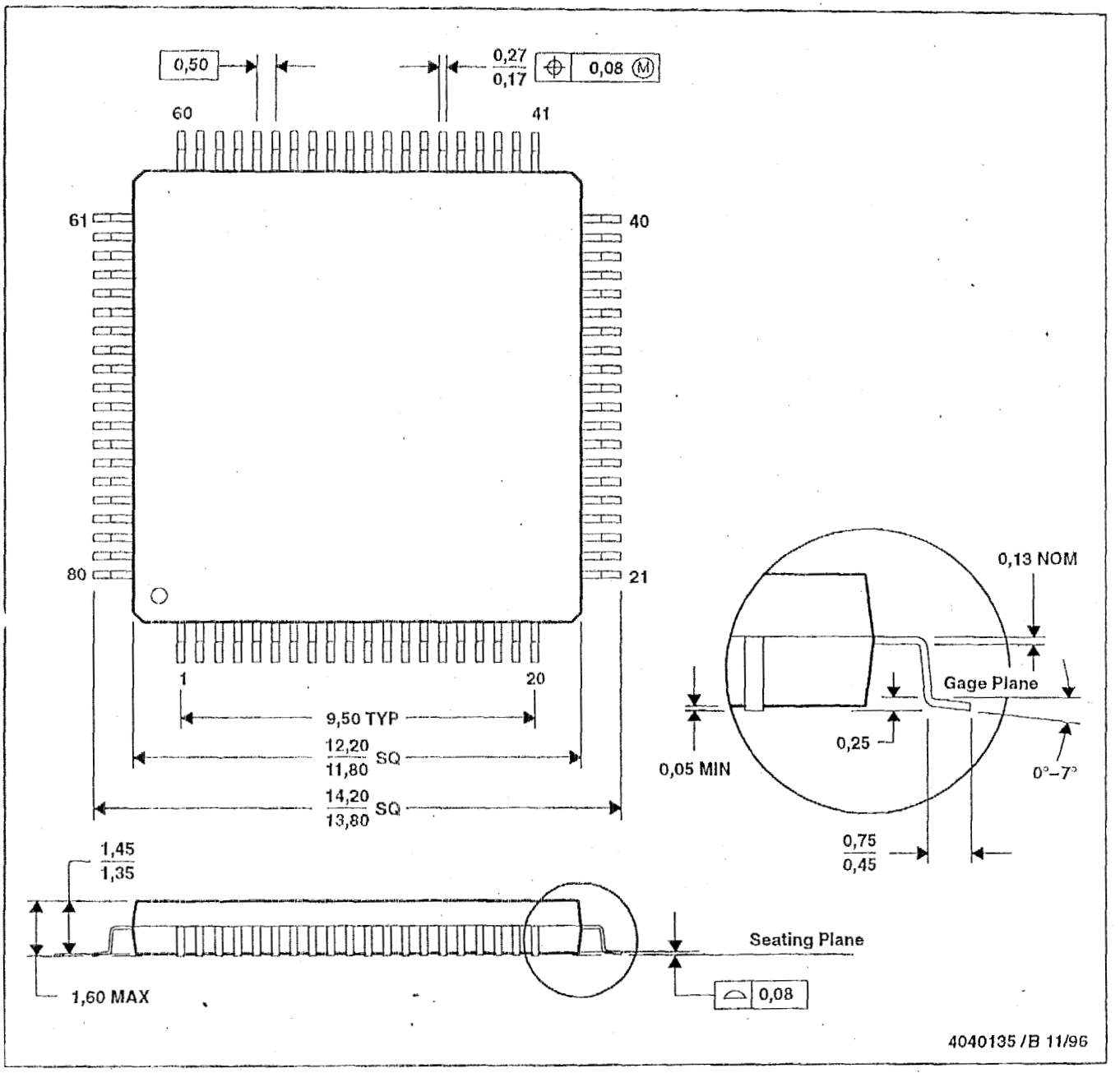
NO. OF PINS **	D/E		D1/E1		D2/E2	
	MIN	MAX	MIN	MAX	MIN	MAX
20	0.385 (9,78)	0.395 (10,03)	0.350 (8,89)	0.356 (9,04)	0.141 (3,58)	0.169 (4,29)
28	0.485 (12,32)	0.495 (12,57)	0.450 (11,43)	0.456 (11,58)	0.191 (4,85)	0.219 (5,56)
44	0.685 (17,40)	0.695 (17,65)	0.650 (16,51)	0.656 (16,66)	0.291 (7,39)	0.319 (8,10)
52	0.785 (19,94)	0.795 (20,19)	0.750 (19,05)	0.756 (19,20)	0.341 (8,66)	0.369 (9,37)
68	0.985 (25,02)	0.995 (25,27)	0.950 (24,13)	0.958 (24,33)	0.441 (11,20)	0.469 (11,91)
84	1.185 (30,10)	1.195 (30,35)	1.150 (29,21)	1.158 (29,41)	0.541 (13,74)	0.569 (14,45)

4040005/B 03/95

NOTES: A. All linear dimensions are in inches (millimeters).
B. This drawing is subject to change without notice.
C. Falls within JEDEC MS-018

MECHANICAL DATA

PN (S-PQFP-G80) **PLASTIC QUAD FLATPACK**



NOTES: A. All linear dimensions are in millimeters.
B. This drawing is subject to change without notice.
C. Falls within JEDEC MS-026

IMPORTANT NOTICE

Texas Instruments and its subsidiaries (TI) reserve the right to make changes to their products or to discontinue any product or service without notice, and advise customers to obtain the latest version of relevant information to verify, before placing orders, that information being relied on is current and complete. All products are sold subject to the terms and conditions of sale supplied at the time of order acknowledgement, including those pertaining to warranty, patent infringement, and limitation of liability.

TI warrants performance of its semiconductor products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

CERTAIN APPLICATIONS USING SEMICONDUCTOR PRODUCTS MAY INVOLVE POTENTIAL RISKS OF DEATH, PERSONAL INJURY, OR SEVERE PROPERTY OR ENVIRONMENTAL DAMAGE ("CRITICAL APPLICATIONS"). TI SEMICONDUCTOR PRODUCTS ARE NOT DESIGNED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS. INCLUSION OF TI PRODUCTS IN SUCH APPLICATIONS IS UNDERSTOOD TO BE FULLY AT THE CUSTOMER'S RISK.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards must be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance or customer product design. TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used. TI's publication of information regarding any third party's products or services does not constitute TI's approval, warranty or endorsement thereof.

4.4 Xilinx, XC5204 – Field Programmable Gate Array

The XC5204 is the primary component for the logic of the Diamond Systems Emerald-MM-DIO card and determines the Digital I/O operations. The specifications and data sheet for this IC is included for extra clarity.

The following data sheet was printed from the Xilinx Internet site at:

http://www.xilinx.com/partinfo/5200.pdf

The logic programmed into the XC5204 FPGA is determined by the cell structure of the IC explained on pages 7-85 through 7-86 of the data sheet. The signal routing within the FPGA is determined by the contents of the 128-kbit serial PROM, 17128EPI. Every time the FPGA is powered up, the programming data is transferred from the PROM to the FPGA.

XILINX®

XC5200 Series
Field Programmable Gate Arrays

November 5, 1998 (Version 5.2)

Product Specification

Features

- Low-cost, register/latch rich, SRAM based reprogrammable architecture
  - 0.5µm three-layer metal CMOS process technology
  - 256 to 1936 logic cells (3,000 to 23,000 "gates")
  - Price competitive with Gate Arrays
- System Level Features
  - System performance beyond 50 MHz
  - 6 levels of interconnect hierarchy
  - VersaRing™ I/O Interface for pin-locking
  - Dedicated carry logic for high-speed arithmetic functions
  - Cascade chain for wide input functions
  - Built-in IEEE 1149.1 JTAG boundary scan test circuitry on all I/O pins
  - Internal 3-state bussing capability
  - Four dedicated low-skew clock or signal distribution nets
- Versatile I/O and Packaging
  - Innovative VersaRing™ I/O interface provides a high logic cell to I/O ratio, with up to 244 I/O signals
  - Programmable output slew-rate control maximizes performance and reduces noise
  - Zero Flip-Flop hold time for input registers simplifies system timing
  - Independent Output Enables for external bussing
  - Footprint compatibility in common packages within the XC5200 Series and with the XC4000 Series
  - Over 150 device/package combinations, including advanced BGA, TQ, and VQ packaging available
- Fully Supported by Xilinx Development System
  - Automatic place and route software
  - Wide selection of PC and Workstation platforms
  - Over 100 3rd-party Alliance interfaces
  - Supported by shrink-wrap Foundation software

Description

The XC5200 Field-Programmable Gate Array Family is engineered to deliver low cost. Building on experiences gained with three previous successful SRAM FPGA families, the XC5200 family brings a robust feature set to programmable logic design. The VersaBlock™ logic module, the VersaRing I/O interface, and a rich hierarchy of interconnect resources combine to enhance design flexibility and reduce time-to-market. Complete support for the XC5200 family is delivered through the familiar Xilinx software environment. The XC5200 family is fully supported on popular workstation and PC platforms. Popular design entry methods are fully supported, including ABEL, schematic capture, VHDL, and Verilog HDL synthesis. Designers utilizing logic synthesis can use their existing tools to design with the XC5200 devices.

Table 1: XC5200 Field-Programmable Gate Array Family Members

Device	XC5202	XC5204	XC5206	XC5210	XC5215
Logic Cells	256	480	784	1,296	1,936
Max Logic Gates	3,000	6,000	10,000	16,000	23,000
Typical Gate Range	2,000 - 3,000	4,000 - 6,000	6,000 - 10,000	10,000 - 16,000	15,000 - 23,000
VersaBlock Array	8 x 8	10 x 12	14 x 14	18 x 18	22 x 22
CLBs	64	120	196	324	484
Flip-Flops	256	480	784	1,296	1,936
I/Os	84	124	148	196	244
TBUFs per Longline	10	14	16	20	24

XC5200 Family Compared to XC4000/Spartan™ and XC3000 Series

For readers already familiar with the XC4000/Spartan and XC3000 FPGA Families, this section describes significant differences between them and the XC5200 family. Unless otherwise indicated, comparisons refer to both XC4000/Spartan and XC3000 devices.

Configurable Logic Block (CLB) Resources

Each XC5200 CLB contains four independent 4-input function generators and four registers, which are configured as four independent Logic Cells™ (LCs). The registers in each XC5200 LC are optionally configurable as edge-triggered D-type flip-flops or as transparent level-sensitive latches.

The XC5200 CLB includes dedicated carry logic that provides fast arithmetic carry capability. The dedicated carry logic may also be used to cascade function generators for implementing wide arithmetic functions.

XC4000 family: XC5200 devices have no wide edge decoders. Wide decoders are implemented using cascade logic. Although sacrificing speed for some designs, lack of wide edge decoders reduces the die area and hence cost of the XC5200.

XC4000/Spartan family: XC5200 dedicated carry logic differs from that of the XC4000/Spartan family in that the sum is generated in an additional function generator in the adjacent column. This design reduces XC5200 die size and hence cost for many applications. Note, however, that a loadable up/down counter requires the same number of function generators in both families. XC3000 has no dedicated carry.

XC4000/Spartan family: XC5200 lookup tables are optimized for cost and hence cannot implement RAM.

Input/Output Block (IOB) Resources

The XC5200 family maintains footprint compatibility with the XC4000 family, but not with the XC3000 family.

To minimize cost and maximize the number of I/O per Logic Cell, the XC5200 I/O does not include flip-flops or latches.

For high performance paths, the XC5200 family provides direct connections from each IOB to the registers in the adjacent CLB in order to emulate IOB registers.

Each XC5200 I/O Pin provides a programmable delay element to control input set-up time. This element can be used to avoid potential hold-time problems. Each XC5200 I/O Pin is capable of 8-mA source and sink currents.

IEEE 1149.1-type boundary scan is supported in each XC5200 I/O.

Table 2: Xilinx Field-Programmable Gate Array Families

Parameter	XC5200	Spartan	XC4000	XC3000
CLB function generators	4	3	3	2
CLB inputs	20	9	9	5
CLB outputs	12	4	4	2
Global buffers	4	8	8	2
User RAM	no	yes	yes	no
Edge decoders	no	no	yes	no
Cascade chain	yes	no	no	no
Fast carry logic	yes	yes	yes	no
Internal 3-state	yes	yes	yes	yes
Boundary scan	yes	yes	yes	no
Slew-rate control	yes	yes	yes	yes

Routing Resources

The XC5200 family provides a flexible coupling of logic and local routing resources called the VersaBlock. The XC5200 VersaBlock element includes the CLB, a Local Interconnect Matrix (LIM), and direct connects to neighboring VersaBlocks.

The XC5200 provides four global buffers for clocking or high-fanout control signals. Each buffer may be sourced by means of its dedicated pad or from any internal source.

Each XC5200 TBUF can drive up to two horizontal and two vertical Longlines. There are no internal pull-ups for XC5200 Longlines.

Configuration and Readback

The XC5200 supports a new configuration mode called Express mode.

XC4000/Spartan family: The XC5200 family provides a global reset but not a global set.

XC5200 devices use a different configuration process than that of the XC3000 family, but use the same process as the XC4000 and Spartan families.

XC3000 family: Although their configuration processes differ, XC5200 devices may be used in daisy chains with XC3000 devices.

XC3000 family: The XC5200 PROGRAM pin is a single-function input pin that overrides all other inputs. The PROGRAM pin does not exist in XC3000.

XC3000 family: XC5200 devices support an additional programming mode: Peripheral Synchronous.

XC3000 family: The XC5200 family does not support Power-down, but offers a Global 3-state input that does not reset any flip-flops.

XC3000 family: The XC5200 family does not provide an on-chip crystal oscillator amplifier, but it does provide an internal oscillator from which a variety of frequencies up to 12 MHz are available.

Architectural Overview

Figure 1 presents a simplified, conceptual overview of the XC5200 architecture. Similar to conventional FPGAs, the XC5200 family consists of programmable IOBs, programmable logic blocks, and programmable interconnect. Unlike other FPGAs, however, the logic and local routing resources of the XC5200 family are combined in flexible VersaBlocks (Figure 2). General-purpose routing connects to the VersaBlock through the General Routing Matrix (GRM).

VersaBlock: Abundant Local Routing Plus Versatile Logic

The basic logic element in each VersaBlock structure is the Logic Cell, shown in Figure 3. Each LC contains a 4-input function generator (F), a storage device (FD), and control logic. There are five independent inputs and three outputs to each LC. The independence of the inputs and outputs allows the software to maximize the resource utilization within each LC. Each Logic Cell also contains a direct feedthrough path that does not sacrifice the use of either the function generator or the register; this feature is a first for FPGAs. The storage device is configurable as either a D flip-flop or a latch. The control logic consists of carry logic for fast implementation of arithmetic functions, which can also be configured as a cascade chain allowing decode of very wide input functions.

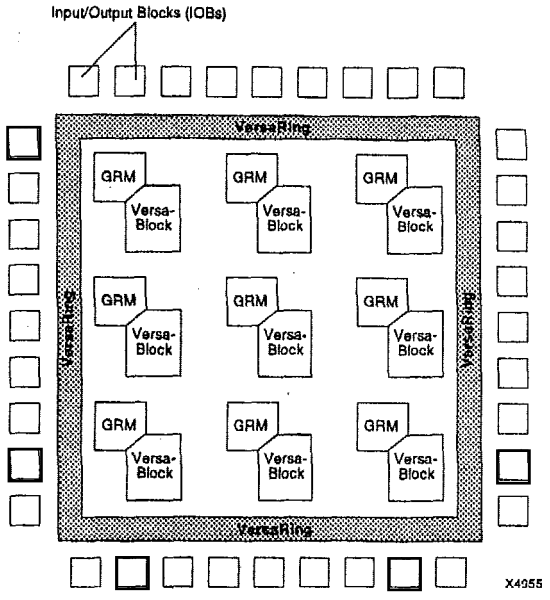


Figure 1: XC5200 Architectural Overview

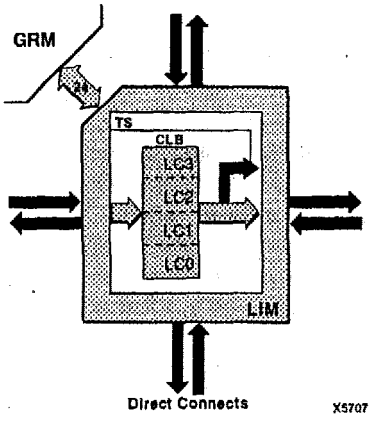


Figure 2: VersaBlock

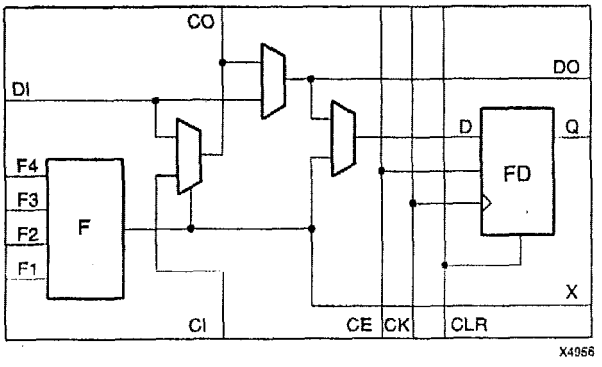


Figure 3: XC5200 Logic Cell (Four LCs per CLB)

The XC5200 CLB consists of four LCs, as shown in Figure 4. Each CLB has 20 independent inputs and 12 independent outputs. The top and bottom pairs of LCs can be configured to implement 5-input functions. The challenge of FPGA implementation software has always been to maximize the usage of logic resources. The XC5200 family addresses this issue by surrounding each CLB with two types of local interconnect — the Local Interconnect Matrix (LIM) and direct connects. These two interconnect resources, combined with the CLB, form the VersaBlock, represented in Figure 2.

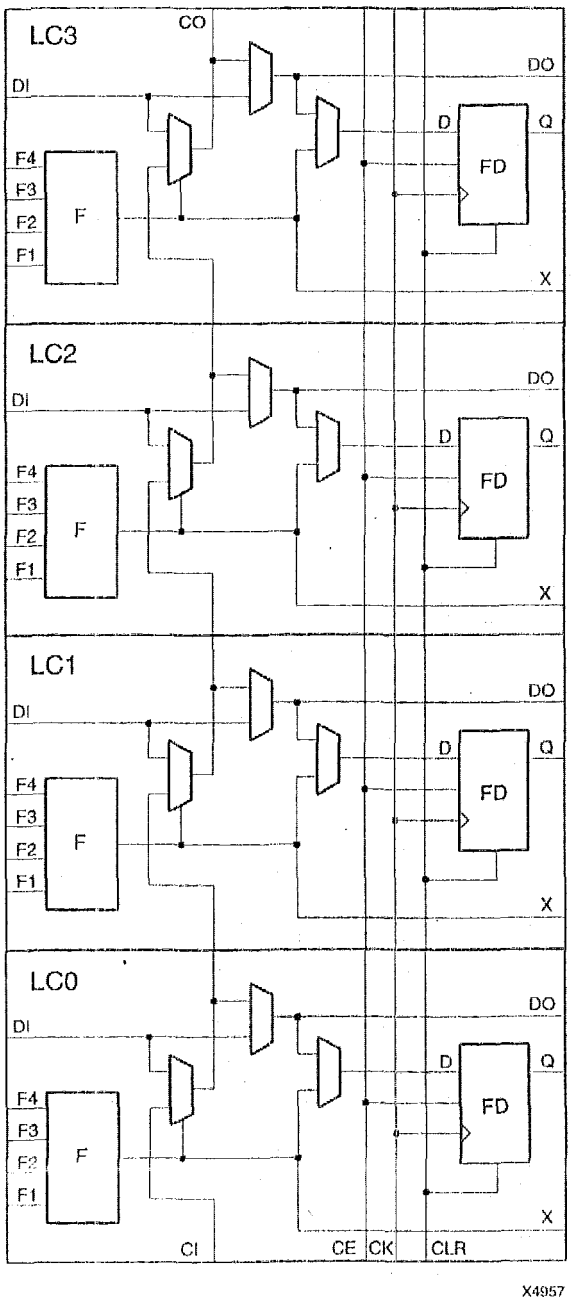


Figure 4: Configurable Logic Block

The LIM provides 100% connectivity of the inputs and outputs of each LC in a given CLB. The benefit of the LIM is that no general routing resources are required to connect feedback paths within a CLB. The LIM connects to the GRM via 24 bidirectional nodes.

The direct connects allow immediate connections to neighboring CLBs, once again without using any of the general interconnect. These two layers of local routing resource improve the granularity of the architecture, effectively making the XC5200 family a "sea of logic cells." Each Versa-Block has four 3-state buffers that share a common enable line and directly drive horizontal and vertical Longlines, creating robust on-chip bussing capability. The VersaBlock allows fast, local implementation of logic functions, effectively implementing user designs in a hierarchical fashion. These resources also minimize local routing congestion and improve the efficiency of the general interconnect, which is used for connecting larger groups of logic. It is this combination of both fine-grain and coarse-grain architecture attributes that maximize logic utilization in the XC5200 family. This symmetrical structure takes full advantage of the third metal layer, freeing the placement software to pack user logic optimally with minimal routing restrictions.

VersaRing I/O Interface

The interface between the IOBs and core logic has been redesigned in the XC5200 family. The IOBs are completely decoupled from the core logic. The XC5200 IOBs contain dedicated boundary-scan logic for added board-level testability, but do not include input or output registers. This approach allows a maximum number of IOBs to be placed around the device, improving the I/O-to-gate ratio and decreasing the cost per I/O. A "freeway" of interconnect cells surrounding the device forms the VersaRing, which provides connections from the IOBs to the internal logic. These incremental routing resources provide abundant connections from each IOB to the nearest VersaBlock, in addition to Longline connections surrounding the device. The VersaRing eliminates the historic trade-off between high logic utilization and pin placement flexibility. These incremental edge resources give users increased flexibility in preassigning (i.e., locking) I/O pins before completing their logic designs. This ability accelerates time-to-market, since PCBs and other system components can be manufactured concurrent with the logic design.

General Routing Matrix

The GRM is functionally similar to the switch matrices found in other architectures, but it is novel in its tight coupling to the logic resources contained in the VersaBlocks. Advanced simulation tools were used during the development of the XC5200 architecture to determine the optimal level of routing resources required. The XC5200 family contains six levels of interconnect hierarchy — a series of

single-length lines, double-length lines, and Longlines all routed through the GRM. The direct connects, LIM, and logic-cell feedthrough are contained within each Versa-Block. Throughout the XC5200 interconnect, an efficient multiplexing scheme, in combination with three layer metal (TLM), was used to improve the overall efficiency of silicon usage.

Performance Overview

The XC5200 family has been benchmarked with many designs running synchronous clock rates beyond 66 MHz. The performance of any design depends on the circuit to be implemented, and the delay through the combinatorial and sequential logic elements, plus the delay in the interconnect routing. A rough estimate of timing can be made by assuming 3-6 ns per logic level, which includes direct-connect routing delays, depending on speed grade. More accurate estimations can be made using the information in the Switching Characteristic Guideline section.

Taking Advantage of Reconfiguration

FPGA devices can be reconfigured to change logic function while resident in the system. This capability gives the system designer a new degree of freedom not available with any other type of logic.

Hardware can be changed as easily as software. Design updates or modifications are easy, and can be made to products already in the field. An FPGA can even be reconfigured dynamically to perform different functions at different times.

Reconfigurable logic can be used to implement system self-diagnostics, create systems capable of being reconfigured for different environments or operations, or implement multi-purpose hardware for a given application. As an added benefit, using reconfigurable FPGA devices simplifies hardware design and debugging and shortens product time-to-market.

Detailed Functional Description

Configurable Logic Blocks (CLBs)

Figure 4 shows the logic in the XC5200 CLB, which consists of four Logic Cells (LC[3:0]). Each Logic Cell consists of an independent 4-input Lookup Table (LUT), and a D-Type flip-flop or latch with common clock, clock enable, and clear, but individually selectable clock polarity. Additional logic features provided in the CLB are:

- An independent 5-input LUT by combining two 4-input LUTs.
- High-speed carry propagate logic.
- High-speed pattern decoding.
- High-speed direct connection to flip-flop D-inputs.
- Individual selection of either a transparent, level-sensitive latch or a D flip-flop.
- Four 3-state buffers with a shared Output Enable.

5-Input Functions

Figure 5 illustrates how the outputs from the LUTs from LC0 and LC1 can be combined with a 2:1 multiplexer (F5_MUX) to provide a 5-input function. The outputs from the LUTs of LC2 and LC3 can be similarly combined.

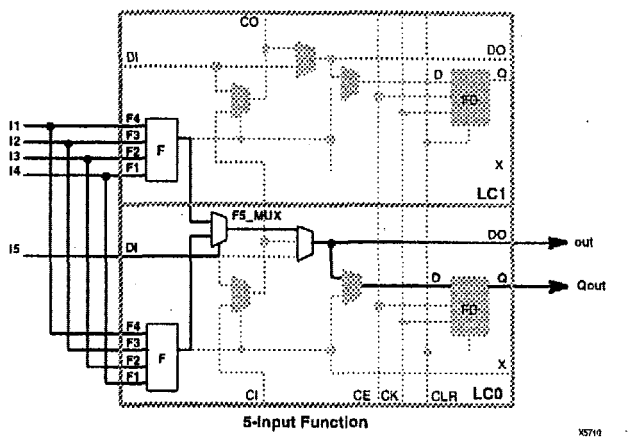


Figure 5: Two LUTs in Parallel Combined to Create a 5-input Function

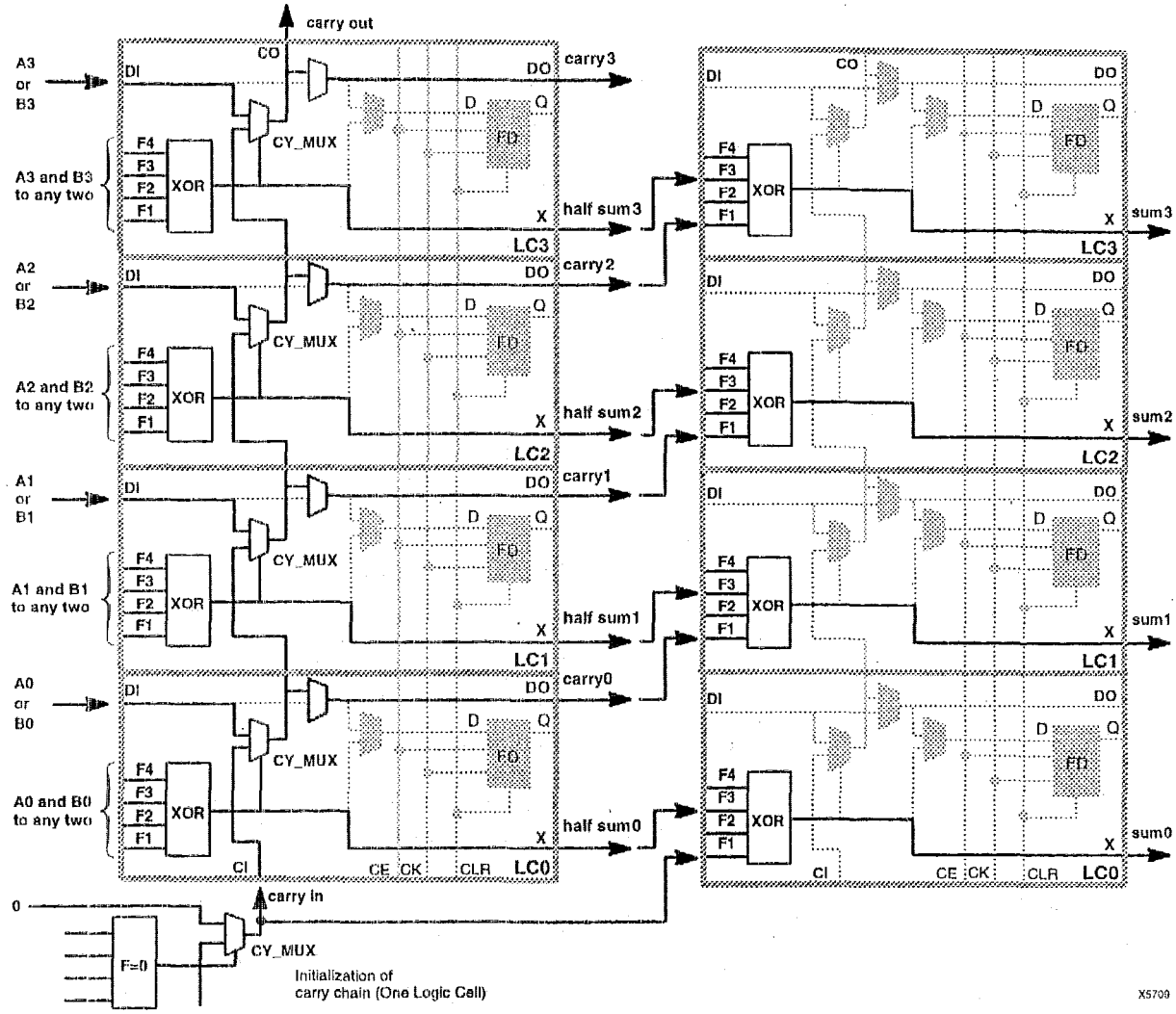


Figure 6: XC5200 CY_MUX Used for Adder Carry Propagate

Carry Function

The XC5200 family supports a carry-logic feature that enhances the performance of arithmetic functions such as counters, adders, etc. A carry multiplexer (CY_MUX) symbol is used to indicate the XC5200 carry logic. This symbol represents the dedicated 2:1 multiplexer in each LC that performs the one-bit high-speed carry propagate per logic cell (four bits per CLB).

While the carry propagate is performed inside the LC, an adjacent LC must be used to complete the arithmetic function. Figure 6 represents an example of an adder function. The carry propagate is performed on the CLB shown, which also generates the half-sum for the four-bit adder. An adjacent CLB is responsible for XORing the half-sum with the corresponding carry-out. Thus an adder or counter requires two LCs per bit. Notice that the carry chain requires an initialization stage, which the XC5200 family accomplishes using the carry initialize (CY_INIT) macro and one additional LC. The carry chain can propagate vertically up a column of CLBs.

The XC5200 library contains a set of Relationally-Placed Macros (RPMs) and arithmetic functions designed to take advantage of the dedicated carry logic. Using and modifying these macros makes it much easier to implement cus-

tomized RPMs, freeing the designer from the need to become an expert on architectures.

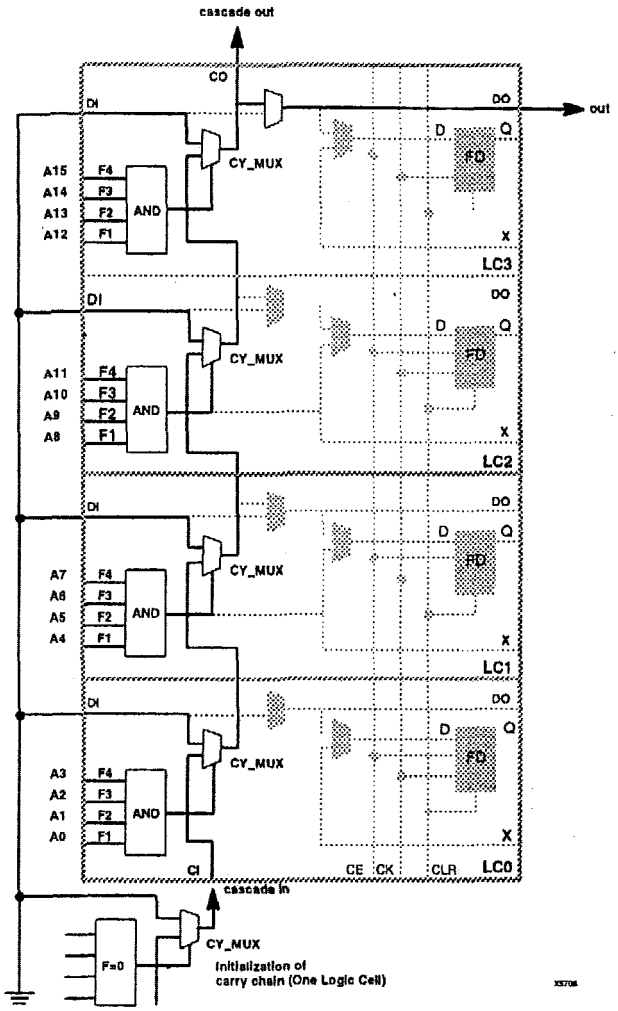


Figure 7: XC5200 CY_MUX Used for Decoder Cascade Logic

Cascade Function

Each CY_MUX can be connected to the CY_MUX in the adjacent LC to provide cascadable decode logic. Figure 7 illustrates how the 4-input function generators can be configured to take advantage of these four cascaded CY_MUXes. Note that AND and OR cascading are specific cases of a general decode. In AND cascading all bits are decoded equal to logic one, while in OR cascading all bits are decoded equal to logic zero. The flexibility of the LUT achieves this result. The XC5200 library contains gate macros designed to take advantage of this function.

CLB Flip-Flops and Latches

The CLB can pass the combinatorial output(s) to the interconnect network, but can also store the combinatorial results or other incoming data in flip-flops, and connect their outputs to the interconnect network as well. The CLB storage elements can also be configured as latches.

Table 3: CLB Storage Element Functionality (active rising edge is shown)

Mode	CK	CE	CLR	D	Q
Power-Up or GR	X	X	X	X	0
Flip-Flop	X	X	1	X	0
	⎍	1*	0*	D	D
	0	X	0*	X	Q
Latch	1	1*	0*	X	Q
	0	1*	0*	D	D
Both	X	0	0*	X	Q

Legend:

X	Don't care
⎍	Rising edge
0*	Input is Low or unconnected (default value)
1*	Input is High or unconnected (default value)

Data Inputs and Outputs

The source of a storage element data input is programmable. It is driven by the function F, or by the Direct In (DI) block input. The flip-flops or latches drive the Q CLB outputs.

Four fast feed-through paths from DI to DO are available, as shown in Figure 4. This bypass is sometimes used by the automated router to repower internal signals. In addition to the storage element (Q) and direct (DO) outputs, there is a combinatorial output (X) that is always sourced by the Lookup Table.

The four edge-triggered D-type flip-flops or level-sensitive latches have common clock (CK) and clock enable (CE) inputs. Any of the clock inputs can also be permanently enabled. Storage element functionality is described in Table 3.

Clock Input

The flip-flops can be triggered on either the rising or falling clock edge. The clock pin is shared by all four storage elements with individual polarity control. Any inverter placed on the clock input is automatically absorbed into the CLB.

Clock Enable

The clock enable signal (CE) is active High. The CE pin is shared by the four storage elements. If left unconnected for any, the clock enable for that storage element defaults to the active state. CE is not invertible within the CLB.

Clear

An asynchronous storage element input (CLR) can be used to reset all four flip-flops or latches in the CLB. This input

can also be independently disabled for any flip-flop. CLR is active High. It is not invertible within the CLB.

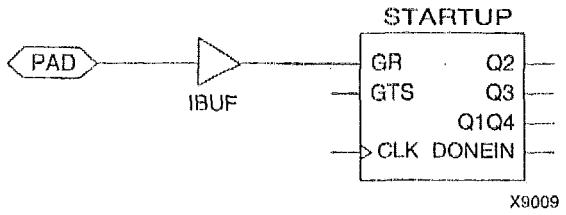


Figure 8: Schematic Symbols for Global Reset

Global Reset

A separate Global Reset line clears each storage element during power-up, reconfiguration, or when a dedicated Reset net is driven active. This global net (GR) does not compete with other routing resources; it uses a dedicated distribution network.

GR can be driven from any user-programmable pin as a global reset input. To use this global net, place an input pad and input buffer in the schematic or HDL code, driving the GR pin of the STARTUP symbol. (See Figure 9.) A specific pin location can be assigned to this input using a LOC attribute or property, just as with any other user-programmable pad. An inverter can optionally be inserted after the input buffer to invert the sense of the Global Reset signal. Alternatively, GR can be driven from any internal node.

Using FPGA Flip-Flops and Latches

The abundance of flip-flops in the XC5200 Series invites pipelined designs. This is a powerful way of increasing performance by breaking the function into smaller subfunctions and executing them in parallel, passing on the results through pipeline flip-flops. This method should be seriously considered wherever throughput is more important than latency.

To include a CLB flip-flop, place the appropriate library symbol. For example, FDCE is a D-type flip-flop with clock enable and asynchronous clear. The corresponding latch symbol is called LDCE.

In XC5200-Series devices, the flip-flops can be used as registers or shift registers without blocking the function generators from performing a different, perhaps unrelated task. This ability increases the functional capacity of the devices.

The CLB setup time is specified between the function generator inputs and the clock input CK. Therefore, the specified CLB flip-flop setup time includes the delay through the function generator.

Three-State Buffers

The XC5200 family has four dedicated Three-State Buffers (TBUFs, or BUFTs in the schematic library) per CLB (see Figure 9). The four buffers are individually configurable through four configuration bits to operate as simple non-inverting buffers or in 3-state mode. When in 3-state mode the CLB output enable (TS) control signal drives the enable to all four buffers. Each TBUF can drive up to two horizontal and/or two vertical Longlines. These 3-state buffers can be used to implement multiplexed or bidirectional buses on the horizontal or vertical longlines, saving logic resources.

The 3-state buffer enable is an active-High 3-state (i.e. an active-Low enable), as shown in Table 4.

Table 4: Three-State Buffer Functionality

IN	T	OUT
X	1	Z
IN	0	IN

Another 3-state buffer with similar access is located near each I/O block along the right and left edges of the array.

The longlines driven by the 3-state buffers have a weak keeper at each end. This circuit prevents undefined floating levels. However, it is overridden by any driver. To ensure the longline goes high when no buffers are on, add an additional BUFT to drive the output High during all of the previously undefined states.

Figure 10 shows how to use the 3-state buffers to implement a multiplexer. The selection is accomplished by the buffer 3-state signal.

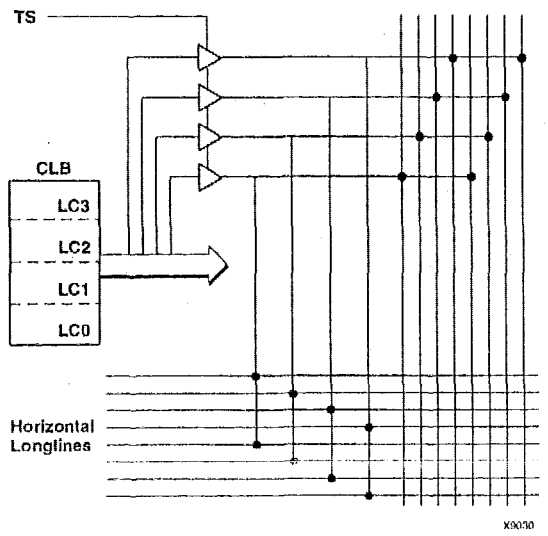


Figure 9: XC5200 3-State Buffers

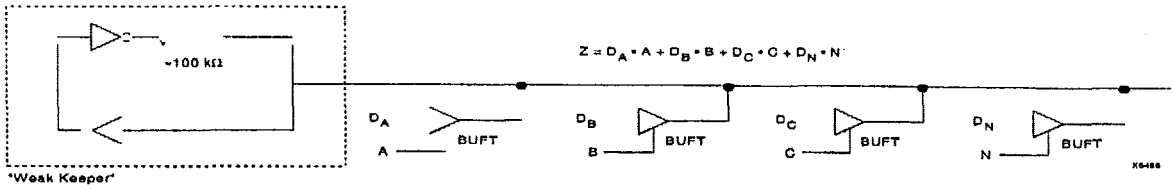


Figure 10: 3-State Buffers Implement a Multiplexer

Input/Output Blocks

User-configurable input/output blocks (IOBs) provide the interface between external package pins and the internal logic. Each IOB controls one package pin and can be configured for input, output, or bidirectional signals.

The I/O block, shown in Figure 11, consists of an input buffer and an output buffer. The output driver is an 8-mA full-rail CMOS buffer with 3-state control. Two slew-rate control modes are supported to minimize bus transients. Both the output buffer and the 3-state control are invertible. The input buffer has globally selected CMOS or TTL input thresholds. The input buffer is invertible and also provides a programmable delay line to assure reliable chip-to-chip set-up and hold times. Minimum ESD protection is 3 KV using the Human Body Model.

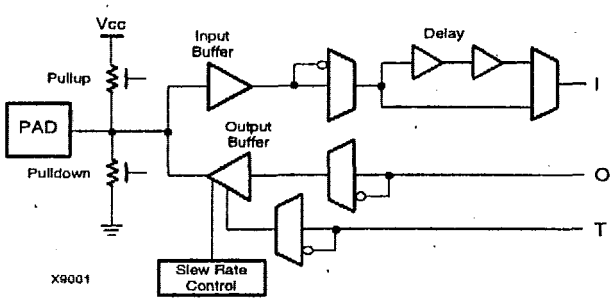


Figure 11: XC5200 I/O Block

IOB Input Signals

The XC5200 inputs can be globally configured for either TTL (1.2V) or CMOS thresholds, using an option in the bitstream generation software. There is a slight hysteresis of about 300mV.

The inputs of XC5200-Series 5-Volt devices can be driven by the outputs of any 3.3-Volt device, if the 5-Volt inputs are in TTL mode.

Supported sources for XC5200-Series device inputs are shown in Table 5.

Table 5: Supported Sources for XC5200-Series Device Inputs

Source	XC5200 Input Mode	
	5 V, TTL	5 V, CMOS
Any device, Vcc = 3.3 V, CMOS outputs	√	Unreliable Data
Any device, Vcc = 5 V, TTL outputs	√	
Any device, Vcc = 5 V, CMOS outputs	√	√

Optional Delay Guarantees Zero Hold Time

XC5200 devices do not have storage elements in the IOBs. However, XC5200 IOBs can be efficiently routed to CLB flip-flops or latches to store the I/O signals.

The data input to the register can optionally be delayed by several nanoseconds. With the delay enabled, the setup time of the input flip-flop is increased so that normal clock routing does not result in a positive hold-time requirement. A positive hold time requirement can lead to unreliable, temperature- or processing-dependent operation.

The input flip-flop setup time is defined between the data measured at the device I/O pin and the clock input at the CLB (not at the clock pin). Any routing delay from the device clock pin to the clock input of the CLB must, therefore, be subtracted from this setup time to arrive at the real setup time requirement relative to the device pins. A short specified setup time might, therefore, result in a negative setup time at the device pins, i.e., a positive hold-time requirement.

When a delay is inserted on the data line, more clock delay can be tolerated without causing a positive hold-time requirement. Sufficient delay eliminates the possibility of a data hold-time requirement at the external pin. The maximum delay is therefore inserted as the software default.

The XC5200 IOB has a one-tap delay element: either the delay is inserted (default), or it is not. The delay guarantees a zero hold time with respect to clocks routed through any of the XC5200 global clock buffers. (See "Global Lines" on page 96 for a description of the global clock buffers in the XC5200.) For a shorter input register setup time, with

non-zero hold, attach a NODELAY attribute or property to the flip-flop or input buffer.

IOB Output Signals

Output signals can be optionally inverted within the IOB, and pass directly to the pad. As with the inputs, a CLB flip-flop or latch can be used to store the output signal.

An active-High 3-state signal can be used to place the output buffer in a high-impedance state, implementing 3-state outputs or bidirectional I/O. Under configuration control, the output (OUT) and output 3-state (T) signals can be inverted. The polarity of these signals is independently configured for each IOB.

The XC5200 devices provide a guaranteed output sink current of 8 mA.

Supported destinations for XC5200-Series device outputs are shown in Table 6.(For a detailed discussion of how to interface between 5 V and 3.3 V devices, see the 3V Products section of *The Programmable Logic Data Book*.)

An output can be configured as open-drain (open-collector) by placing an OBUFT symbol in a schematic or HDL code, then tying the 3-state pin (T) to the output signal, and the input pin (I) to Ground. (See Figure 12.)

Table 6: Supported Destinations for XC5200-Series Outputs

Destination	XC5200 Output Mode
	5 V, CMOS
XC5200 device, V_{CC}=3.3 V, CMOS-threshold inputs	√
Any typical device, V_{CC} = 3.3 V, CMOS-threshold inputs	some[1]
Any device, V_{CC} = 5 V, TTL-threshold inputs	√
Any device, V_{CC} = 5 V, CMOS-threshold inputs	√

1. Only if destination device has 5-V tolerant inputs

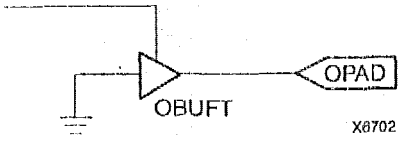


Figure 12: Open-Drain Output

Output Slew Rate

The slew rate of each output buffer is, by default, reduced, to minimize power bus transients when switching non-critical signals. For critical signals, attach a FAST attribute or property to the output buffer or flip-flop.

For XC5200 devices, maximum total capacitive load for simultaneous fast mode switching in the same direction is 200 pF for all package pins between each Power/Ground pin pair. For some XC5200 devices, additional internal Power/Ground pin pairs are connected to special Power and Ground planes within the packages, to reduce ground bounce.

For slew-rate limited outputs this total is two times larger for each device type: 400 pF for XC5200 devices. This maximum capacitive load should not be exceeded, as it can result in ground bounce of greater than 1.5 V amplitude and more than 5 ns duration. This level of ground bounce may cause undesired transient behavior on an output, or in the internal logic. This restriction is common to all high-speed digital ICs, and is not particular to Xilinx or the XC5200 Series.

XC5200-Series devices have a feature called "Soft Start-up," designed to reduce ground bounce when all outputs are turned on simultaneously at the end of configuration. When the configuration process is finished and the device starts up, the first activation of the outputs is automatically slew-rate limited. Immediately following the initial activation of the I/O, the slew rate of the individual outputs is determined by the individual configuration option for each IOB.

Global Three-State

A separate Global 3-State line (not shown in Figure 11) forces all FPGA outputs to the high-impedance state, unless boundary scan is enabled and is executing an EXTEST instruction. This global net (GTS) does not compete with other routing resources; it uses a dedicated distribution network.

GTS can be driven from any user-programmable pin as a global 3-state input. To use this global net, place an input pad and input buffer in the schematic or HDL code, driving the GTS pin of the STARTUP symbol. A specific pin location can be assigned to this input using a LOC attribute or property, just as with any other user-programmable pad. An inverter can optionally be inserted after the input buffer to invert the sense of the Global 3-State signal. Using GTS is similar to Global Reset. See Figure 8 on page 90 for details. Alternatively, GTS can be driven from any internal node.

Other IOB Options

There are a number of other programmable options in the XC5200-Series IOB.

Pull-up and Pull-down Resistors

Programmable IOB pull-up and pull-down resistors are useful for tying unused pins to Vcc or Ground to minimize power consumption and reduce noise sensitivity. The configurable pull-up resistor is a p-channel transistor that pulls

to Vcc. The configurable pull-down resistor is an n-channel transistor that pulls to Ground.

The value of these resistors is 20 kΩ – 100 kΩ. This high value makes them unsuitable as wired-AND pull-up resistors.

The pull-up resistors for most user-programmable IOBs are active during the configuration process. See Table 13 on page 124 for a list of pins with pull-ups active before and during configuration.

After configuration, voltage levels of unused pads, bonded or unbonded, must be valid logic levels, to reduce noise sensitivity and avoid excess current. Therefore, by default, unused pads are configured with the internal pull-up resistor active. Alternatively, they can be individually configured with the pull-down resistor, or as a driven output, or to be driven by an external source. To activate the internal pull-up, attach the PULLUP library component to the net attached to the pad. To activate the internal pull-down, attach the PULLDOWN library component to the net attached to the pad.

JTAG Support

Embedded logic attached to the IOBs contains test structures compatible with IEEE Standard 1149.1 for boundary scan testing, simplifying board-level testing. More information is provided in "Boundary Scan" on page 98.

Oscillator

XC5200 devices include an internal oscillator. This oscillator is used to clock the power-on time-out, clear configuration memory, and source CCLK in Master configuration modes. The oscillator runs at a nominal 12 MHz frequency that varies with process, Vcc, and temperature. The output CCLK frequency is selectable as 1 MHz (default), 6 MHz, or 12 MHz.

The XC5200 oscillator divides the internal 12-MHz clock or a user clock. The user then has the choice of dividing by 4, 16, 64, or 256 for the "OSC1" output and dividing by 2, 8, 32, 128, 1024, 4096, 16384, or 65536 for the "OSC2" output. The division is specified via a "DIVIDEn_BY=x" attribute on the symbol, where n=1 for OSC1, or n=2 for OSC2. These frequencies can vary by as much as -50% or + 50%.

The OSC5 macro is used where an internal oscillator is required. The CK_DIV macro is applicable when a user clock input is specified (see Figure 13).

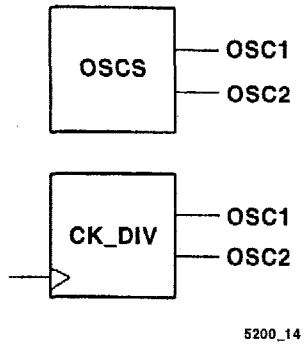


Figure 13: XC5200 Oscillator Macros

VersaBlock Routing

The General Routing Matrix (GRM) connects to the Versa-Block via 24 bidirectional ports (M0-M23). Excluding direct connections, global nets, and 3-statable Longlines, all VersaBlock inputs and outputs connect to the GRM via these 24 ports. Four 3-statable unidirectional signals (TQ0-TQ3) drive out of the VersaBlock directly onto the horizontal and vertical Longlines. Two horizontal global nets and two vertical global nets connect directly to every CLB clock pin; they can connect to other CLB inputs via the GRM. Each CLB also has four unidirectional direct connects to each of its four neighboring CLBs. These direct connects can also feed directly back to the CLB (see Figure 14).

In addition, each CLB has 16 direct inputs, four direct connections from each of the neighboring CLBs. These direct connections provide high-speed local routing that bypasses the GRM.

Local Interconnect Matrix

The Local Interconnect Matrix (LIM) is built from input and output multiplexers. The 13 CLB outputs (12 LC outputs plus a V_{cc}/GND signal) connect to the eight VersaBlock outputs via the output multiplexers, which consist of eight fully populated 13-to-1 multiplexers. Of the eight VersaBlock outputs, four signals drive each neighboring CLB directly, and provide a direct feedback path to the input multiplexers. The four remaining multiplexer outputs can drive the GRM through four TBUFs (TQ0-TQ3). All eight multiplexer outputs can connect to the GRM through the bidirectional M0-M23 signals. All eight signals also connect to the input multiplexers and are potential inputs to that CLB.

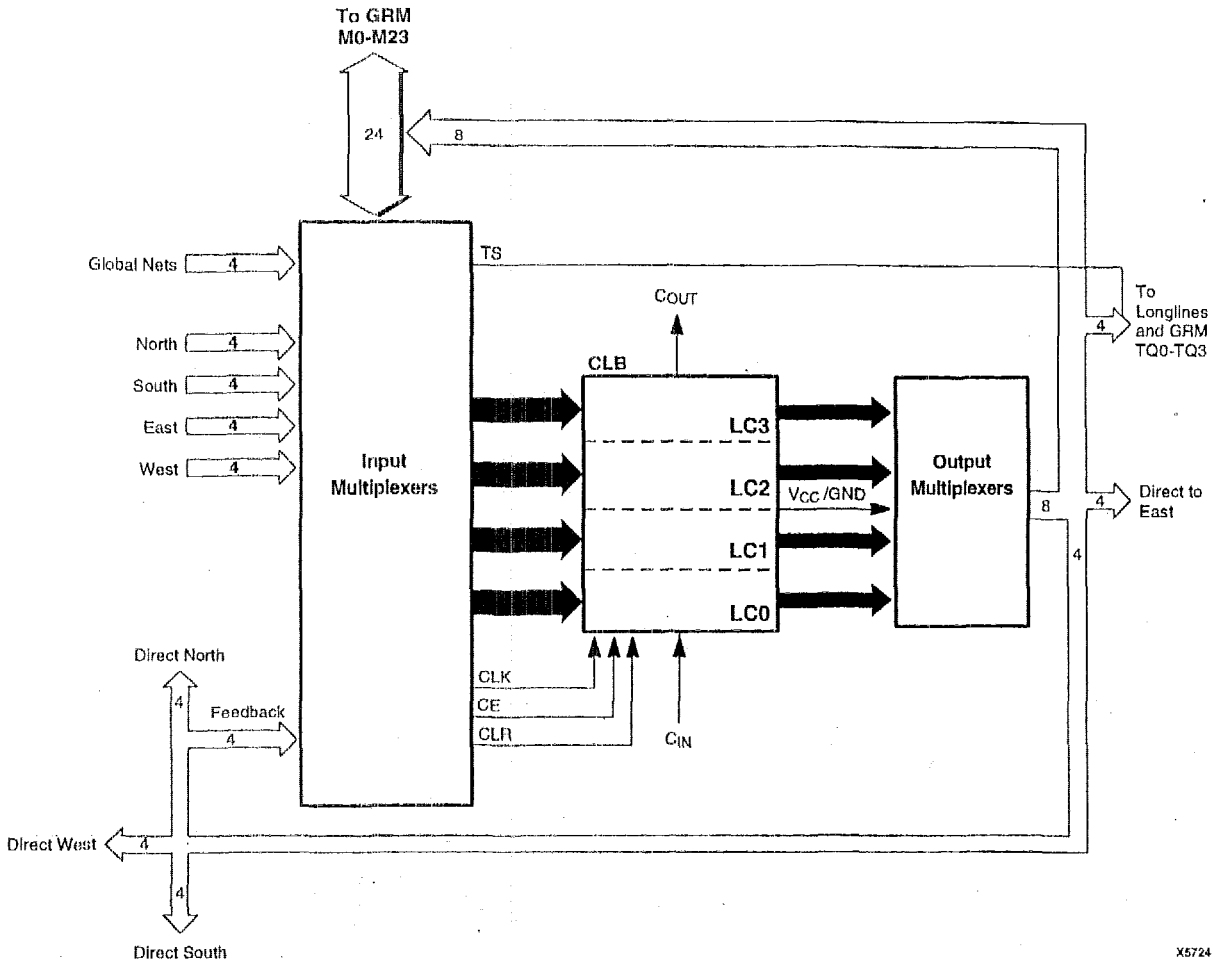


Figure 14: VersaBlock Details

CLB inputs have several possible sources: the 24 signals from the GRM, 16 direct connections from neighboring VersaBlocks, four signals from global, low-skew buffers, and the four signals from the CLB output multiplexers. Unlike the output multiplexers, the input multiplexers are not fully populated; i.e., only a subset of the available signals can be connected to a given CLB input. The flexibility of LUT input swapping and LUT mapping compensates for this limitation. For example, if a 2-input NAND gate is required, it can be mapped into any of the four LUTs, and use any two of the four inputs to the LUT.

Direct Connects

The unidirectional direct-connect segments are connected to the logic input/output pins through the CLB input and output multiplexer arrays, and thus bypass the general routing matrix altogether. These lines increase the routing channel utilization, while simultaneously reducing the delay incurred in speed-critical connections.

The direct connects also provide a high-speed path from the edge CLBs to the VersaRing input/output buffers, and thus reduce pin-to-pin set-up time, clock-to-out, and combinational propagation delay. Direct connects from the input buffers to the CLB DI pin (direct flip-flop input) are only available on the left and right edges of the device. CLB look-up table inputs and combinatorial/registered outputs have direct connects to input/output buffers on all four sides.

The direct connects are ideal for developing customized RPM cells. Using direct connects improves the macro performance, and leaves the other routing channels intact for improved routing. Direct connects can also route through a CLB using one of the four cell-feedthrough paths.

General Routing Matrix

The General Routing Matrix, shown in Figure 15, provides flexible bidirectional connections to the Local Interconnect

Matrix through a hierarchy of different-length metal segments in both the horizontal and vertical directions. A pro-

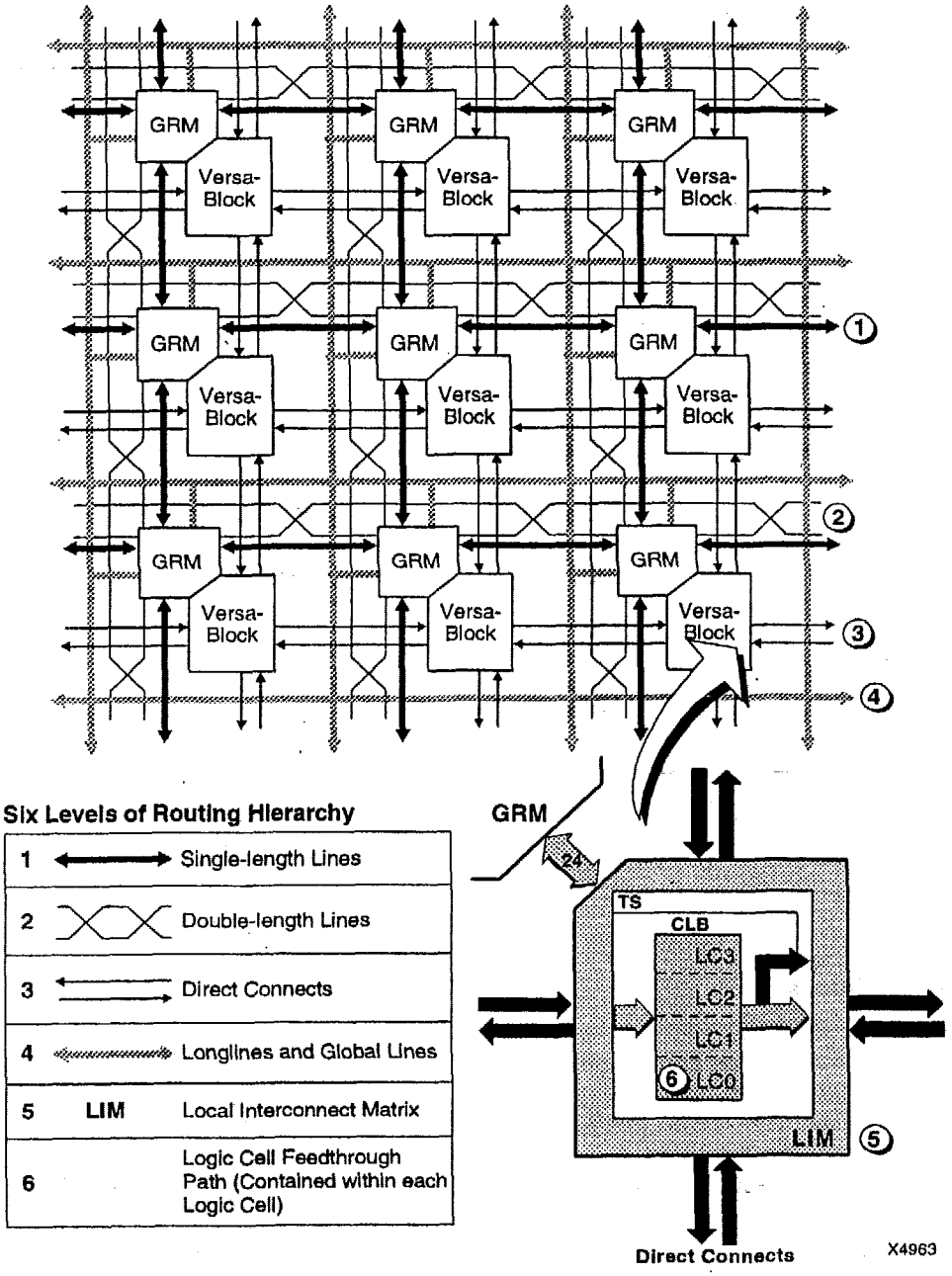


Figure 15: XC5200 Interconnect Structure

grammable interconnect point (PIP) establishes an electrical connection between two wire segments. The PIP, consisting of a pass transistor switch controlled by a memory element, provides bidirectional (in some cases, unidirectional) connection between two adjoining wires. A collection of PIPs inside the General Routing Matrix and in the Local Interconnect Matrix provides connectivity between various types of metal segments. A hierarchy of PIPs and associated routing segments combine to provide a powerful interconnect hierarchy:

- Forty bidirectional single-length segments per CLB provide ten routing channels to each of the four neighboring CLBs in four directions.
- Sixteen bidirectional double-length segments per CLB provide four routing channels to each of four other (non-neighboring) CLBs in four directions.
- Eight horizontal and eight vertical bidirectional Longline

segments span the width and height of the chip, respectively.

Two low-skew horizontal and vertical unidirectional global-line segments span each row and column of the chip, respectively.

Single- and Double-Length Lines

The single- and double-length bidirectional line segments make up the bulk of the routing channels. The double-length lines hop across every other CLB to reduce the propagation delays in speed-critical nets. Regenerating the signal strength is recommended after traversing three or four such segments. Xilinx place-and-route software automatically connects buffers in the path of the signal as necessary. Single- and double-length lines cannot drive onto Longlines and global lines; Longlines and global lines can, however, drive onto single- and double-length lines. As a general rule, Longline and global-line connections to the general routing matrix are unidirectional, with the signal direction from these lines toward the routing matrix.

Longlines

Longlines are used for high-fan-out signals, 3-state busses, low-skew nets, and faraway destinations. Row and column splitter PIPs in the middle of the array effectively double the total number of Longlines by electrically dividing them into two separated half-lines. Longlines are driven by the 3-state buffers in each CLB, and are driven by similar buffers at the periphery of the array from the VersaRing I/O Interface.

Bus-oriented designs are easily implemented by using Longlines in conjunction with the 3-state buffers in the CLB and in the VersaRing. Additionally, weak keeper cells at the periphery retain the last valid logic level on the Longlines when all buffers are in 3-state mode.

Longlines connect to the single-length or double-length lines, or to the logic inside the CLB, through the General Routing Matrix. The only manner in which a Longline can be driven is through the four 3-state buffers; therefore, a Longline-to-Longline or single-line-to-Longline connection through PIPs in the General Routing Matrix is not possible. Again, as a general rule, long- and global-line connections to the General Routing Matrix are unidirectional, with the signal direction from these lines toward the routing matrix.

The XC5200 family has no pull-ups on the ends of the Longlines sourced by TBUFs, unlike the XC4000 Series. Consequently, wired functions (i.e., WAND and WORAND) and wide multiplexing functions requiring pull-ups for undefined states (i.e., bus applications) must be implemented in a different way. In the case of the wired functions, the same functionality can be achieved by taking advantage of the carry/cascade logic described above, implementing a wide logic function in place of the wired function. In the case of 3-state bus applications, the user must insure that all states of the multiplexing function are defined. This process is as simple as adding an additional TBUF to drive the bus High when the previously undefined states are activated.

Global Lines

Global buffers in Xilinx FPGAs are special buffers that drive a dedicated routing network called Global Lines, as shown in Figure 16. This network is intended for high-fanout clocks or other control signals, to maximize speed and minimize skewing while distributing the signal to many loads.

The XC5200 family has a total of four global buffers (BUFG symbol in the library), each with its own dedicated routing channel. Two are distributed vertically and two horizontally throughout the FPGA.

The global lines provide direct input only to the CLB clock pins. The global lines also connect to the General Routing Matrix to provide access from these lines to the function generators and other control signals.

Four clock input pads at the corners of the chip, as shown in Figure 16, provide a high-speed, low-skew clock network to each of the four global-line buffers. In addition to the dedicated pad, the global lines can be sourced by internal logic. PIPs from several routing channels within the VersaRing can also be configured to drive the global-line buffers.

Details of all the programmable interconnect for a CLB is shown in Figure 17.

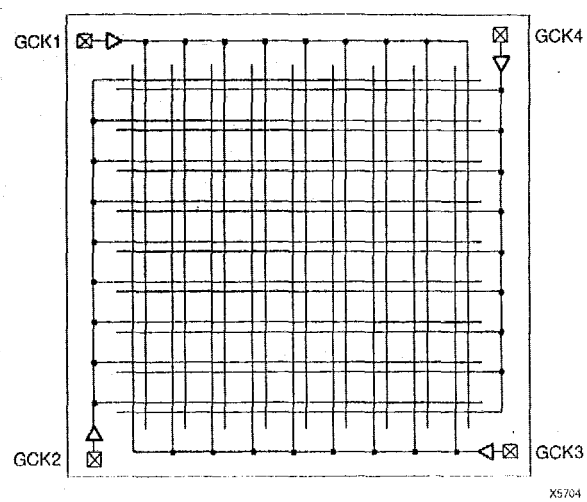


Figure 16: Global Lines

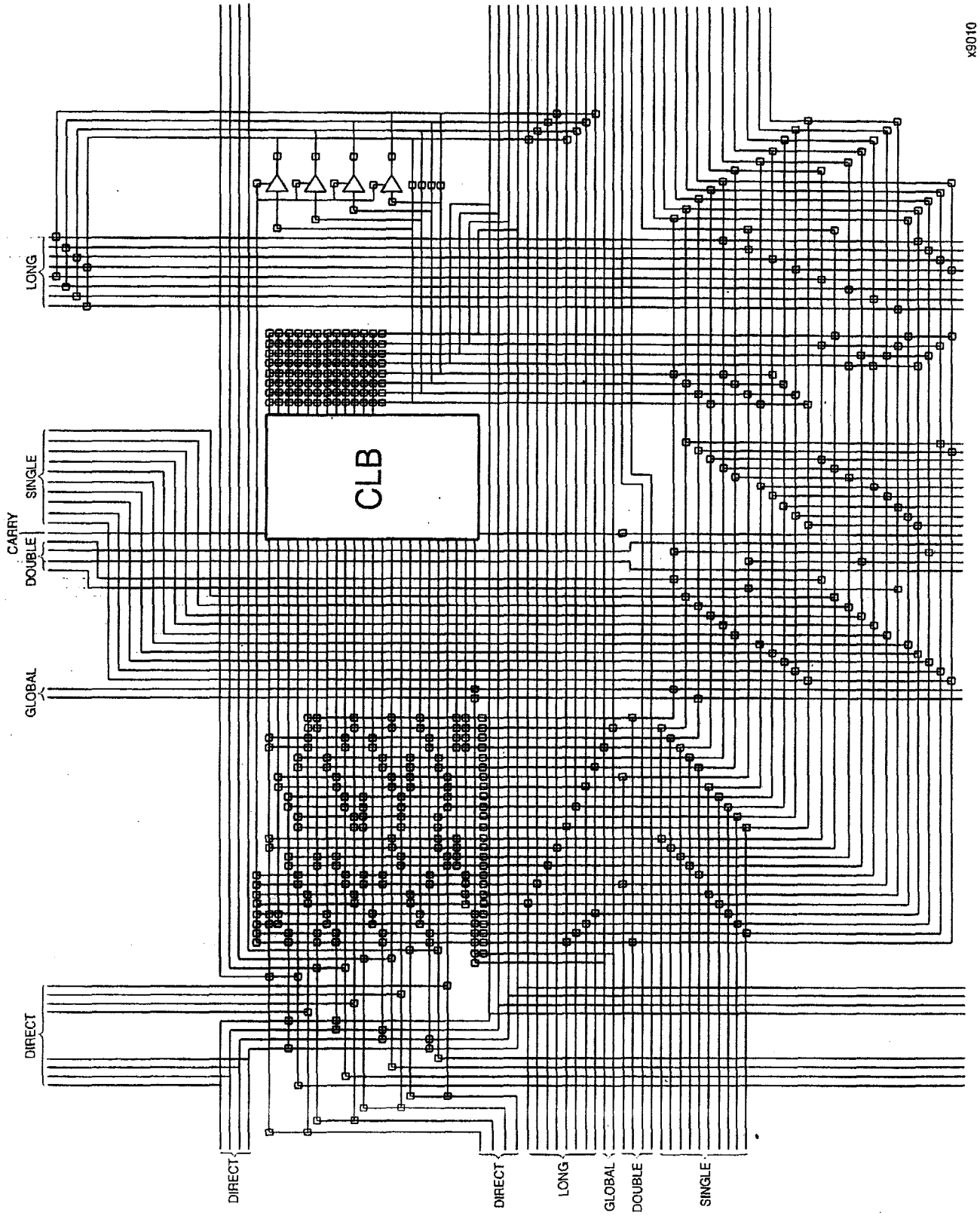


Figure 17: Detail of Programmable Interconnect Associated with XC5200 Series CLB

VersaRing Input/Output Interface

The VersaRing, shown in Figure 18, is positioned between the core logic and the pad ring; it has all the routing resources of a VersaBlock without the CLB logic. The VersaRing decouples the core logic from the I/O pads. Each VersaRing Cell provides up to four pad-cell connections on one side, and connects directly to the CLB ports on the other side.

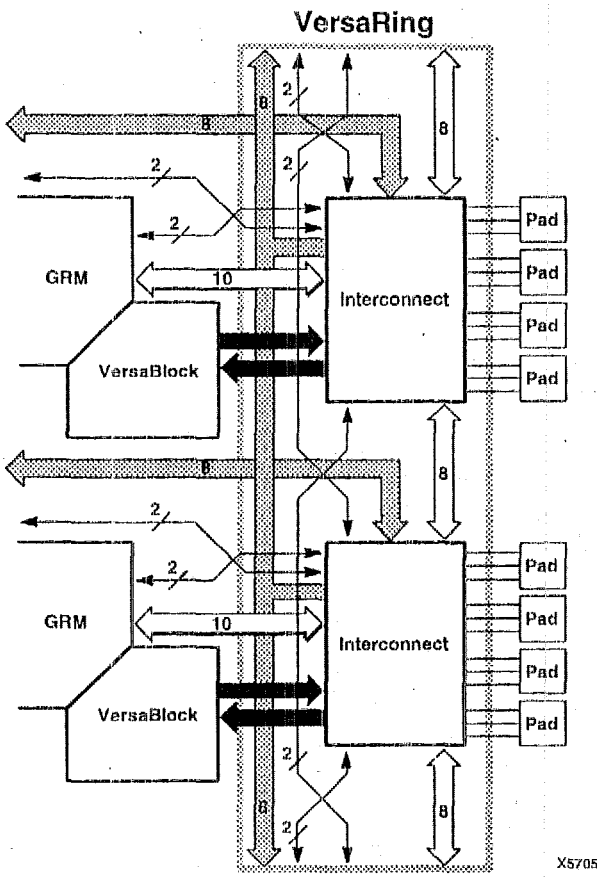


Figure 18: VersaRing I/O Interface

Boundary Scan

The "bed of nails" has been the traditional method of testing electronic assemblies. This approach has become less appropriate, due to closer pin spacing and more sophisticated assembly methods like surface-mount technology and multi-layer boards. The IEEE boundary scan standard 1149.1 was developed to facilitate board-level testing of electronic assemblies. Design and test engineers can imbed a standard test logic structure in their device to achieve high fault coverage for I/O and internal logic. This structure is easily implemented with a four-pin interface on any boundary scan-compatible IC. IEEE 1149.1-compatible devices may be serial daisy-chained together, connected in parallel, or a combination of the two.

XC5200 devices support all the mandatory boundary-scan instructions specified in the IEEE standard 1149.1. A Test Access Port (TAP) and registers are provided that implement the EXTEST, SAMPLE/PRELOAD, and BYPASS instructions. The TAP can also support two USERCODE instructions. When the boundary scan configuration option is selected, three normal user I/O pins become dedicated inputs for these functions. Another user output pin becomes the dedicated boundary scan output.

Boundary-scan operation is independent of individual IOB configuration and package type. All IOBs are treated as independently controlled bidirectional pins, including any unbonded IOBs. Retaining the bidirectional test capability after configuration provides flexibility for interconnect testing.

Also, internal signals can be captured during EXTEST by connecting them to unbonded IOBs, or to the unused outputs in IOBs used as unidirectional input pins. This technique partially compensates for the lack of INTEST support.

The user can serially load commands and data into these devices to control the driving of their outputs and to examine their inputs. This method is an improvement over bed-of-nails testing. It avoids the need to over-drive device outputs, and it reduces the user interface to four pins. An optional fifth pin, a reset for the control logic, is described in the standard but is not implemented in Xilinx devices.

The dedicated on-chip logic implementing the IEEE 1149.1 functions includes a 16-state machine, an instruction register and a number of data registers. The functional details can be found in the IEEE 1149.1 specification and are also discussed in the Xilinx application note XAPP 017: *"Boundary Scan in XC4000 and XC5200 Series devices"*

Figure 19 on page 99 is a diagram of the XC5200-Series boundary scan logic. It includes three bits of Data Register per IOB, the IEEE 1149.1 Test Access Port controller, and the Instruction Register with decodes.

The public boundary-scan instructions are always available prior to configuration. After configuration, the public instructions and any USERCODE instructions are only available if specified in the design. While SAMPLE and BYPASS are available during configuration, it is recommended that boundary-scan operations not be performed during this transitory period.

In addition to the test instructions outlined above, the boundary-scan circuitry can be used to configure the FPGA device, and to read back the configuration data.

All of the XC4000 boundary-scan modes are supported in the XC5200 family. Three additional outputs for the User-Register are provided (Reset, Update, and Shift), repre-

senting the decoding of the corresponding state of the boundary-scan internal state machine.

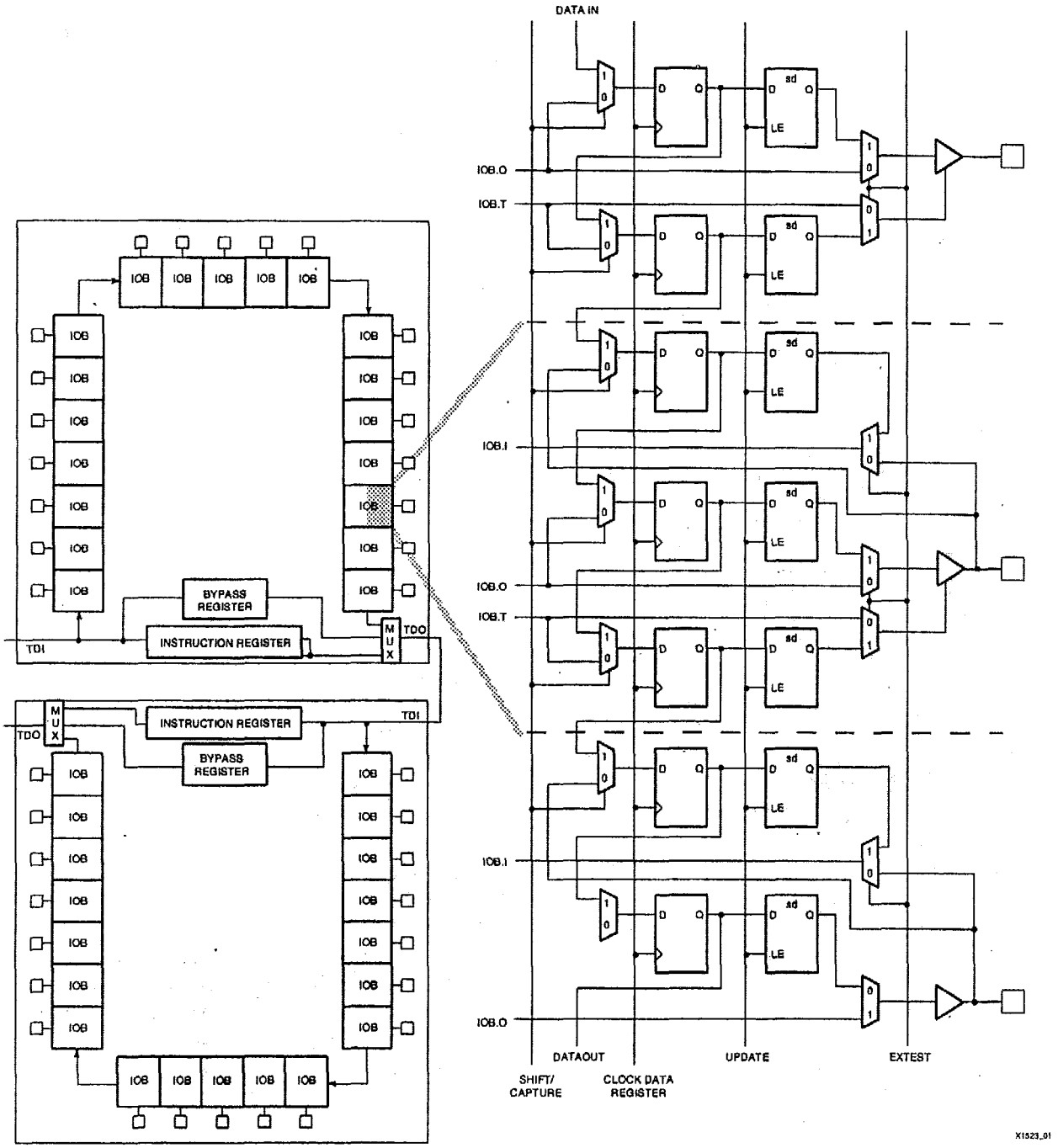


Figure 19: XC5200-Series Boundary Scan Logic

XC5200-Series devices can also be configured through the boundary scan logic. See XAPP 017 for more information.

Data Registers

The primary data register is the boundary scan register. For each IOB pin in the FPGA, bonded or not, it includes three bits for In, Out and 3-State Control. Non-IOB pins have appropriate partial bit population for In or Out only. PROGRAM, CCLK and DONE are not included in the boundary scan register. Each EXTEST CAPTURE-DR state captures all In, Out, and 3-State pins.

The data register also includes the following non-pin bits: TDO.T, and TDO.O, which are always bits 0 and 1 of the data register, respectively, and BSCANT.UPD, which is always the last bit of the data register. These three boundary scan bits are special-purpose Xilinx test signals.

The other standard data register is the single flip-flop BYPASS register. It synchronizes data being passed through the FPGA to the next downstream boundary scan device.

The FPGA provides two additional data registers that can be specified using the BSCAN macro. The FPGA provides two user pins (BSCAN.SEL1 and BSCAN.SEL2) which are the decodes of two user instructions, USER1 and USER2. For these instructions, two corresponding pins (BSCAN.TDO1 and BSCAN.TDO2) allow user scan data to be shifted out on TDO. The data register clock (BSCAN.DRCK) is available for control of test logic which the user may wish to implement with CLBs. The NAND of TCK and RUN-TEST-IDLE is also provided (BSCAN.IDLE).

Instruction Set

The XC5200-Series boundary scan instruction set also includes instructions to configure the device and read back the configuration data. The instruction set is coded as shown in Table 7.

Table 7: Boundary Scan Instructions

Instruction I2	I1	I0	Test Selected	TDO Source	I/O Data Source
0	0	0	EXTEST	DR	DR
0	0	1	SAMPLE/PRELOAD	DR	Pin/Logic
0	1	0	USER 1	BSCAN.TDO1	User Logic
0	1	1	USER 2	BSCAN.TDO2	User Logic
1	0	0	READBACK	Readback Data	Pin/Logic
1	0	1	CONFIGURE	DOUT	Disabled
1	1	0	Reserved	—	—
1	1	1	BYPASS	Bypass Register	—

Bit Sequence

The bit sequence within each IOB is: 3-State, Out, In. The data-register cells for the TAP pins TMS, TCK, and TDI have an OR-gate that permanently disables the output buffer if boundary-scan operation is selected. Consequently, it is impossible for the outputs in IOBs used by TAP inputs to conflict with TAP operation. TAP data is taken directly from the pin, and cannot be overwritten by injected boundary-scan data.

The primary global clock inputs (PGCK1-PGCK4) are taken directly from the pins, and cannot be overwritten with boundary-scan data. However, if necessary, it is possible to drive the clock input from boundary scan. The external clock source is 3-stated, and the clock net is driven with boundary scan data through the output driver in the clock-pad IOB. If the clock-pad IOBs are used for non-clock signals, the data may be overwritten normally.

Pull-up and pull-down resistors remain active during boundary scan. Before and during configuration, all pins are pulled up. After configuration, the choice of internal pull-up or pull-down resistor must be taken into account when designing test vectors to detect open-circuit PC traces.

From a cavity-up view of the chip (as shown in XDE or Epic), starting in the upper right chip corner, the boundary scan data-register bits are ordered as shown in Table 8. The device-specific pinout tables for the XC5200 Series include the boundary scan locations for each IOB pin.

Table 8: Boundary Scan Bit Sequence

Bit Position	I/O Pad Location
Bit 0 (TDO)	Top-edge I/O pads (right to left)
Bit 1	...
...	Left-edge I/O pads (top to bottom)
...	Bottom-edge I/O pads (left to right)
...	Right-edge I/O pads (bottom to top)
Bit N (TDI)	BSCANT.UPD

BSDL (Boundary Scan Description Language) files for XC5200-Series devices are available on the Xilinx web site in the File Download area.

Including Boundary Scan

If boundary scan is only to be used during configuration, no special elements need be included in the schematic or HDL code. In this case, the special boundary scan pins TDI, TMS, TCK and TDO can be used for user functions after configuration.

To indicate that boundary scan remain enabled after configuration, include the BSCAN library symbol and connect pad symbols to the TDI, TMS, TCK and TDO pins, as shown in Figure 20.

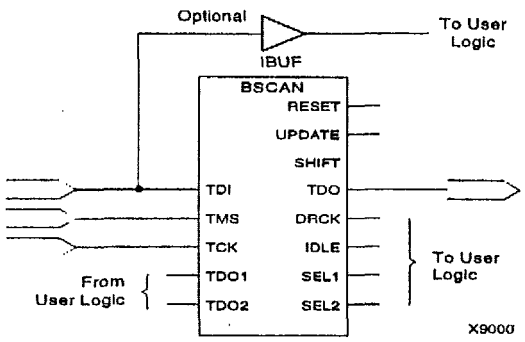


Figure 20: Boundary Scan Schematic Example

Even if the boundary scan symbol is used in a schematic, the input pins TMS, TCK, and TDI can still be used as inputs to be routed to internal logic. Care must be taken not to force the chip into an undesired boundary scan state by inadvertently applying boundary scan input patterns to these pins. The simplest way to prevent this is to keep TMS High, and then apply whatever signal is desired to TDI and TCK.

Avoiding Inadvertent Boundary Scan

If TMS or TCK is used as user I/O, care must be taken to ensure that at least one of these pins is held constant during configuration. In some applications, a situation may occur where TMS or TCK is driven during configuration. This may cause the device to go into boundary scan mode and disrupt the configuration process.

To prevent activation of boundary scan during configuration, do either of the following:

- TMS: Tie High to put the Test Access Port controller in a benign RESET state
- TCK: Tie High or Low—do not toggle this clock input.

For more information regarding boundary scan, refer to the Xilinx Application Note XAPP 017, "*Boundary Scan in XC4000 and XC5200 Devices.*"

Power Distribution

Power for the FPGA is distributed through a grid to achieve high noise immunity and isolation between logic and I/O. Inside the FPGA, a dedicated Vcc and Ground ring surrounding the logic array provides power to the I/O drivers, as shown in Figure 21. An independent matrix of Vcc and Ground lines supplies the interior logic of the device.

This power distribution grid provides a stable supply and ground for all internal logic, providing the external package power pins are all connected and appropriately decoupled. Typically, a 0.1 µF capacitor connected near the Vcc and Ground pins of the package will provide adequate decoupling.

Output buffers capable of driving/sinking the specified 8 mA loads under specified worst-case conditions may be capable of driving/sinking up to 10 times as much current under best case conditions.

Noise can be reduced by minimizing external load capacitance and reducing simultaneous output transitions in the same direction. It may also be beneficial to locate heavily loaded output buffers near the Ground pads. The I/O Block output buffers have a slew-rate limited mode (default) which should be used where output rise and fall times are not speed-critical.

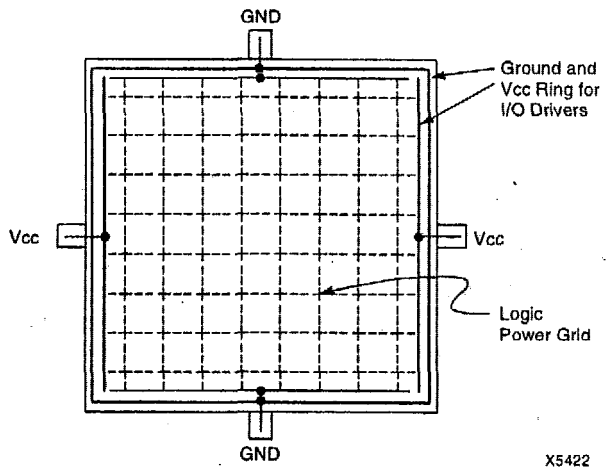


Figure 21: XC5200-Series Power Distribution

Pin Descriptions

There are three types of pins in the XC5200-Series devices:

- Permanently dedicated pins
- User I/O pins that can have special functions
- Unrestricted user-programmable I/O pins.

Before and during configuration, all outputs not used for the configuration process are 3-stated and pulled high with a 20 kΩ - 100 kΩ pull-up resistor.

After configuration, if an IOB is unused it is configured as an input with a 20 kΩ - 100 kΩ pull-up resistor.

Device pins for XC5200-Series devices are described in Table 9. Pin functions during configuration for each of the seven configuration modes are summarized in "Pin Func-

tions During Configuration" on page 124, in the "Configuration Timing" section.

Table 9: Pin Descriptions

Pin Name	I/O During Config.	I/O After Config.	Pin Description
Permanently Dedicated Pins			
VCC	I	I	Five or more (depending on package) connections to the nominal +5 V supply voltage. All must be connected, and each must be decoupled with a 0.01 - 0.1 µF capacitor to Ground.
GND	I	I	Four or more (depending on package type) connections to Ground. All must be connected.
CCLK	I or O	I	During configuration, Configuration Clock (CCLK) is an output in Master modes or Asynchronous Peripheral mode, but is an input in Slave mode, Synchronous Peripheral mode, and Express mode. After configuration, CCLK has a weak pull-up resistor and can be selected as the Readback Clock. There is no CCLK High time restriction on XC5200-Series devices, except during Readback. See "Violating the Maximum High and Low Time Specification for the Readback Clock" on page 113 for an explanation of this exception.
DONE	I/O	O	DONE is a bidirectional signal with an optional internal pull-up resistor. As an output, it indicates the completion of the configuration process. As an input, a Low level on DONE can be configured to delay the global logic initialization and the enabling of outputs. The exact timing, the clock source for the Low-to-High transition, and the optional pull-up resistor are selected as options in the program that creates the configuration bitstream. The resistor is included by default.
$\overline{\text{PROGRAM}}$	I	I	PROGRAM is an active Low input that forces the FPGA to clear its configuration memory. It is used to initiate a configuration cycle. When $\overline{\text{PROGRAM}}$ goes High, the FPGA executes a complete clear cycle, before it goes into a WAIT state and releases $\overline{\text{INIT}}$. The $\overline{\text{PROGRAM}}$ pin has an optional weak pull-up after configuration.
User I/O Pins That Can Have Special Functions			
RDY/$\overline{\text{BUSY}}$	O	I/O	During Peripheral mode configuration, this pin indicates when it is appropriate to write another byte of data into the FPGA. The same status is also available on D7 in Asynchronous Peripheral mode, if a read operation is performed when the device is selected. After configuration, RDY/$\overline{\text{BUSY}}$ is a user-programmable I/O pin. RDY/$\overline{\text{BUSY}}$ is pulled High with a high-impedance pull-up prior to $\overline{\text{INIT}}$ going High.
$\overline{\text{RCLK}}$	O	I/O	During Master Parallel configuration, each change on the A0-A17 outputs is preceded by a rising edge on $\overline{\text{RCLK}}$, a redundant output signal. $\overline{\text{RCLK}}$ is useful for clocked PROMs. It is rarely used during configuration. After configuration, $\overline{\text{RCLK}}$ is a user-programmable I/O pin.
M0, M1, M2	I	I/O	As Mode inputs, these pins are sampled before the start of configuration to determine the configuration mode to be used. After configuration, M0, M1, and M2 become user-programmable I/O. During configuration, these pins have weak pull-up resistors. For the most popular configuration mode, Slave Serial, the mode pins can thus be left unconnected. A pull-down resistor value of 3.3 kΩ is recommended for other modes.
TDO	O	O	If boundary scan is used, this pin is the Test Data Output. If boundary scan is not used, this pin is a 3-state output, after configuration is completed. This pin can be user output only when called out by special schematic definitions. To use this pin, place the library component TDO instead of the usual pad symbol. An output buffer must still be used.

Table 9: Pin Descriptions (Continued)

Pin Name	I/O During Config.	I/O After Config.	Pin Description
TDI, TCK, TMS	I	I/O or I (JTAG)	If boundary scan is used, these pins are Test Data In, Test Clock, and Test Mode Select inputs respectively. They come directly from the pads, bypassing the IOBs. These pins can also be used as inputs to the CLB logic after configuration is completed. If the BSCAN symbol is not placed in the design, all boundary scan functions are inhibited once configuration is completed, and these pins become user-programmable I/O. In this case, they must be called out by special schematic definitions. To use these pins, place the library components TDI, TCK, and TMS instead of the usual pad symbols. Input or output buffers must still be used.
HDC	O	I/O	High During Configuration (HDC) is driven High until the I/O go active. It is available as a control output indicating that configuration is not yet completed. After configuration, HDC is a user-programmable I/O pin.
$\overline{\text{LDC}}$	O	I/O	Low During Configuration ($\overline{\text{LDC}}$) is driven Low until the I/O go active. It is available as a control output indicating that configuration is not yet completed. After configuration, $\overline{\text{LDC}}$ is a user-programmable I/O pin.
$\overline{\text{INIT}}$	I/O	I/O	Before and during configuration, $\overline{\text{INIT}}$ is a bidirectional signal. A 1 kΩ - 10 kΩ external pull-up resistor is recommended. As an active-Low open-drain output, $\overline{\text{INIT}}$ is held Low during the power stabilization and internal clearing of the configuration memory. As an active-Low input, it can be used to hold the FPGA in the internal WAIT state before the start of configuration. Master mode devices stay in a WAIT state an additional 50 to 250 µs after $\overline{\text{INIT}}$ has gone High. During configuration, a Low on this output indicates that a configuration data error has occurred. After the I/O go active, $\overline{\text{INIT}}$ is a user-programmable I/O pin.
GCK1 - GCK4	Weak Pull-up	I or I/O	Four Global inputs each drive a dedicated internal global net with short delay and minimal skew. These internal global nets can also be driven from internal logic. If not used to drive a global net, any of these pins is a user-programmable I/O pin. The GCK1-GCK4 pins provide the shortest path to the four Global Buffers. Any input pad symbol connected directly to the input of a BUFG symbol is automatically placed on one of these pins.
$\overline{\text{CS0}}$, CS1, $\overline{\text{WS}}$, $\overline{\text{RS}}$	I	I/O	These four inputs are used in Asynchronous Peripheral mode. The chip is selected when $\overline{\text{CS0}}$ is Low and CS1 is High. While the chip is selected, a Low on Write Strobe ($\overline{\text{WS}}$) loads the data present on the D0 - D7 inputs into the internal data buffer. A Low on Read Strobe ($\overline{\text{RS}}$) changes D7 into a status output — High if Ready, Low if Busy — and drives D0 - D6 High. In Express mode, CS1 is used as a serial-enable signal for daisy-chaining. $\overline{\text{WS}}$ and $\overline{\text{RS}}$ should be mutually exclusive, but if both are Low simultaneously, the Write Strobe overrides. After configuration, these are user-programmable I/O pins.
A0 - A17	O	I/O	During Master Parallel configuration, these 18 output pins address the configuration EPROM. After configuration, they are user-programmable I/O pins.
D0 - D7	I	I/O	During Master Parallel, Peripheral, and Express configuration, these eight input pins receive configuration data. After configuration, they are user-programmable I/O pins.
DIN	I	I/O	During Slave Serial or Master Serial configuration, DIN is the serial configuration data input receiving data on the rising edge of CCLK. During Parallel configuration, DIN is the D0 input. After configuration, DIN is a user-programmable I/O pin.
DOUT	O	I/O	During configuration in any mode but Express mode, DOUT is the serial configuration data output that can drive the DIN of daisy-chained slave FPGAs. DOUT data changes on the falling edge of CCLK. In Express mode, DOUT is the status output that can drive the CS1 of daisy-chained FPGAs, to enable and disable downstream devices. After configuration, DOUT is a user-programmable I/O pin.

Table 9: Pin Descriptions (Continued)

Pin Name	I/O During Config.	I/O After Config.	Pin Description
Unrestricted User-Programmable I/O Pins			
I/O	Weak Pull-up	I/O	These pins can be configured to be input and/or output after configuration is completed. Before configuration is completed, these pins have an internal high-value pull-up resistor (20 kΩ - 100 kΩ) that defines the logic level as High.

Configuration

Configuration is the process of loading design-specific programming data into one or more FPGAs to define the functional operation of the internal blocks and their interconnections. This is somewhat like loading the command registers of a programmable peripheral chip. XC5200-Series devices use several hundred bits of configuration data per CLB and its associated interconnects. Each configuration bit defines the state of a static memory cell that controls either a function look-up table bit, a multiplexer input, or an interconnect pass transistor. The development system translates the design into a netlist file. It automatically partitions, places and routes the logic and generates the configuration data in PROM format.

Special Purpose Pins

Three configuration mode pins (M2, M1, M0) are sampled prior to configuration to determine the configuration mode. After configuration, these pins can be used as auxiliary I/O connections. The development system does not use these resources unless they are explicitly specified in the design entry. This is done by placing a special pad symbol called MD2, MD1, or MD0 instead of the input or output pad symbol.

In XC5200-Series devices, the mode pins have weak pull-up resistors during configuration. With all three mode pins High, Slave Serial mode is selected, which is the most popular configuration mode. Therefore, for the most common configuration mode, the mode pins can be left unconnected. (Note, however, that the internal pull-up resistor value can be as high as 100 kΩ.) After configuration, these pins can individually have weak pull-up or pull-down resistors, as specified in the design. A pull-down resistor value of 3.3kΩ is recommended.

These pins are located in the lower left chip corner and are near the readback nets. This location allows convenient routing if compatibility with the XC2000 and XC3000 family conventions of M0/RT, M1/RD is desired.

Configuration Modes

XC5200 devices have seven configuration modes. These modes are selected by a 3-bit input code applied to the M2, M1, and M0 inputs. There are three self-loading Master modes, two Peripheral modes, and a Serial Slave mode,

Table 10: Configuration Modes

Mode	M2	M1	M0	CCLK	Data
Master Serial	0	0	0	output	Bit-Serial
Slave Serial	1	1	1	input	Bit-Serial
Master Parallel Up	1	0	0	output	Byte-Wide, increment from 00000
Master Parallel Down	1	1	0	output	Byte-Wide, decrement from 3FFFF
Peripheral Synchronous*	0	1	1	input	Byte-Wide
Peripheral Asynchronous	1	0	1	output	Byte-Wide
Express	0	1	0	input	Byte-Wide
Reserved	0	0	1	—	—

Note :*Peripheral Synchronous can be considered byte-wide Slave Parallel

which is used primarily for daisy-chained devices. The seventh mode, called Express mode, is an additional slave mode that allows high-speed parallel configuration. The coding for mode selection is shown in Table 10.

Note that the smallest package, VQ64, only supports the Master Serial, Slave Serial, and Express modes.A detailed description of each configuration mode, with timing information, is included later in this data sheet. During configuration, some of the I/O pins are used temporarily for the configuration process. All pins used during configuration are shown in Table 13 on page 124.

Master Modes

The three Master modes use an internal oscillator to generate a Configuration Clock (CCLK) for driving potential slave devices. They also generate address and timing for external PROM(s) containing the configuration data.

Master Parallel (Up or Down) modes generate the CCLK signal and PROM addresses and receive byte parallel data. The data is internally serialized into the FPGA data-frame format. The up and down selection generates starting addresses at either zero or 3FFFF, for compatibility with different microprocessor addressing conventions. The

Master Serial mode generates CCLK and receives the configuration data in serial form from a Xilinx serial-configuration PROM.

CCLK speed is selectable as 1 MHz (default), 6 MHz, or 12 MHz. Configuration always starts at the default slow frequency, then can switch to the higher frequency during the first frame. Frequency tolerance is -50% to +50%.

Peripheral Modes

The two Peripheral modes accept byte-wide data from a bus. A RDY/$\overline{\text{BUSY}}$ status is available as a handshake signal. In Asynchronous Peripheral mode, the internal oscillator generates a CCLK burst signal that serializes the byte-wide data. CCLK can also drive slave devices. In the synchronous mode, an externally supplied clock input to CCLK serializes the data.

Slave Serial Mode

In Slave Serial mode, the FPGA receives serial configuration data on the rising edge of CCLK and, after loading its configuration, passes additional data out, resynchronized on the next falling edge of CCLK.

Multiple slave devices with identical configurations can be wired with parallel DIN inputs. In this way, multiple devices can be configured simultaneously.

Serial Daisy Chain

Multiple devices with different configurations can be connected together in a "daisy chain," and a single combined bitstream used to configure the chain of slave devices.

To configure a daisy chain of devices, wire the CCLK pins of all devices in parallel, as shown in Figure 28 on page 114. Connect the DOUT of each device to the DIN of the next. The lead or master FPGA and following slaves each passes resynchronized configuration data coming from a single source. The header data, including the length count, is passed through and is captured by each FPGA when it recognizes the 0010 preamble. Following the length-count data, each FPGA outputs a High on DOUT until it has received its required number of data frames.

After an FPGA has received its configuration data, it passes on any additional frame start bits and configuration data on DOUT. When the total number of configuration clocks applied after memory initialization equals the value of the 24-bit length count, the FPGAs begin the start-up sequence and become operational together. FPGA I/O are normally released two CCLK cycles after the last configuration bit is received. Figure 25 on page 109 shows the start-up timing for an XC5200-Series device.

The daisy-chained bitstream is not simply a concatenation of the individual bitstreams. The PROM file formatter must be used to combine the bitstreams for a daisy-chained configuration.

Multi-Family Daisy Chain

All Xilinx FPGAs of the XC2000, XC3000, XC4000, and XC5200 Series use a compatible bitstream format and can, therefore, be connected in a daisy chain in an arbitrary sequence. There is, however, one limitation. If the chain contains XC5200-Series devices, the master normally cannot be an XC2000 or XC3000 device.

The reason for this rule is shown in Figure 25 on page 109. Since all devices in the chain store the same length count value and generate or receive one common sequence of CCLK pulses, they all recognize length-count match on the same CCLK edge, as indicated on the left edge of Figure 25. The master device then generates additional CCLK pulses until it reaches its finish point F. The different families generate or require different numbers of additional CCLK pulses until they reach F. Not reaching F means that the device does not really finish its configuration, although DONE may have gone High, the outputs became active, and the internal reset was released. For the XC5200-Series device, not reaching F means that readback cannot be initiated and most boundary scan instructions cannot be used.

The user has some control over the relative timing of these events and can, therefore, make sure that they occur at the proper time and the finish point F is reached. Timing is controlled using options in the bitstream generation software.

XC5200 devices always have the same number of CCLKs in the power up delay, independent of the configuration mode, unlike the XC3000/XC4000 Series devices. To guarantee all devices in a daisy chain have finished the power-up delay, tie the INIT pins together, as shown in Figure 27.

XC3000 Master with an XC5200-Series Slave

Some designers want to use an XC3000 lead device in peripheral mode and have the I/O pins of the XC5200-Series devices all available for user I/O. Figure 22 provides a solution for that case.

This solution requires one CLB, one IOB and pin, and an internal oscillator with a frequency of up to 5 MHz as a clock source. The XC3000 master device must be configured with late Internal Reset, which is the default option.

One CLB and one IOB in the lead XC3000-family device are used to generate the additional CCLK pulse required by the XC5200-Series devices. When the lead device removes the internal RESET signal, the 2-bit shift register responds to its clock input and generates an active Low output signal for the duration of the subsequent clock period. An external connection between this output and CCLK thus creates the extra CCLK pulse.

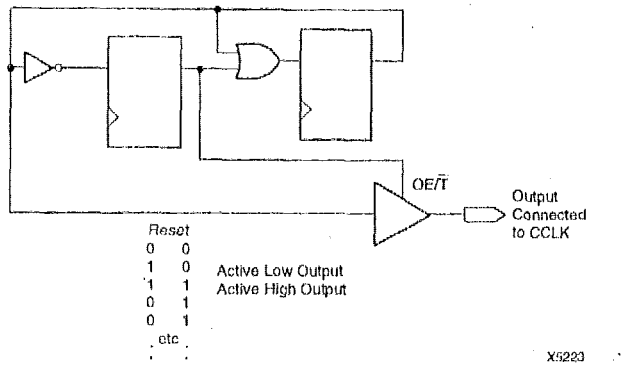


Figure 22: CCLK Generation for XC3000 Master Driving an XC5200-Series Slave

Express Mode

Express mode is similar to Slave Serial mode, except the data is presented in parallel format, and is clocked into the target device a byte at a time rather than a bit at a time. The data is loaded in parallel into eight different columns: it is not internally serialized. Eight bits of configuration data are loaded with every CCLK cycle, therefore this configuration mode runs at eight times the data rate of the other six modes. In this mode the XC5200 family is capable of supporting a CCLK frequency of 10 MHz, which is equivalent to an 80 MHz serial rate, because eight bits of configuration data are being loaded per CCLK cycle. An XC5210 in the Express mode, for instance, can be configured in about 2 ms. The Express mode does not support CRC error checking, but does support constant-field error checking. A length count is not used in Express mode.

In the Express configuration mode, an external signal drives the CCLK input(s). The first byte of parallel configuration data must be available at the D inputs of the FPGA devices a short set-up time before the second rising CCLK edge. Subsequent data bytes are clocked in on each consecutive rising CCLK edge. See Figure 38 on page 123.

Bitstream generation currently generates a bitstream sufficient to program in all configuration modes except Express. Extra CCLK cycles are necessary to complete the configuration, since in this mode data is read at a rate of eight bits per CCLK cycle instead of one bit per cycle. Normally the entire start-up sequence requires a number of bits that is equal to the number of CCLK cycles needed. An additional five CCLKs (equivalent to 40 extra bits) will guarantee completion of configuration, regardless of the start-up options chosen.

Multiple slave devices with identical configurations can be wired with parallel D0-D7 inputs. In this way, multiple devices can be configured simultaneously.

Pseudo Daisy Chain

Multiple devices with different configurations can be connected together in a pseudo daisy chain, provided that all of the devices are in Express mode. A single combined bitstream is used to configure the chain of Express mode devices, but the input data bus must drive D0-D7 of each device. Tie High the CS1 pin of the first device to be configured, or leave it floating in the XC5200 since it has an internal pull-up. Connect the DOUT pin of each FPGA to the CS1 pin of the next device in the chain. The D0-D7 inputs are wired to each device in parallel. The DONE pins are wired together, with one or more internal DONE pull-ups activated. Alternatively, a 4.7 kΩ external resistor can be used, if desired. (See Figure 37 on page 122.) CCLK pins are tied together.

The requirement that all DONE pins in a daisy chain be wired together applies only to Express mode, and only if all devices in the chain are to become active simultaneously. All devices in Express mode are synchronized to the DONE pin. User I/O for each device become active after the DONE pin for that device goes High. (The exact timing is determined by options to the bitstream generation software.) Since the DONE pin is open-drain and does not drive a High value, tying the DONE pins of all devices together prevents all devices in the chain from going High until the last device in the chain has completed its configuration cycle.

The status pin DOUT is pulled LOW two internal-oscillator cycles (nominally 1 MHz) after $\overline{\text{INIT}}$ is recognized as High, and remains Low until the device's configuration memory is full. Then DOUT is pulled High to signal the next device in the chain to accept the configuration data on the D7-D0 bus. All devices receive and recognize the six bytes of preamble and length count, irrespective of the level on CS1; but subsequent frame data is accepted only when CS1 is High and the device's configuration memory is not already full.

Setting CCLK Frequency

For Master modes, CCLK can be generated in one of three frequencies. In the default slow mode, the frequency is nominally 1 MHz. In fast CCLK mode, the frequency is nominally 12 MHz. In medium CCLK mode, the frequency is nominally 6 MHz. The frequency range is -50% to +50%. The frequency is selected by an option when running the bitstream generation software. If an XC5200-Series Master is driving an XC3000- or XC2000-family slave, slow CCLK mode must be used. Slow mode is the default.

Table 11: XC5200 Bitstream Format

Data Type	Value	Occurrences
Fill Byte	11111111	Once per bitstream
Preamble	11110010	
Length Counter	COUNT(23:0)	
Fill Byte	11111111	

Table 11: XC5200 Bitstream Format

Data Type	Value	Occurrences
Start Byte	11111110	Once per data frame
Data Frame *	DATA(N-1:0)	
Cyclic Redundancy Check or Constant Field Check	CRC(3:0) or 0110	
Fill Nibble	1111	
Extend Write Cycle	FFFFFF	
Postamble	11111110	Once per device
Fill Bytes (30)	FFFF...FF	
Start-Up Byte	FF	Once per bitstream
*Bits per Frame (N) depends on device size, as described for table 11.		

Data Stream Format

The data stream ("bitstream") format is identical for all configuration modes, with the exception of Express mode. In Express mode, the device becomes active when DONE goes High, therefore no length count is required. Additionally, CRC error checking is not supported in Express mode.

The data stream formats are shown in Table 11. Express mode data is shown with D0 at the left and D7 at the right. For all other modes, bit-serial data is read from left to right, and byte-parallel data is effectively assembled from this serial bitstream, with the first bit in each byte assigned to D0.

The configuration data stream begins with a string of eight ones, a preamble code, followed by a 24-bit length count and a separator field of ones (or 24 fill bits, in Express mode). This header is followed by the actual configuration data in frames. The length and number of frames depends on the device type (see Table 12). Each frame begins with a start field and ends with an error check. In all modes except Express mode, a postamble code is required to signal the end of data for a single device. In all cases, additional start-up bytes of data are required to provide four clocks for the startup sequence at the end of configuration. Long daisy chains require additional startup bytes to shift the last data through the chain. All startup bytes are don't-cares; these bytes are not included in bitstreams created by the Xilinx software.

In Express mode, only non-CRC error checking is supported. In all other modes, a selection of CRC or non-CRC error checking is allowed by the bitstream generation software. The non-CRC error checking tests for a designated end-of-frame field for each frame. For CRC error checking, the software calculates a running CRC and inserts a unique four-bit partial check at the end of each frame. The 11-bit CRC check of the last frame of an FPGA includes the last seven data bits.

Detection of an error results in the suspension of data loading and the pulling down of the $\overline{\text{INIT}}$ pin. In Master modes, CCLK and address signals continue to operate externally. The user must detect $\overline{\text{INIT}}$ and initialize a new configuration by pulsing the $\overline{\text{PROGRAM}}$ pin Low or cycling Vcc.

Table 12: Internal Configuration Data Structure

Device	VersaBlock Array	PROM Size (bits)	Xilinx Serial PROM Needed
XC5202	8 x 8	42,416	XC1765D
XC5204	10 x 12	70,704	XC17128D
XC5206	14 x 14	106,288	XC17128D
XC5210	18 x 18	165,488	XC17256D
XC5215	22 x 22	237,744	XC17256D

Bits per Frame = (34 x number of Rows) + 28 for the top + 28 for the bottom + 4 splitter bits + 8 start bits + 4 error check bits + 4 fill bits * + 24 extended write bits
= (34 x number of Rows) + 100
* In the XC5202 (8 x 8), there are 8 fill bits per frame, not 4

Number of Frames = (12 x number of Columns) + 7 for the left edge + 8 for the right edge + 1 splitter bit
= (12 x number of Columns) + 16

Program Data = (Bits per Frame x Number of Frames) + 48 header bits + 8 postamble bits + 240 fill bits + 8 start-up bits
= (Bits per Frame x Number of Frames) + 304

PROM Size = Program Data

Cyclic Redundancy Check (CRC) for Configuration and Readback

The Cyclic Redundancy Check is a method of error detection in data transmission applications. Generally, the transmitting system performs a calculation on the serial bitstream. The result of this calculation is tagged onto the data stream as additional check bits. The receiving system performs an identical calculation on the bitstream and compares the result with the received checksum.

Each data frame of the configuration bitstream has four error bits at the end, as shown in Table 11. If a frame data error is detected during the loading of the FPGA, the configuration process with a potentially corrupted bitstream is terminated. The FPGA pulls the $\overline{\text{INIT}}$ pin Low and goes into a Wait state.

During Readback, 11 bits of the 16-bit checksum are added to the end of the Readback data stream. The checksum is computed using the CRC-16 CCITT polynomial, as shown in Figure 23. The checksum consists of the 11 most significant bits of the 16-bit code. A change in the checksum indicates a change in the Readback bitstream. A comparison to a previous checksum is meaningful only if the readback data is independent of the current device state. CLB outputs should not be included (Read Capture option not used). Statistically, one error out of 2048 might go undetected.

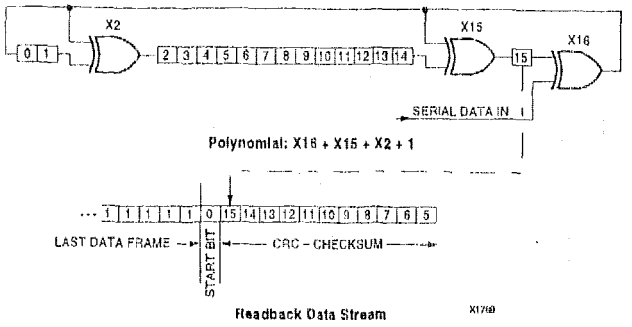


Figure 23: Circuit for Generating CRC-16

Configuration Sequence

There are four major steps in the XC5200-Series power-up configuration sequence.

- Power-On Time-Out
- Initialization
- Configuration
- Start-Up

The full process is illustrated in Figure 24.

Power-On Time-Out

An internal power-on reset circuit is triggered when power is applied. When V_{CC} reaches the voltage at which portions of the FPGA begin to operate (i.e., performs a write-and-read test of a sample pair of configuration memory bits), the programmable I/O buffers are 3-stated with active high-impedance pull-up resistors. A time-out delay — nominally 4 ms — is initiated to allow the power-supply voltage to stabilize. For correct operation the power supply must reach V_{CC}(min) by the end of the time-out, and must not dip below it thereafter.

There is no distinction between master and slave modes with regard to the time-out delay. Instead, the $\overline{INIT}$ line is used to ensure that all daisy-chained devices have completed initialization. Since XC2000 devices do not have this signal, extra care must be taken to guarantee proper operation when daisy-chaining them with XC5200 devices. For proper operation with XC3000 devices, the $\overline{RESET}$ signal, which is used in XC3000 to delay configuration, should be connected to $\overline{INIT}$.

If the time-out delay is insufficient, configuration should be delayed by holding the $\overline{INIT}$ pin Low until the power supply has reached operating levels.

This delay is applied only on power-up. It is not applied when reconfiguring an FPGA by pulsing the $\overline{PROGRAM}$ pin Low. During all three phases — Power-on, Initialization, and Configuration — DONE is held Low; HDC, $\overline{LDC}$, and $\overline{INIT}$ are active; DOUT is driven; and all I/O buffers are disabled.

Initialization

This phase clears the configuration memory and establishes the configuration mode.

The configuration memory is cleared at the rate of one frame per internal clock cycle (nominally 1 MHz). An open-drain bidirectional signal, $\overline{INIT}$, is released when the configuration memory is completely cleared. The device then tests for the absence of an external active-low level on $\overline{INIT}$. The mode lines are sampled two internal clock cycles later (nominally 2 µs).

The master device waits an additional 32 µs to 256 µs (nominally 64-128 µs) to provide adequate time for all of the slave devices to recognize the release of $\overline{INIT}$ as well. Then the master device enters the Configuration phase.

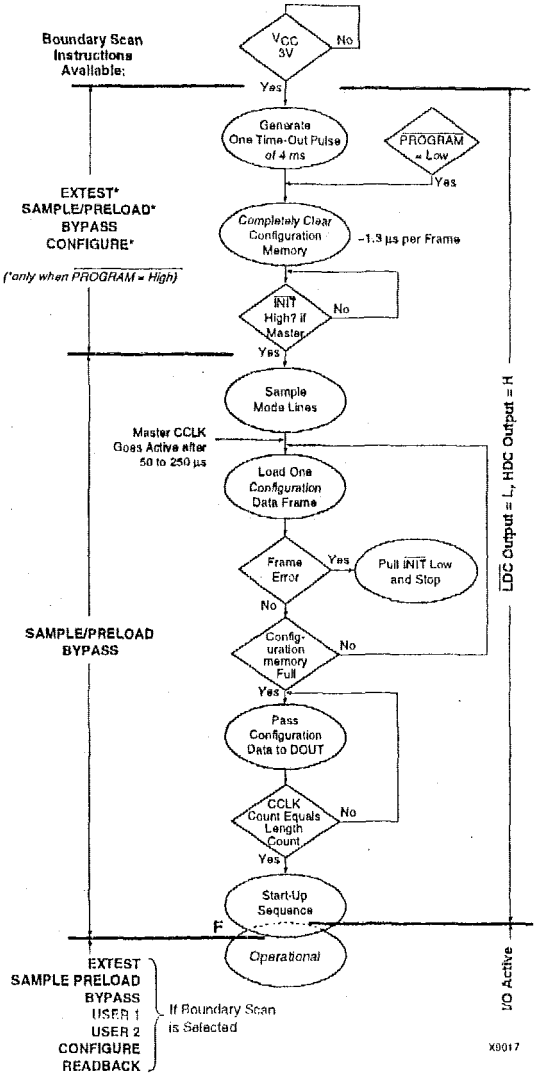


Figure 24: Configuration Sequence

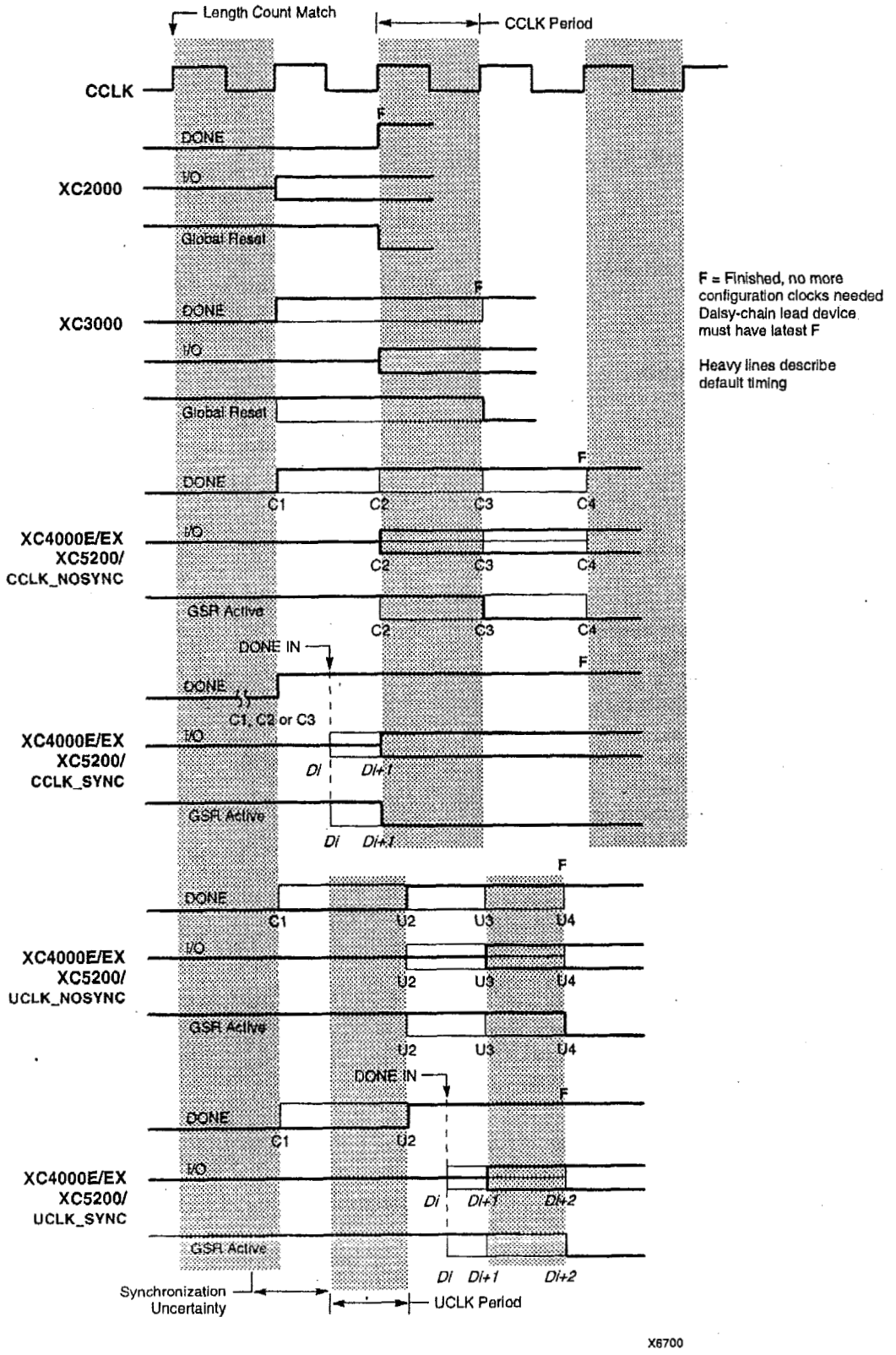


Figure 25: Start-up Timing

Configuration

The length counter begins counting immediately upon entry into the configuration state. In slave-mode operation it is important to wait at least two cycles of the internal 1-MHz clock oscillator after $\overline{INIT}$ is recognized before toggling CCLK and feeding the serial bitstream. Configuration will not begin until the internal configuration logic reset is released, which happens two cycles after $\overline{INIT}$ goes High. A master device's configuration is delayed from 32 to 256 µs to ensure proper operation with any slave devices driven by the master device.

The 0010 preamble code, included for all modes except Express mode, indicates that the following 24 bits represent the length count. The length count is the total number of configuration clocks needed to load the complete configuration data. (Four additional configuration clocks are required to complete the configuration process, as discussed below.) After the preamble and the length count have been passed through to all devices in the daisy chain, DOUT is held High to prevent frame start bits from reaching any daisy-chained devices. In Express mode, the length count bits are ignored, and DOUT is held Low, to disable the next device in the pseudo daisy chain.

A specific configuration bit, early in the first frame of a master device, controls the configuration-clock rate and can increase it by a factor of eight. Therefore, if a fast configuration clock is selected by the bitstream, the slower clock rate is used until this configuration bit is detected.

Each frame has a start field followed by the frame-configuration data bits and a frame error field. If a frame data error is detected, the FPGA halts loading, and signals the error by pulling the open-drain $\overline{INIT}$ pin Low. After all configuration frames have been loaded into an FPGA, DOUT again follows the input data so that the remaining data is passed on to the next device. In Express mode, when the first device is fully programmed, DOUT goes High to enable the next device in the chain.

Delaying Configuration After Power-Up

To delay master mode configuration after power-up, pull the bidirectional $\overline{INIT}$ pin Low, using an open-collector (open-drain) driver. (See Figure 12.)

Using an open-collector or open-drain driver to hold $\overline{INIT}$ Low before the beginning of master mode configuration causes the FPGA to wait after completing the configuration memory clear operation. When $\overline{INIT}$ is no longer held Low externally, the device determines its configuration mode by capturing its mode pins, and is ready to start the configuration process. A master device waits up to an additional 250 µs to make sure that any slaves in the optional daisy chain have seen that $\overline{INIT}$ is High.

Start-Up

Start-up is the transition from the configuration process to the intended user operation. This transition involves a change from one clock source to another, and a change from interfacing parallel or serial configuration data where most outputs are 3-stated, to normal operation with I/O pins active in the user-system. Start-up must make sure that the user-logic 'wakes up' gracefully, that the outputs become active without causing contention with the configuration signals, and that the internal flip-flops are released from the global Reset at the right time.

Figure 25 describes start-up timing for the three Xilinx families in detail. Express mode configuration always uses either CCLK_SYNC or UCLK_SYNC timing, the other configuration modes can use any of the four timing sequences.

To access the internal start-up signals, place the STARTUP library symbol.

Start-up Timing

Different FPGA families have different start-up sequences.

The XC2000 family goes through a fixed sequence. DONE goes High and the internal global Reset is de-activated one CCLK period after the I/O become active.

The XC3000A family offers some flexibility. DONE can be programmed to go High one CCLK period before or after the I/O become active. Independent of DONE, the internal global Reset is de-activated one CCLK period before or after the I/O become active.

The XC4000/XC5200 Series offers additional flexibility. The three events — DONE going High, the internal Reset being de-activated, and the user I/O going active — can all occur in any arbitrary sequence. Each of them can occur one CCLK period before or after, or simultaneous with, any of the others. This relative timing is selected by means of software options in the bitstream generation software.

The default option, and the most practical one, is for DONE to go High first, disconnecting the configuration data source and avoiding any contention when the I/Os become active one clock later. Reset is then released another clock period later to make sure that user-operation starts from stable internal conditions. This is the most common sequence, shown with heavy lines in Figure 25, but the designer can modify it to meet particular requirements.

Normally, the start-up sequence is controlled by the internal device oscillator output (CCLK), which is asynchronous to the system clock.

XC4000/XC5200 Series offers another start-up clocking option, UCLK_NOSYNC. The three events described above need not be triggered by CCLK. They can, as a configuration option, be triggered by a user clock. This means that the device can wake up in synchronism with the user system.

When the UCLK_SYNC option is enabled, the user can externally hold the open-drain DONE output Low, and thus stall all further progress in the start-up sequence until DONE is released and has gone High. This option can be used to force synchronization of several FPGAs to a common user clock, or to guarantee that all devices are successfully configured before any I/Os go active.

If either of these two options is selected, and no user clock is specified in the design or attached to the device, the chip could reach a point where the configuration of the device is complete and the Done pin is asserted, but the outputs do not become active. The solution is either to recreate the bitstream specifying the start-up clock as CCLK, or to supply the appropriate user clock.

Start-up Sequence

The Start-up sequence begins when the configuration memory is full, and the total number of configuration clocks received since $\overline{\text{INIT}}$ went High equals the loaded value of the length count.

The next rising clock edge sets a flip-flop Q0, shown in Figure 26. Q0 is the leading bit of a 5-bit shift register. The outputs of this register can be programmed to control three events.

- The release of the open-drain DONE output
- The change of configuration-related pins to the user function, activating all IOBs.
- The termination of the global Set/Reset initialization of all CLB and IOB storage elements.

The DONE pin can also be wire-ANDed with DONE pins of other FPGAs or with other external signals, and can then be used as input to bit Q3 of the start-up register. This is called "Start-up Timing Synchronous to Done In" and is selected by either CCLK_SYNC or UCLK_SYNC.

When DONE is not used as an input, the operation is called "Start-up Timing Not Synchronous to DONE In," and is selected by either CCLK_NOSYNC or UCLK_NOSYNC.

As a configuration option, the start-up control register beyond Q0 can be clocked either by subsequent CCLK pulses or from an on-chip user net called STARTUP.CLK. These signals can be accessed by placing the STARTUP library symbol.

Start-up from CCLK

If CCLK is used to drive the start-up, Q0 through Q3 provide the timing. Heavy lines in Figure 25 show the default timing, which is compatible with XC2000 and XC3000 devices using early DONE and late Reset. The thin lines indicate all other possible timing options.

Start-up from a User Clock (STARTUP.CLK)

When, instead of CCLK, a user-supplied start-up clock is selected, Q1 is used to bridge the unknown phase relationship between CCLK and the user clock. This arbitration causes an unavoidable one-cycle uncertainty in the timing of the rest of the start-up sequence.

DONE Goes High to Signal End of Configuration

In all configuration modes except Express mode, XC5200-Series devices read the expected length count from the bitstream and store it in an internal register. The length count varies according to the number of devices and the composition of the daisy chain. Each device also counts the number of CCLKs during configuration.

Two conditions have to be met in order for the DONE pin to go high:

- the chip's internal memory must be full, and
- the configuration length count must be met, *exactly.*

This is important because the counter that determines when the length count is met begins with the very first CCLK, not the first one after the preamble.

Therefore, if a stray bit is inserted before the preamble, or the data source is not ready at the time of the first CCLK, the internal counter that holds the number of CCLKs will be one ahead of the actual number of data bits read. At the end of configuration, the configuration memory will be full, but the number of bits in the internal counter will not match the expected length count.

As a consequence, a Master mode device will continue to send out CCLKs until the internal counter turns over to zero, and then reaches the correct length count a second time. This will take several seconds [2^{24} * CCLK period] — which is sometimes interpreted as the device not configuring at all.

If it is not possible to have the data ready at the time of the first CCLK, the problem can be avoided by increasing the number in the length count by the appropriate value.

In Express mode, there is no length count. The DONE pin for each device goes High when the device has received its quota of configuration data. Wiring the DONE pins of several devices together delays start-up of all devices until all are fully configured.

Note that DONE is an open-drain output and does not go High unless an internal pull-up is activated or an external pull-up is attached. The internal pull-up is activated as the default by the bitstream generation software.

Release of User I/O After DONE Goes High

By default, the user I/O are released one CCLK cycle after the DONE pin goes High. If CCLK is not clocked after DONE goes High, the outputs remain in their initial state — 3-stated, with a 20 kΩ - 100 kΩ pull-up. The delay from

DONE High to active user I/O is controlled by an option to the bitstream generation software.

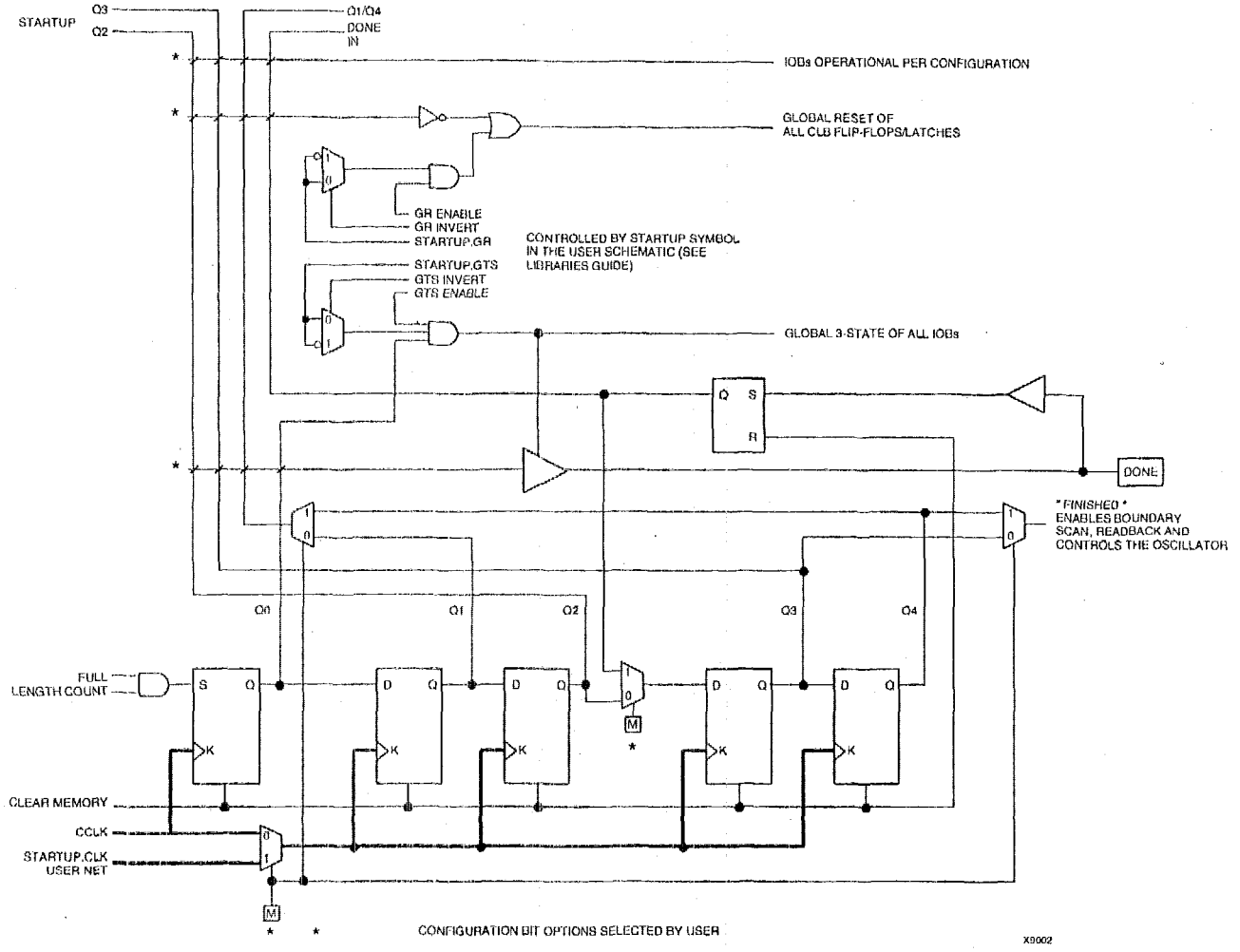


Figure 26: Start-up Logic

Release of Global Reset After DONE Goes High

By default, Global Reset (GR) is released two CCLK cycles after the DONE pin goes High. If CCLK is not clocked twice after DONE goes High, all flip-flops are held in their initial reset state. The delay from DONE High to GR inactive is controlled by an option to the bitstream generation software.

Configuration Complete After DONE Goes High

Three full CCLK cycles are required after the DONE pin goes High, as shown in Figure 25 on page 109. If CCLK is not clocked three times after DONE goes High, readback cannot be initiated and most boundary scan instructions cannot be used.

Configuration Through the Boundary Scan Pins

XC5200-Series devices can be configured through the boundary scan pins.

For detailed information, refer to the Xilinx application note XAPP017, "*Boundary Scan in XC4000 and XC5200 Devices*."

Readback

The user can read back the content of configuration memory and the level of certain internal nodes without interfering with the normal operation of the device.

Readback not only reports the downloaded configuration bits, but can also include the present state of the device, represented by the content of all flip-flops and latches in CLBs.

Note that in XC5200-Series devices, configuration data is *not* inverted with respect to configuration as it is in XC2000 and XC3000 families.

Readback of Express mode bitstreams results in data that does not resemble the original bitstream, because the bitstream format differs from other modes.

XC5200-Series Readback does not use any dedicated pins, but uses four internal nets (RDBK.TRIG, RDBK.DATA, RDBK.RIP and RDBK.CLK) that can be routed to any IOB. To access the internal Readback signals, place the READBACK library symbol and attach the appropriate pad symbols, as shown in Figure 27.

After Readback has been initiated by a Low-to-High transition on RDBK.TRIG, the RDBK.RIP (Read In Progress) output goes High on the next rising edge of RDBK.CLK. Subsequent rising edges of this clock shift out Readback data on the RDBK.DATA net.

Readback data does not include the preamble, but starts with five dummy bits (all High) followed by the Start bit (Low) of the first frame. The first two data bits of the first frame are always High.

Each frame ends with four error check bits. They are read back as High. The last seven bits of the last frame are also read back as High. An additional Start bit (Low) and an 11-bit Cyclic Redundancy Check (CRC) signature follow, before RDBK.RIP returns Low.

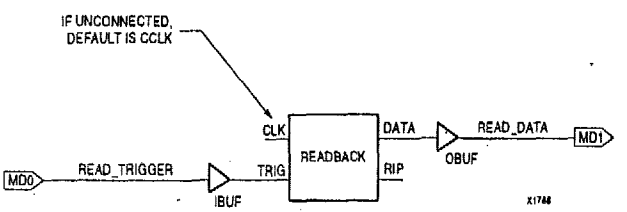


Figure 27: Readback Schematic Example

Readback Options

Readback options are: Read Capture, Read Abort, and Clock Select. They are set with the bitstream generation software.

Read Capture

When the Read Capture option is selected, the readback data stream includes sampled values of CLB and IOB signals. The rising edge of RDBK.TRIG latches the inverted values of the CLB outputs and the IOB output and input signals. Note that while the bits describing configuration (interconnect and function generators) are *not* inverted, the CLB and IOB output signals *are* inverted.

When the Read Capture option is not selected, the values of the capture bits reflect the configuration data originally written to those memory locations.

The readback signals are located in the lower-left corner of the device.

Read Abort

When the Read Abort option is selected, a High-to-Low transition on RDBK.TRIG terminates the readback operation and prepares the logic to accept another trigger.

After an aborted readback, additional clocks (up to one readback clock per configuration frame) may be required to re-initialize the control logic. The status of readback is indicated by the output control net RDBK.RIP. RDBK.RIP is High whenever a readback is in progress.

Clock Select

CCLK is the default clock. However, the user can insert another clock on RDBK.CLK. Readback control and data are clocked on rising edges of RDBK.CLK. If readback must be inhibited for security reasons, the readback control nets are simply not connected.

Violating the Maximum High and Low Time Specification for the Readback Clock

The readback clock has a maximum High and Low time specification. In some cases, this specification cannot be met. For example, if a processor is controlling readback, an interrupt may force it to stop in the middle of a readback. This necessitates stopping the clock, and thus violating the specification.

The specification is mandatory only on clocking data at the end of a frame prior to the next start bit. The transfer mechanism will load the data to a shift register during the last six clock cycles of the frame, prior to the start bit of the following frame. This loading process is dynamic, and is the source of the maximum High and Low time requirements.

Therefore, the specification only applies to the six clock cycles prior to and including any start bit, including the clocks before the first start bit in the readback data stream. At other times, the frame data is already in the register and the register is not dynamic. Thus, it can be shifted out just like a regular shift register.

The user must precisely calculate the location of the readback data relative to the frame. The system must keep track of the position within a data frame, and disable interrupts before frame boundaries. Frame lengths and data formats are listed in Table 11 and Table 12.

Readback with the XChecker Cable

The XChecker Universal Download/Readback Cable and Logic Probe uses the readback feature for bitstream verification. It can also display selected internal signals on the PC or workstation screen, functioning as a low-cost in-circuit emulator.

Configuration Timing

The seven configuration modes are discussed in detail in this section. Timing specifications are included.

Slave Serial Mode

In Slave Serial mode, an external signal drives the CCLK input of the FPGA. The serial configuration bitstream must be available at the DIN input of the lead FPGA a short setup time before each rising CCLK edge.

The lead FPGA then presents the preamble data—and all data that overflows the lead device—on its DOUT pin.

There is an internal delay of 0.5 CCLK periods, which means that DOUT changes on the falling CCLK edge, and the next FPGA in the daisy chain accepts data on the subsequent rising CCLK edge.

Figure 28 shows a full master/slave system. An XC5200-Series device in Slave Serial mode should be connected as shown in the third device from the left.

Slave Serial mode is selected by a <111> on the mode pins (M2, M1, M0). Slave Serial is the default mode if the mode pins are left unconnected, as they have weak pull-up resistors during configuration.

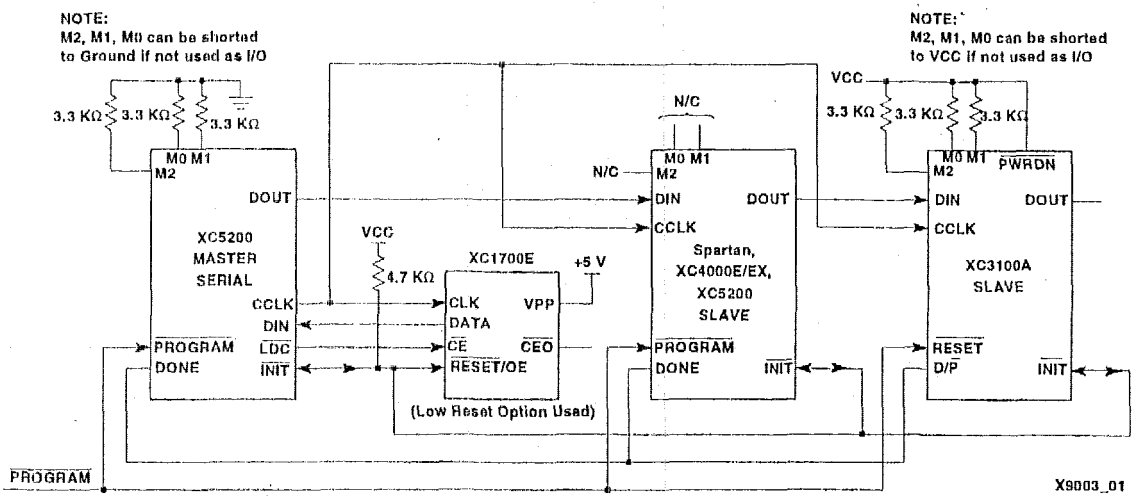
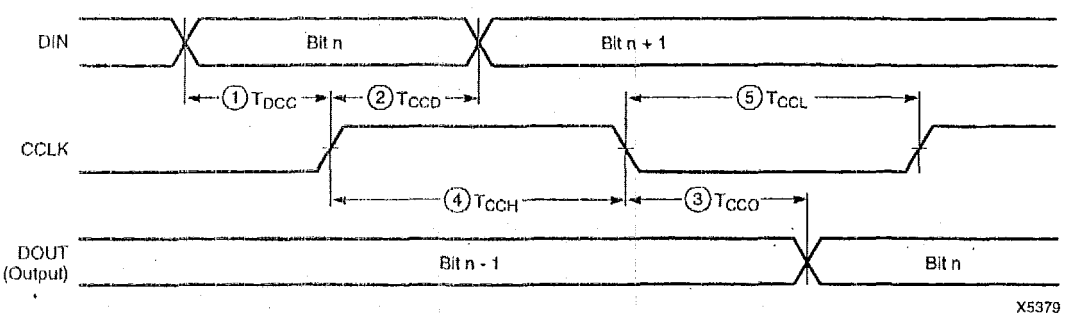


Figure 28: Master/Slave Serial Mode Circuit Diagram



	Description	Symbol		Min	Max	Units
CCLK	DIN setup	1	T_{DCC}	20		ns
	DIN hold	2	T_{CCD}	0		ns
	DIN to DOUT	3	T_{CCO}		30	ns
	High time	4	T_{CCH}	45		ns
	Low time	5	T_{CCL}	45		ns
	Frequency		F_{CC}		10	MHz

Note: Configuration must be delayed until the $\overline{INIT}$ pins of all daisy-chained FPGAs are High.

Figure 29: Slave Serial Mode Programming Switching Characteristics

Master Serial Mode

In Master Serial mode, the CCLK output of the lead FPGA drives a Xilinx Serial PROM that feeds the FPGA DIN input. Each rising edge of the CCLK output increments the Serial PROM internal address counter. The next data bit is put on the SPROM data output, connected to the FPGA DIN pin. The lead FPGA accepts this data on the subsequent rising CCLK edge.

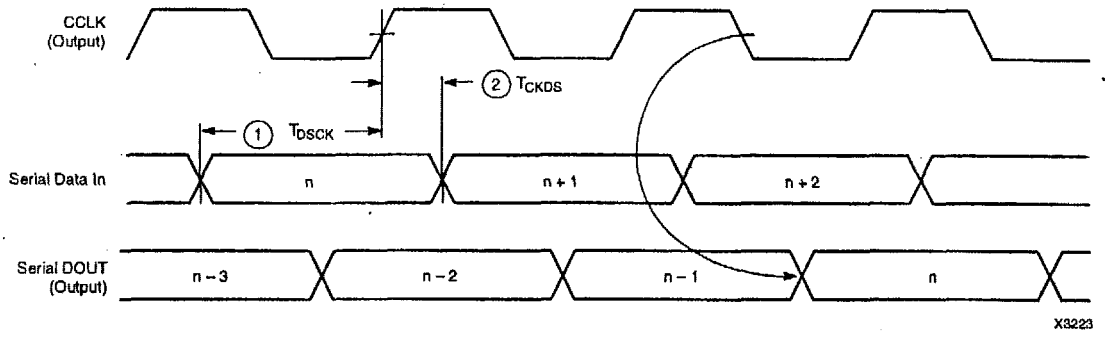
The lead FPGA then presents the preamble data—and all data that overflows the lead device—on its DOUT pin. There is an internal pipeline delay of 1.5 CCLK periods, which means that DOUT changes on the falling CCLK edge, and the next FPGA in the daisy chain accepts data on the subsequent rising CCLK edge.

In the bitstream generation software, the user can specify Fast ConfigRate, which, starting several bits into the first frame, increases the CCLK frequency by a factor of twelve. The value increases from a nominal 1 MHz, to a nominal 12 MHz. Be sure that the serial PROM and slaves are fast enough to support this data rate. The Medium ConfigRate option changes the frequency to a nominal 6 MHz. XC2000, XC3000/A, and XC3100A devices do not support the Fast or Medium ConfigRate options.

The SPROM CE input can be driven from either $\overline{LDC}$ or DONE. Using $\overline{LDC}$ avoids potential contention on the DIN pin, if this pin is configured as user-I/O, but $\overline{LDC}$ is then restricted to be a permanently High user output after configuration. Using DONE can also avoid contention on DIN, provided the DONE before I/O enable option is invoked.

Figure 28 on page 114 shows a full master/slave system. The leftmost device is in Master Serial mode.

Master Serial mode is selected by a <000> on the mode pins (M2, M1, M0).



	Description	Symbol		Min	Max	Units
CCLK	DIN setup	1	T_{DSCK}	20		ns
	DIN hold	2	T_{CKDS}	0		ns

Notes: 1. At power-up, Vcc must rise from 2.0 V to Vcc min in less than 25 ms, otherwise delay configuration by pulling $\overline{PROGRAM}$ Low until Vcc is valid.
2. Master Serial mode timing is based on testing in slave mode.

Figure 30: Master Serial Mode Programming Switching Characteristics

In the two Master Parallel modes, the lead FPGA directly addresses an industry-standard byte-wide EPROM, and accepts eight data bits just before incrementing or decrementing the address outputs.

The eight data bits are serialized in the lead FPGA, which then presents the preamble data—and all data that overflows the lead device—on its DOUT pin. There is an internal delay of 1.5 CCLK periods, after the rising CCLK edge that accepts a byte of data (and also changes the EPROM address) until the falling CCLK edge that makes the LSB (D0) of this byte appear at DOUT. This means that DOUT changes on the falling CCLK edge, and the next FPGA in the daisy chain accepts data on the subsequent rising CCLK edge.

The PROM address pins can be incremented or decremented, depending on the mode pin settings. This option allows the FPGA to share the PROM with a wide variety of microprocessors and microcontrollers. Some processors must boot from the bottom of memory (all zeros) while others must boot from the top. The FPGA is flexible and can load its configuration bitstream from either end of the memory.

Master Parallel Up mode is selected by a <100> on the mode pins (M2, M1, M0). The EPROM addresses start at 00000 and increment.

Master Parallel Down mode is selected by a <110> on the mode pins. The EPROM addresses start at 3FFFF and decrement.

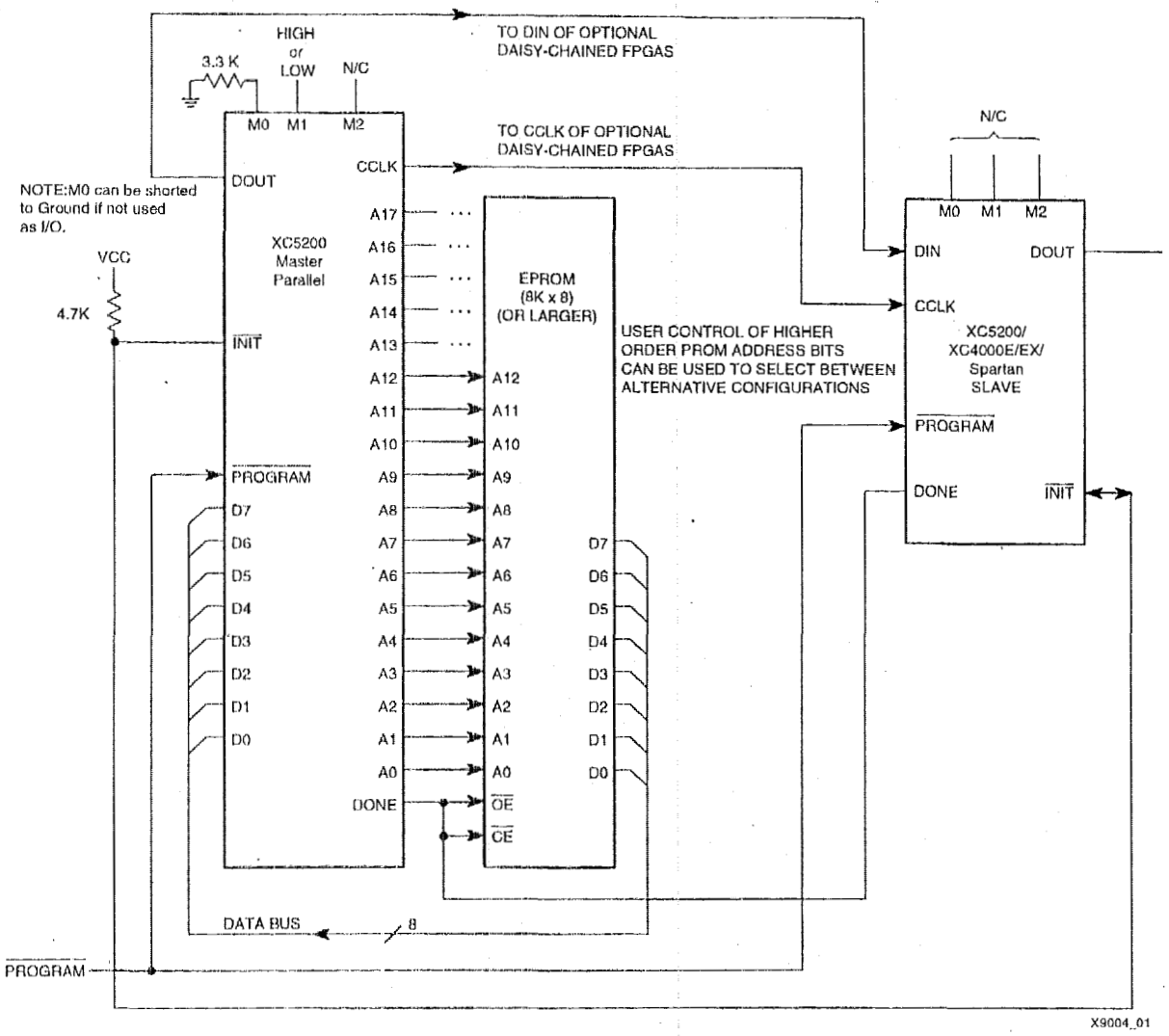
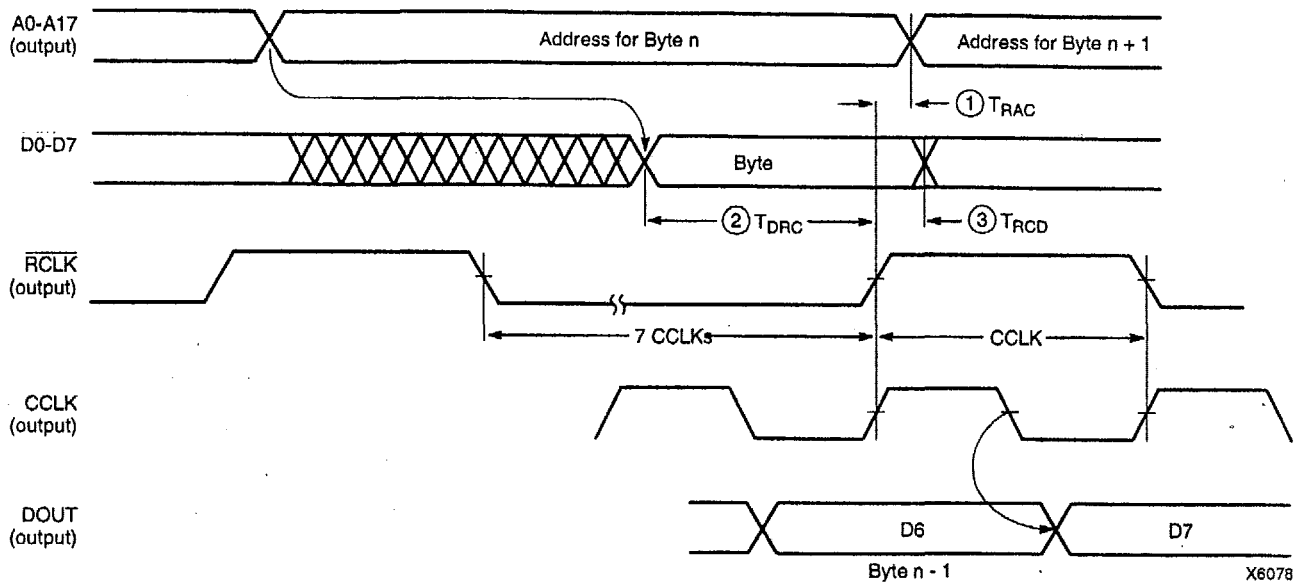


Figure 31: Master Parallel Mode Circuit Diagram



	Description	Symbol		Min	Max	Units
CCLK	Delay to Address valid	1	T_{RAC}	0	200	ns
	Data setup time	2	T_{DRC}	60		ns
	Data hold time	3	T_{RCD}	0		ns

Note: 1. At power-up, V_{CC} must rise from 2.0 V to V_{CC} min in less then 25 ms, otherwise delay configuration by pulling $\overline{\text{PROGRAM}}$ Low until V_{CC} is Valid.
2. The first Data byte is loaded and CCLK starts at the end of the first $\overline{\text{RCLK}}$ active cycle (rising edge).

This timing diagram shows that the EPROM requirements are extremely relaxed. EPROM access time can be longer than 500 ns. EPROM data output has no hold-time requirements.

Figure 32: Master Parallel Mode Programming Switching Characteristics

Synchronous Peripheral Mode

Synchronous Peripheral mode can also be considered Slave Parallel mode. An external signal drives the CCLK input(s) of the FPGA(s). The first byte of parallel configuration data must be available at the Data inputs of the lead FPGA a short setup time before the rising CCLK edge. Subsequent data bytes are clocked in on every eighth consecutive rising CCLK edge.

The same CCLK edge that accepts data, also causes the RDY/$\overline{\text{BUSY}}$ output to go High for one CCLK period. The pin name is a misnomer. In Synchronous Peripheral mode it is really an ACKNOWLEDGE signal. Synchronous operation does not require this response, but it is a meaningful signal for test purposes. Note that RDY/$\overline{\text{BUSY}}$ is pulled High with a high-impedance pullup prior to $\overline{\text{INIT}}$ going High.

The lead FPGA serializes the data and presents the preamble data (and all data that overflows the lead device) on its DOUT pin. There is an internal delay of 1.5 CCLK periods, which means that DOUT changes on the falling CCLK edge, and the next FPGA in the daisy chain accepts data on the subsequent rising CCLK edge.

In order to complete the serial shift operation, 10 additional CCLK rising edges are required after the last data byte has been loaded, plus one more CCLK cycle for each daisy-chained device.

Synchronous Peripheral mode is selected by a <011> on the mode pins (M2, M1, M0).

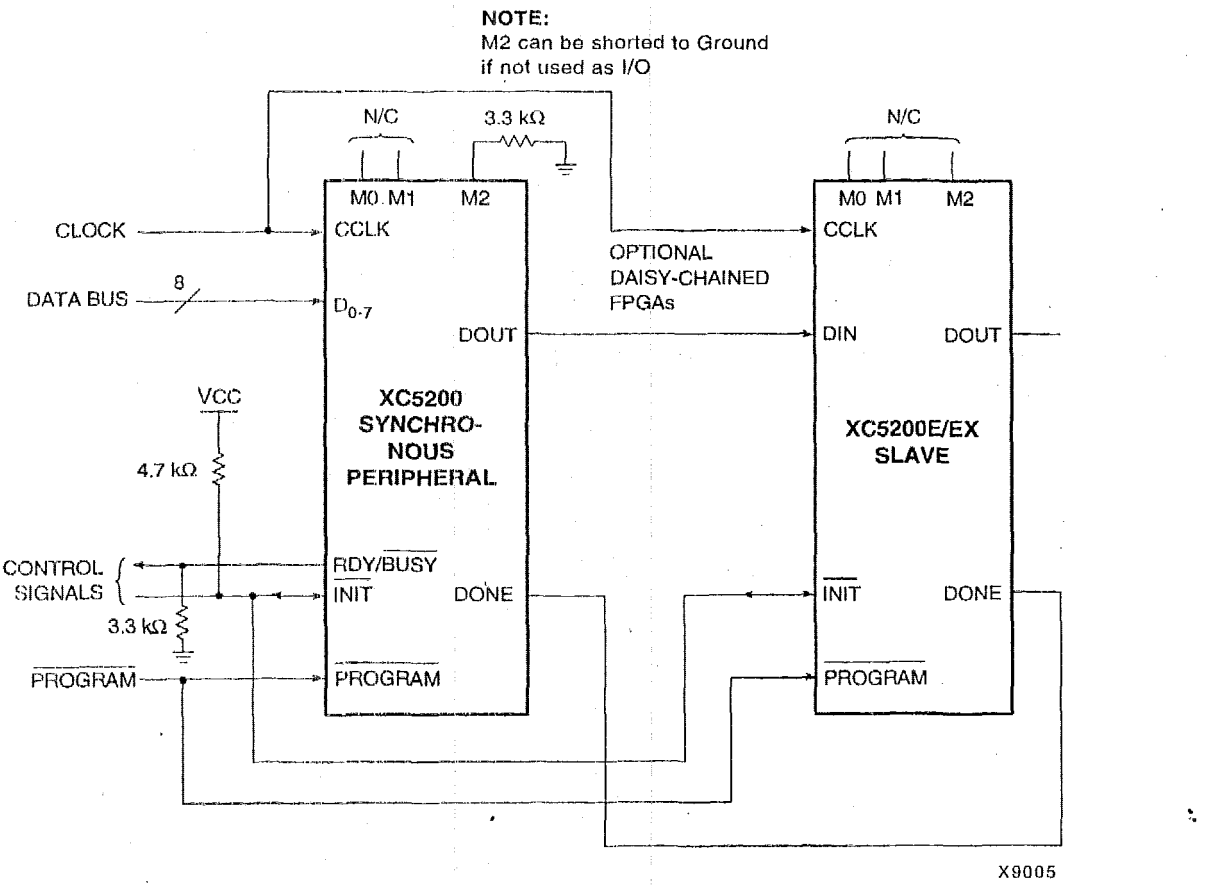
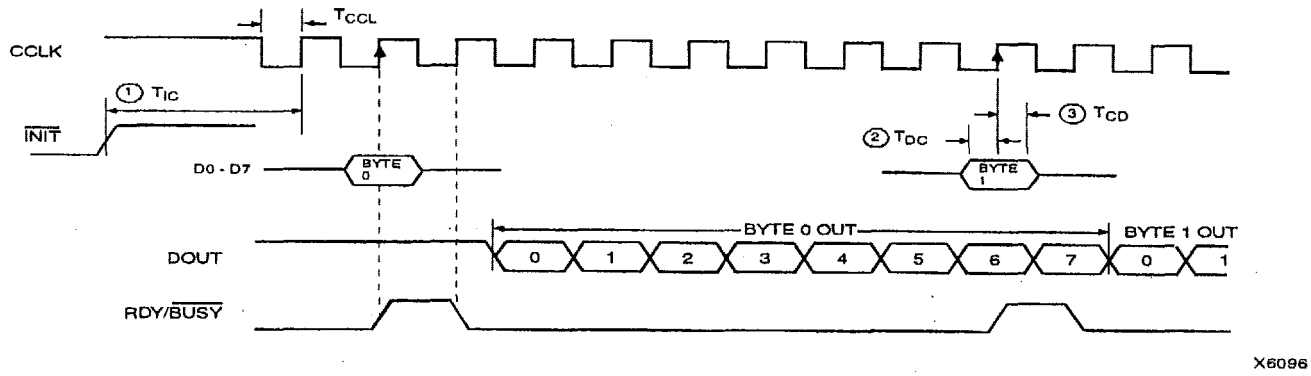


Figure 33: Synchronous Peripheral Mode Circuit Diagram



	Description	Symbol		Min	Max	Units
CCLK	INIT (High) setup time	1	T_{IC}	5		µs
	D0 - D7 setup time	2	T_{DC}	60		ns
	D0 - D7 hold time	3	T_{CD}	0		ns
	CCLK High time		T_{CCH}	50		ns
	CCLK Low time		T_{CCL}	60		ns
	CCLK Frequency		F_{CC}		8	MHz

Notes: 1. Peripheral Synchronous mode can be considered Slave Parallel mode. An external CCLK provides timing, clocking in the **first** data byte on the **second** rising edge of CCLK after $\overline{INIT}$ goes high. Subsequent data bytes are clocked in on every eighth consecutive rising edge of CCLK.
2. The $\overline{RDY/BUSY}$ line goes High for one CCLK period after data has been clocked in, although synchronous operation does not require such a response.
3. The pin name $\overline{RDY/BUSY}$ is a misnomer. In synchronous peripheral mode this is really an ACKNOWLEDGE signal.
4. Note that data starts to shift out serially on the DOUT pin 0.5 CCLK periods after it was loaded in parallel. Therefore, additional CCLK pulses are clearly required after the last byte has been loaded.

Figure 34: Synchronous Peripheral Mode Programming Switching Characteristics

Asynchronous Peripheral Mode

Write to FPGA

Asynchronous Peripheral mode uses the trailing edge of the logic AND condition of $\overline{WS}$ and $\overline{CS0}$ being Low and $\overline{RS}$ and CS1 being High to accept byte-wide data from a microprocessor bus. In the lead FPGA, this data is loaded into a double-buffered UART-like parallel-to-serial converter and is serially shifted into the internal logic.

The lead FPGA presents the preamble data (and all data that overflows the lead device) on its DOUT pin. The RDY/$\overline{BUSY}$ output from the lead FPGA acts as a handshake signal to the microprocessor. RDY/$\overline{BUSY}$ goes Low when a byte has been received, and goes High again when the byte-wide input buffer has transferred its information into the shift register, and the buffer is ready to receive new data. A new write may be started immediately, as soon as the RDY/$\overline{BUSY}$ output has gone Low, acknowledging receipt of the previous data. Write may not be terminated until RDY/$\overline{BUSY}$ is High again for one CCLK period. Note that RDY/$\overline{BUSY}$ is pulled High with a high-impedance pull-up prior to $\overline{INIT}$ going High.

The length of the $\overline{BUSY}$ signal depends on the activity in the UART. If the shift register was empty when the new byte was received, the $\overline{BUSY}$ signal lasts for only two CCLK periods. If the shift register was still full when the new byte was received, the $\overline{BUSY}$ signal can be as long as nine CCLK periods.

Note that after the last byte has been entered, only seven of its bits are shifted out. CCLK remains High with DOUT equal to bit 6 (the next-to-last bit) of the last byte entered.

The READY/$\overline{BUSY}$ handshake can be ignored if the delay from any one Write to the end of the next Write is guaranteed to be longer than 10 CCLK periods.

Status Read

The logic AND condition of the $\overline{CS0}$, CS1 and $\overline{RS}$ inputs puts the device status on the Data bus.

- D7 High indicates Ready
- D7 Low indicates Busy
- D0 through D6 go unconditionally High

It is mandatory that the whole start-up sequence be started and completed by one byte-wide input. Otherwise, the pins used as Write Strobe or Chip Enable might become active outputs and interfere with the final byte transfer. If this transfer does not occur, the start-up sequence is not completed all the way to the finish (point F in Figure 25 on page 109).

In this case, at worst, the internal reset is not released. At best, Readback and Boundary Scan are inhibited. The length-count value, as generated by the software, ensures that these problems never occur.

Although RDY/$\overline{BUSY}$ is brought out as a separate signal, microprocessors can more easily read this information on one of the data lines. For this purpose, D7 represents the RDY/$\overline{BUSY}$ status when $\overline{RS}$ is Low, $\overline{WS}$ is High, and the two chip select lines are both active.

Asynchronous Peripheral mode is selected by a <101> on the mode pins (M2, M1, M0).

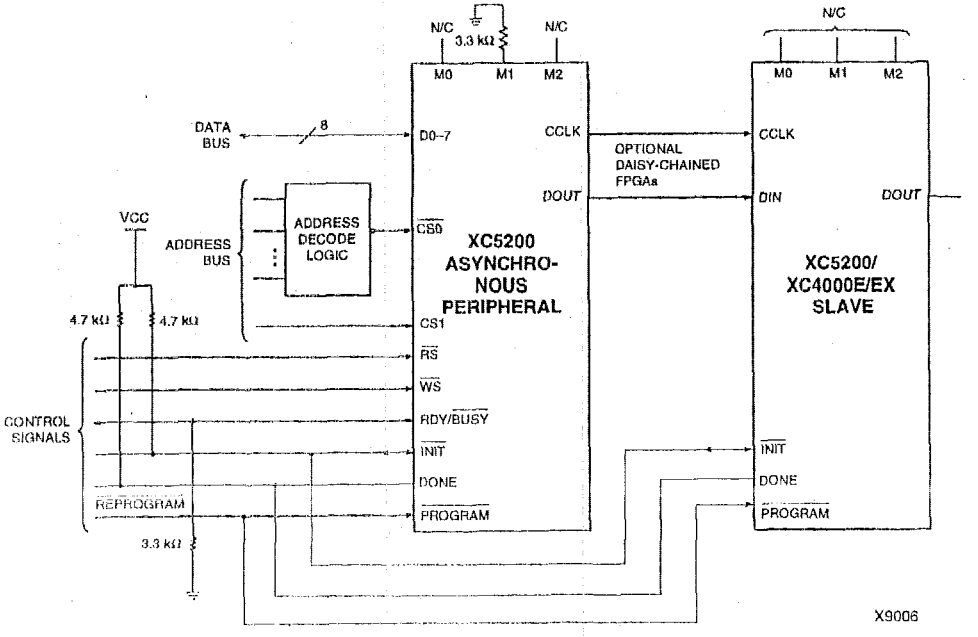
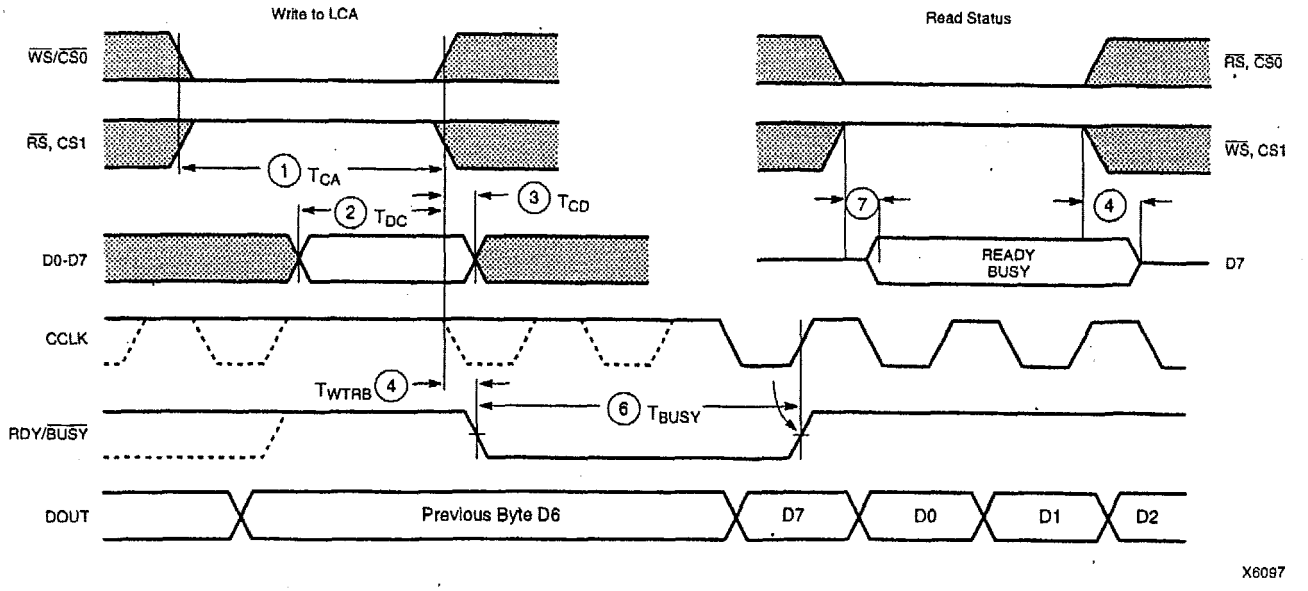


Figure 35: Asynchronous Peripheral Mode Circuit Diagram



	Description	Symbol		Min	Max	Units
Write	Effective Write time ($\overline{CSO}$, $\overline{WS}$=Low; $\overline{RS}$, CS1=High	1	T_{CA}	100		ns
	DIN setup time	2	T_{DC}	60		ns
	DIN hold time	3	T_{CD}	0		ns
RDY	RDY/BUSY delay after end of Write or Read	4	T_{WTRB}		60	ns
	RDY/BUSY active after beginning of Read	7			60	ns
	RDY/BUSY Low output (Note 4)	6	T_{BUSY}	2	9	CCLK periods

Notes:
1. Configuration must be delayed until $\overline{INIT}$ pins of all daisy-chained FPGAs are high.
2. The time from the end of $\overline{WS}$ to CCLK cycle for the new byte of data depends on the completion of previous byte processing and the phase of internal timing generator for CCLK.
3. CCLK and DOUT timing is tested in slave mode.
4. T_{BUSY} indicates that the double-buffered parallel-to-serial converter is not yet ready to receive new data. The shortest T_{BUSY} occurs when a byte is loaded into an empty parallel-to-serial converter. The longest T_{BUSY} occurs when a new word is loaded into the input register before the second-level buffer has started shifting out data.

This timing diagram shows very relaxed requirements. Data need not be held beyond the rising edge of $\overline{WS}$. RDY/$\overline{BUSY}$ will go active within 60 ns after the end of $\overline{WS}$. A new write may be asserted immediately after RDY/$\overline{BUSY}$ goes Low, but write may not be terminated until RDY/$\overline{BUSY}$ has been High for one CCLK period.

Figure 36: Asynchronous Peripheral Mode Programming Switching Characteristics

Express Mode

Express mode is similar to Slave Serial mode, except that data is processed one byte per CCLK cycle instead of one bit per CCLK cycle. An external source is used to drive CCLK, while byte-wide data is loaded directly into the configuration data shift registers. A CCLK frequency of 10 MHz is equivalent to an 80 MHz serial rate, because eight bits of configuration data are loaded per CCLK cycle. Express mode does not support CRC error checking, but does support constant-field error checking.

In Express mode, an external signal drives the CCLK input of the FPGA device. The first byte of parallel configuration data must be available at the D inputs of the FPGA a short setup time before the second rising CCLK edge. Subsequent data bytes are clocked in on each consecutive rising CCLK edge.

If the first device is configured in Express mode, additional devices may be daisy-chained only if every device in the chain is also configured in Express mode. CCLK pins are tied together and D0-D7 pins are tied together for all devices along the chain. A status signal is passed from DOUT to CS1 of successive devices along the chain. The lead device in the chain has its CS1 input tied High (or floating, since there is an internal pullup). Frame data is accepted only when CS1 is High and the device's configuration memory is not already full. The status pin DOUT is pulled Low two internal-oscillator cycles after $\overline{INIT}$ is recognized as High, and remains Low until the device's configuration memory is full. DOUT is then pulled High to signal the next device in the chain to accept the configuration data on the D0-D7 bus.

The DONE pins of all devices in the chain should be tied together, with one or more active internal pull-ups. If a large number of devices are included in the chain, deactivate some of the internal pull-ups, since the Low-driving DONE pin of the last device in the chain must sink the current from all pull-ups in the chain. The DONE pull-up is activated by default. It can be deactivated using an option in the bitstream generation software.

XC5200 devices in Express mode are always synchronized to DONE. The device becomes active after DONE goes High. DONE is an open-drain output. With the DONE pins tied together, therefore, the external DONE signal stays low until all devices are configured, then all devices in the daisy chain become active simultaneously. If the DONE pin of a device is left unconnected, the device becomes active as soon as that device has been configured.

Express mode is selected by a <010> on the mode pins (M2, M1, M0).

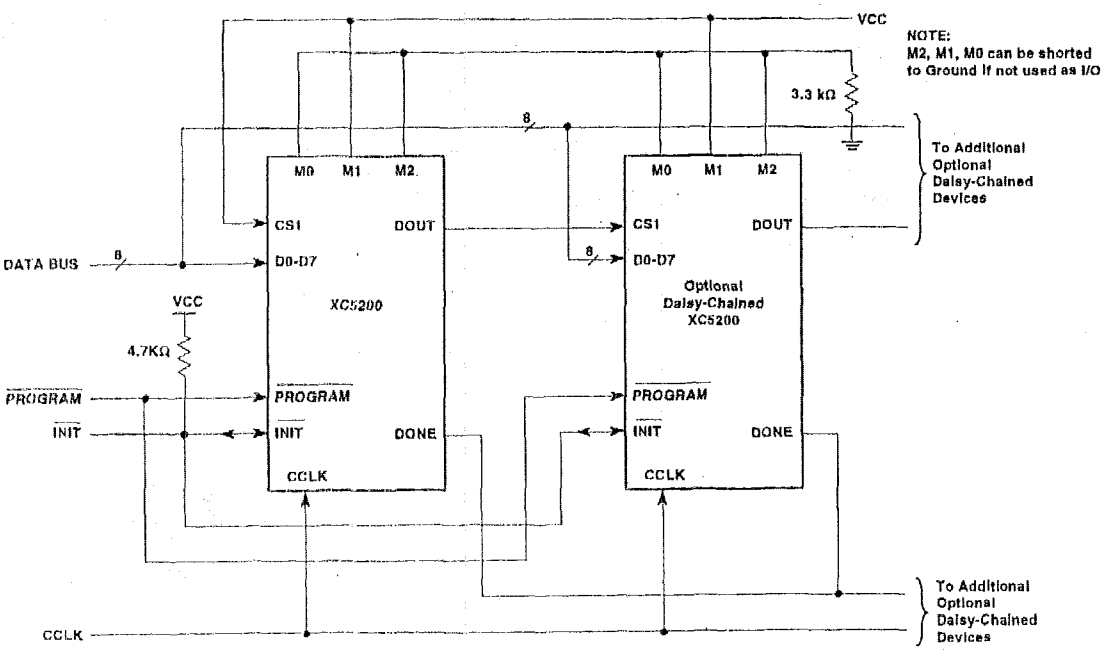
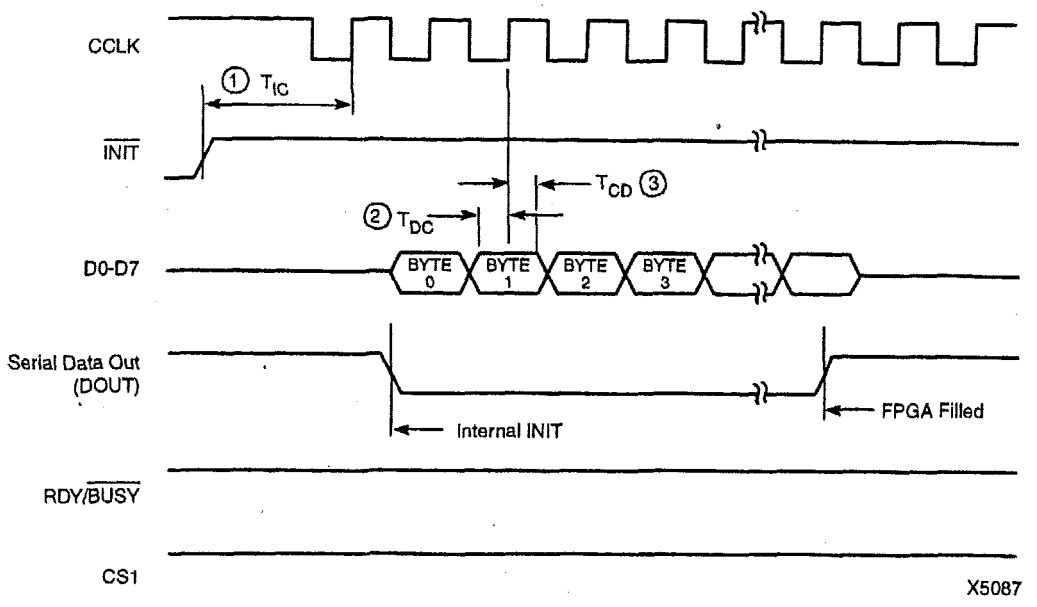


Figure 37: Express Mode Circuit Diagram



	Description	Symbol		Min	Max	Units
CCLK	INIT (High) Setup time required	1	T_{IC}	5		µs
	DIN Setup time required	2	T_{DC}	30		ns
	DIN hold time required	3	T_{CD}	0		ns
	CCLK High time		T_{CCH}	30		ns
	CCLK Low time		T_{CCL}	30		ns
	CCLK frequency		F_{CC}		10	MHz

Note: If not driven by the preceding DOUT, CS1 must remain high until the device is fully configured.

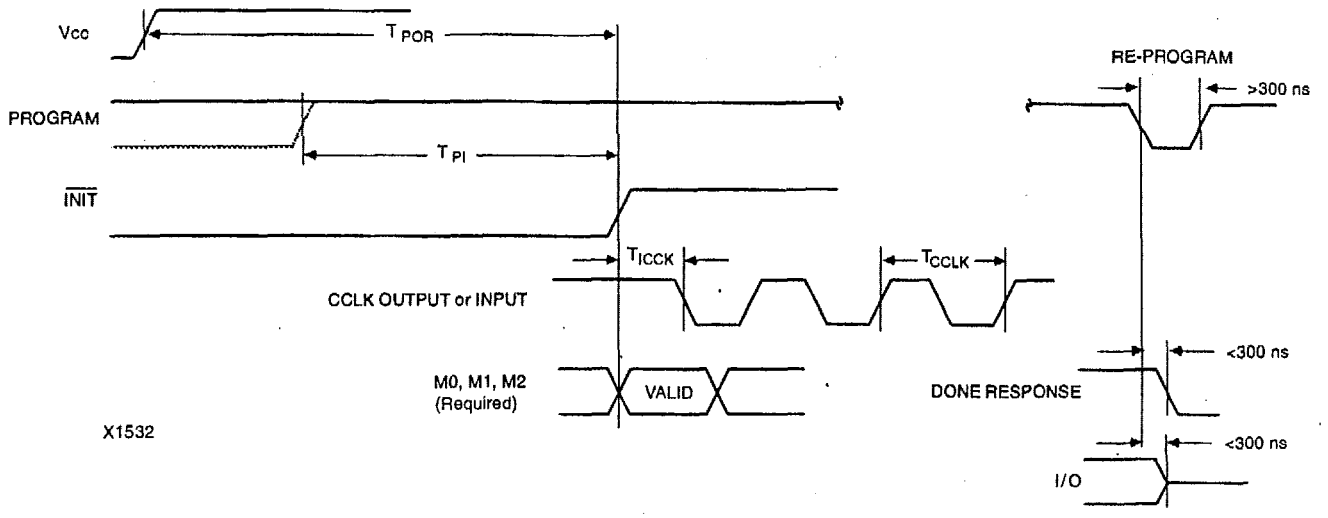
Figure 38: Express Mode Programming Switching Characteristics

Table 13. Pin Functions During Configuration

CONFIGURATION MODE: <M2:M1:M0>							USER OPERATION
SLAVE <1:1:1>	MASTER-SER <0:0:0>	SYN.PERIPH <0:1:1>	ASYN.PERIPH <1:0:1>	MASTER-HIGH <1:1:0>	MASTER-LOW <1:0:0>	EXPRESS <0:1:0>	
				A16	A16		GCK1-I/O
				A17	A17		I/O
TDI	TDI	TDI	TDI	TDI	TDI	TDI	TDI-I/O
TCK	TCK	TCK	TCK	TCK	TCK	TCK	TCK-I/O
TMS	TMS	TMS	TMS	TMS	TMS	TMS	TMS-I/O
							I/O
M1 (HIGH) (I)	M1 (LOW) (I)	M1 (HIGH) (I)	M1 (LOW) (I)	M1 (HIGH) (I)	M1 (LOW) (I)	M1 (HIGH) (I)	I/O
M0 (HIGH) (I)	M0 (LOW) (I)	M0 (HIGH) (I)	M0 (HIGH) (I)	M0 (LOW) (I)	M0 (LOW) (I)	M0 (LOW) (I)	I/O
M2 (HIGH) (I)	M2 (LOW) (I)	M2 (LOW) (I)	M2 (HIGH) (I)	M2 (HIGH) (I)	M2 (HIGH) (I)	M2 (LOW) (I)	I/O
							GCK2-I/O
HDC (HIGH)	HDC (HIGH)	HDC (HIGH)	HDC (HIGH)	HDC (HIGH)	HDC (HIGH)	HDC (HIGH)	I/O
LDC (LOW)	LDC (LOW)	LDC (LOW)	LDC (LOW)	LDC (LOW)	LDC (LOW)	LDC (LOW)	I/O
INIT-ERROR	INIT-ERROR	INIT-ERROR	INIT-ERROR	INIT-ERROR	INIT-ERROR	INIT-ERROR	I/O
							I/O
DONE	DONE	DONE	DONE	DONE	DONE	DONE	DONE
PROGRAM (I)	PROGRAM (I)	PROGRAM (I)	PROGRAM (I)	PROGRAM (I)	PROGRAM (I)	PROGRAM (I)	PROGRAM
		DATA 7 (I)	DATA 7 (I)	DATA 7 (I)	DATA 7 (I)	DATA 7 (I)	I/O
							GCK3-I/O
		DATA 6 (I)	DATA 6 (I)	DATA 6 (I)	DATA 6 (I)	DATA 6 (I)	I/O
		DATA 5 (I)	DATA 5 (I)	DATA 5 (I)	DATA 5 (I)	DATA 5 (I)	I/O
			CSO (I)				I/O
		DATA 4 (I)	DATA 4 (I)	DATA 4 (I)	DATA 4 (I)	DATA 4 (I)	I/O
		DATA 3 (I)	DATA 3 (I)	DATA 3 (I)	DATA 3 (I)	DATA 3 (I)	I/O
			RS (I)				I/O
		DATA 2 (I)	DATA 2 (I)	DATA 2 (I)	DATA 2 (I)	DATA 2 (I)	I/O
		DATA 1 (I)	DATA 1 (I)	DATA 1 (I)	DATA 1 (I)	DATA 1 (I)	I/O
		RDY/BUSY	RDY/BUSY	RCLK	RCLK		I/O
DIN (I)	DIN (I)	DATA 0 (I)	DATA 0 (I)	DATA 0 (I)	DATA 0 (I)	DATA 0 (I)	I/O
DOUT	DOUT	DOUT	DOUT	DOUT	DOUT	DOUT	I/O
CCLK (I)	CCLK (O)	CCLK (I)	CCLK (O)	CCLK (O)	CCLK (O)	CCLK (I)	CCLK (I)
TDO	TDO	TDO	TDO	TDO	TDO	TDO	TDO-I/O
			WS (I)	A0	A0		I/O
				A1	A1		GCK4-I/O
			CS1 (I)	A2	A2	CS1 (I)	I/O
				A3	A3		I/O
				A4	A4		I/O
				A5	A5		I/O
				A6	A6		I/O
				A7	A7		I/O
				A8	A8		I/O
				A9	A9		I/O
				A10	A10		I/O
				A11	A11		I/O
				A12	A12		I/O
				A13	A13		I/O
				A14	A14		I/O
				A15	A15		I/O
							ALL OTHERS

Notes:
1. A shaded table cell represents a 20-kΩ to 100-kΩ pull-up resistor before and during configuration.
2. (I) represents an input (O) represents an output.
3. $\overline{INIT}$ is an open-drain output during configuration.

Configuration Switching Characteristics



Master Modes

Description	Symbol	Min	Max	Units
Power-On-Reset	T_{POR}	2	15	ms
Program Latency	T_{PI}	6	70	µs per CLB column
CCLK (output) Delay	T_{ICCK}	40	375	µs
period (slow)	T_{CCLK}	640	3000	ns
period (fast)	T_{CCLK}	100	375	ns

Slave and Peripheral Modes

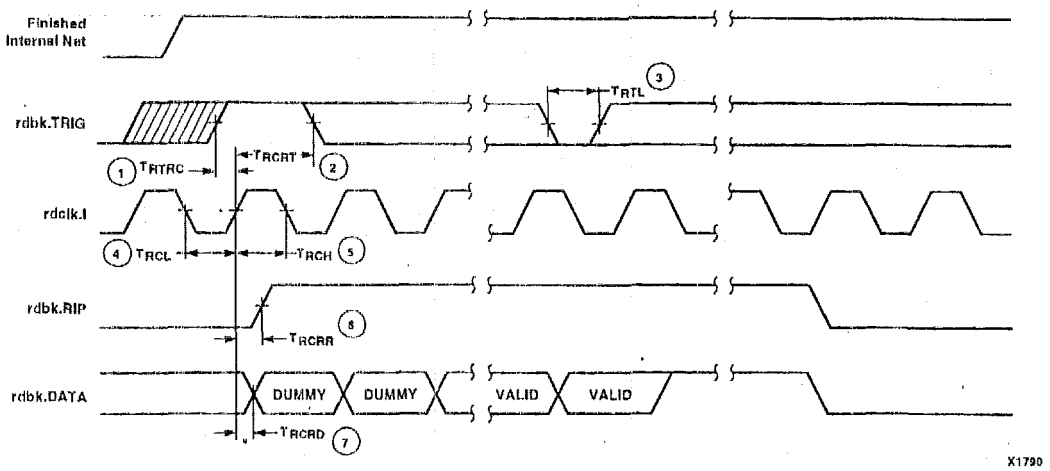
Description	Symbol	Min	Max	Units
Power-On-Reset	T_{POR}	2	15	ms
Program Latency	T_{PI}	6	70	µs per CLB column
CCLK (input) Delay (required)	T_{ICCK}	5		µs
period (required)	T_{CCLK}	100		ns

Note: At power-up, V_{CC} must rise from 2.0 to V_{CC} min in less than 15 ms, otherwise delay configuration using PROGRAM until V_{CC} is valid.

XC5200 Program Readback Switching Characteristic Guidelines

Testing of the switching parameters is modeled after testing methods specified by MIL-M-38510/605. All devices are 100% functionally tested. Internal timing parameters are not measured directly. They are derived from benchmark timing patterns that are taken at device introduction, prior to any process improvements.

The following guidelines reflect worst-case values over the recommended operating conditions.



	Description	Symbol		Min	Max	Units
rdbk.TRIG	rdbk.TRIG setup to initiate and abort Readback	1	T_{RTRC}	200	-	ns
	rdbk.TRIG hold to initiate and abort Readback	2	T_{RCRT}	50	-	ns
rdclk.1	rdbk.DATA delay	7	T_{RCRD}	-	250	ns
	rdbk.RIP delay	6	T_{RCRR}	-	250	ns
	High time	5	T_{RCH}	250	500	ns
	Low time	4	T_{RCL}	250	500	ns

Note 1: Timing parameters apply to all speed grades.

Note 2: rdbk.TRIG is High prior to Finished, Finished will trigger the first Readback

XC5200 Switching Characteristics

Definition of Terms

In the following tables, some specifications may be designated as Advance or Preliminary. These terms are defined as follows:

Advance: Initial estimates based on simulation and/or extrapolation from other speed grades, devices, or device families. Use as estimates, not for production.
Preliminary: Based on preliminary characterization. Further changes are not expected.
Unmarked: Specifications not identified as either Advance or Preliminary are to be considered Final.[1]

XC5200 Operating Conditions

Symbol	Description	Min	Max	Units
V_{CC}	Supply voltage relative to GND Commercial: 0°C to 85°C junction	4.75	5.25	V
	Supply voltage relative to GND Industrial: -40°C to 100°C junction	4.5	5.5	V
V_{IHT}	High-level input voltage — TTL configuration	2.0	V_{CC}	V
V_{ILT}	Low-level input voltage — TTL configuration	0	0.8	V
V_{IHC}	High-level input voltage — CMOS configuration	70%	100%	V_{CC}
V_{ILC}	Low-level input voltage — CMOS configuration	0	20%	V_{CC}
T_{IN}	Input signal transition time		250	ns

XC5200 DC Characteristics Over Operating Conditions

Symbol	Description	Min	Max	Units
V_{OH}	High-level output voltage @ I_{OH} = -8.0 mA, V_{CC} min	3.86		V
V_{OL}	Low-level output voltage @ I_{OL} = 8.0 mA, V_{CC} max		0.4	V
I_{CCO}	Quiescent FPGA supply current (Note 1)		15	mA
I_{IL}	Leakage current	-10	+10	µA
C_{IN}	Input capacitance (sample tested)		15	pF
I_{RIN}	Pad pull-up (when selected) @ V_{IN} = 0V (sample tested)	0.02	0.30	mA

Note: 1. With no output current loads, all package pins at Vcc or GND, either TTL or CMOS inputs, and the FPGA configured with a tie option.

XC5200 Absolute Maximum Ratings

Symbol	Description		Units
V_{CC}	Supply voltage relative to GND	-0.5 to +7.0	V
V_{IN}	Input voltage with respect to GND	-0.5 to V_{CC} +0.5	V
V_{TS}	Voltage applied to 3-state output	-0.5 to V_{CC} +0.5	V
T_{STG}	Storage temperature (ambient)	-65 to +150	°C
T_{SOL}	Maximum soldering temperature (10 s @ 1/16 in. = 1.5 mm)	+260	°C
T_J	Junction temperature in plastic packages	+125	°C
	Junction temperature in ceramic packages	+150	°C

Note: Stresses beyond those listed under Absolute Maximum Ratings may cause permanent damage to the device. These are stress ratings only, and functional operation of the device at these or any other conditions beyond those listed under Recommended Operating Conditions is not implied. Exposure to Absolute Maximum Ratings conditions for extended periods of time may affect device reliability.

1. Notwithstanding the definition of the above terms, all specifications are subject to change without notice.

XC5200 Global Buffer Switching Characteristic Guidelines

Testing of the switching parameters is modeled after testing methods specified by MIL-M-38510/605. All devices are 100% functionally tested. Since many internal timing parameters cannot be measured directly, they are derived from benchmark timing patterns. The following guidelines reflect worst-case values over the recommended operating conditions. For more detailed, more precise, and more up-to-date timing information, use the values provided by the timing calculator and used in the simulator.

		Speed Grade	-6	-5	-4	-3
Description	**Symbol**	**Device**	**Max (ns)**	**Max (ns)**	**Max (ns)**	**Max (ns)**
Global Signal Distribution	T_{BUFG}	XC5202	9.1	8.5	8.0	6.9
From pad through global buffer, to any clock (CK)		XC5204	9.3	8.7	8.2	7.6
		XC5206	9.4	8.8	8.3	7.7
		XC5210	9.4	8.8	8.5	7.7
		XC5215	10.5	9.9	9.8	9.6

XC5200 Longline Switching Characteristic Guidelines

Testing of the switching parameters is modeled after testing methods specified by MIL-M-38510/605. All devices are 100% functionally tested. Since many internal timing parameters cannot be measured directly, they are derived from benchmark timing patterns. The following guidelines reflect worst-case values over the recommended operating conditions. For more detailed, more precise, and more up-to-date timing information, use the values provided by the timing calculator and used in the simulator.

		Speed Grade	-6	-5	-4	-3
Description	**Symbol**	**Device**	**Max (ns)**	**Max (ns)**	**Max (ns)**	**Max (ns)**
TBUF driving a Longline TS I — O TBUF I to Longline, while TS is Low; i.e., buffer is constantly active	T_{IO}	XC5202	6.0	3.8	3.0	2.0
		XC5204	6.4	4.1	3.2	2.3
		XC5206	6.6	4.2	3.3	2.7
		XC5210	6.6	4.2	3.3	2.9
		XC5215	7.3	4.6	3.8	3.2
TS going Low to Longline going from floating High or Low to active Low or High	T_{ON}	XC5202	7.8	5.6	4.7	4.0
		XC5204	8.3	5.9	4.9	4.3
		XC5206	8.4	6.0	5.0	4.4
		XC5210	8.4	6.0	5.0	4.4
		XC5215	8.9	6.3	5.3	4.5
TS going High to TBUF going inactive, not driving Longline	T_{OFF}	XC52xx	3.0	2.8	2.6	2.4

Note: 1. Die-size-dependent parameters are based upon XC5215 characterization. Production specifications will vary with array size.

XC5200 CLB Switching Characteristic Guidelines

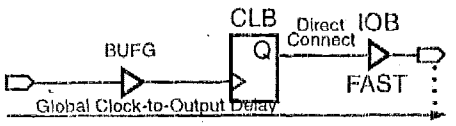
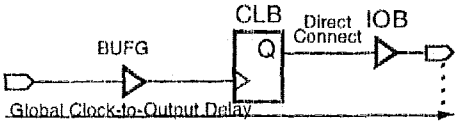
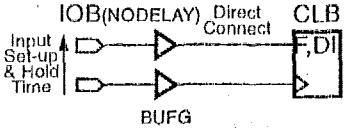
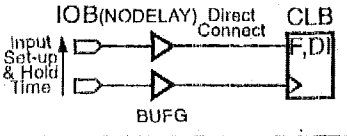
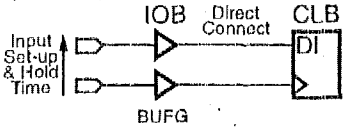
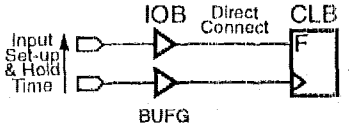
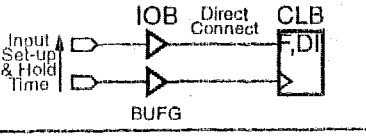
Testing of the switching parameters is modeled after testing methods specified by MIL-M-38510/605. All devices are 100% functionally tested. Since many internal timing parameters cannot be measured directly, they are derived from benchmark timing patterns. The following guidelines reflect worst-case values over the recommended operating conditions. For more detailed, more precise, and more up-to-date timing information, use the values provided by the timing calculator and used in the simulator.

Speed Grade		-6		-5		-4		-3	
Description	**Symbol**	**Min (ns)**	**Max (ns)**	**Min (ns)**	**Max (ns)**	**Min (ns)**	**Max (ns)**	**Min (ns)**	**Max (ns)**
Combinatorial Delays									
F inputs to X output	T_{ILO}		5.6		4.6		3.8		3.0
F inputs via transparent latch to Q	T_{ITO}		8.0		6.6		5.4		4.3
DI inputs to DO output (Logic-Cell Feedthrough)	T_{IDO}		4.3		3.5		2.8		2.4
F inputs via F5_MUX to DO output	T_{IMO}		7.2		5.8		5.0		4.3
Carry Delays									
Incremental delay per bit	T_{CY}		0.7		0.6		0.5		0.5
Carry-in overhead from DI	T_{CYDI}		1.8		1.6		1.5		1.4
Carry-in overhead from F	T_{CYL}		3.7		3.2		2.9		2.4
Carry-out overhead to DO	T_{CYO}		4.0		3.2		2.5		2.1
Sequential Delays									
Clock (CK) to out (Q) (Flip-Flop)	T_{CKO}		5.8		4.9		4.0		4.0
Gate (Latch enable) going active to out (Q)	T_{GO}		9.2		7.4		5.9		5.5
Set-up Time Before Clock (CK)									
F inputs	T_{ICK}	2.3		1.8		1.4		1.3	
F inputs via F5_MUX	T_{MICK}	3.8		3.0		2.5		2.4	
DI input	T_{DICK}	0.8		0.5		0.4		0.4	
CE input	T_{EICK}	1.6		1.2		0.9		0.9	
Hold Times After Clock (CK)									
F inputs	T_{CKI}	0		0		0		0	
F inputs via F5_MUX	T_{CKMI}	0		0		0		0	
DI input	T_{CKDI}	0		0		0		0	
CE input	T_{CKEI}	0		0		0		0	
Clock Widths									
Clock High Time	T_{CH}	6.0		6.0		6.0		6.0	
Clock Low Time	T_{CL}	6.0		6.0		6.0		6.0	
Toggle Frequency (MHz) (Note 3)	F_{TOG}		83		83		83		83
Reset Delays									
Width (High)	T_{CLRW}	6.0		6.0		6.0		6.0	
Delay from CLR to Q (Flip-Flop)	T_{CLR}		7.7		6.3		5.1		4.0
Delay from CLR to Q (Latch)	T_{CLRL}		6.5		5.2		4.2		3.0
Global Reset Delays									
Width (High)	T_{GCLRW}	6.0		6.0		6.0		6.0	
Delay from internal GR to Q	T_{GCLR}		14.7		12.1		9.1		8.0

Note: 1. The CLB K to Q output delay (T_{CKO}) of any CLB, plus the shortest possible interconnect delay, is always longer than the Data In hold-time requirement (T_{CKDI}) of any CLB on the same die.
2. Timing is based upon the XC5215 device. For other devices, see Timing Calculator.
3. Maximum flip-flop toggle rate for export control purposes.

XC5200 Guaranteed Input and Output Parameters (Pin-to-Pin)

All values listed below are tested directly, and guaranteed over the operating conditions. The same parameters can also be derived indirectly from the Global Buffer specifications. The delay calculator uses this indirect method, and may overestimate because of worst-case assumptions. When there is a discrepancy between these two methods, the values listed below should be used, and the derived values should be considered conservative overestimates.

		Speed Grade	-6	-5	-4	-3
Description	**Symbol**	**Device**	**Max (ns)**	**Max (ns)**	**Max (ns)**	**Max (ns)**
Global Clock to Output Pad (fast) 	T_{ICKOF} (Max)	XC5202	16.9	15.1	10.9	9.8
		XC5204	17.1	15.3	11.3	9.9
		XC5206	17.2	15.4	11.9	10.8
		XC5210	17.2	15.4	12.8	11.2
		XC5215	19.0	17.0	12.8	11.7
Global Clock to Output Pad (slew-limited) 	T_{ICKO} (Max)	XC5202	21.4	18.7	12.6	11.5
		XC5204	21.6	18.9	13.3	11.9
		XC5206	21.7	19.0	13.6	12.5
		XC5210	21.7	19.0	15.0	12.9
		XC5215	24.3	21.2	15.0	13.1
Input Set-up Time (no delay) to CLB Flip-Flop 	T_{PSUF} (Min)	XC5202	2.5	2.0	1.9	1.9
		XC5204	2.3	1.9	1.9	1.9
		XC5206	2.2	1.9	1.9	1.9
		XC5210	2.2	1.9	1.9	1.8
		XC5215	2.0	1.8	1.7	1.7
Input Hold Time (no delay) to CLB Flip-Flop 	T_{PHF} (Min)	XC5202	3.8	3.8	3.5	3.5
		XC5204	3.9	3.9	3.8	3.6
		XC5206	4.4	4.4	4.4	4.3
		XC5210	5.1	5.1	4.9	4.8
		XC5215	5.8	5.8	5.7	5.6
Input Set-up Time (with delay) to CLB Flip-Flop DI Input 	T_{PSU}	XC5202	7.3	6.6	6.6	6.6
		XC5204	7.3	6.6	6.6	6.6
		XC5206	7.2	6.5	6.4	6.3
		XC5210	7.2	6.5	6.0	6.0
		XC5215	6.8	5.7	5.7	5.7
Input Set-up Time (with delay) to CLB Flip-Flop F Input 	T_{PSUL} (Min)	XC5202	8.8	7.7	7.5	7.5
		XC5204	8.6	7.5	7.5	7.5
		XC5206	8.5	7.4	7.4	7.4
		XC5210	8.5	7.4	7.4	7.3
		XC5215	8.5	7.4	7.4	7.2
Input Hold Time (with delay) to CLB Flip-Flop 	T_{PH} (Min)	XC52xx	0	0	0	0

Note: 1. These measurements assume that the CLB flip-flop uses a direct interconnect to or from the IOB. The INREG/ OUTREG properties, or XACT-Performance, can be used to assure that direct connects are used. t_{PSU} applies only to the CLB input DI that bypasses the look-up table, which only offers direct connects to IOBs on the left and right edges of the die. t_{PSUL} applies to the CLB inputs F that feed the look-up table, which offers direct connect to IOBs on all four edges, as do the CLB Q outputs.

2. When testing outputs (fast or slew-limited), half of the outputs on one side of the device are switching.

XC5200 IOB Switching Characteristic Guidelines

Testing of the switching parameters is modeled after testing methods specified by MIL-M-38510/605. All devices are 100% functionally tested. Since many internal timing parameters cannot be measured directly, they are derived from benchmark timing patterns. The following guidelines reflect worst-case values over the recommended operating conditions. For more detailed, more precise, and more up-to-date timing information, use the values provided by the timing calculator and used in the simulator.

Speed Grade		-6	-5	-4	-3
Description	**Symbol**	**Max (ns)**	**Max (ns)**	**Max (ns)**	**Max (ns)**
Input					
Propagation Delays from CMOS or TTL Levels					
Pad to I (no delay)	T_{PI}	5.7	5.0	4.8	3.3
Pad to I (with delay)	T_{PID}	11.4	10.2	10.2	9.5
Output					
Propagation Delays to CMOS or TTL Levels					
Output (O) to Pad (fast)	T_{OPF}	4.6	4.5	4.5	3.5
Output (O) to Pad (slew-limited)	T_{OPS}	9.5	8.4	8.0	5.0
From clock (CK) to output pad (fast), using direct connect between Q and output (O)	T_{OKPOF}	10.1	9.3	8.3	7.5
From clock (CK) to output pad (slew-limited), using direct connect between Q and output (O)	T_{OKPOS}	14.9	13.1	11.8	10.0
3-state to Pad active (fast)	T_{TSONF}	5.6	5.2	4.9	4.6
3-state to Pad active (slew-limited)	T_{TSONS}	10.4	9.0	8.3	6.0
Internal GTS to Pad active	T_{GTS}	17.7	15.9	14.7	13.5

Note: 1. Timing is measured at pin threshold, with 50-pF external capacitance loads. **Slew-limited** output rise/fall times are approximately two times longer than **fast** output rise/fall times.
2. Unused and unbonded IOBs are configured by default as inputs with internal pull-up resistors.
3. Timing is based upon the XC5215 device. For other devices, see Timing Calculator.

XC5200 Boundary Scan (JTAG) Switching Characteristic Guidelines

The following guidelines reflect worst-case values over the recommended operating conditions. They are expressed in units of nanoseconds and apply to all XC5200 devices unless otherwise noted.

Speed Grade		-6		-5		-4		-3	
Description	**Symbol**	**Min**	**Max**	**Min**	**Max**	**Min**	**Max**	**Min**	**Max**
Setup and Hold									
Input (TDI) to clock (TCK) setup time	T_{TDITCK}	30.0		30.0		30.0		30.0	
Input (TDI) to clock (TCK) hold time	T_{TCKTDI}	0		0		0		0	
Input (TMS) to clock (TCK) setup time	T_{TMSTCK}	15.0		15.0		15.0		15.0	
Input (TMS) to clock (TCK) hold time	T_{TCKTMS}	0		0		0		0	
Propagation Delay									
Clock (TCK) to Pad (TDO)	T_{TCKPO}		30.0		30.0		30.0		30.0
Clock									
Clock (TCK) High	T_{TCKH}	30.0		30.0		30.0		30.0	
Clock (TCK) Low	T_{TCKL}	30.0		30.0		30.0		30.0	
F_{MAX} (MHz)	F_{MAX}		10.0		10.0		10.0		10.0

Note 1: Input pad setup and hold times are specified with respect to the internal clock.

Device-Specific Pinout Tables

Device-specific tables include all packages for each XC5200-Series device. They follow the pad locations around the die, and include boundary scan register locations.

Pin Locations for XC5202 Devices

The following table may contain pinout information for unsupported device/package combinations. Please see the availability charts elsewhere in the XC5200 Series data sheet for availability information.

Pin	Description	VQ64*	PC84	PQ100	VQ100	TQ144	PG156	Boundary Scan Order
	VCC	-	2	92	89	128	H3	-
1.	I/O (A8)	57	3	93	90	129	H1	51
2.	I/O (A9)	58	4	94	91	130	G1	54
3.	I/O	-	-	95	92	131	G2	57
4.	I/O	-	-	96	93	132	G3	63
5.	I/O (A10)	-	5	97	94	133	F1	66
6.	I/O (A11)	59	6	98	95	134	F2	69
	GND	-	-	-	-	137	F3	-
7.	I/O (A12)	60	7	99	96	138	E3	78
8.	I/O (A13)	61	8	100	97	139	C1	81
9.	I/O (A14)	62	9	1	98	142	B1	90
10.	I/O (A15)	63	10	2	99	143	B2	93
	VCC	64	11	3	100	144	C3	-
	GND	-	12	4	1	1	C4	-
11.	GCK1 (A16, I/O)	1	13	5	2	2	B3	102
12.	I/O (A17)	2	14	6	3	3	A1	105
13.	I/O (TDI)	3	15	7	4	6	B4	111
14.	I/O (TCK)	4	16	8	5	7	A3	114
	GND	-	-	-	-	8	C6	-
15.	I/O (TMS)	5	17	9	6	11	A5	117
16.	I/O	6	18	10	7	12	C7	123
17.	I/O	-	-	-	-	13	B7	126
18.	I/O	-	-	11	8	14	A6	129
19.	I/O	-	19	12	9	15	A7	135
20.	I/O	7	20	13	10	16	A8	138
	GND	8	21	14	11	17	C8	-
	VCC	9	22	15	12	18	B8	-
21.	I/O	-	23	16	13	19	C9	141
22.	I/O	10	24	17	14	20	B9	147
23.	I/O		-	18	15	21	A9	150
24.	I/O		-	-	-	22	B10	153
25.	I/O	-	25	19	16	23	C10	159
26.	I/O	11	26	20	17	24	A10	162
	GND		-	-	-	27	C11	-
27.	I/O	12	27	21	18	28	B12	165
28.	I/O		-	22	19	29	A13	171
29.	I/O	13	28	23	20	32	B13	174
30.	I/O	14	29	24	21	33	B14	177
31.	M1 (I/O)	15	30	25	22	34	A15	186
	GND	-	31	26	23	35	C13	-
32.	M0 (I/O)	16	32	27	24	36	A16	189
	VCC	-	33	28	25	37	C14	-
33.	M2 (I/O)	17	34	29	26	38	B15	192
34.	GCK2 (I/O)	18	35	30	27	39	B16	195

Pin	Description	VQ64*	PC84	PQ100	VQ100	TQ144	PG156	Boundary Scan Order
35.	I/O (HDC)	19	36	31	28	40	D14	204
36.	I/O	-	-	32	29	43	E14	207
37.	I/O (LDC)	20	37	33	30	44	C16	210
	GND	-	-	-	-	45	F14	-
38.	I/O	-	38	34	31	48	F16	216
39.	I/O	21	39	35	32	49	G14	219
40.	I/O	-	-	36	33	50	G15	222
41.	I/O	-	-	37	34	51	G16	228
42.	I/O	22	40	38	35	52	H16	231
43.	I/O (ERR, INIT)	23	41	39	36	53	H15	234
	VCC	24	42	40	37	54	H14	-
	GND	25	43	41	38	55	J14	-
44.	I/O	26	44	42	39	56	J15	240
45.	I/O	27	45	43	40	57	J16	243
46.	I/O	-	-	44	41	58	K16	246
47.	I/O	-	-	45	42	59	K15	252
48.	I/O	28	46	46	43	60	K14	255
49.	I/O	29	47	47	44	61	L16	258
	GND	-	-	-	-	64	L14	-
50.	I/O	-	48	48	45	65	P16	264
51.	I/O	30	49	49	46	66	M14	267
52.	I/O	-	50	50	47	69	N14	276
53.	I/O	31	51	51	48	70	R16	279
	GND	-	52	52	49	71	P14	-
	DONE	32	53	53	50	72	R15	-
	VCC	33	54	54	51	73	P13	-
	PROG	34	55	55	52	74	R14	-
54.	I/O (D7)	35	56	56	53	75	T16	288
55.	GCK3 (I/O)	36	57	57	54	76	T15	291
56.	I/O (D6)	37	58	58	55	79	T14	300
57.	I/O	-	-	59	56	80	T13	303
	GND	-	-	-	-	81	P11	-
58.	I/O (D5)	38	59	60	57	84	T10	306
59.	I/O ($\overline{CS0}$)	-	60	61	58	85	P10	312
60.	I/O	-	-	62	59	86	R10	315
61.	I/O	-	-	63	60	87	T9	318
62.	I/O (D4)	39	61	64	61	88	R9	324
63.	I/O	-	62	65	62	89	P9	327
	VCC	40	63	66	63	90	R8	-
	GND	41	64	67	64	91	P8	-
64.	I/O (D3)	42	65	68	65	92	T8	336
65.	I/O ($\overline{RS}$)	43	66	69	66	93	T7	339
66.	I/O	-	-	70	67	94	T6	342
67.	I/O	-	-	-	-	95	R7	348
68.	I/O (D2)	44	67	71	68	96	P7	351
69.	I/O	-	68	72	69	97	T5	360
	GND	-	-	-	-	100	P6	-
70.	I/O (D1)	45	69	73	70	101	T3	363
71.	I/O ($\overline{RCLK}$-BUSY/RDY)	-	70	74	71	102	P5	366
72.	I/O (D0, DIN)	46	71	75	72	105	P4	372
73.	I/O (DOUT)	47	72	76	73	106	T2	375

Pin	Description	VQ64*	PC84	PQ100	VQ100	TQ144	PG156	Boundary Scan Order
	CCLK	48	73	77	74	107	R2	-
	VCC	-	74	78	75	108	P3	-
74.	I/O (TDO)	49	75	79	76	109	T1	0
	GND	-	76	80	77	110	N3	-
75.	I/O (A0, WS)	50	77	81	78	111	R1	9
76.	GCK4 (A1, I/O)	51	78	82	79	112	P2	15
77.	I/O (A2, CS1)	52	79	83	80	115	P1	18
78.	I/O (A3)	-	80	84	81	116	N1	21
	GND	-	-	-	-	118	L3	-
79.	I/O (A4)	-	81	85	82	121	K3	27
80.	I/O (A5)	53	82	86	83	122	K2	30
81.	I/O	-	-	87	84	123	K1	33
82.	I/O	-	-	88	85	124	J1	39
83.	I/O (A6)	54	83	89	86	125	J2	42
84.	I/O (A7)	55	84	90	87	126	J3	45
	GND	56	1	91	88	127	H2	-

* VQ64 package supports Master Serial, Slave Serial, and Express configuration modes only.

Additional No Connect (N.C.) Connections on TQ144 Package

TQ144					
135	9	41	67	98	117
136	10	42	68	99	119
140	25	46	77	103	120
141	26	47	78	104	
4	30	62	82	113	
5	31	63	83	114	

Notes: Boundary Scan Bit 0 = TDO.T
Boundary Scan Bit 1 = TDO.O
Boundary Scan Bit 1056 = BSCAN.UPD

Pin Locations for XC5204 Devices

The following table may contain pinout information for unsupported device/package combinations. Please see the availability charts elsewhere in the XC5200 Series data sheet for availability information.

Pin	Description		PC84	PQ100	VQ100	TQ144	PG156	PQ160	Boundary Scan Order
	VCC		2	92	89	128	H3	142	-
1.	I/O (A8)		3	93	90	129	H1	143	78
2.	I/O (A9)		4	94	91	130	G1	144	81
3.	I/O		-	95	92	131	G2	145	87
4.	I/O		-	96	93	132	G3	146	90
5.	I/O (A10)		5	97	94	133	F1	147	93
6.	I/O (A11)		6	98	95	134	F2	148	99
7.	I/O		-	-	-	135	E1	149	102
8.	I/O		-	-	-	136	E2	150	105
	GND		-	-	-	137	F3	151	-
9.	I/O		-	-	-	-	D1	152	111
10.	I/O		-	-	-	-	D2	153	114
11.	I/O (A12)		7	99	96	138	E3	154	117
12.	I/O (A13)		8	100	97	139	C1	155	123
13.	I/O		-	-	-	140	C2	156	126

Pin	Description		PC84	PQ100	VQ100	TQ144	PG156	PQ160	Boundary Scan Order
14.	I/O		-	-	-	141	D3	157	129
15.	I/O (A14)		9	1	98	142	B1	158	138
16.	I/O (A15)		10	2	99	143	B2	159	141
	VCC		11	3	100	144	C3	160	-
	GND		12	4	1	1	C4	1	-
17.	GCK1 (A16, I/O)		13	5	2	2	B3	2	150
18.	I/O (A17)		14	6	3	3	A1	3	153
19.	I/O		-	-	-	4	A2	4	159
20.	I/O		-	-	-	5	C5	5	162
21.	I/O (TDI)		15	7	4	6	B4	6	165
22.	I/O (TCK)		16	8	5	7	A3	7	171
	GND		-	-	-	8	C6	10	-
23.	I/O		-	-	-	9	B5	11	174
24.	I/O		-	-	-	10	B6	12	177
25.	I/O (TMS)		17	9	6	11	A5	13	180
26.	I/O		18	10	7	12	C7	14	183
27.	I/O		-	-	-	13	B7	15	186
28.	I/O		-	11	8	14	A6	16	189
29.	I/O		19	12	9	15	A7	17	195
30.	I/O		20	13	10	16	A8	18	198
	GND		21	14	11	17	C8	19	-
	VCC		22	15	12	18	B8	20	-
31.	I/O		23	16	13	19	C9	21	201
32.	I/O		24	17	14	20	B9	22	207
33.	I/O		-	18	15	21	A9	23	210
34.	I/O		-	-	-	22	B10	24	213
35.	I/O		25	19	16	23	C10	25	219
36.	I/O		26	20	17	24	A10	26	222
37.	I/O		-	-	-	25	A11	27	225
38.	I/O		-	-	-	26	B11	28	231
	GND		-	-	-	27	C11	29	-
39.	I/O		27	21	18	28	B12	32	234
40.	I/O		-	22	19	29	A13	33	237
41.	I/O		-	-	-	30	A14	34	240
42.	I/O		-	-	-	31	C12	35	243
43.	I/O		28	23	20	32	B13	36	246
44.	I/O		29	24	21	33	B14	37	249
45.	M1 (I/O)		30	25	22	34	A15	38	258
	GND		31	26	23	35	C13	39	-
46.	M0 (I/O)		32	27	24	36	A16	40	261
	VCC		33	28	25	37	C14	41	-
47.	M2 (I/O)		34	29	26	38	B15	42	264
48.	GCK2 (I/O)		35	30	27	39	B16	43	267
49.	I/O (HDC)		36	31	28	40	D14	44	276
50.	I/O		-	-	-	41	C15	45	279
51.	I/O		-	-	-	42	D15	46	282
52.	I/O		-	32	29	43	E14	47	288
53.	I/O (LDC)		37	33	30	44	C16	48	291
54.	I/O		-	-	-	-	E15	49	294
55.	I/O		-	-	-	-	D16	50	300
	GND		-	-	-	45	F14	51	-
56.	I/O		-	-	-	46	F15	52	303

Pin	Description		PC84	PQ100	VQ100	TQ144	PG156	PQ160	Boundary Scan Order
57.	I/O		-	-	-	47	E16	53	306
58.	I/O		38	34	31	48	F16	54	312
59.	I/O		39	35	32	49	G14	55	315
60.	I/O		-	36	33	50	G15	56	318
61.	I/O		-	37	34	51	G16	57	324
62.	I/O		40	38	35	52	H16	58	327
63.	I/O ($\overline{ERR}$, $\overline{INIT}$)		41	39	36	53	H15	59	330
	VCC		42	40	37	54	H14	60	-
	GND		43	41	38	55	J14	61	-
64.	I/O		44	42	39	56	J15	62	336
65.	I/O		45	43	40	57	J16	63	339
66.	I/O		-	44	41	58	K16	64	348
67.	I/O		-	45	42	59	K15	65	351
68.	I/O		46	46	43	60	K14	66	354
69.	I/O		47	47	44	61	L16	67	360
70.	I/O		-	-	-	62	M16	68	363
71.	I/O		-	-	-	63	L15	69	366
	GND		-	-	-	64	L14	70	-
72.	I/O		-	-	-	-	N16	71	372
73.	I/O		-	-	-	-	M15	72	375
74.	I/O		48	48	45	65	P16	73	378
75.	I/O		49	49	46	66	M14	74	384
76.	I/O		-	-	-	67	N15	75	387
77.	I/O		-	-	-	68	P15	76	390
78.	I/O		50	50	47	69	N14	77	396
79.	I/O		51	51	48	70	R16	78	399
	GND		52	52	49	71	P14	79	-
	DONE		53	53	50	72	R15	80	-
	VCC		54	54	51	73	P13	81	-
	PROG		55	55	52	74	R14	82	-
80.	I/O (D7)		56	56	53	75	T16	83	408
81.	GCK3 (I/O)		57	57	54	76	T15	84	411
82.	I/O		-	-	-	77	R13	85	420
83.	I/O		-	-	-	78	P12	86	423
84.	I/O (D6)		58	58	55	79	T14	87	426
85.	I/O		-	59	56	80	T13	88	432
	GND		-	-	-	81	P11	91	-
86.	I/O		-	-	-	82	R11	92	435
87.	I/O		-	-	-	83	T11	93	438
88.	I/O (D5)		59	60	57	84	T10	94	444
89.	I/O ($\overline{CS0}$)		60	61	58	85	P10	95	447
90.	I/O		-	62	59	86	R10	96	450
91.	I/O		-	63	60	87	T9	97	456
92.	I/O (D4)		61	64	61	88	R9	98	459
93.	I/O		62	65	62	89	P9	99	462
	VCC		63	66	63	90	R8	100	-
	GND		64	67	64	91	P8	101	-
94.	I/O (D3)		65	68	65	92	T8	102	468
95.	I/O ($\overline{RS}$)		66	69	66	93	T7	103	471
96.	I/O		-	70	67	94	T6	104	474
97.	I/O		-	-	-	95	R7	105	480
98.	I/O (D2)		67	71	68	96	P7	106	483

Pin	Description		PC84	PQ100	VQ100	TQ144	PG156	PQ160	Boundary Scan Order
99.	I/O		68	72	69	97	T5	107	486
100.	I/O		-	-	-	98	R6	108	492
101.	I/O		-	-	-	99	T4	109	495
	GND		-	-	-	100	P6	110	-
102.	I/O (D1)		69	73	70	101	T3	113	498
103.	I/O ($\overline{\text{RCLK-BUSY/RDY}}$)		70	74	71	102	P5	114	504
104.	I/O		-	-	-	103	R4	115	507
105.	I/O		-	-	-	104	R3	116	510
106.	I/O (D0, DIN)		71	75	72	105	P4	117	516
107.	I/O (DOUT)		72	76	73	106	T2	118	519
	CCLK		73	77	74	107	R2	119	-
	VCC		74	78	75	108	P3	120	-
108.	I/O (TDO)		75	79	76	109	T1	121	0
	GND		76	80	77	110	N3	122	-
109.	I/O (A0, $\overline{\text{WS}}$)		77	81	78	111	R1	123	9
110.	GCK4 (A1, I/O)		78	82	79	112	P2	124	15
111.	I/O		-	-	-	113	N2	125	18
112.	I/O		-	-	-	114	M3	126	21
113.	I/O (A2, CS1)		79	83	80	115	P1	127	27
114.	I/O (A3)		80	84	81	116	N1	128	30
115.	I/O		-	-	-	117	M2	129	33
116.	I/O		-	-	-	-	M1	130	39
	GND		-	-	-	118	L3	131	-
117.	I/O		-	-	-	119	L2	132	42
118.	I/O		-	-	-	120	L1	133	45
119.	I/O (A4)		81	85	82	121	K3	134	51
120.	I/O (A5)		82	86	83	122	K2	135	54
121.	I/O		-	87	84	123	K1	137	57
122.	I/O		-	88	85	124	J1	138	63
123.	I/O (A6)		83	89	86	125	J2	139	66
124.	I/O (A7)		84	90	87	126	J3	140	69
	GND		1	91	88	127	H2	141	-

Additional No Connect (N.C.) Connections for PQ160 Package

PQ160				
8	30	89	111	136
9	31	90	112	

Notes: Boundary Scan Bit 0 = TDO.T
Boundary Scan Bit 1 = TDO.O
Boundary Scan Bit 1056 = BSCAN.UPD

Pin Locations for XC5206 Devices

The following table may contain pinout information for unsupported device/package combinations. Please see the availability charts elsewhere in the XC5200 Series data sheet for availability information.

Pin	Description	PC84	PQ100	VQ100	TQ144	PQ160	TQ176	PG191	PQ208	Boundary Scan Order
	VCC	2	92	89	128	142	155	J4	183	-
1.	I/O (A8)	3	93	90	129	143	156	J3	184	87
2.	I/O (A9)	4	94	91	130	144	157	J2	185	90
3.	I/O	-	95	92	131	145	158	J1	186	93
4.	I/O	-	96	93	132	146	159	H1	187	99
5.	I/O	-	-	-	-	-	160	H2	188	102
6.	I/O	-	-	-	-	-	161	H3	189	105
7.	I/O (A10)	5	97	94	133	147	162	G1	190	111
8.	I/O (A11)	6	98	95	134	148	163	G2	191	114
9.	I/O	-	-	-	135	149	164	F1	192	117
10.	I/O	-	-	-	136	150	165	E1	193	123
	GND	-	-	-	137	151	166	G3	194	-
11.	I/O	-	-	-	-	152	168	C1	197	126
12.	I/O	-	-	-	-	153	169	E2	198	129
13.	I/O (A12)	7	99	96	138	154	170	F3	199	138
14.	I/O (A13)	8	100	97	139	155	171	D2	200	141
15.	I/O	-	-	-	140	156	172	B1	201	150
16.	I/O	-	-	-	141	157	173	E3	202	153
17.	I/O (A14)	9	1	98	142	158	174	C2	203	162
18.	I/O (A15)	10	2	99	143	159	175	B2	204	165
	VCC	11	3	100	144	160	176	D3	205	-
	GND	12	4	1	1	1	1	D4	2	-
19.	GCK1 (A16, I/O)	13	5	2	2	2	2	C3	4	174
20.	I/O (A17)	14	6	3	3	3	3	C4	5	177
21.	I/O	-	-	-	4	4	4	B3	6	183
22.	I/O	-	-	-	5	5	5	C5	7	186
23.	I/O (TDI)	15	7	4	6	6	6	A2	8	189
24.	I/O (TCK)	16	8	5	7	7	7	B4	9	195
25.	I/O	-	-	-	-	8	8	C6	10	198
26.	I/O	-	-	-	-	9	9	A3	11	201
	GND	-	-	-	8	10	10	C7	14	-
27.	I/O	-	-	-	9	11	11	A4	15	207
28.	I/O	-	-	-	10	12	12	A5	16	210
29.	I/O (TMS)	17	9	6	11	13	13	B7	17	213
30.	I/O	18	10	7	12	14	14	A6	18	219
31.	I/O	-	-	-	-	-	15	C8	19	222
32.	I/O	-	-	-	-	-	16	A7	20	225
33.	I/O	-	-	-	13	15	17	B8	21	234
34.	I/O	-	11	8	14	16	18	A8	22	237
35.	I/O	19	12	9	15	17	19	B9	23	246
36.	I/O	20	13	10	16	18	20	C9	24	249
	GND	21	14	11	17	19	21	D9	25	-
	VCC	22	15	12	18	20	22	D10	26	-
37.	I/O	23	16	13	19	21	23	C10	27	255
38.	I/O	24	17	14	20	22	24	B10	28	258
39.	I/O	-	18	15	21	23	25	A9	29	261
40.	I/O	-	-	-	22	24	26	A10	30	267
41.	I/O	-	-	-	-	-	27	A11	31	270

Pin	Description	PC84	PQ100	VQ100	TQ144	PQ160	TQ176	PG191	PQ208	Boundary Scan Order
42.	I/O	-	-	-	-	-	28	C11	32	273
43.	I/O	25	19	16	23	25	29	B11	33	279
44.	I/O	26	20	17	24	26	30	A12	34	282
45.	I/O	-	-	-	25	27	31	B12	35	285
46.	I/O	-	-	-	26	28	32	A13	36	291
	GND	-	-	-	27	29	33	C12	37	-
47.	I/O	-	-	-	-	30	34	A15	40	294
48.	I/O	-	-	-	-	31	35	C13	41	297
49.	I/O	27	21	18	28	32	36	B14	42	303
50.	I/O	-	22	19	29	33	37	A16	43	306
51.	I/O	-	-	-	30	34	38	B15	44	309
52.	I/O	-	-	-	31	35	39	C14	45	315
53.	I/O	28	23	20	32	36	40	A17	46	318
54.	I/O	29	24	21	33	37	41	B16	47	321
55.	M1 (I/O)	30	25	22	34	38	42	C15	48	330
	GND	31	26	23	35	39	43	D15	49	-
56.	M0 (I/O)	32	27	24	36	40	44	A18	50	333
	VCC	33	28	25	37	41	45	D16	55	-
57.	M2 (I/O)	34	29	26	38	42	46	C16	56	336
58.	GCK2 (I/O)	35	30	27	39	43	47	B17	57	339
59.	I/O (HDC)	36	31	28	40	44	48	E16	58	348
60.	I/O	-	-	-	41	45	49	C17	59	351
61.	I/O	-	-	-	42	46	50	D17	60	354
62.	I/O	-	32	29	43	47	51	B18	61	360
63.	I/O (LDC)	37	33	30	44	48	52	E17	62	363
64.	I/O	-	-	-	-	49	53	F16	63	372
65.	I/O	-	-	-	-	50	54	C18	64	375
	GND	-	-	-	45	51	55	G16	67	-
66.	I/O	-	-	-	46	52	56	E18	68	378
67.	I/O	-	-	-	47	53	57	F18	69	384
68.	I/O	38	34	31	48	54	58	G17	70	387
69.	I/O	39	35	32	49	55	59	G18	71	390
70.	I/O	-	-	-	-	-	60	H16	72	396
71.	I/O	-	-	-	-	-	61	H17	73	399
72.	I/O	-	36	33	50	56	62	H18	74	402
73.	I/O	-	37	34	51	57	63	J18	75	408
74.	I/O	40	38	35	52	58	64	J17	76	411
75.	I/O (ERR, INIT)	41	39	36	53	59	65	J16	77	414
	VCC	42	40	37	54	60	66	J15	78	-
	GND	43	41	38	55	61	67	K15	79	-
76.	I/O	44	42	39	56	62	68	K16	80	420
77.	I/O	45	43	40	57	63	69	K17	81	423
78.	I/O	-	44	41	58	64	70	K18	82	426
79.	I/O	-	45	42	59	65	71	L18	83	432
80.	I/O	-	-	-	-	-	72	L17	84	435
81.	I/O	-	-	-	-	-	73	L16	85	438
82.	I/O	46	46	43	60	66	74	M18	86	444
83.	I/O	47	47	44	61	67	75	M17	87	447
84.	I/O	-	-	-	62	68	76	N18	88	450
85.	I/O	-	-	-	63	69	77	P18	89	456
	GND	-	-	-	64	70	78	M16	90	-
86.	I/O	-	-	-	-	71	79	T18	93	459

Pin	Description	PC84	PQ100	VQ100	TQ144	PQ160	TQ176	PG191	PQ208	Boundary Scan Order
87.	I/O	-	-	-	-	72	80	P17	94	468
88.	I/O	48	48	45	65	73	81	N16	95	471
89.	I/O	49	49	46	66	74	82	T17	96	480
90.	I/O	-	-	-	67	75	83	R17	97	483
91.	I/O	-	-	-	68	76	84	P16	98	486
92.	I/O	50	50	47	69	77	85	U18	99	492
93.	I/O	51	51	48	70	78	86	T16	100	495
	GND	52	52	49	71	79	87	R16	101	-
	DONE	53	53	50	72	80	88	U17	103	-
	VCC	54	54	51	73	81	89	R15	106	-
	PROG	55	55	52	74	82	90	V18	108	-
94.	I/O (D7)	56	56	53	75	83	91	T15	109	504
95.	GCK3 (I/O)	57	57	54	76	84	92	U16	110	507
96.	I/O	-	-	-	77	85	93	T14	111	516
97.	I/O	-	-	-	78	86	94	U15	112	519
98.	I/O (D6)	58	58	55	79	87	95	V17	113	522
99.	I/O	-	59	56	80	88	96	V16	114	528
100.	I/O	-	-	-	-	89	97	T13	115	531
101.	I/O	-	-	-	-	90	98	U14	116	534
	GND	-	-	-	81	91	99	T12	119	-
102.	I/O	-	-	-	82	92	100	U13	120	540
103.	I/O	-	-	-	83	93	101	V13	121	543
104.	I/O (D5)	59	60	57	84	94	102	U12	122	552
105.	I/O (CS0)	60	61	58	85	95	103	V12	123	555
106.	I/O	-	-	-	-	-	104	T11	124	558
107.	I/O	-	-	-	-	-	105	U11	125	564
108.	I/O	-	62	59	86	96	106	V11	126	567
109.	I/O	-	63	60	87	97	107	V10	127	570
110.	I/O (D4)	61	64	61	88	98	108	U10	128	576
111.	I/O	62	65	62	89	99	109	T10	129	579
	VCC	63	66	63	90	100	110	R10	130	-
	GND	64	67	64	91	101	111	R9	131	-
112.	I/O (D3)	65	68	65	92	102	112	T9	132	588
113.	I/O ($\overline{RS}$)	66	69	66	93	103	113	U9	133	591
114.	I/O	-	70	67	94	104	114	V9	134	600
115.	I/O	-	-	-	95	105	115	V8	135	603
116.	I/O	-	-	-	-	-	116	U8	136	612
117.	I/O	-	-	-	-	-	117	T8	137	615
118.	I/O (D2)	67	71	68	96	106	118	V7	138	618
119.	I/O	68	72	69	97	107	119	U7	139	624
120.	I/O	-	-	-	98	108	120	V6	140	627
121.	I/O	-	-	-	99	109	121	U6	141	630
	GND	-	-	-	100	110	122	T7	142	-
122.	I/O	-	-	-	-	111	123	U5	145	636
123.	I/O	-	-	-	-	112	124	T6	146	639
124.	I/O (D1)	69	73	70	101	113	125	V3	147	642
125.	I/O ($\overline{RCLK-BUSY/RDY}$)	70	74	71	102	114	126	V2	148	648
126.	I/O	-	-	-	103	115	127	U4	149	651
127.	I/O	-	-	-	104	116	128	T5	150	654
128.	I/O (D0, DIN)	71	75	72	105	117	129	U3	151	660
129.	I/O (DOUT)	72	76	73	106	118	130	T4	152	663

Pin	Description	PC84	PQ100	VQ100	TQ144	PQ160	TQ176	PG191	PQ208	Boundary Scan Order
	CCLK	73	77	74	107	119	131	V1	153	-
	VCC	74	78	75	108	120	132	R4	154	-
130.	I/O (TDO)	75	79	76	109	121	133	U2	159	-
	GND	76	80	77	110	122	134	R3	160	-
131.	I/O (A0, WS)	77	81	78	111	123	135	T3	161	9
132.	GCK4 (A1, I/O)	78	82	79	112	124	136	U1	162	15
133.	I/O	-	-	-	113	125	137	P3	163	18
134.	I/O	-	-	-	114	126	138	R2	164	21
135.	I/O (A2, CS1)	79	83	80	115	127	139	T2	165	27
136.	I/O (A3)	80	84	81	116	128	140	N3	166	30
137.	I/O	-	-	-	117	129	141	P2	167	33
138.	I/O	-	-	-	-	130	142	T1	168	42
	GND	-	-	-	118	131	143	M3	171	-
139.	I/O	-	-	-	119	132	144	P1	172	45
140.	I/O	-	-	-	120	133	145	N1	173	51
141.	I/O (A4)	81	85	82	121	134	146	M2	174	54
142.	I/O (A5)	82	86	83	122	135	147	M1	175	57
143.	I/O	-	-	-	-	-	148	L3	176	63
144.	I/O	-	-	-	-	136	149	L2	177	66
145.	I/O	-	87	84	123	137	150	L1	178	69
146.	I/O	-	88	85	124	138	151	K1	179	75
147.	I/O (A6)	83	89	86	125	139	152	K2	180	78
148.	I/O (A7)	84	90	87	126	140	153	K3	181	81
	GND	1	91	88	127	141	154	K4	182	-

Additional No Connect (N.C.) Connections for PQ208 and TQ176 Packages

PQ208							TQ176
195	1	39	65	104	143	158	167
196	3	51	66	105	144	169	
206	12	52	91	107	155	170	
207	13	53	92	117	156		
208	38	54	102	118	157		

Notes: Boundary Scan Bit 0 = TDO.T
Boundary Scan Bit 1 = TDO.O
Boundary Scan Bit 1056 = BSCAN.UPD

Pin Locations for XC5210 Devices

The following table may contain pinout information for unsupported device/package combinations. Please see the availability charts elsewhere in the XC5200 Series data sheet for availability information.

Pin	Description	PC84	TQ144	PQ160	TQ176	PQ208	PG223	BG225	PQ240	Boundary Scan Order
	VCC	2	128	142	155	183	J4	VCC*	212	-
1.	I/O (A8)	3	129	143	156	184	J3	E8	213	111
2.	I/O (A9)	4	130	144	157	185	J2	B7	214	114
3.	I/O	-	131	145	158	186	J1	A7	215	117
4.	I/O	-	132	146	159	187	H1	C7	216	123
5.	I/O	-	-	-	160	188	H2	D7	217	126
6.	I/O	-	-	-	161	189	H3	E7	218	129

Pin	Description	PC84	TQ144	PQ160	TQ176	PQ208	PG223	BG225	PQ240	Boundary Scan Order
7.	I/O (A10)	5	133	147	162	190	G1	A6	220	135
8.	I/O (A11)	6	134	148	163	191	G2	B6	221	138
	VCC	-	-	-	-	-	-	VCC*	222	-
9.	I/O	-	-	-	-	-	H4	C6	223	141
10.	I/O	-	-	-	-	-	G4	F7	224	150
11.	I/O	-	135	149	164	192	F1	A5	225	153
12.	I/O	-	136	150	165	193	E1	B5	226	162
	GND	-	137	151	166	194	G3	GND*	227	-
13.	I/O	-	-	-	-	195	F2	D6	228	165
14.	I/O	-	-	-	167	196	D1	C5	229	171
15.	I/O	-	-	152	168	197	C1	A4	230	174
16.	I/O	-	-	153	169	198	E2	E6	231	177
17.	I/O (A12)	7	138	154	170	199	F3	B4	232	183
18.	I/O (A13)	8	139	155	171	200	D2	D5	233	186
19.	I/O	-	-	-	-	-	F4	A3	234	189
20.	I/O	-	-	-	-	-	E4	C4	235	195
21.	I/O	-	140	156	172	201	B1	B3	236	198
22.	I/O	-	141	157	173	202	E3	F6	237	201
23.	I/O (A14)	9	142	158	174	203	C2	A2	238	210
24.	I/O (A15)	10	143	159	175	204	B2	C3	239	213
	VCC	11	144	160	176	205	D3	VCC*	240	-
	GND	12	1	1	1	2	D4	GND*	1	-
25.	GCK1 (A16, I/O)	13	2	2	2	4	C3	D4	2	222
26.	I/O (A17)	14	3	3	3	5	C4	B1	3	225
27.	I/O	-	4	4	4	6	B3	C2	4	231
28.	I/O	-	5	5	5	7	C5	E5	5	234
29.	I/O (TDI)	15	6	6	6	8	A2	D3	6	237
30.	I/O (TCK)	16	7	7	7	9	B4	C1	7	243
31.	I/O	-	-	8	8	10	C6	D2	8	246
32.	I/O	-	-	9	9	11	A3	G6	9	249
33.	I/O	-	-	-	-	12	B5	E4	10	255
34.	I/O	-	-	-	-	13	B6	D1	11	258
35.	I/O	-	-	-	-	-	D5	E3	12	261
36.	I/O	-	-	-	-	-	D6	E2	13	267
	GND	-	8	10	10	14	C7	GND*	14	-
37.	I/O	-	9	11	11	15	A4	F5	15	270
38.	I/O	-	10	12	12	16	A5	E1	16	273
39.	I/O (TMS)	17	11	13	13	17	B7	F4	17	279
40.	I/O	18	12	14	14	18	A6	F3	18	282
	VCC	-	-	-	-	-	-	VCC*	19	-
41.	I/O	-	-	-	-	-	D7	F2	20	285
42.	I/O	-	-	-	-	-	D8	F1	21	291
43.	I/O	-	-	-	15	19	C8	G4	23	294
44.	I/O	-	-	-	16	20	A7	G3	24	297
45.	I/O	-	13	15	17	21	B8	G2	25	306
46.	I/O	-	14	16	18	22	A8	G1	26	309
47.	I/O	19	15	17	19	23	B9	G5	27	318
48.	I/O	20	16	18	20	24	C9	H3	28	321
	GND	21	17	19	21	25	D9	GND*	29	-
	VCC	22	18	20	22	26	D10	VCC*	30	-
49.	I/O	23	19	21	23	27	C10	H4	31	327

Pin	Description	PC84	TQ144	PQ160	TQ176	PQ208	PG223	BG225	PQ240	Boundary Scan Order
50.	I/O	24	20	22	24	28	B10	H5	32	330
51.	I/O	-	21	23	25	29	A9	J2	33	333
52.	I/O	-	22	24	26	30	A10	J1	34	339
53.	I/O	-	-	-	27	31	A11	J3	35	342
54.	I/O	-	-	-	28	32	C11	J4	36	345
55.	I/O	-	-	-	-	-	D11	J5	38	351
56.	I/O	-	-	-	-	-	D12	K1	39	354
	VCC	-	-	-	-	-	-	VCC*	40	-
57.	I/O	25	23	25	29	33	B11	K2	41	357
58.	I/O	26	24	26	30	34	A12	K3	42	363
59.	I/O	-	25	27	31	35	B12	J6	43	366
60.	I/O	-	26	28	32	36	A13	L1	44	369
	GND	-	27	29	33	37	C12	GND*	45	-
61.	I/O	-	-	-	-	-	D13	L2	46	375
62.	I/O	-	-	-	-	-	D14	K4	47	378
63.	I/O	-	-	-	-	38	B13	L3	48	381
64.	I/O	-	-	-	-	39	A14	M1	49	387
65.	I/O	-	-	30	34	40	A15	K5	50	390
66.	I/O	-	-	31	35	41	C13	M2	51	393
67.	I/O	27	28	32	36	42	B14	L4	52	399
68.	I/O	-	29	33	37	43	A16	N1	53	402
69.	I/O	-	30	34	38	44	B15	M3	54	405
70.	I/O	-	31	35	39	45	C14	N2	55	411
71.	I/O	28	32	36	40	46	A17	K6	56	414
72.	I/O	29	33	37	41	47	B16	P1	57	417
73.	M1 (I/O)	30	34	38	42	48	C15	N3	58	426
	GND	31	35	39	43	49	D15	GND*	59	-
74.	M0 (I/O)	32	36	40	44	50	A18	P2	60	429
	VCC	33	37	41	45	55	D16	VCC*	61	-
75.	M2 (I/O)	34	38	42	46	56	C16	M4	62	432
76.	GCK2 (I/O)	35	39	43	47	57	B17	R2	63	435
77.	I/O (HDC)	36	40	44	48	58	E16	P3	64	444
78.	I/O	-	41	45	49	59	C17	L5	65	447
79.	I/O	-	42	46	50	60	D17	N4	66	450
80.	I/O	-	43	47	51	61	B18	R3	67	456
81.	I/O (LDC)	37	44	48	52	62	E17	P4	68	459
82.	I/O	-	-	49	53	63	F16	K7	69	462
83.	I/O	-	-	50	54	64	C18	M5	70	468
84.	I/O	-	-	-	-	65	D18	R4	71	471
85.	I/O	-	-	-	-	66	F17	N5	72	474
86.	I/O	-	-	-	-	-	E15	P5	73	480
87.	I/O	-	-	-	-	-	F15	L6	74	483
	GND	-	45	51	55	67	G16	GND*	75	-
88.	I/O	-	46	52	56	68	E18	R5	76	486
89.	I/O	-	47	53	57	69	F18	M6	77	492
90.	I/O	38	48	54	58	70	G17	N6	78	495
91.	I/O	39	49	55	59	71	G18	P6	79	504
	VCC	-	-	-	-	-	-	VCC*	80	-
92.	I/O	-	-	-	60	72	H16	R6	81	507
93.	I/O	-	-	-	61	73	H17	M7	82	510
94.	I/O	-	-	-	-	-	G15	N7	84	516

Pin	Description	PC84	TQ144	PQ160	TQ176	PQ208	PG223	BG225	PQ240	Boundary Scan Order
95.	I/O	-	-	-	-	-	H15	P7	85	519
96.	I/O	-	50	56	62	74	H18	R7	86	522
97.	I/O	-	51	57	63	75	J18	L7	87	528
98.	I/O	40	52	58	64	76	J17	N8	88	531
99.	I/O (ERR, INIT)	41	53	59	65	77	J16	P8	89	534
	VCC	42	54	60	66	78	J15	VCC*	90	-
	GND	43	55	61	67	79	K15	GND*	91	-
100.	I/O	44	56	62	68	80	K16	L8	92	540
101.	I/O	45	57	63	69	81	K17	P9	93	543
102.	I/O	-	58	64	70	82	K18	R9	94	546
103.	I/O	-	59	65	71	83	L18	N9	95	552
104.	I/O	-	-	-	72	84	L17	M9	96	555
105.	I/O	-	-	-	73	85	L16	L9	97	558
106.	I/O	-	-	-	-	-	L15	R10	99	564
107.	I/O	-	-	-	-	-	M15	P10	100	567
	VCC	-	-	-	-	-	-	VCC*	101	-
108.	I/O	46	60	66	74	86	M18	N10	102	570
109.	I/O	47	61	67	75	87	M17	K9	103	576
110.	I/O	-	62	68	76	88	N18	R11	104	579
111.	I/O	-	63	69	77	89	P18	P11	105	588
	GND	-	64	70	78	90	M16	GND*	106	-
112.	I/O	-	-	-	-	-	N15	M10	107	591
113.	I/O	-	-	-	-	-	P15	N11	108	600
114.	I/O	-	-	-	-	91	N17	R12	109	603
115.	I/O	-	-	-	-	92	R18	L10	110	606
116.	I/O	-	-	71	79	93	T18	P12	111	612
117.	I/O	-	-	72	80	94	P17	M11	112	615
118.	I/O	48	65	73	81	95	N16	R13	113	618
119.	I/O	49	66	74	82	96	T17	N12	114	624
120.	I/O	-	67	75	83	97	R17	P13	115	627
121.	I/O	-	68	76	84	98	P16	K10	116	630
122.	I/O	50	69	77	85	99	U18	R14	117	636
123.	I/O	51	70	78	86	100	T16	N13	118	639
	GND	52	71	79	87	101	R16	GND*	119	-
	DONE	53	72	80	88	103	U17	P14	120	-
	VCC	54	73	81	89	106	R15	VCC*	121	-
	PROG	55	74	82	90	108	V18	M12	122	-
124.	I/O (D7)	56	75	83	91	109	T15	P15	123	648
125.	GCK3 (I/O)	57	76	84	92	110	U16	N14	124	651
126.	I/O	-	77	85	93	111	T14	L11	125	660
127.	I/O	-	78	86	94	112	U15	M13	126	663
128.	I/O	-	-	-	-	-	R14	N15	127	666
129.	I/O	-	-	-	-	-	R13	M14	128	672
130.	I/O (D6)	58	79	87	95	113	V17	J10	129	675
131.	I/O	-	80	88	96	114	V16	L12	130	678
132.	I/O	-	-	89	97	115	T13	M15	131	684
133.	I/O	-	-	90	98	116	U14	L13	132	687
134.	I/O	-	-	-	-	117	V15	L14	133	690
135.	I/O	-	-	-	-	118	V14	K11	134	696
	GND	-	81	91	99	119	T12	GND*	135	-
136.	I/O	-	-	-	-	-	R12	L15	136	699

Pin	Description	PC84	TQ144	PQ160	TQ176	PQ208	PG223	BG225	PQ240	Boundary Scan Order
137.	I/O	-	-	-	-	-	R11	K12	137	708
138.	I/O	-	82	92	100	120	U13	K13	138	711
139.	I/O	-	83	93	101	121	V13	K14	139	714
	VCC	-	-	-	-	-	-	VCC*	140	-
140.	I/O (D5)	59	84	94	102	122	U12	K15	141	720
141.	I/O ($\overline{CS0}$)	60	85	95	103	123	V12	J12	142	723
142.	I/O	-	-	-	104	124	T11	J13	144	726
143.	I/O	-	-	-	105	125	U11	J14	145	732
144.	I/O	-	86	96	106	126	V11	J15	146	735
145.	I/O	-	87	97	107	127	V10	J11	147	738
146.	I/O (D4)	61	88	98	108	128	U10	H13	148	744
147.	I/O	62	89	99	109	129	T10	H14	149	747
	VCC	63	90	100	110	130	R10	VCC*	150	-
	GND	64	91	101	111	131	R9	GND*	151	-
148.	I/O (D3)	65	92	102	112	132	T9	H12	152	756
149.	I/O ($\overline{RS}$)	66	93	103	113	133	U9	H11	153	759
150.	I/O	-	94	104	114	134	V9	G14	154	768
151.	I/O	-	95	105	115	135	V8	G15	155	771
152.	I/O	-	-	-	116	136	U8	G13	156	780
153.	I/O	-	-	-	117	137	T8	G12	157	783
154.	I/O (D2)	67	96	106	118	138	V7	G11	159	786
155.	I/O	68	97	107	119	139	U7	F15	160	792
	VCC	-	-	-	-	-	-	VCC*	161	-
156.	I/O	-	98	108	120	140	V6	F14	162	795
157.	I/O	-	99	109	121	141	U6	F13	163	798
158.	I/O	-	-	-	-	-	R8	G10	164	804
159.	I/O	-	-	-	-	-	R7	E15	165	807
	GND	-	100	110	122	142	T7	GND*	166	-
160.	I/O	-	-	-	-	-	R6	E14	167	810
161.	I/O	-	-	-	-	-	R5	F12	168	816
162.	I/O	-	-	-	-	143	V5	E13	169	819
163.	I/O	-	-	-	-	144	V4	D15	170	822
164.	I/O	-	-	111	123	145	U5	F11	171	828
165.	I/O	-	-	112	124	146	T6	D14	172	831
166.	I/O (D1)	69	101	113	125	147	V3	E12	173	834
167.	I/O ($\overline{RCLK}$-BUSY/RDY)	70	102	114	126	148	V2	C15	174	840
168.	I/O	-	103	115	127	149	U4	D13	175	843
169.	I/O	-	104	116	128	150	T5	C14	176	846
170.	I/O (D0, DIN)	71	105	117	129	151	U3	F10	177	855
171.	I/O (DOUT)	72	106	118	130	152	T4	B15	178	858
	CCLK	73	107	119	131	153	V1	C13	179	-
	VCC	74	108	120	132	154	R4	VCC*	180	-
172.	I/O (TDO)	75	109	121	133	159	U2	A15	181	-
	GND	76	110	122	134	160	R3	GND*	182	-
173.	I/O (A0, $\overline{WS}$)	77	111	123	135	161	T3	A14	183	9
174.	GCK4 (A1, I/O)	78	112	124	136	162	U1	B13	184	15
175.	I/O	-	113	125	137	163	P3	E11	185	18
176.	I/O	-	114	126	138	164	R2	C12	186	21
177.	I/O ($\overline{CS1}$, A2)	79	115	127	139	165	T2	A13	187	27
178.	I/O (A3)	80	116	128	140	166	N3	B12	188	30
179.	I/O	-	-	-	-	-	P4	F9	189	33

Pin	Description	PC84	TQ144	PQ160	TQ176	PQ208	PG223	BG225	PQ240	Boundary Scan Order
180.	I/O	-	-	-	-	-	N4	D11	190	39
181.	I/O	-	117	129	141	167	P2	A12	191	42
182.	I/O	-	-	130	142	168	T1	C11	192	45
183.	I/O	-	-	-	-	169	R1	B11	193	51
184.	I/O	-	-	-	-	170	N2	E10	194	54
	-	-	-	-	-	-	-	GND*		-
	GND	-	118	131	143	171	M3	-	196	-
185.	I/O	-	119	132	144	172	P1	A11	197	57
186.	I/O	-	120	133	145	173	N1	D10	198	66
187.	I/O	-	-	-	-	-	M4	C10	199	69
188.	I/O	-	-	-	-	-	L4	B10	200	75
	VCC	-	-	-	-	-	-	VCC*	201	-
189.	I/O (A4)	81	121	134	146	174	M2	A10	202	78
190.	I/O (A5)	82	122	135	147	175	M1	D9	203	81
191.	I/O	-	-	-	148	176	L3	C9	205	87
192.	I/O	-	-	136	149	177	L2	B9	206	90
193.	I/O	-	123	137	150	178	L1	A9	207	93
194.	I/O	-	124	138	151	179	K1	E9	208	99
195.	I/O (A6)	83	125	139	152	180	K2	C8	209	102
196.	I/O (A7)	84	126	140	153	181	K3	B8	210	105
	GND	1	127	141	154	182	K4	GND*	211	-

Additional No Connect (N.C.) Connections for PQ208 and PQ240 Packages

PQ208					PQ240		
1	53	105	157	208	22	143	219
3	54	107	158		37	158	
51	102	155	206		83	195	
52	104	156	207		98	204	

Notes: * Pins labeled VCC* are internally bonded to a VCC plane within the BG225 package. The external pins are: B2, D8, H15, R8, B14, R1, H1, and R15.
Pins labeled GND* are internally bonded to a ground plane within the BG225 package. The external pins are: A1, D12, G7, G9, H6, H8, H10, J8, K8, A8, F8, G8, H2, H7, H9, J7, J9, M8.
Boundary Scan Bit 0 = TDO.T
Boundary Scan Bit 1 = TDO.O
Boundary Scan Bit 1056 = BSCAN.UPD

Pin Locations for XC5215 Devices

The following table may contain pinout information for unsupported device/package combinations. Please see the availability charts elsewhere in the XC5200 Series data sheet for availability information.

Pin	Description	PQ160	HQ208	HQ240	PG299	BG225	BG352	Boundary Scan Order
	VCC	142	183	212	K1	VCC*	VCC*	-
1.	I/O (A8)	143	184	213	K2	E8	D14	138
2.	I/O (A9)	144	185	214	K3	B7	C14	141
3.	I/O	145	186	215	K5	A7	A15	147
4.	I/O	146	187	216	K4	C7	B15	150
5.	I/O	-	188	217	J1	D7	C15	153
6.	I/O	-	189	218	J2	E7	D15	159
7.	I/O (A10)	147	190	220	H1	A6	A16	162

Pin	Description	PQ160	HQ208	HQ240	PG299	BG225	BG352	Boundary Scan Order
8.	I/O (A11)	148	191	221	J3	B6	B16	165
9.	I/O	-	-	-	H2	-	C17	171
10.	I/O	-	-	-	G1	-	B18	174
	VCC	-	-	222	E1	VCC*	VCC*	-
11.	I/O	-	-	223	H3	C6	C18	177
12.	I/O	-	-	224	G2	F7	D17	183
13.	I/O	149	192	225	H4	A5	A20	186
14.	I/O	150	193	226	F2	B5	B19	189
	GND	151	194	227	F1	GND*	GND*	-
15.	I/O	-	-	-	H5	-	C19	195
16.	I/O	-	-	-	G3	-	D18	198
17.	I/O	-	195	228	D1	D6	A21	201
18.	I/O	-	196	229	G4	C5	B20	207
19.	I/O	152	197	230	E2	A4	C20	210
20.	I/O	153	198	231	F3	E6	B21	213
21.	I/O (A12)	154	199	232	G5	B4	B22	219
22.	I/O (A13)	155	200	233	C1	D5	C21	222
23.	I/O	-	-	-	F4	-	D20	225
24.	I/O	-	-	-	E3	-	A23	234
25.	I/O	-	-	234	D2	A3	D21	237
26.	I/O	-	-	235	C2	C4	C22	243
27.	I/O	156	201	236	F5	B3	B24	246
28.	I/O	157	202	237	E4	F6	C23	249
29.	I/O (A14)	158	203	238	D3	A2	D22	258
30.	I/O (A15)	159	204	239	C3	C3	C24	261
	VCC	160	205	240	A2	VCC*	VCC*	-
	GND	1	2	1	B1	GND*	GND*	-
31.	GCK1 (A16, I/O)	2	4	2	D4	D4	D23	270
32.	I/O (A17)	3	5	3	B2	B1	C25	273
33.	I/O	4	6	4	B3	C2	D24	279
34.	I/O	5	7	5	E6	E5	E23	282
35.	I/O (TDI)	6	8	6	D5	D3	C26	285
36.	I/O (TCK)	7	9	7	C4	C1	E24	294
37.	I/O	-	-	-	A3	-	F24	297
38.	I/O	-	-	-	D6	-	E25	303
39.	I/O	8	10	8	E7	D2	D26	306
40.	I/O	9	11	9	B4	G6	G24	309
41.	I/O	-	12	10	C5	E4	F25	315
42.	I/O	-	13	11	A4	D1	F26	318
43.	I/O	-	-	12	D7	E3	H23	321
44.	I/O	-	-	13	C6	E2	H24	327
45.	I/O	-	-	-	E8	-	G25	330
46.	I/O	-	-	-	B5	-	G26	333
	GND	10	14	14	A5	GND*	GND*	-
47.	I/O	11	15	15	B6	F5	J23	339
48.	I/O	12	16	16	D8	E1	J24	342
49.	I/O (TMS)	13	17	17	C7	F4	H25	345
50.	I/O	14	18	18	B7	F3	K23	351
	VCC	-	-	19	A6	VCC*	VCC*	-
51.	I/O	-	-	20	C8	F2	L24	354
52.	I/O	-	-	21	E9	F1	K25	357
53.	I/O	-	-	-	B8	-	L25	363

Pin	Description	PQ160	HQ208	HQ240	PG299	BG225	BG352	Boundary Scan Order
54.	I/O	-	-	-	A8	-	L26	366
55.	I/O	-	19	23	C9	G4	M23	369
56.	I/O	-	20	24	B9	G3	M24	375
57.	I/O	15	21	25	E10	G2	M25	378
58.	I/O	16	22	26	A9	G1	M26	381
59.	I/O	17	23	27	D10	G5	N24	390
60.	I/O	18	24	28	C10	H3	N25	393
	GND	19	25	29	A10	GND*	GND*	-
	VCC	20	26	30	A11	VCC*	VCC*	-
61.	I/O	21	27	31	B10	H4	N26	399
62.	I/O	22	28	32	B11	H5	P25	402
63.	I/O	23	29	33	C11	J2	P23	405
64.	I/O	24	30	34	E11	J1	P24	411
65.	I/O	-	31	35	D11	J3	R26	414
66.	I/O	-	32	36	A12	J4	R25	417
67.	I/O	-	-	-	B12	-	R24	423
68.	I/O	-	-	-	A13	-	R23	426
69.	I/O	-	-	38	E12	J5	T26	429
70.	I/O	-	-	39	B13	K1	T25	435
	VCC	-	-	40	A16	VCC*	VCC*	-
71.	I/O	25	33	41	A14	K2	U24	438
72.	I/O	26	34	42	C13	K3	V25	441
73.	I/O	27	35	43	B14	J6	V24	447
74.	I/O	28	36	44	D13	L1	U23	450
	GND	29	37	45	A15	GND*	GND*	-
75.	I/O	-	-	-	B15	-	Y26	453
76.	I/O	-	-	-	E13	-	W25	459
77.	I/O	-	-	46	C14	L2	W24	462
78.	I/O	-	-	47	A17	K4	V23	465
79.	I/O	-	38	48	D14	L3	AA26	471
80.	I/O	-	39	49	B16	M1	Y25	474
81.	I/O	30	40	50	C15	K5	Y24	477
82.	I/O	31	41	51	E14	M2	AA25	483
83.	I/O	-	-	-	A18	-	AB25	486
84.	I/O	-	-	-	D15	-	AA24	489
85.	I/O	32	42	52	C16	L4	Y23	495
86.	I/O	33	43	53	B17	N1	AC26	498
87.	I/O	34	44	54	B18	M3	AA23	501
88.	I/O	35	45	55	E15	N2	AB24	507
89.	I/O	36	46	56	D16	K6	AD25	510
90.	I/O	37	47	57	C17	P1	AC24	513
91.	M1 (I/O)	38	48	58	A20	N3	AB23	522
	GND	39	49	59	A19	GND*	GND*	-
92.	M0 (I/O)	40	50	60	C18	P2	AD24	525
	VCC	41	55	61	B20	VCC*	VCC*	-
93.	M2 (I/O)	42	56	62	D17	M4	AC23	528
94.	GCK2 (I/O)	43	57	63	B19	R2	AE24	531
95.	I/O (HDC)	44	58	64	C19	P3	AD23	540
96.	I/O	45	59	65	F16	L5	AC22	543
97.	I/O	46	60	66	E17	N4	AF24	546
98.	I/O	47	61	67	D18	R3	AD22	552
99.	I/O (LDC)	48	62	68	C20	P4	AE23	555

Pin	Description	PQ160	HQ208	HQ240	PG299	BG225	BG352	Boundary Scan Order
100.	I/O	-	-	-	F17	-	AE22	558
101.	I/O	-	-	-	G16	-	AF23	564
102.	I/O	49	63	69	D19	K7	AD20	567
103.	I/O	50	64	70	E18	M5	AE21	570
104.	I/O	-	65	71	D20	R4	AF21	576
105.	I/O	-	66	72	G17	N5	AC19	579
106.	I/O	-	-	73	F18	P5	AD19	582
107.	I/O	-	-	74	H16	L6	AE20	588
108.	I/O	-	-	-	E19	-	AF20	591
109.	I/O	-	-	-	F19	-	AC18	594
	GND	51	67	75	E20	GND*	GND*	-
110.	I/O	52	68	76	H17	R5	AD18	600
111.	I/O	53	69	77	G18	M6	AE19	603
112.	I/O	54	70	78	G19	N6	AC17	606
113.	I/O	55	71	79	H18	P6	AD17	612
	VCC	-	-	80	F20	VCC*	VCC*	-
114.	I/O	-	72	81	J16	R6	AE17	615
115.	I/O	-	73	82	G20	M7	AE16	618
116.	I/O	-	-	-	H20	-	AF16	624
117.	I/O	-	-	-	J18	-	AC15	627
118.	I/O	-	-	84	J19	N7	AD15	630
119.	I/O	-	-	85	K16	P7	AE15	636
120.	I/O	56	74	86	J20	R7	AF15	639
121.	I/O	57	75	87	K17	L7	AD14	642
122.	I/O	58	76	88	K18	N8	AE14	648
123.	I/O (ERR, INIT)	59	77	89	K19	P8	AF14	651
	VCC	60	78	90	L20	VCC*	VCC*	-
	GND	61	79	91	K20	GND*	GND*	-
124.	I/O	62	80	92	L19	L8	AE13	660
125.	I/O	63	81	93	L18	P9	AC13	663
126.	I/O	64	82	94	L16	R9	AD13	672
127.	I/O	65	83	95	L17	N9	AF12	675
128.	I/O	-	84	96	M20	M9	AE12	678
129.	I/O	-	85	97	M19	L9	AD12	684
130.	I/O	-	-	-	N20	-	AC12	687
131.	I/O	-	-	-	M18	-	AF11	690
132.	I/O	-	-	99	N19	R10	AE11	696
133.	I/O	-	-	100	P20	P10	AD11	699
	VCC	-	-	101	T20	VCC*	VCC*	-
134.	I/O	66	86	102	N18	N10	AE9	702
135.	I/O	67	87	103	P19	K9	AD9	708
136.	I/O	68	88	104	N17	R11	AC10	711
137.	I/O	69	89	105	R19	P11	AF7	714
	GND	70	90	106	R20	GND*	GND*	-
138.	I/O	-	-	-	N16	-	AE8	720
139.	I/O	-	-	-	P18	-	AD8	723
140.	I/O	-	-	107	U20	M10	AC9	726
141.	I/O	-	-	108	P17	N11	AF6	732
142.	I/O	-	91	109	T19	R12	AE7	735
143.	I/O	-	92	110	R18	L10	AD7	738
144.	I/O	71	93	111	P16	P12	AE6	744
145.	I/O	72	94	112	V20	M11	AE5	747

Pin	Description	PQ160	HQ208	HQ240	PG299	BG225	BG352	Boundary Scan Order
146.	I/O	-	-	-	R17	-	AD6	750
147.	I/O	-	-	-	T18	-	AC7	756
148.	I/O	73	95	113	U19	R13	AF4	759
149.	I/O	74	96	114	V19	N12	AF3	768
150.	I/O	75	97	115	R16	P13	AD5	771
151.	I/O	76	98	116	T17	K10	AE3	774
152.	I/O	77	99	117	U18	R14	AD4	780
153.	I/O	78	100	118	X20	N13	AC5	783
	GND	79	101	119	W20	GND*	GND*	-
	DONE	80	103	120	V18	P14	AD3	-
	VCC	81	106	121	X19	VCC*	VCC*	-
	PROG	82	108	122	U17	M12	AC4	-
154.	I/O (D7)	83	109	123	W19	P15	AD2	792
155.	GCK3 (I/O)	84	110	124	W18	N14	AC3	795
156.	I/O	85	111	125	T15	L11	AB4	804
157.	I/O	86	112	126	U16	M13	AD1	807
158.	I/O	-	-	127	V17	N15	AA4	810
159.	I/O	-	-	128	X18	M14	AA3	816
160.	I/O	-	-	-	U15	-	AB2	819
161.	I/O	-	-	-	T14	-	AC1	828
162.	I/O (D6)	87	113	129	W17	J10	Y3	831
163.	I/O	88	114	130	V16	L12	AA2	834
164.	I/O	89	115	131	X17	M15	AA1	840
165.	I/O	90	116	132	U14	L13	W4	843
166.	I/O	-	117	133	V15	L14	W3	846
167.	I/O	-	118	134	T13	K11	Y2	852
168.	I/O	-	-	-	W16	-	Y1	855
169.	I/O	-	-	-	W15	-	V4	858
	GND	91	119	135	X16	GND*	GND*	-
170.	I/O	-	-	136	U13	L15	V3	864
171.	I/O	-	-	137	V14	K12	W2	867
172.	I/O	92	120	138	W14	K13	U4	870
173.	I/O	93	121	139	V13	K14	U3	876
	VCC	-	-	140	X15	VCC*	VCC*	-
174.	I/O (D5)	94	122	141	T12	K15	V2	879
175.	I/O (CS0)	95	123	142	X14	J12	V1	882
176.	I/O	-	-	-	X13	-	T1	888
177.	I/O	-	-	-	V12	-	R4	891
178.	I/O	-	124	144	W12	J13	R3	894
179.	I/O	-	125	145	T11	J14	R2	900
180.	I/O	96	126	146	X12	J15	R1	903
181.	I/O	97	127	147	U11	J11	P3	906
182.	I/O (D4)	98	128	148	V11	H13	P2	912
183.	I/O	99	129	149	W11	H14	P1	915
	VCC	100	130	150	X10	VCC*	VCC*	-
	GND	101	131	151	X11	GND*	GND*	-
184.	I/O (D3)	102	132	152	W10	H12	N2	924
185.	I/O (RS)	103	133	153	V10	H11	N4	927
186.	I/O	104	134	154	T10	G14	N3	936
187.	I/O	105	135	155	U10	G15	M1	939
188.	I/O	-	136	156	X9	G13	M2	942
189.	I/O	-	137	157	W9	G12	M3	948

Pin	Description	PQ160	HQ208	HQ240	PG299	BG225	BG352	Boundary Scan Order
190.	I/O	-	-	-	X8	-	M4	951
191.	I/O	-	-	-	V9	-	L1	954
192.	I/O (D2)	106	138	159	W8	G11	J1	960
193.	I/O	107	139	160	X7	F15	K3	963
	VCC	-	-	161	X5	VCC*	VCC*	
194.	I/O	108	140	162	V8	F14	J2	966
195.	I/O	109	141	163	W7	F13	J3	972
196.	I/O	-	-	164	U8	G10	K4	975
197.	I/O	-	-	165	W6	E15	G1	978
	GND	110	142	166	X6	GND*	GND*	
198.	I/O	-	-	-	T8	-	H2	984
199.	I/O	-	-	-	V7	-	H3	987
200.	I/O	-	-	167	X4	E14	J4	990
201.	I/O	-	-	168	U7	F12	F1	996
202.	I/O	-	143	169	W5	E13	G2	999
203.	I/O	-	144	170	V6	D15	G3	1002
204.	I/O	111	145	171	T7	F11	F2	1008
205.	I/O	112	146	172	X3	D14	E2	1011
206.	I/O (D1)	113	147	173	U6	E12	F3	1014
207.	I/O (RCLK-BUSY/RDY)	114	148	174	V5	C15	G4	1020
208.	I/O	-	-	-	W4	-	D2	1023
209.	I/O	-	-	-	W3	-	F4	1032
210.	I/O	115	149	175	T6	D13	E3	1035
211.	I/O	116	150	176	U5	C14	C2	1038
212.	I/O (D0, DIN)	117	151	177	V4	F10	D3	1044
213.	I/O (DOUT)	118	152	178	X1	B15	E4	1047
	CCLK	119	153	179	V3	C13	C3	-
	VCC	120	154	180	W1	VCC*	VCC*	-
214.	I/O (TDO)	121	159	181	U4	A15	D4	0
	GND	122	160	182	X2	GND*	GND*	-
215.	I/O (A0, $\overline{WS}$)	123	161	183	W2	A14	B3	9
216.	GCK4 (A1, I/O)	124	162	184	V2	B13	C4	15
217.	I/O	125	163	185	R5	E11	D5	18
218.	I/O	126	164	186	T4	C12	A3	21
219.	I/O (A2, CS1)	127	165	187	U3	A13	D6	27
220.	I/O (A3)	128	166	188	V1	B12	C6	30
221.	I/O	-	-	-	R4	-	B5	33
222.	I/O	-	-	-	P5	-	A4	39
223.	I/O	-	-	189	U2	F9	C7	42
224.	I/O	-	-	190	T3	D11	B6	45
225.	I/O	129	167	191	U1	A12	A6	51
226.	I/O	130	168	192	P4	C11	D8	54
227.	I/O	-	169	193	R3	B11	B7	57
228.	I/O	-	170	194	N5	E10	A7	63
229.	I/O	-	-	195	T2	-	D9	66
230.	I/O	-	-	-	R2	-	C9	69
	GND	131	171	196	T1	GND*	GND*	-
231.	I/O	132	172	197	N4	A11	B8	75
232.	I/O	133	173	198	P3	D10	D10	78
233.	I/O	-	-	199	P2	C10	C10	81
234.	I/O	-	-	200	N3	B10	B9	87
	VCC	-	-	201	R1	VCC*	VCC*	-

Pin	Description	PQ160	HQ208	HQ240	PG299	BG225	BG352	Boundary Scan Order
235.	I/O	-	-	-	M5	-	B11	90
236.	I/O	-	-	-	P1	-	A11	93
237.	I/O (A4)	134	174	202	N1	A10	D12	99
238.	I/O (A5)	135	175	203	M3	D9	C12	102
239.	I/O	-	176	205	M2	C9	B12	105
240.	I/O	136	177	206	L5	B9	A12	111
241.	I/O	137	178	207	M1	A9	C13	114
242.	I/O	138	179	208	L4	E9	B13	117
243.	I/O (A6)	139	180	209	L3	C8	A13	126
244.	I/O (A7)	140	181	210	L2	B8	B14	129
	GND	141	182	211	L1	GND*	GND*	-

Additional No Connect (N.C.) Connections for HQ208 and HQ240 Packages

HQ208		HQ240
206	102	219
207	104	22
208	105	37
1	107	83
3	155	98
51	156	143
52	157	158
53	158	204
54	-	-

Notes: * Pins labeled VCC* are internally bonded to a VCC plane within the BG225 and BG352 packages. The external pins for the BG225 are: B2, D8, H15, R8, B14, R1, H1, and R15. The external pins for the BG352 are: A10, A17, B2, B25, D13, D19, D7, G23, H4, K1, K26, N23, P4, U1, U26, W23, Y4, AC14, AC20, AC8, AE2, AE25, AF10, and AF17.
Pins labeled GND* are internally bonded to a ground plane within the BG225 and BG352 packages. The external pins for the BG225 are: A1, D12, G7, G9, H6, H8, H10, J8, K8, A8, F8, G8, H2, H7, H9, J7, J9, M8. The external pins for the BG352 are: A1, A2, A5, A8, A14, A19, A22, A25, A26, B1, B26, E1, E26, H1, H26, N1, P26, W1, W26, AB1, AB26, AE1, AE26, AF1, AF13, AF19, AF2, AF22, AF25, AF26, AF5, AF8.

Boundary Scan Bit 0 = TDO.T
Boundary Scan Bit 1 = TDO.O
Boundary Scan Bit 1056 = BSCAN.UPD

Product Availability

	PINS	64	84	100	100	144	156	160	176	191	208	208	223	225	240	240	299	352
	TYPE	Plast. VQFP	Plast. PLCC	Plast. PQFP	Plast. VQFP	Plast. TQFP	Ceram. PGA	Plast. PQFP	Plast. TQFP	Ceram. PGA	High-Perf. QFP	Plast. PQFP	Ceram. PGA	Plast. BGA	High-Perf. QFP	Plast. PQFP	Ceram. PGA	Plast. BGA
	CODE	VQ64*	PC84	PQ100	VQ100	TQ144	PG156	PQ160	TQ176	PG191	HQ208	PQ208	PG223	BG225	HQ240	PQ240	PG299	BG352
XC5202	-6	CI	CI	CI	CI	CI	CI											
	-5	CI	CI	CI	CI	CI	CI											
	-4	C	C	C	C	C	C											
	-3	C	C	C	C	C	C											
XC5204	-6		CI	CI	CI	CI	CI	CI										
	-5		CI	CI	CI	CI	CI	CI										
	-4		C	C	C	C	C	C										
	-3		C	C	C	C	C	C										
XC5206	-6		CI	CI	CI	CI		CI	CI	CI		CI						
	-5		CI	CI	CI	CI		CI	CI	CI		CI						
	-4		C	C	C	C		C	C	C		C						
	-3		C	C	C	C		C	C	C		C						
XC5210	-6		CI			CI		CI	CI			CI	CI	CI		CI		
	-5		CI			CI		CI	CI			CI	CI	CI		CI		
	-4		C			C		C	C			C	C	C		C		
	-3		C			C		C	C			C	C	C		C		
XC5215	-6							CI			CI			CI	CI		CI	CI
	-5							C			C			C	C		C	C
	-4							C			C			C	C		C	C
	-3							C			C			C	C		C	C

7/8/98

C = Commercial T_J = 0° to +85°C

I= Industrial T_J = -40°C to +100°C

* VQ64 package supports Master Serial, Slave Serial, and Express configuration modes only.

User I/O Per Package

Device	Max I/O	Package Type																
		VQ64	PC84	PQ100	VQ100	TQ144	PG156	PQ160	TQ176	PG191	HQ208	PQ208	PG223	BG225	HQ240	PQ240	PG299	BG352
XC5202	84	52	65	81	81	84	84											
XC5204	124		65	81	81	117	124	124										
XC5206	148		65	81	81	117		133	148	148		148						
XC5210	196		65			117		133	149			164	196	196		196		
XC5215	244							133			164			196	197		244	244

7/8/98

Ordering Information

Example: **XC5210-6PQ208C**

- Device Type
- Speed Grade
- Temperature Range
- Number of Pins
- Package Type

Revisions

Version	Description
12/97	Rev 5.0 added -3, -4 specification
7/98	Rev 5.1 added Spartan family to comparison, removed HQ304
11/98	Rev 5.2 All specifications made final.

4.1 High-Level Description of I/O Card Operations

The Emerald-MM-DIO card is used to provide four extra serial ports to input from the four analysis computers. The processor card has only two serial ports and must be supplemented to allow serial communication with four analysis computers.

The Emerald-MM-DIO card is used to provide digital I/O bits to input instructions from the operator switch used to select the mode of the system (Assay, Background, Measurement Control, Gamma Calibration). It also provided digital I/O bits to output the unclassified results of the measurements to the data barrier.

4.1.1 Brief Overview Narrative for the I/O Card

The Emerald-MM-DIO card is a PC104 card that provides four extra serial ports. The port address and interrupt levels can be selected from a wide range of valid choices to avoid any conflicts with the two serial ports provided on the 3SXi processor card.

The Emerald-MM-DIO card also provides 48 bits of digital I/O arranged as six 8-bit ports. The port address can be selected from a wide range of valid choices to avoid any conflicts with other I/O operations. The card also has the ability to cause an interrupt whenever an edge of the selected polarity is detected on any of 24 input bits (unused for this application).

4.1.2 Critical Specifications for the I/O Card

The single I/O board must be able to handle four separate serial ports each with a selectable interrupt line. The nearly unconstrained IRQ selection ability for the Emerald-MM-DIO card is very useful. The same board must handle at least three bytes of digital I/O.

4.2 Diamond Systems, Emerald-MM-DIO – Digital I/O Card

The following sections provide information regarding the Emerald-MM-DIO card, which is used as the system I/O card.

4.2.1 Specifications for the Diamond Systems Emerald-MM-DIO Card

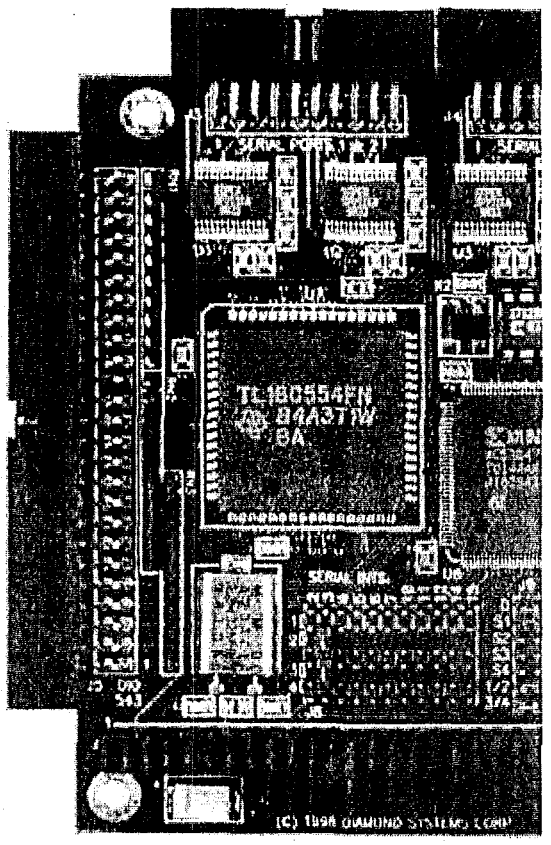
The following are the vendor specifications for the Emerald-MM-DIO card from the vendor's Internet site at:

http://www.diamondsystems.com/emmdio.htm

DIAMOND SYSTEMS CORPORATION

Emerald-MM-DIO-XT™

4 - Port RS-232 + 48 Digital I/O PC/104 Module



Features

- 4 RS-232 serial ports
- Flexible address and interrupt selection
- Full Interrupt Sharing Capability
- Interrupt status register for Windows NT
- Dual 20-pin I/O headers (2 ports per header) for serial ports
- 48 digital I/O lines
- Edge detection capability
- +5V only power supply
- Compatible with the WinSystems PCM-COM4 and PCM-UIO48A
- Two boards in one saves space and lowers cost

Description

For users who need a higher level of integration than our standard products deliver, we offer this two-in-one board with 4 RS-232 serial ports, 48 digital I/O lines, and extended temperature operation.

Each serial port provides the standard set of PC serial port signals. The ST16C554 quad UART chip on board contains a 16-byte FIFO for each port and supports data rates of up to 115kbps. Interrupt sharing is fully supported among the four serial ports, and an interrupt status register enables the board to operate under Windows NT. Eight different address combinations are possible, and any of 11 different interrupt levels can be used.

The 48 digital I/O lines feature programmable direction on a line by line basis, and all lines have 10KOhms pull-up resistors. One group of 24 lines includes the additional feature of programmable edge detection. Both active line selection and edge polarity are programmable. Status registers indicate which, if any, lines have changed logic levels in the selected manner since the last status check. An interrupt can be generated on the PC/104 bus whenever the indicated activity occurs.

♦Edge Detection Feature

The 24 digital I/O lines on ports 0 - 2 have edge detection capability. Active lines and edge polarity (rising or falling edges) are selectable line by line. When a selected line changes in the selected direction, a corresponding bit is set in an edge detection register. The 24 bits in these detection registers can be read and cleared by software. They are also funneled into an interrupt circuit, so that interrupts can be generated on the PC/104 bus when a change of state occurs.

♦Interrupt Status Register

The interrupt status register indicates the status of each port's interrupt request line. This register is required for proper operation under Windows NT. The status register will operate regardless of whether interrupt sharing is enabled. If two or more ports are sharing the same interrupt level, the status register will still indicate the correct status of each port's interrupt request line.

7	6	5	4	3	2	1	0
X	X	X	X	INT4	INT3	INT3	INT1

X	Bit not used; generally reads back as a 1
INT4-INT1	Status of interrupt request for each port: **0** no interrupt request active **1** interrupt request active

♦Interrupt Sharing

Each serial port requires an interrupt line for proper operation. Since there are a limited number of interrupts on the PC/104 bus, using multiple serial ports on a single computer can quickly become a problem. For this reason Emerald-MM-DIO fully supports interrupt sharing, using tristate bus drivers and a status register to indicate the interrupt status of each port. Any number of ports or boards can share the same interrupt level.

♦I/O Header Pinout

Serial Ports

(Serial ports 1 and 2 shown; same pinout used for ports 3 and 4)

DCD 1	1	2	DSR 1
RXD 1	3	4	RTS 1
TXD 1	5	6	CTS 1
DTR 1	7	8	RI 1
GND	9	10	N/C
DCD 2	11	12	DSR 2
RXD 2	13	14	RTS 2
TXD 2	15	16	CTS 2
DTR 2	17	18	RI 2
GND	19	20	N/C

Digital I/O

Digital I/O (Digital I/O ports 0, 1, 2 shown; same pinout used for ports 3, 4 and 5).

Port 2 Bit 7	1	2	GND
Port 2 Bit 6	3	4	GND
Port 2 Bit 5	5	6	GND
Port 2 Bit 4	7	8	GND
Port 2 Bit 3	9	10	GND
Port 2 Bit 2	11	12	GND
Port 2 Bit 1	13	14	GND
Port 2 Bit 0	15	16	GND
Port 1 Bit 7	17	18	GND
Port 1 Bit 6	19	20	GND
Port 1 Bit 5	21	22	GND
Port 1 Bit 4	23	24	GND
Port 1 Bit 3	25	26	GND
Port 1 Bit 2	27	28	GND
Port 1 Bit 1	29	30	GND
Port 1 Bit 0	31	32	GND
Port 0 Bit 7	33	34	GND
Port 0 Bit 6	35	36	GND
Port 0 Bit 5	37	38	GND
Port 0 Bit 4	39	40	GND
Port 0 Bit 3	41	42	GND
Port 0 Bit 2	43	44	GND
Port 0 Bit 1	45	46	GND
Port 0 Bit 0	47	48	GND
+5V	49	50	GND

♦Specifications

Serial Ports	
No. of serial ports	4, RS-232
Maximum baud rate	115kbps
Communications parameters	5, 6, 7, or 8 data bits; Even, odd, or no parity
Short circuit protection	All outputs protected against continuous short circuit
Input impedance	3KOhms min
Input voltage swing	±30V max
Output voltage swing	±5V min, ±7V typical
Digital I/O	
No. of lines	48
Direction	Programmable bit by bit
Input voltage:	
- Logic 0	0.0V min, 0.8V max
- Logic 1	2.0V min, 5.0V max
Output voltage:	
Logic 0	0.0V min, 0.4V max
Logic 1	3.86V min, 5.0V max
Output current	±8mA max per line
General	
I/O headers:	
- Serial ports	2 20-position (2x10) headers
- Digital I/O	2 50-pin (2x25) headers All I/O headers mate with standard ribbon cable (IDC) connectors
Dimensions	3.55" x 3.775" LxW (PC/104 standard)
Power supply	+5VDC ±5%
Current consumption	100mA typical, all outputs unloaded
Operating temperature	-40 to +85°C
Operating humidity	5% to 95% noncondensing
PC/104 bus	8 bit and 16-bit bus headers are installed (16-bit header is used for interrupt levels only)

♦ Ordering Information

Emerald-MM-DIO-XT 4 RS-232 serial ports + 48 Digital I/O PC/104 Module
Extended Temperature

♦ For More Information

Call 1 (800) 36-PC104 or (650) 813-1100 for more technical information on this product, for a free copy of our full-line catalog, or to place an order.

Download the Emerald-MM-DIO Manual - Adobe Acrobat format

Made in USA.

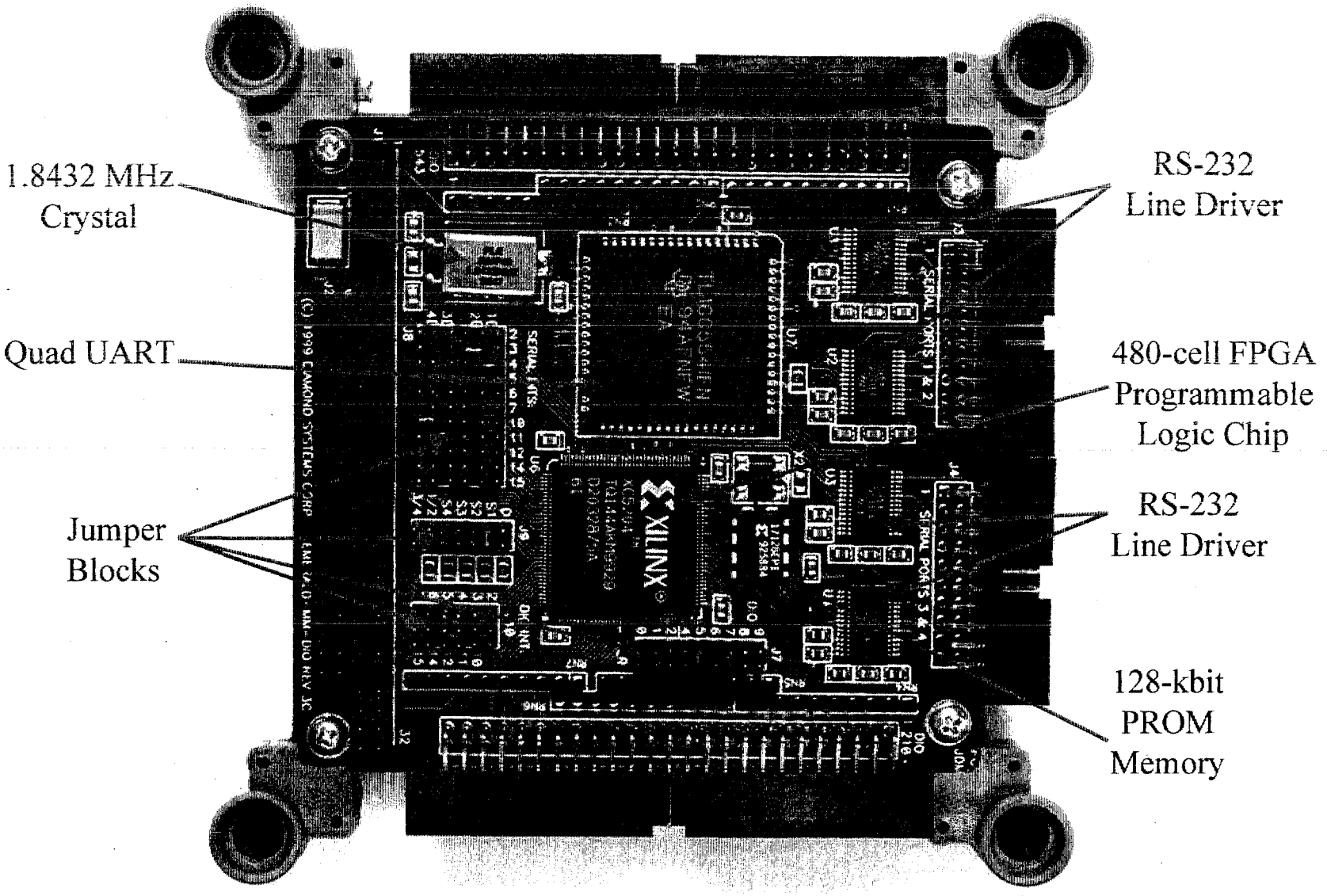
TOP OF PAGE | ABOUT DIAMOND SYSTEMS | NEXT PRODUCT | PREVIOUS PRODUCT | PRODUCT INDEX | PRICE LIST | REQUEST MORE INFO | ORDERING INFO | SEND EMAIL

4.2.2 Photographs of the Diamond Systems Emerald-MM-DIO Card

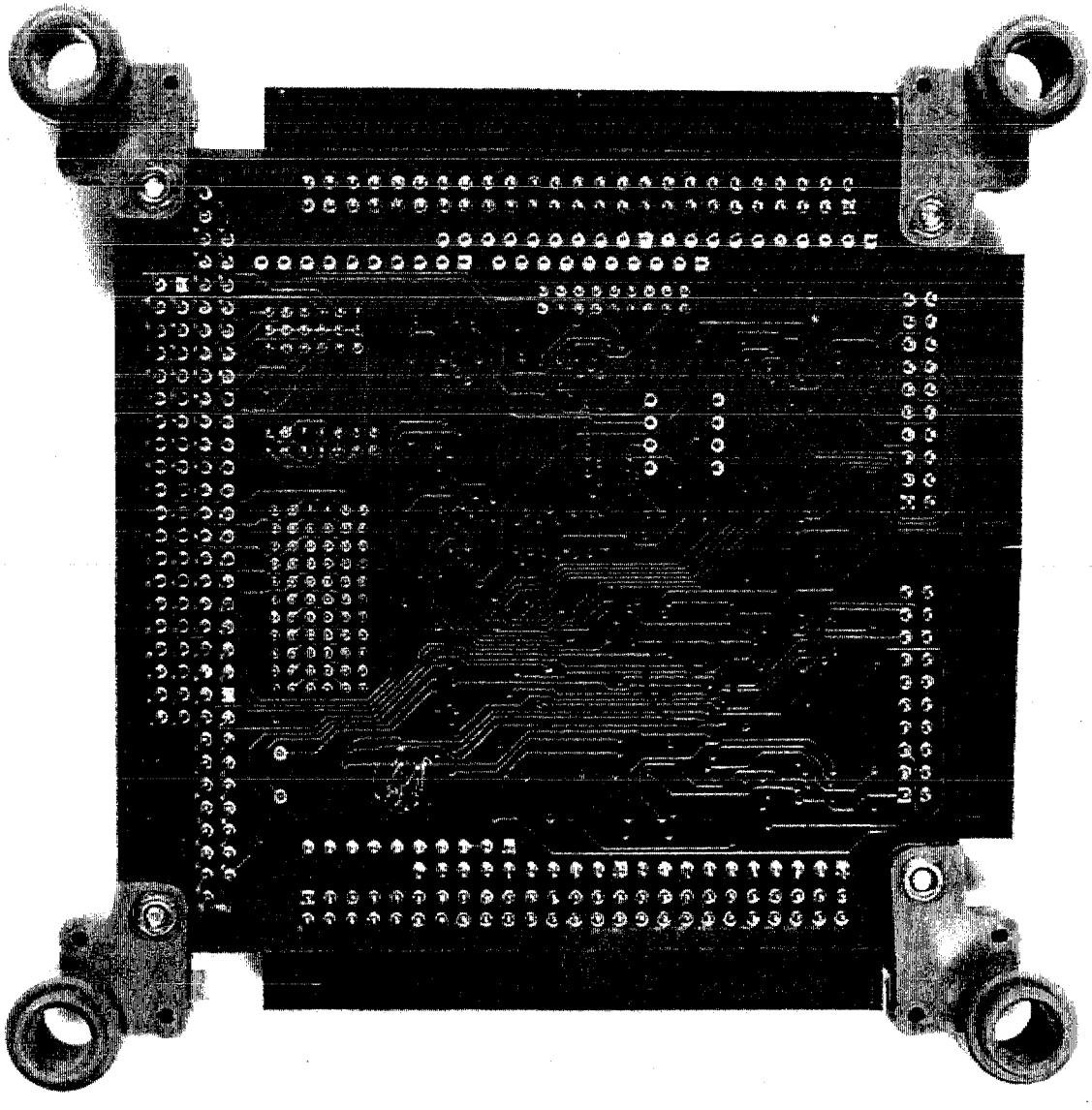
This section contains:

- Labeled top-view photograph of the Diamond Systems Emerald-MM-DIO Card
- Labeled bottom-view photograph of the Diamond Systems Emerald-MM-DIO Card.

EMERALD-MM-DIO – Top-View Photograph



EMERALD-MM-DIO – Bottom-View Photograph



4.2.3 X-Ray of the Diamond Systems Emerald-MM-DIO Card

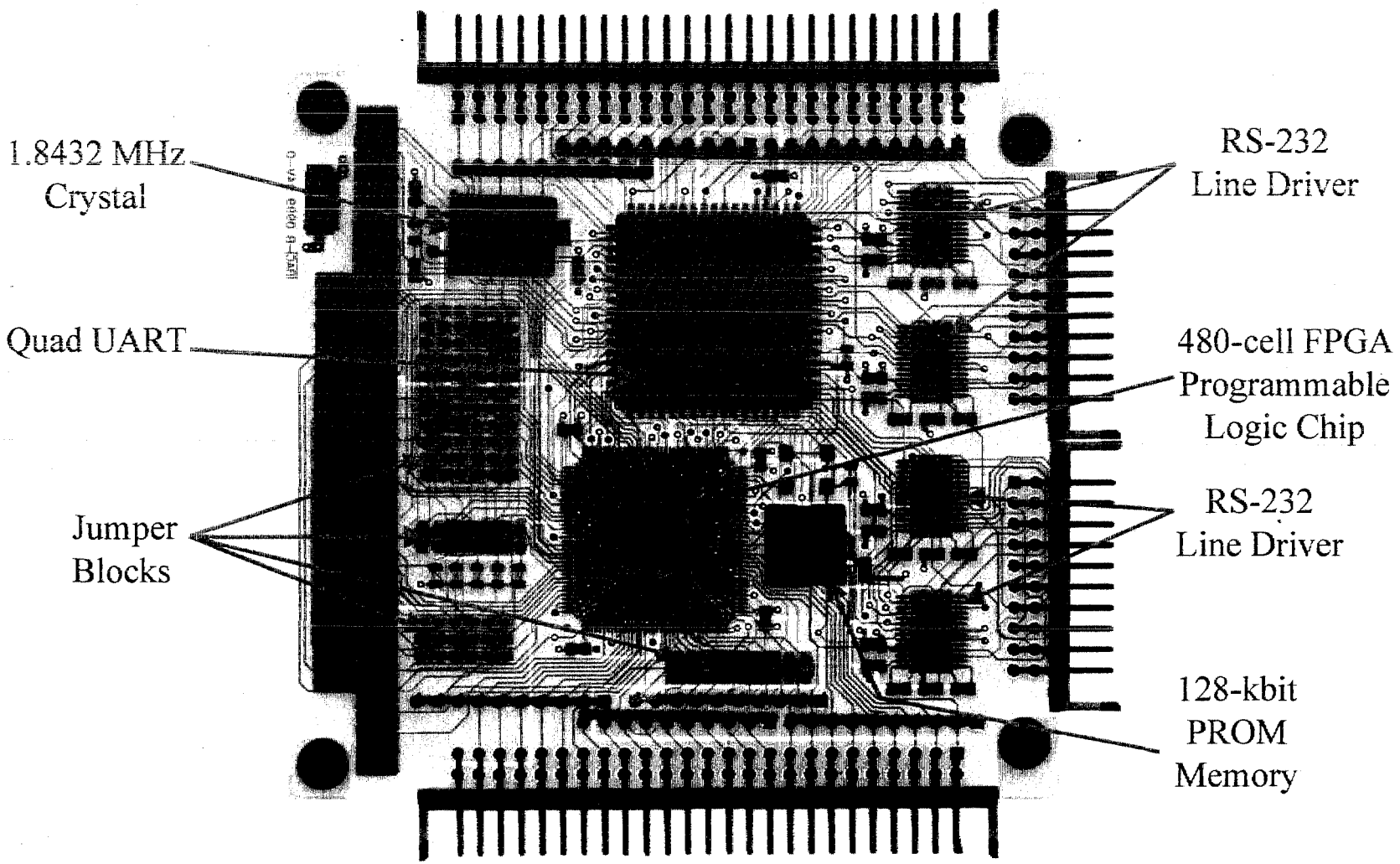
This section contains:

- Labeled top-view x-ray of the Emerald-MM-DIO Card.

The x-ray can be compared to the top-view photograph in section 4.2.2. Note that the ICs are mounted on only one side of the Emerald-MM-DIO printed circuit (PC) board. Thus, the large die in the center of both the UART and the FPGA ICs can be seen without interfering ICs from the other side of the PC board. The traces internal to the IC between the legs and the die can be seen. However, PC-feed-through holes can be seen in the x-ray under the FPGA. These holes correspond to those observed in the bottom-view photograph. The PC board traces and the internal IC traces can easily be distinguished for the FPGA. Traces from the top and bottom sides of the PC board can readily be distinguished where the bottom-side traces cross topside traces. The resolution of the actual x-ray negatives is considerably better than used for these digital images. For example, the very fine wires internal to the FPGA between the die and the internal traces nearly can be seen even at this digital-scan resolution.

The x-ray energy can be varied to become more penetrating. Presumably, the crystal in the can could be better viewed with x-rays that are more energetic. Ceramic ICs (UART) and plastic ICs (FPGA) warrant different x-ray energies for ideal viewing. The brightness, contrast, and gamma were optimized to maximize total detail for this print.

EMERALD-MM-DIO – Top-View X-Ray



4.2.4 Jumper Settings for the Diamond Systems Emerald-MM-DIO Card

Note locations of the jumpers is shown the figure on page 3 of the technical manual.

The Emerald-MM-DIO card jumpers used select:

- Serial-port (COM) addresses at 100 hex, 108 hex, 110 hex, and 118 hex (non-standard)
- Digital I/O (DIO) port addresses 300 hex to 305 hex
- IRQ levels 4, 3, 11, and 10 for the COM ports respectively (standard for COM1 to COM4)
- Terminating resisters (1-kohm) used for the COM port IRQ lines
- No IRQ selected for the DIO ports.

Table 1. Base-Address Selections for COM and DIO Ports via Jumper-7

Jumper	**Function**	**Default**	**Used**	**Description**
J7-S0	Serial Ports & Interrupts	Out = 1	In = 0	See Page 7 or Table below
J7-S1	Serial Ports & Interrupts	In = 0	In = 0	
J7-S2	Serial Ports & Interrupts	Out = 1	Out = 1	
-D0	Digital I/O Base address	0	0	No jumpers assumed 0
-D1	Digital I/O Base address	0	0	No jumpers assumed 0
-D1	Digital I/O Base address	0	0	No jumpers assumed 0
-D3	Digital I/O Base address	0	0	No jumpers assumed 0
J7-D4	Digital I/O Base address	In = 0	In = 0	Out = 1, In = 0 in base address bit
J7-D5	Digital I/O Base address	In = 0	In = 0	Default = 10 0000 0000 b = 200 h
J7-D6	Digital I/O Base address	In = 0	In = 0	Used = 11 0000 0000 b = 300 h
J7-D7	Digital I/O Base address	In = 0	In = 0	
J7-D8	Digital I/O Base address	In = 0	Out = 1	
J7-D9	Digital I/O Base address	Out = 1	Out = 1	

Table 2. Base-Address Values for Serial Ports Based on Jumper-7 S-value from Table 1

S-Value Selected by J7	**Port1 Base Address**	**Port2 Base Address**	**Port3 Base Address**	**Port4 Base Address**	**Interrupt Status**
0 = Std COMs	3F8	2F8	3E8	2E8	220
1	3E8	2E8	3A8	2A8	220
2	380	388	288	230	224
3	240	248	260	268	224
4 = Used	100	108	110	118	240
5 = Default	120	128	130	138	244
6	140	148	150	158	248
7	160	168	170	178	24C

Table 3. Interrupt Level Selection for Serial Ports

Jumper	Function	Default	Used	Description
J8	Interrupt levels IRQ			Select IRQ for each Port
	Port1 Rows 1 & 2	IRQ3	IRQ4	
	Port2 Rows 2 & 3	IRQ3	IRQ3	
	Port3 Rows 4 & 5	IRQ3	IRQ11	
	Port4 Rows 5 & 6	IRQ3	IRQ10	

Table 4. Pull-Down Resister Enabling for Interrupt Lines and Interrupt Sharing for Serial Ports

Jumper	Function	Default	Used	Description
J9-D	DIO Interrupt	In	Out	In = 1Kohm resistor enabled
J9-S1	Serial Port1 Interrupt	In	In	
J9-S2	Serial Port2 Interrupt	Out	In	Out = 1Kohm resistor disabled
J9-S3	Serial Port3 Interrupt	Out	In	
J9-S4	Serial Port4 Interrupt	Out	In	
J9-1=2	Ports 1&2 Share Interrupt	In	Out	In = Interrupt Sharing enabled
J9-3=4	Ports 3&4 Share Interrupt	In	Out	Out = No Interrupt Sharing

Table 5. Interrupt Level Selection for DIO Ports

Jumper	Function	Default	Used	Description
J10-2	DIO Interrupt Level 2	Out	Out	In = DIO uses IRQ2 (Cascade)
J10-3	DIO Interrupt Level 3	Out	Out	In = DIO uses IRQ3 (COM2)
J10-4	DIO Interrupt Level 4	Out	Out	In = DIO uses IRQ4 (COM1)
J10-5	DIO Interrupt Level 5	In	Out	In = DIO uses IRQ5 = Default (LPT2)
J10-6	DIO Interrupt Level 6	Out	Out	In = DIO uses IRQ6 (Floppy)
J10-7	DIO Interrupt Level 7	Out	Out	In = DIO uses IRQ7 (LPT)
J10-10	DIO Interrupt Level 10	Out	Out	In = DIO uses IRQ10 (COM4)
J10-11	DIO Interrupt Level 11	Out	Out	In = DIO uses IRQ11 (COM3)
J10-12	DIO Interrupt Level 12	Out	Out	In = DIO uses IRQ12
J10-14	DIO Interrupt Level 14	Out	Out	In = DIO uses IRQ14 (IDE hard disk)
J10-15	DIO Interrupt Level 15	Out	Out	In = DIO uses IRQ15

4.2.5 Parameter Settings Set by Software for the Diamond Systems Emerald-MM-DIO Card

The software in DATA_ATT.EXE sets the serial port parameters to the values shown in the following table. The DATA_ATT.EXE application program handles reading information from the four measurement computers via the serial ports and outputting unclassified pass/fail indications to the DIO ports.

Table of Software Set Parameters for the DIO Card

Parameter or Characteristic	**Setting Used**
Baud Rate	9600
Parity	None
Character length	8 bits
Number of stop bits	1 stop bit
Protocol	PROT_NONNON

4.2.6 Schematics for the Diamond Systems Emerald-MM-DIO Card

The schematics for the Emerald-MM-DIO card require a non-disclosure agreement with Diamond Systems.

4.2.7 Parts List for the Diamond Systems Emerald-MM-DIO Card

The following table lists the integrated circuits used on the DIO card.

Table of ICs Used on the DIO Card

Socket	Vendor	Part Number	Description
U1-4	SIPEX	SP211CA	RS-232 Line Driver
U5	XILINX	17128EPI 925884	128-kbit Once Programmable Memory for the FPGA
U6	XILINX	XC5204	FPGA – 480 Cells
U7	Texas Instruments	TL 16C554	Quad UART
Y1	PLE	SRMP49	1.8432-MHz Crystal for UART

4.2.8 Voltage and Current Requirements for the Diamond Systems Emerald-MM-DIO Card

Power is provided to the DIO card via the standard PC104 AT bus extension. The following table provides the pins used and the power consumption.

Table Providing Power Information for DIO Card

Pin	**PC-104 Standard**	**Comments**
B1	Ground	
B3	+5 VDC	Typical: 0.5 watts 100 ma (all outputs unloaded)
B5	-5 VDC	Not used
B7	-12 VDC	Not used
B9	+12 VDC	Not used
B31	Ground	
B29	+5 VDC	

4.2.9 Supplementary Narrative Describing Functional Blocks and Implementation for the Diamond Systems Emerald-MM-DIO Card

The TL16C554 asynchronous communications element contains the logic to handle four serial ports. The communication between the CPU and this IC is handled by a Xilinx XC5204 field programmable gate array (FPGA) IC via jumper-selected port addresses. The logic programmed into the FPGA selects which of the four universal asynchronous receiver/transmitter (UART) sections the 8-bits of port-related data are intended for depending on the port address used by the software. The TL16C554's multiplexed 8-bit data (D7-D0) is routed between the several CPU ports and the four UART sections via four chip select lines (CSa*, CSb*, CSc* & CSd*) and common read and write strobes (IOR* & IOW*). Traffic control for the multiplexed, 8-bit, bi-directional data lines (D7-D0) is by two common control lines (RXRDY* = receiver ready and TXRDY* = transmitter ready). The data lines have access to eight registers within each UART section determined by three IC address lines (A2-A0). Details are listed on page 16 of the TL16C554 data sheet provided for clarity in section 4.3.2 of this document.

The TL16C554 communicates directly with four independent RS-232 line driver/ receiver ICs (SIPEX SP211CA) via four sets of six control lines (e.g., CTSa* = clear to send, DCDa* = data carrier detect, DTRa* = data terminal ready, INTa = interrupt output, RIa* = ring detect indicator, and RTSa* = request to send). It also uses four pairs of serial data lines (e.g., RXa = serial input and TXa = serial output).

The 48-bits of digital I/O are organized into six 8-bit ports. This data is routed between the CPU ports to the individual I/O pins exclusively by the logic programmed into the Xilinx XC5204 FPGA. The DIO card has the ability to cause an interrupt when a selected-polarity transition occurs on any of the bit values associated with ports0-2. Thus, use of ports0-2 for input lines is preferred. See the DIO manual pages 9-12 for potential interrupt features, which are disabled by the jumper configuration utilized.

The logic is programmed into the 160-pin XC5204 FPGA at power up from data stored in the 8-pin 17128EPI 128-kbit once programmable read-only memory contained in a socket on the DIO card. The PROM uses a serial configuration.

The DIO card contains a 1.8452-MHz crystal for the clock internal to the TL16C554. The maximum baud rate is 115 kbps and the crystal frequency is the usual 16 times the baud rate.

4.2.10 Test Point Information for the Diamond Systems Emerald-MM-DIO Card

The following table contains illustrative examples of test points determined from the schematics and the individual IC data sheets. More test points can be readily added for a more thorough joint inspection.

Table of Recommended Test Points for the DIO Card

Location IC-Pin	**Test Object**	**Expected Result**	**Signal Name**	**Comments**
P1-B1	Input Power	Ground		Input Power Connector
P1-B3	Input Power	+5VDC ±5%		Input Power Connector
U7-36	Crystal Freq.	1.8452 MHz	XTAL2	Crystal output
U7-38		Pulses	RXRDY*	Receive ready
U7-39		Pulses	TXRDY*	Transmit ready
J3 & J4 Pins	Com Ports	Pulses	several	Test each COM port while transmitting
J5 & J6 Pins	DIO Ports	Pulses	several	Test each DIO port while changing values
U6-	FPGA Freq.	Pulses	CCLK	XILINX clock

4.2.11 Technical Manual for the Diamond Systems Emerald-MM-DIO Card

The following technical manual was printed from the Diamond Systems Internet site at:

http://www.diamondsystems.com/files/emmdio.pdf.

EMERALD-MM-DIO

Quad RS-232 + 48 Digital I/O PC/104 Module

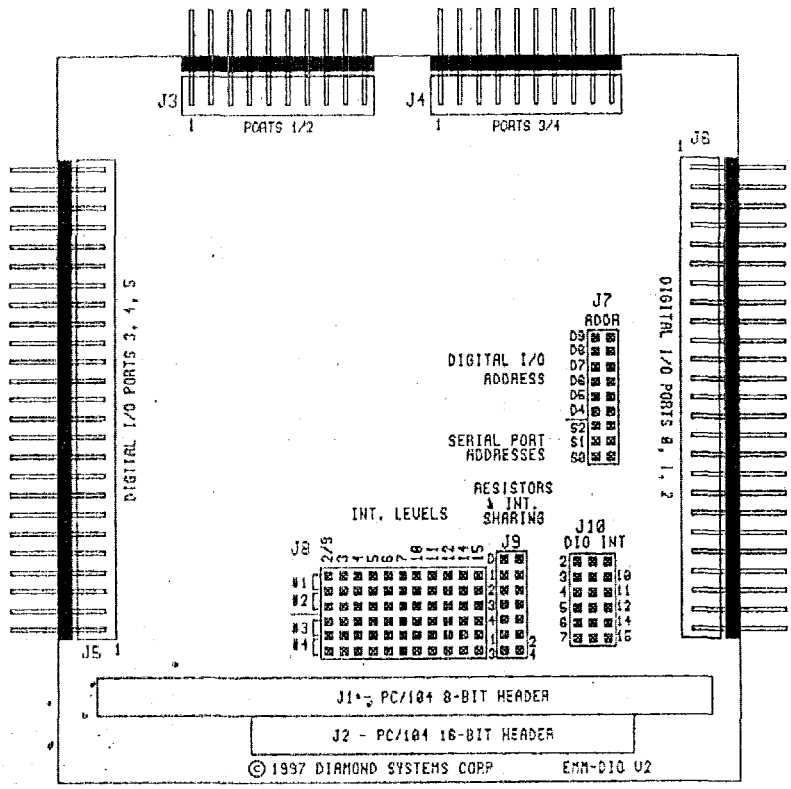
Board Revision V2

User Manual V1.0

Diamond Systems Corporation
450 San Antonio Rd.
Palo Alto, CA 94306
Tel (650) 813-1100
Fax (650) 813-1130
techinfo@diamondsys.com

TABLE OF CONTENTS

1. EMERALD-MM-DIO BOARD DRAWING



2. I/O HEADER PINOUTS

All I/O headers mate with .1" x .1" pitch dual-row ribbon cable connectors.

2.1 RS-232 Headers

J3

DCD 1	1	2	DSR 1
RXD 1	3	4	RTS 1
TXD 1	5	6	CTS 1
DTR 1	7	8	RI 1
GND	9	10	NC
DCD 2	11	12	DSR 2
RXD 2	13	14	RTS 2
TXD 2	15	16	CTS 2
DTR 2	17	18	RI 2
GND	19	20	NC

J4

DCD 3	1	2	DSR 3
RXD 3	3	4	RTS 3
TXD 3	5	6	CTS 3
DTR 3	7	8	RI 3
GND	9	10	NC
DCD 4	11	12	DSR 4
RXD 4	13	14	RTS 4
TXD 4	15	16	CTS 4
DTR 4	17	18	RI 4
GND	19	20	NC

Signal Definitions:

Signal Name	Definition	Direction with respect to board
DCD	Data Carrier Detect	Input
DSR	Data Set Ready	Input
RXD	Receive Data	Input
RTS	Request To Send	Output
TXD	Transmit Data	Output
CTS	Clear To Send	Input
DTR	Data Terminal Ready	Output
RI	Ring Indicator	Input
GND	Ground	--

2.2 Digital I/O Headers

J5

Digital IO Ports 5, 4, 3

Port 5 Bit 7	1	2	GND
Port 5 Bit 6	3	4	GND
Port 5 Bit 5	5	6	GND
Port 5 Bit 4	7	8	GND
Port 5 Bit 3	9	10	GND
Port 5 Bit 2	11	12	GND
Port 5 Bit 1	13	14	GND
Port 5 Bit 0	15	16	GND
Port 4 Bit 7	17	18	GND
Port 4 Bit 6	19	20	GND
Port 4 Bit 5	32	22	GND
Port 4 Bit 4	23	24	GND
Port 4 Bit 3	25	26	GND
Port 4 Bit 2	27	28	GND
Port 4 Bit 1	29	30	GND
Port 4 Bit 0	31	32	GND
Port 3 Bit 7	33	34	GND
Port 3 Bit 6	35	36	GND
Port 3 Bit 5	37	38	GND
Port 3 Bit 4	39	40	GND
Port 3 Bit 3	41	42	GND
Port 3 Bit 2	43	44	GND
Port 3 Bit 1	45	46	GND
Port 3 Bit 0	47	48	GND
+5V	49	50	GND

J6

Digital I/O Ports 2, 1, 0

Port 2 Bit 7	1	2	GND
Port 2 Bit 6	3	4	GND
Port 2 Bit 5	5	6	GND
Port 2 Bit 4	7	8	GND
Port 2 Bit 3	9	10	GND
Port 2 Bit 2	11	12	GND
Port 2 Bit 1	13	14	GND
Port 2 Bit 0	15	16	GND
Port 1 Bit 7	17	18	GND
Port 1 Bit 6	19	20	GND
Port 1 Bit 5	32	22	GND
Port 1 Bit 4	23	24	GND
Port 1 Bit 3	25	26	GND
Port 1 Bit 2	27	28	GND
Port 1 Bit 1	29	30	GND
Port 1 Bit 0	31	32	GND
Port 0 Bit 7	33	34	GND
Port 0 Bit 6	35	36	GND
Port 0 Bit 5	37	38	GND
Port 0 Bit 4	39	40	GND
Port 0 Bit 3	41	42	GND
Port 0 Bit 2	43	44	GND
Port 0 Bit 1	45	46	GND
Port 0 Bit 0	47	48	GND
+5V	49	50	GND

Signal Definitions:

Signal Name	Definition	Direction with respect to board
Port n Bit m	Digital I/O bit m on port n	Bidirectional Ports 0 through 2 have interrupt on state change capability
+5V	+5V power from PC/104 bus	---
GND	Ground connection on PC/104 bus	---

Note that pin 1 on J6 is at the top of the board (at the opposite end from the PC/104 bus connectors), and pin 1 on J5 is at the bottom of the board (next to the PC/104 bus connectors).

3. BOARD CONFIGURATION

Refer to the board drawing on page 3 for locations of the configuration items mentioned here.

3.1 Address Selection

EMERALD-MM-DIO occupies 6 distinct blocks in I/O memory: One for the digital I/O, four for the serial ports, and one for the serial port interrupt status register.

Digital I/O Base Address

The digital I/O base address is set with jumper block J7, located on the right side of the board next to I/O header J6. The locations labeled D9 through D4 are used for the digital I/O address. These lines correspond to address lines 9 through 4. The lowest 4 address bits, numbered 3 - 0, are assumed to be 0 for the base address. The following table lists some example base address settings; many others are possible. Note that when a jumper is installed, the corresponding address line must be 0, and when a jumper is removed, the corresponding address line must be 1. Addresses below 100 Hex are reserved for the CPU, so only combinations above 100 Hex are shown here.

Table 3.1: Jumper Block J7 - Digital I/O Base Address Configuration Examples

D9	D8	D7	D6	D5	D4	Base Address
In	Out	In	In	In	In	100
In	Out	In	In	Out	In	120
In	Out	In	Out	In	In	140
In	Out	Out	In	In	In	180
Out	In	In	In	In	In	200
Out	In	In	Out	In	In	240
Out	In	Out	In	In	In	280
Out	In	Out	Out	In	In	2C0
Out	Out	In	In	In	In	300
Out	Out	In	Out	In	In	340
Out	Out	Out	In	In	In	380
Out	Out	Out	Out	In	In	3C0

Serial Port and Interrupt Register Address Selection

These addresses are also set with jumper block J7, located at the right side of the board by I/O header J6. Eight different I/O address combinations are selectable, using the three locations marked S2, S1, S0. The address shown below for each port is the base address of that port, i.e. the lowest address of the port's I/O address block.

Table 3.2: Jumper Block J7 - Serial Port and Interrupt Status Register Addresses

No.	S2	S1	S0	Port 1	Port 2	Port 3	Port 4	Interrupt Status
0	In	In	In	3F8	2F8	3E8	2E8	220
1	In	In	Out	3E8	2E8	3A8	2A8	220
2	In	Out	In	380	388	288	230	224
3	In	Out	Out	240	248	260	268	224
4	Out	In	In	100	108	110	118	240
5	Out	In	Out	120	128	130	138	244
6	Out	Out	In	140	148	150	158	248
7	Out	Out	Out	160	168	170	178	24C

3.2 Serial Port Interrupt Level Selection

The serial port interrupt levels are selected with jumper block J8 at the bottom of the board. The interrupt level for each port is selected by installing a jumper across two rows underneath the desired level number as follows:

Port 1 Rows 1 and 2
Port 2 Rows 2 and 3
Port 3 Rows 4 and 5
Port 4 Rows 5 and 6

Note that ports 1 and 2 share row 2, and ports 3 and 4 share row 5. As long as ports 1 and 2 do not share the same interrupt level, there is no conflict, and the same applies for ports 3 and 4.

If sharing is to be enabled for ports 1 and 2, then install a jumper under the desired interrupt level for either port 1 or port 2, and then install another jumper across the pins marked 1 and 2 in jumper block J9, located to the right of J8. These pins are on the row second from the bottom.

Similarly, if sharing is to be enabled for ports 3 and 4, then install a jumper under the desired interrupt level for either port 3 or port 4, and then install another jumper across the pins marked 3 and 4 in jumper block J9, located to the right of J8. These pins are on the bottom row of J9.

3.3 Digital Interrupt Level Selection

The digital input interrupt level is selected by installing a jumper next to the desired level number on jumper block J10. The jumper is installed so that one of the pins it fits over is in the middle column, and the second pin is on one of the outer columns, depending on the level selected. For example, to select level 3, install a jumper next to the number 3 so that it fits over the left and middle columns. To select level 14, install a jumper next to the number 14 so that it fits over the middle and right columns.

3.4 Interrupt Pulldown Resistor Enabling

On the PC/104 bus, interrupt lines that are to be shared require a 1K Ω pulldown resistor to pull the line low when no device is driving it active to request interrupt service. Jumper block J9 isused to enable the pulldown resistors. Install a jumper across both pins next to the device for which you want to enable the resistor: 1 - 4 selects the same-numbered serial port, and D selects the digital interrupt. To disable the pulldown resistor, pull out the jumper or install it over only one pin.

Note that if you have interrupt sharing enabled, only one jumper should be installed for all ports sharing the same level. Otherwise you will have too low an impedance on the line, and the board will not be able to drive the signal to a logic 1 level to request interrupt service.

For example, if you are sharing interrupts on ports 1 and 2, but ports 3 and 4 are using different interrupt levels, then install jumpers in J9 next to positions 1, 3, and 4 or positions 2, 3, and 4. Do not install a jumper in both the 1 and 2 positions.

If the digital I/O is sharing an interrupt level with one of the serial ports, then the same applies: do not install a jumper in both the D position and the serial port number position.

3.5 Default Settings

The default settings for EMERALD-MM-DIO are as follows:

I/O address	200 Hex, see Table 3.1 above
Serial ports	Selection 5 in Table 3.2 above
Serial port interrupts	All 4 ports sharing level 3
Interrupt pulldown resistors	Enabled on serial interrupts and digital interrupt; remaining serial port jumpers are present over only one pin

4. DIGITAL I/O OPERATION

The digital I/O circuitry occupies 16 bytes, with the base address selected with jumper block J7 as described on page 6. Not all addresses are used; however all are reserved by the board.

All internal registers reset to 0 asynchronously upon power up or system reset.

4.1 Digital I/O Address Map

Base +	Write Function	Read Function
0	Port 0 DIO	Port 0 DIO
1	Port 1 DIO	Port 1 DIO
2	Port 2 DIO	Port 2 DIO
3	Port 3 DIO	Port 3 DIO
4	Port 4 DIO	Port 4 DIO
5	Port 5 DIO	Port 5 DIO
6	N/A	Digital I/O interrupt status register
7	Page / Lock register (3 pages)	N/A
8	Maps to register 0 of current page	
9	Maps to register 1 of current page	
10	Maps to register 2 of current page	
11	N/A	N/A
12	N/A	N/A
13	N/A	N/A
14	N/A	N/A
15	N/A	N/A

4.2 Page Registers

The Page/Lock register at address base + 7 contains two bits which select the current page accessed by locations 8 - 10. Four pages are addressable with these two bits. Page 0 registers are not used; only pages 1 through 3 are defined. Each page contains three registers as described below. These registers are used to control the behavior of the edge detection circuitry which exists for the 24 I/O bits in DIO registers 0 through 2.

Page 1 registers:

Offset	Write Function	Read Function
0	Pol_0	Pol_0
1	Pol_1	Pol_1
2	Pol_2	Pol_2

These three 8 bit read/write registers are used to indicate the polarity of edges detected by the edge detection circuit. A 0 in a bit position enables falling edge detection on the corresponding I/O bit, and a 1 enables rising edge detection. Pol_0 corresponds to DIO register 0, Pol_1 to DIO register 1, and Pol_2 to DIO register 2.

Page 2 registers:

Offset	Write Function	Read Function
0	Enab_0	Enab_0
1	Enab_1	Enab_1
2	Enab_2	Enab_2

These three 8-bit read/write registers are used to individually enable or disable edge detection on each bit in the corresponding I/O registers. A 0 in a bit position disables edge detection on the corresponding bit, and a 1 enables edge detection for that bit. Enab_0 corresponds to DIO register 0, Enab_1 to DIO register 1, and Enab_2 to DIO register 2.

Page 3 registers:

Offset	Write Function	Read Function
0	Clear edge detect bits for DIO 0	Int_ID_0
1	Clear edge detect bits for DIO 1	Int_ID_1
2	Clear edge detect bits for DIO 2	Int_ID_2

Reading from these three locations returns the status of the edge detection circuit for each bit. A 1 in a bit position indicates that that bit experienced the edge indicated by the corresponding bit in the corresponding Pol register AND that bit is enabled with a 1 in the corresponding bit in the corresponding Enab register.

Writing to any of these addresses clears all the edge detect bits for the corresponding group of 8 I/O lines. Register 0 corresponds to DIO register 0, etc.

5. DIGITAL I/O REGISTER DESCRIPTIONS

These are the on-board registers decoded with DIOSW[9-4].

Base + 0 to 5 Read/Write Digital I/O Data

Bit No.	7	6	5	4	3	2	1	0
Name	D[n][7]	D[n][6]	D[n][5]	D[n][4]	D[n][3]	D[n][2]	D[n][1]	D[n][0]

Definitions:

D[n][7-0] Digital I/O data; n is the port number 0 - 5, corresponding to addresses base + 0 through base + 5.

When reading from a port, the value read is the opposite polarity from the state of the pin.

When a 0 is written to the port, the port's output pin is tristated. When a 1 is written to the port, the port's output pin is driven low.

Base + 6 Read Interrupt Status Register

Bit No.	7	6	5	4	3	2	1	0
Name	X	X	X	X	X	INT2	INT1	INT0

Definitions:

X Not used

INT2-0 Interrupt pending on port 2-0:

0 = no interrupt pending, 1 = interrupt pending

Base + 7 Write Page / Lock Register

Bit No.	7	6	5	4	3	2	1	0
Name	P1	P0	Lock5	Lock4	Lock3	Lock2	Lock1	Lock0

Definitions:

P1-0 Page number. Defines which page of registers is accessed at base + 8 through base + 10.

P1	P0	Page
0	0	Not used
0	1	Polarity registers
1	0	Enable registers
1	1	Interrupt ID registers / Clear Interrupts

Lock5-0 Controls write access to ports 5-0. A 1 in a bit position prohibits data being written to the associated port. A 0 enables writing data to that port.

Page Registers:

Page 1:

Base + 8 to 10 Read/Write Edge Polarity POL[0-2]

Bit No.	7	6	5	4	3	2	1	0
Name	Pol[n][7]	Pol[n][6]	Pol[n][5]	Pol[n][4]	Pol[n][3]	Pol[n][2]	Pol[n][1]	Pol[n][0]

Definitions:

Pol[n][m] Polarity for edge detection circuit on port n, bit m. n = 0 to 2, m = 0 to 7. If the bit is a 0, negative edges are detected by the edge detection circuit, and if the bit is a 1, positive edges are detected. Edges are only detected if the associated Enable bit is a 1 (see page 2 registers below).

Page 2:

Base + 8 to 10 Read/Write Edge Detect Enable ENAB[0-2]

Bit No.	7	6	5	4	3	2	1	0
Name	Enab[n][7]	Enab[n][6]	Enab[n][5]	Enab[n][4]	Enab[n][3]	Enab[n][2]	Enab[n][1]	Enab[n][0]

Definitions:

Enab[n][m] Enables edge detection on port n, bit m. n = 0 to 2, m = 0 to 7. If the bit is a 0, edge detection is disabled for that input line, and if the bit is a 1, edge detection is enabled. The polarity of the edge that is detected is controlled with the polarity registers in page 0 (see page 1 registers above).

Page 3:

Base + 8 to 10 Write Edge Detection Clear CLEAR[0-2]

Bit No.	7	6	5	4	3	2	1	0
Name	X	X	X	X	X	X	X	X

Definitions:

X Not used

Writing to the page 3 register at base + 8 clears all edge detect circuits for port 0, writing to base + 9 clears port 1, and writing to base + 10 clears port 2. The value written does not matter.

Base + 8 to 10 Read Edge Detection Status INT_ID[0-2]

Bit No.	7	6	5	4	3	2	1	0
Name	ID[n][7]	ID[n][6]	ID[n][5]	ID[n][4]	ID[n][3]	ID[n][2]	ID[n][1]	ID[n][0]

Definitions:

ID[n][7-0] Edge detection status for port n, bit 7-0. A 0 indicates that the programmed e dge was detected since that last clear, and a 0 indicates that no edge was detected.

4.3 Texas Instruments, TL16C554 – Quad UART IC

The TL16C554 is the primary component of the Diamond Systems Emerald-MM-DIO card and determines most of the I/O operations. The specifications and data sheet for this IC is included for extra clarity.

4.3.1 Specifications for the TL16C554

The following specification sheet was printed from the Texas Instruments Internet site at:

http://www.ti.com/sc/docs/products/analog/tl16c554.html.

TL16C554, QUAD UART WITH 16-BYTE FIFOS

Device Status: Active

Parameter Name	TL16C554
Number of Channels	Quad
FIFOs	16-Byte
Operating Voltage (V)	5

Description

The TL16C554 and the TL16C554I are enhanced quadruple versions of the TL16C550B asynchronous communications element (ACE). Each channel performs serial-to-parallel conversion on data characters received from peripheral devices or modems and parallel-to-serial conversion on data characters transmitted by the CPU. The complete status of each channel of the quadruple ACE can be read at any time during functional operation by the CPU. The information obtained includes the type and condition of the operation performed and any error conditions encountered.

The TL16C554 and the TL16C554I quadruple ACE can be placed in an alternate FIFO mode, which activates the internal FIFOs to allow 16 bytes (plus three bits of error data per byte in the receiver FIFO) to be stored in both receive and transmit modes. To minimize system overhead and maximize system efficiency, all logic is on the chip. Two terminal functions allow signaling of direct memory access (DMA) transfers. Each ACE includes a programmable baud rate generator that can divide the timing reference clock input by a divisor between 1 and $(2^{16}-1)$.

The TL16C554 and the TL16C554I are available in a 68-pin plastic-leaded chip-carrier (PLCC) FN package and in an 80-pin (TQFP) PN package.

Features

- Integrated Asynchronous Communications Element
- Consists of Four Improved TL16C550 ACEs Plus Steering Logic
- In FIFO Mode, Each ACE Transmitter and Receiver Is Buffered With 16-Byte FIFO to Reduce the Number of Interrupts to CPU
- In TL16C450 Mode, Hold and Shift Registers Eliminate Need for Precise Synchronization Between the CPU and Serial Data
- Up to 16-MHz Clock Rate for up to 1-Mbaud Operation
- Programmable Baud Rate Generators Which Allow Division of Any Input Reference Clock by 1 to $(2^{16} - 1)$ and Generate an Internal 16 × Clock
- Adds or Deletes Standard Asynchronous Communication Bits (Start, Stop, and Parity) to or From the Serial Data Stream
- Independently Controlled Transmit, Receive, Line Status, and Data Set Interrupts
- Fully Programmable Serial Interface Characteristics:
  - 5-, 6-, 7-, or 8-Bit Characters
  - Even-, Odd-, or No-Parity Bit
  - 1-, 11/2-, or 2-Stop Bit Generation
  - Baud Generation (DC to 1-Mbit Per Second)
- False Start Bit Detection
- Complete Status Reporting Capabilities
- Line Break Generation and Detection
- Internal Diagnostic Capabilities:
  - Loopback Controls for Communications Link Fault Isolation
  - Break, Parity, Overrun, Framing Error Simulation
- Fully Prioritized Interrupt System Controls
- Modem Control Functions (CTS\, RTS\, DSR\, DTR\, RI\, and DCD\)
- 3-State Outputs Provide TTL Drive Capabilities for Bidirectional Data Bus and Control Bus

To view the following documents, Acrobat Reader 3.x is required.
To download a document to your hard drive, right-click on the link and choose 'Save'.

Datasheets

Full datasheet in Acrobat PDF: sllS165d.pdf (478 KB)
Full datasheet in Zipped PostScript: sllS165d.psz (421 KB)

Pricing/Samples/Availability

Orderable Device	Package	Pins	Temp	Status	Price/unit USD (100-999)	Pack Qty	Availability / Samples
TL16C554IFN	FN	68		ACTIVE	8.40	18	Check stock or order
TL16C554FNR	FN	68		ACTIVE	7.28	1	Check stock or order
TL16C554IFN	FN	68		ACTIVE	8.72	18	Check stock or order
TL16C554IFNR	FN	68		ACTIVE	7.56	250	Check stock or order
TL16C554IPN	PN	80		ACTIVE	9.05	119	Check stock or order
TL16C554PN	PN	80		ACTIVE	8.72	119	Check stock or order

Application Reports

- 422 AND 485 OVERVIEW AND SYSTEM CONFIGURATIONS (SLLA070)
- ANALOG APPLICATIONS JOURNAL (SLYT010)
- ANALOG APPLICATIONS JOURNAL FEBRUARY 2000 (SLYT012, 781 KB)
- ELECTROSTATIC DISCHARGE APPLICATION NOTE (SSYA008)
- THERMAL CHARACTERISTICS OF LINEAR AND LOGIC PACKAGES USING JEDEC PCB DESIGNS (SZZA017A)

Table Data Updated on: 2/23/2000

4.3.2 Data Sheet for the TL16C554

The following data sheet was printed from the Texas Instruments Internet site at:

http://www-s.ti.com/sc/psheets/slls165d/slls165d.pdf#xml=http://www-search.ti.com/search97cgi/s97r_cgi?action=View&VdkVgwKey=http%3A%2F%2Fwww%2Ds%2Eti%2Ecom%2Fsc%2Fpsheets%2Fslls165d%2Fslls165d%2Epdf&doctype=xml&Collection=TechDocs&QueryZip=tl16c554&.

- Integrated Asynchronous Communications Element
- Consists of Four Improved TL16C550 ACEs Plus Steering Logic
- In FIFO Mode, Each ACE Transmitter and Receiver Is Buffered With 16-Byte FIFO to Reduce the Number of Interrupts to CPU
- In TL16C450 Mode, Hold and Shift Registers Eliminate Need for Precise Synchronization Between the CPU and Serial Data
- Up to 16-MHz Clock Rate for up to 1-Mbaud Operation
- Programmable Baud Rate Generators Which Allow Division of Any Input Reference Clock by 1 to ($2^{16}-1$) and Generate an Internal 16 × Clock
- Adds or Deletes Standard Asynchronous Communication Bits (Start, Stop, and Parity) to or From the Serial Data Stream
- Independently Controlled Transmit, Receive, Line Status, and Data Set Interrupts
- Fully Programmable Serial Interface Characteristics:
  - 5-, 6-, 7-, or 8-Bit Characters
  - Even-, Odd-, or No-Parity Bit
  - 1-, 1 1/2-, or 2-Stop Bit Generation
  - Baud Generation (DC to 1-Mbit Per Second)
- False Start Bit Detection
- Complete Status Reporting Capabilities
- Line Break Generation and Detection
- Internal Diagnostic Capabilities:
  - Loopback Controls for Communications Link Fault Isolation
  - Break, Parity, Overrun, Framing Error Simulation
- Fully Prioritized Interrupt System Controls
- Modem Control Functions ($\overline{CTS}$, $\overline{RTS}$, $\overline{DSR}$, $\overline{DTR}$, $\overline{RI}$, and $\overline{DCD}$)
- 3-State Outputs Provide TTL Drive Capabilities for Bidirectional Data Bus and Control Bus

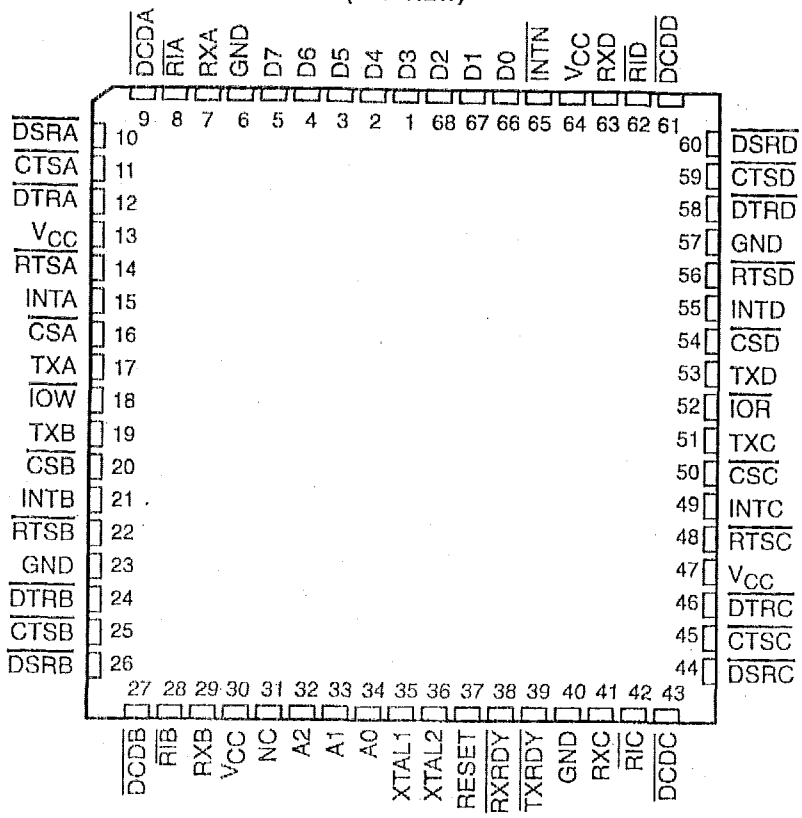
description

The TL16C554 and the TL16C554I are enhanced quadruple versions of the TL16C550B asynchronous communications element (ACE). Each channel performs serial-to-parallel conversion on data characters received from peripheral devices or modems and parallel-to-serial conversion on data characters transmitted by the CPU. The complete status of each channel of the quadruple ACE can be read at any time during functional operation by the CPU. The information obtained includes the type and condition of the operation performed and any error conditions encountered.

The TL16C554 and the TL16C554I quadruple ACE can be placed in an alternate FIFO mode, which activates the internal FIFOs to allow 16 bytes (plus three bits of error data per byte in the receiver FIFO) to be stored in both receive and transmit modes. To minimize system overhead and maximize system efficiency, all logic is on the chip. Two terminal functions allow signaling of direct memory access (DMA) transfers. Each ACE includes a programmable baud rate generator that can divide the timing reference clock input by a divisor between 1 and ($2^{16}-1$).

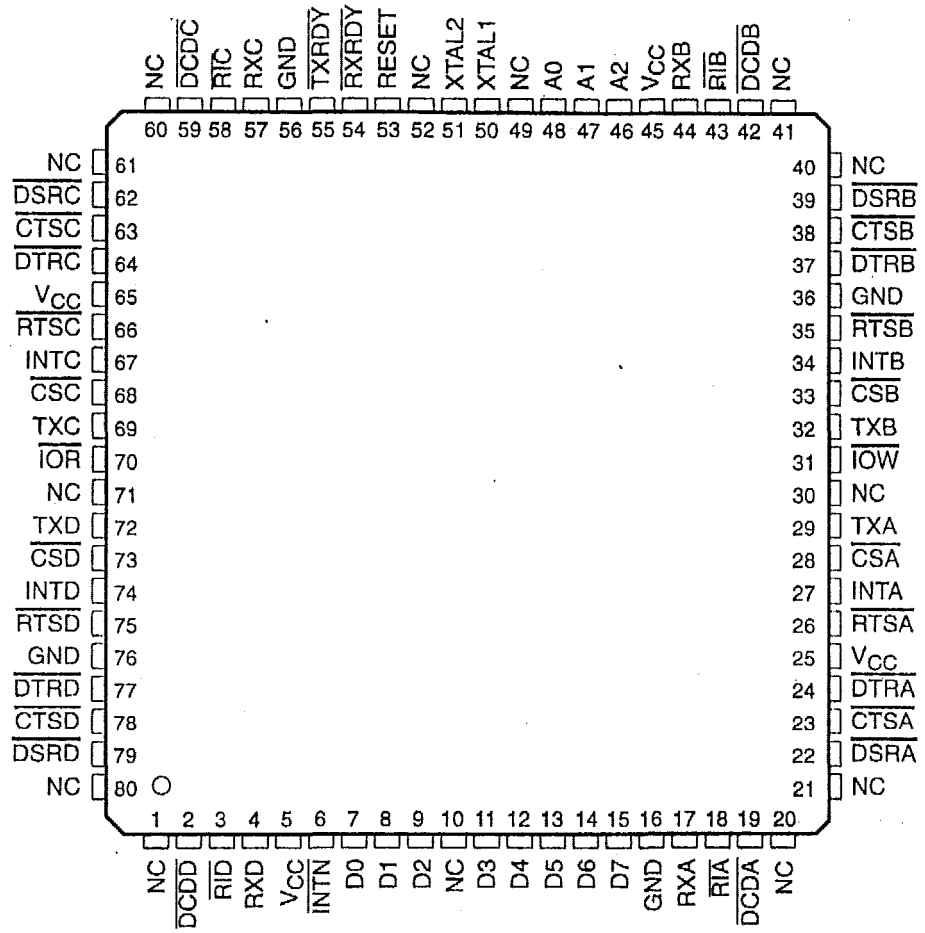
The TL16C554 and the TL16C554I are available in a 68-pin plastic-leaded chip-carrier (PLCC) FN package and in an 80-pin (TQFP) PN package.

FN PACKAGE
(TOP VIEW)



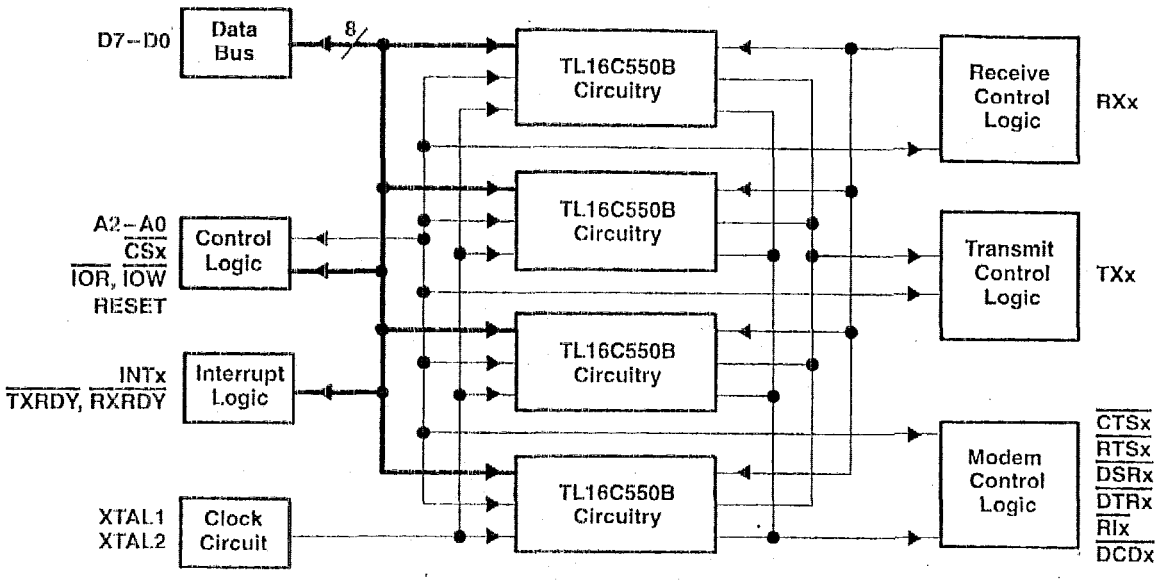
NC – No internal connection

PN PACKAGE
(TOP VIEW)



NC – No internal connection

functional block diagram†



† For TL16C550 circuitry, refer to the TL16C550B data sheet.

Terminal Functions

TERMINAL NAME	FN NO.	PN NO.	I/O	DESCRIPTION
A0 A1 A2	34 33 32	48 47 46	I	Register select terminals. A0, A1, and A2 are three inputs used during read and write operations to select the ACE register to read or write.
$\overline{CSA}$, $\overline{CSB}$, $\overline{CSC}$, $\overline{CSD}$	16, 20, 50, 54	28, 33, 68, 73	I	Chip select. Each chip select ($\overline{CSx}$) enables read and write operations to its respective channel.
$\overline{CTSA}$, $\overline{CTSB}$, $\overline{CTSC}$, $\overline{CTSD}$	11, 25, 45, 59	23, 38, 63, 78	I	Clear to send. $\overline{CTSx}$ is a modem status signal. Its condition can be checked by reading bit 4 (CTS) of the modem status register. $\overline{CTS}$ has no affect on the transmit or receive operation.
D7–D0	66–68 1–5	15–11, 9–7	I/O	Data bus. Eight data lines with 3-state outputs provide a bidirectional path for data, control, and status information between the TL16C554 and the CPU. D0 is the least significant bit (LSB).
$\overline{DCDA}$, $\overline{DCDB}$, $\overline{DCDC}$, $\overline{DCDD}$	9, 27, 43, 61	19,42, 59, 2	I	Data carrier detect. A low on $\overline{DCDx}$ indicates the carrier has been detected by the modem. The condition of this signal is checked by reading bit 7 of the modem status register.
$\overline{DSRA}$, $\overline{DSRB}$, $\overline{DSRC}$, $\overline{DSRD}$	10, 26, 44, 60	22, 39, 62, 79	I	Data set ready. $\overline{DSRx}$ is a modem status signal. Its condition can be checked by reading bit 5 (DSR) of the modem status register. $\overline{DSR}$ has no affect on the transmit or receive operation.
$\overline{DTRA}$, $\overline{DTRB}$, $\overline{DTRC}$, $\overline{DTRD}$	12, 24, 46, 58	24, 37, 64, 77	O	Data terminal ready. $\overline{DTRx}$ is an output that indicates to a modem or data set that the ACE is ready to establish communications. It is placed in the active state by setting the DTR bit of the modem control register. $\overline{DTRx}$ is placed in the inactive state (high) either as a result of the master reset during loop mode operation or clearing bit 0 ($\overline{DTR}$) of the modem control register.
GND	6, 23, 40, 57	16, 36, 56, 76		Signal and power ground
$\overline{INTN}$	65	6	I	Interrupt normal. $\overline{INTN}$ operates in conjunction with bit 3 of the modem status register and affects operation of the interrupts (INTA, INTB, INTC, and INTD) for the four universal asynchronous receiver/transceivers (UARTs) per the following table.

$\overline{INTN}$	OPERATION OF INTERRUPTS
Brought low or allowed to float	Interrupts are enabled according to the state of OUT2 (MCR bit 3). When the MCR bit 3 is cleared, the 3-state interrupt output of that UART is in the high-impedance state. When the MCR bit 3 is set, the interrupt output of the UART is enabled.
Brought high	Interrupts are always enabled, overriding the OUT2 enables.

Terminal Functions (Continued)

TERMINAL NAME	TERMINAL FN NO.	TERMINAL PN NO.	I/O	DESCRIPTION
INTA, INTB, INTC, INTD	15, 21, 49, 55	27, 34, 67, 74	O	External interrupt output. The INTx outputs go high (when enabled by the interrupt register) and inform the CPU that the ACE has an interrupt to be serviced. Four conditions that cause an interrupt to be issued are: a receiver error, receiver data available or timeout (FIFO mode only), transmitter holding register empty, and an enabled modem status interrupt. The interrupt is disabled when it is serviced or as the result of a master reset.
$\overline{IOR}$	52	70	I	Read strobe. A low level on $\overline{IOR}$ transfers the contents of the TL16C554 data bus to the external CPU bus.
$\overline{IOW}$	18	31	I	Write strobe. $\overline{IOW}$ allows the CPU to write into the selected address by the address register.
RESET	37	53	I	Master reset. When active, RESET clears most ACE registers and sets the state of various signals. The transmitter output and the receiver input is disabled during reset time.
$\overline{RIA}$, $\overline{RIB}$, $\overline{RIC}$, $\overline{RID}$	8, 28, 42, 62	18, 43, 58, 3	I	Ring detect indicator. A low on $\overline{RIx}$ indicates the modem has received a ring signal from the telephone line. The condition of this signal can be checked by reading bit 6 of the modem status register.
$\overline{RTSA}$, $\overline{RTSB}$, $\overline{RTSC}$, $\overline{RTSD}$	14, 22, 48, 56	26, 35, 66, 75	O	Request to send. When active, $\overline{RTSx}$ informs the modem or data set that the ACE is ready to receive data. Writing a 1 in the modem control register sets this bit to a low state. After reset, this terminal is set high. These terminals have no affect on the transmit or receive operation.
RXA, RXB RXC, RXD	7, 29, 41, 63	17, 44, 57, 4	I	Serial input. RXx is a serial data input from a connected communications device. During loopback mode, the RXx input is disabled from external connection and connected to the TXx output internally.
$\overline{RXRDY}$	38	54	O	Receive ready. $\overline{RXRDY}$ goes low when the receive FIFO is full. It can be used as a single transfer or multitransfer.
TXA, TXB TXC, TXD	17, 19, 51, 53	29, 32, 69, 72	O	Transmit outputs. TXx is a composite serial data output that is connected to a communications device. TXA, TXB, TXC, and TXD are set to the marking (high) state as a result of reset.
$\overline{TXRDY}$	39	55	O	Transmit ready. $\overline{TXRDY}$ goes low when the transmit FIFO is full. It can be used as a single transfer or multitransfer function.
VCC	13, 30, 47, 64	5, 25, 45, 65		Power supply
XTAL1	35	50	I	Crystal input 1 or external clock input. A crystal can be connected to XTAL1 and XTAL2 to utilize the internal oscillator circuit. An external clock can be connected to drive the internal clock circuits.
XTAL2	36	51	O	Crystal output 2 or buffered clock output (see XTAL1).

absolute maximum ratings over free-air temperature range (unless otherwise noted)†

Supply voltage range, V_{CC} (see Note 1) ..... –0.5 V to 7 V
Input voltage range at any input, V_I ..... –0.5 V to 7 V
Output voltage range, V_O ..... –0.5 V to V_{CC} + 3 V
Continuous total power dissipation at (or below) 70°C ..... 500 mW
Operating free-air temperature range, T_A: TL16C554 ..... –0°C to 70°C
TL16C554I ..... –40°C to 85°C
Storage temperature range, T_{stg} ..... –65°C to 150°C

† Stresses beyond those listed under "absolute maximum ratings" may cause permanent damage to the device. These are stress ratings only, and functional operation of the device at these or any other conditions beyond those indicated under "recommended operating conditions" is not implied. Exposure to absolute-maximum-rated conditions for extended periods may affect device reliability.

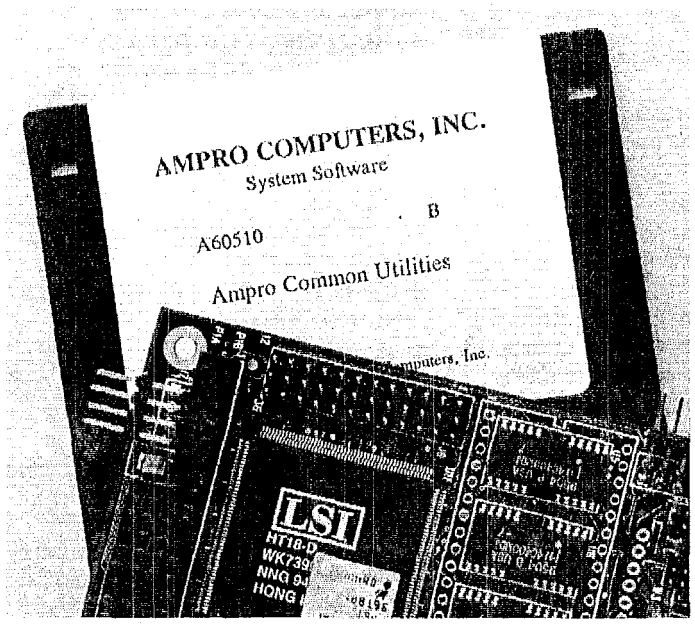
NOTE 1: All voltage levels are with respect to GND.

AMPRO
COMMON UTILITIES

Several software programs are provided by Ampro to access the CPU hardware.

Embedded-PC BIOS and Utilities

Enhanced PC-compatible BIOS and utilities for embedded systems



- **Proven: field tested in thousands of OEM applications for over seven years**
- **Compatible: based on industry standard Award BIOS core, plus extensively tested Ampro enhancements**
- **Flexible: extended functions and options support the requirements of embedded applications**
- **Reliable: enhanced to support the demands of robust, non-stop, unattended system operation**
- **Complete: includes utilities for extended function support, system development, and field maintenance**
- **Included with all Ampro CPU products**

Ampro's exclusive Embedded-PC BIOS is an enhanced, fully PC-compatible ROM-BIOS that enables reliable, unattended non-stop operation of Ampro Little Board™ and CoreModule™ CPUs in embedded system applications. The Embedded-PC BIOS, included with all Ampro CPU products, greatly increases system reliability, functionality, and flexibility, making it ideal for the broadest possible range of embedded system applications.

Proven Compatibility

To assure the highest degree of PC compatibility, Ampro purchased a source license from Award Software and has used the well-proven Award BIOS as the "kernel" of the Ampro Embedded-PC BIOS. To this fully compatible core, Ampro has added an extensive set of embedded-PC enhancements which have set the standard in the industry for reliability, functionality, flexibility, and support. In all cases, great care in implementation plus thorough internal qualification and field testing have been used to ensure that full compatibility with the standard PC application environment is preserved while offering the highest possible level of reliability and robustness.

Extended Flexibility and Functionality

Embedded systems demand great flexibility in configuring the CPU and system peripherals to support application-specific requirements. Ampro has included BIOS functions which offer a level of system flexibility not found in the normal desktop PC environment. Included are such features as SCSI and solid state disk support, serial console and serial program-download options, non-volatile parameter storage support, and OEM "hooks" that let you customize the system without modifying the BIOS.

Enhanced Reliability

Embedded systems must be able to run reliably in rugged, hostile, and mission-critical environments without operator intervention. A normal PC BIOS, developed for the desktop environment, cannot adequately support these requirements. Over a seven year period, Ampro has evolved a comprehensive set of BIOS enhancements and extensions that increase the robustness and reliability of embedded-PC based system operation. These enhancements include such features as watchdog timer support, fail-safe boot, battery-free boot, and solid state disk support which lets you replace conventional disk drives with rugged and reliable silicon memory devices.

Embedded-PC Utilities

Each Ampro CPU development kit or system includes a set of Embedded-PC Utilities which simplify use of the extended features in the Ampro Embedded-PC BIOS and which assist in system development, operation, and maintenance. Included are utilities for SCSI, watchdog timer control, and support for serially connected remote-hosted system access. A partial list of Ampro's Embedded-PC Utilities appears in the specifications section of this data sheet.

EMBEDDED-PC BIOS EXTENDED FEATURES

- Solid state disk (SSD) support:
  - —Reliability enhancement for rugged, hostile environments
  - —Substitutes EPROM, Flash, or nonvolatile RAM devices for normal floppy drives
  - —Usable as DOS boot device
  - —SSDs may coexist with standard floppy and/or hard disk drives or with PCMCIA devices
  - —NOVRAM and Flash SSDs may be programmed from a resident DOS utility or remotely via a serial access utility

    NOTE: Flash SSD is supported as a (re)programmable read-only DOS drive
- Extended floppy services:
  - —Supports standard PC/AT floppy formats on PC- and XT-compatible CPUs
- SCSI services:
  - —Supports DOS-based system boot and operation using a wide variety of SCSI drives
  - —Extensive configuration options and flexibility
  - —SCSI and IDE drives may coexist
  - —Supports "ASPI" drivers and utilities via a DOS device driver
  - —BIOS SCSI function provides API for application software access and non-standard device support
- Watchdog timer support:
  - —SETUP parameter defines startup WDT time constant, to ensure reliable system BOOT
  - —BIOS function provides API for application software access, to ensure system operation integrity
- Fast boot option:
  - —Offers normal or accelerated power-on self test options for increased system boot speed
  - —Provides microcontroller-like quick startup
- Fail-safe boot support:
  - —Continuously cycles through a defined list of boot devices until a boot device is available
  - —Accommodates slow boot or malfunctioning devices
  - —Supports unattended operation
- Battery-free boot support:
  - —CMOS SETUP data backup is stored in an EEPROM device
  - —Allows system boot and operation with a bad or non-present battery
  - —Supports environments in which batteries are not permitted
  - —Prevents total system failure due to bad batteries (only the real time-of-day function is impaired)
- Serial console option:
  - —Substitutes an external serially-connected device for conventional PC keyboard and video
  - —Saves embedded system cost and space
  - —Allows remote access by users or external computers via serial connection or modem
  - —Useful for diagnostics, system status monitoring, or remote system control
- Serial loader option:
  - —Allows external loading of executable code prior to system boot
  - —Increases embedded system flexibility
  - —Useful for diagnostics, remote system control, and SSD device programming
- EEPROM access function:
  - —BIOS function provides API for application use of 512-bit nonvolatile data area
  - —Useful for implementing features like serialization, copy protection, and passwords
- OEM customization hooks:
  - —Executes custom code prior to system boot via ROM extensions or code "patches"
  - —Allows system customization without BIOS modification
- Advanced Power Management (Refer to Ampro CPU datasheets for further details.)

EMBEDDED-PC UTILITIES

- Serial loader program: DOS utility for transferring a binary executable file from a remote "host" PC to an embedded CPU "target" Useful for a wide variety of purposes including loading of a number of remote debuggers.
- Serial SSD programmer: DOS utility for (re)programming the SSD devices on an embedded CPU "target" under control of a remote serially connected "host" PC
- Setup access utility: DOS program for entering into the in-BIOS SETUP function during system operation
- SCSI support: SCSI device format, initialization and maintenance utilities
- Watchdog timer utility: DOS program which can be used to "tickle" the CPU's WDT
- Terminal emulation utility: allows a standard PC to emulate an ASCII terminal; for use with the serial console function of an embedded Ampro CPU module
- Advanced Power Management device driver:
  - —DOS device driver supports power management on Ampro CPU modules

    NOTE: APM support is provided as part of the standard embedded-PC BIOS in Ampro's newer products including the Little Board/486*i*.

Common Utilities

Technical Manual

P/N: 5000931

Revision: B

Ampro Computers Incorporated
4757 Hellyer Ave ■ San Jose, CA 95138
Tel (408) 360-0200 ■ FAX (408) 360-0220
http://www.ampro.com

NOTICE

DISCLAIMER

Ampro Computers, Incorporated makes no representations or warranties with respect to the contents of this manual or of the associated Ampro software products, and specifically disclaims any implied warranties of merchantability or fitness for any particular purpose. Ampro shall under no circumstances be liable for incidental or consequential damages or related expenses resulting from the use of this product, even if it has been notified of the possibility of such damages. Ampro reserves the right to revise this publication from time to time without obligation to notify any person of such revisions. If errors are found, please contact Ampro at the address listed on the title page of this document.

TRADEMARKS

The Ampro logo is a registered trademark, and Ampro, Little Board, StackPlane, MiniBackplane, MiniModule, and CoreModule are trademarks of Ampro Computers, Inc. All other marks are the property of their respective companies.

TECHNICAL SUPPORT

Technical Support is available from 8:00 AM to 5:00 PM, Pacific time. Please have the product you wish to discuss at hand when calling. The number is 800-966-5200

Revision History

Revision	Reason for Change	Date
1	Preliminary Release	4/95
A	Production Release	7/95
B	Add FLASHWRITE	9/96

TABLE OF CONTENTS

AMPRO COMMON UTILITIES

1 INTRODUCTION

This document contains detailed information about the utility programs provided on the Ampro Common Utilities diskette. The Ampro Common Utilities diskette is included in each CPU product's Development Kit.

This manual explains what each program does and how it is used. Program descriptions are in alphabetical order, so this document can serve as a technical reference.

Each utility program is identified by a version and revision level. When the program is run, its version number (and a revision level) appear in a sign-on message, such as the following:

Ampro Hard Disk Format Utility
Copyright (C) 1986 Ampro Computers, Inc.
Version 2.1

In this case the program is Version 2, Revision 1. Versions of a program which have the same "version" number operate in the same manner. When a change is made to a program which necessitates a new description, its version number is changed, indicating that the old description is no longer accurate.

Note

The Common Utilities diskette may also contain files with names of the form *name.txt*. We recommend that you read these files when you find them, as they contain information on recent program revisions, enhancements, or additions.

2 DISKETTE ORGANIZATION

The Common Utilities diskette is arranged so that similar programs share common subdirectories. Table 1 lists the subdirectories, programs found in each directory, and a brief description of each program.

Directory	Utility Name	Function
\ (root)	SETUP.COM	CPU SETUP utility
AMPRO.UTL	SETSPEED.COM	Set CPU speed
	WATCHDOG.COM	Controls CPUs watchdog timer
	XTCLK.COM	Sets XT's DOS clock from hardware clock
	XTPEMM.COM	Expanded memory manager for CM/XT *Plus*
APM.UTL	APM.SYS	Advanced power management driver
OTHER.UTL	SERLOAD.COM	Downloads program to CPU via serial port
	TVTERM.COM	Televideo 900 series emulator
SCSI.UTL	SCSI-ID.COM	Returns system's SCSI initiator ID to console
	SCSI-VER.COM	Returns SCSI-BIOS version to console
	SCSI.REV	SCSI version history; text file
	SCSICOMP.COM	SCSI block compare utility
	SCSICOPY.COM	SCSI block copy utility
	SCSIFMT.COM	SCSI hard disk format utility
	SCSIPARK.COM	SCSI hard disk head parking utility
	SCSITOOL.COM	SCSI common command set debugging tool
SERPROG.UTL	SERPROG.EXE	Programs byte-wide devices from serial port
	ADDFLASH.COM	Updates SERPROG.EXE for new Flash EPROM devices
	SOCKETRW.BIN	Companion program to SERPROG.EXE

Table 1 Common Utilities Summary

3 ADDFLASH

The Ampro ADDFLASH.COM utility works in conjunction with SERPROG.EXE and FLASHWRITE. SERPROG is an intelligent Flash EPROM programming utility that gets its programming data from a serial port. SERPROG is intelligent in that it interrogates the target Flash device to determine its size, type, and so forth. As new Flash memory devices become available, SERPROG must be updated to be able to recognize them. This is the purpose of ADDFLASH. ADDFLASH is a utility that adds new Flash EPROM data to the SERPROG program so that it will recognize new Flash EPROM devices.

To start the program, enter the following on the DOS command line:

```
C:>ADDFLASH
```

ADDFLASH will take you through a series of questions:

```
What is the 4-digit ID of the device? >>
```

Each JEDEC byte-wide Flash memory device has a 4-digit manufacturers code. Enter the code at the prompt. (You do not need to press <Enter>.) Look on the manufactures data sheet to find out the ID code.

```
What is the device capacity in kilobytes?
Please enter: 1 - 32K; 2 - 64K; 3 - 128K; 4 - 256K; 5- 512K >>
```

Enter a single digit, between 1 and 5 (inclusive), representing the size of the device.

```
What is the device type?
Please enter: 1 - +12v; 2 - +5v >>
```

The *device type* represents the programming voltage needed to program the new device. Enter 1 or 2.

```
About to add <size> <type> device, <ID> to external support table
FLASH.ID.
Type "C" to confirm or any other key to abort >>
```

The program shows you what you entered and asks that you confirm your entries before proceeding. If you type "C" (case insensitive), the program will display the following message:

```
Success! FLASHWRITE.EXE will now support the new device.
```

This is slightly misleading. FLASHWRITE.EXE and SERPROG will now be able to access the new device module that was incorporated into the ADDFLASH program. If the message is displayed, ADDFLASH has updated the file called FLASH.ID, which must reside in the current directory, or created a new file called FLASH.ID if it did not already exist.

4 FLASHWRITE

This utility programs the Flash EPROMs in the target system with image files generated by EPROMGEN. Syntax terminology is explained in the introduction of this chapter.

SSD/DOS Flash Device Programming Utility Ver 5.31

Usage: FLASHWRITE *Socket FileName*

Where:

Socket:	Choose One: S0, S1, or S2 for Little Board or CoreModule products Xna for the SSD board at 274h Yna for the SSD board at 275h n = board number a = socket number
Filename	Name of binary file to program into the device

For a list of supported devices, enter FLASHWRITE/T.

Here is an example using FLASHWRITE to program Flash EPROMs on the target system.

Example 1 - Writing to an Expansion Module

This Example shows how simple it is to write a sample file OUTPUT.X1A to a Flash EPROM to create an SSD. This example shows writing to a MiniModule/SSD expansion module designated as Board Number 1, socket A, at Control Port Address 274h.

```
FLASHWRITE  X1A  OUTPUT.X1A
```

5 APMDRIVER.SYS

APMDRIVER.SYS is an Advanced Power Management (APM) driver compatible with the Intel/Microsoft APM Specification 1.0.

This driver does not execute any automatic power saving. The default APM state is disabled. POWER.EXE supplied with MS-DOS may be used with this APM driver. Or, you can incorporate power management into your application programs directly, using the calls documented below.

APMDRIVER.SYS is loaded via the DOS CONFIG.SYS file. To install APMDRIVER.SYS in your CONFIG.SYS file, add the following line to the file:

```
device = APMDRVR.SYS [option]
```

There is a single option that can be added to the command line:

```
option = /NOCLOCKCHANGE
```

Maximum power savings are gained when the CPU clock speed is reduced. However interrupt routines used to exit from low power states (controlled by APM) will execute at the lower clock speed. For maximum performance, but at some sacrifice of power, you can add the NOCLOCKCHANGE option to the command line.

The interface with APM is via interrupt 15h. The following functions are supported.

Installation check allows to determine if support for APM is present and if so which version is supported.

```
;================================================================================
; APM installation check
;================================================================================
; entry:    ax = 5300h
;           bx = 0000h, system BIOS
; returns:  cy = 0
;           ah = major version
;           al = minor version
;           bh = ascii "P"
;           bl = ascii "M"
;           cx = flags
;                bit 0 = 1 if 16 bit PM interface is supported
;                bit 1 = 1 if 32 bit PM interface is supported
;                bit 2 = 1 if CPU idle call slows CPU clock
;                bit 3 = 1 if BIOS power management is disabled
;
; NOTE: call is real mode int 15h only
;================================================================================
```

Interface Connect establishes the cooperative interface between the caller and the APM driver. Prior to this establishing this connection no APM functions are executed.

```
;================================================================================
; Interface connect
;================================================================================
; entry:    ax = 5301h
;           bx = 0000h, system BIOS
; returns:  cy = 0 if successful
```

```
;          cy = 1 if unsuccessful
;          ah = error code
;               02h = interface connect already in effect
;               09h = unrecognized device id
;
; NOTE: this call is only available using the APM int 15h interface
; ================================================================
```

Interface connect establishes a cooperative interaction between the APM driver and the caller. Interface disconnect breaks the cooperative interaction between the APM driver and the caller. It also restores any default APM functionality.

```
;================================================================
; 16 bit Interface connect
;================================================================
; entry:   ax = 5302h
;          bx = 0000h, system BIOS
; returns: cy = 0 if successful
;          cy = 1 if unsuccessful
;          ah = error code
;               05h = 16 bit interface already established
;               06h = 16 bit interface not supported
;               09h = unrecognized device id
;
; NOTE: this call is available using the APM int 15h or PM 16/32
interface. This function returns error code 06h.
;================================================================

;================================================================
; 32 bit Interface connect
;================================================================
; entry:   ax = 5303h
;          bx = 0000h, system BIOS
; returns: cy = 0 if successful
;          cy = 1 if unsuccessful
;          ah = error code
;               07h = 32 bit interface already established
;               08h = 32 bit interface not supported
;               09h = unrecognized device id
;
; NOTE: this call is available using the APM int 15h or PM 16/32
; interface. This function returns error code 08h.
;================================================================
```

```
;=============================================================================
; Interface disconnect
;=============================================================================
; entry:    ax = 5304h
;           bx = 0000h, system BIOS
; returns:  cy = 0 if successful
;           cy = 1 if unsuccessful
;           ah = error code
;                03h = interface not connected
;                09h = unrecognized device id
;
; NOTE: this call is available using the APM int 15h or PM 16/32
; interface
;=============================================================================
```

CPU Idle informs the APM interface that the system is currently idle and processing can be suspended until the next system event (typically an interrupt) occurs.

```
;=============================================================================
; CPU idle
;=============================================================================
; Inform the APM that the system is currently idle and specific power
; saving measures such as halting the CPU may be accomplished.
; entry:    ax = 5305h
; returns:  cy = 0, success
;
; NOTE: this call is available using the APM int 15h or PM 16/32
; interface
;=============================================================================
```

CPU busy informs the APM interface that the system is now busy and processing should continue at full speed. If the APM has placed the system in slow speed this will restore the CPU clock maximum processing speed.

```
;=============================================================================
; CPU busy
;=============================================================================
; Inform the APM that the system is currently busy processing should
; be accomplished at full speed.
; entry:    ax = 5306h
; returns:  cy = 0, success
;
; NOTE: this call is available using the APM int 15h or PM 16/32
; interface
;=============================================================================
```

Set power state places the requested system or device specified in the device ID into the requested state.

```
;=============================================================================
; Set power state
;=============================================================================
; entry:    ax = 5307h
;           bx = power device id
;                00xxh = system
;                00h = system BIOS
;                01h = all devices power managed by the system
;                      BIOS
;                01xxh = display
;                02xxh = secondary storage
;                03xxh = parallel ports
;                04xxh = serial ports
; where:                 xxh = physical device number (0 based)
;                        ffh = all devices in this class
;                cx = system state id
;                0000h = ready (not supported for system device id
;                        0001h)
;                0001h = stand-by
;                0002h = suspend
;                0003h = off
;
; returns: cy = 0, success
;          cy = 1, error
;          ah = error code
;               01h = power management functionality disabled
;               09h = unrecognized device id
;               0ah = parameter value in cx out of range
;               60h = cannot enter requested state
;
; - System stand-by
; entry:    ax = 5307h
;           bx = 0001h
;           cx = 0001h
;
; - Suspend system
; entry:    ax = 5307h
;           bx = 0001h
;           cx = 0002h
;
; NOTE: this call is available using the APM int 15h or PM 16/32
; interface
;=============================================================================
```

Enable/Disable allows the caller to enable or disable all automatic power down functionality.

```
;=============================================================================
; Enable/Disable all automatic BIOS power management functionality
;=============================================================================
; entry:    ax = 5308h
;           bx = ffffh
;           cx = 0 disable, 1 enable
; returns: cy = 0, success
;           cy = 1, error
;           ah = error code
;                01h = power management functionality disabled
;                09h = unrecognized device id
;                0ah = parameter value in cx out of range
;
; NOTE: this call is available using the APM int 15h or PM 16/32
; interface
;=============================================================================
```

Restore defaults requests that all APM default settings be restored.

```
;=============================================================================
; Restore BIOS power-on defaults
;=============================================================================
; entry:    ax = 5309h
;           bx = FFFFh
; returns: cy = 0, success
;           cy = 1, error
;           ah = error code
;                09h = unrecognized device id
;
; NOTE: this call is available using the APM int 15h or PM 16/32
; interface
;=============================================================================
```

Get power status requests APM to return current power status.

```
;=============================================================================
; Get power status
;=============================================================================
; return current power status
; entry:    ax = 530Ah
;           bx = 0001h
; returns: cy = 0, success
;           bh = AC line status
;                0 = off line
;                1 = on line
;               -1 = unknown
;           bl = battery status
;                0 = high
;                1 = low
;                2 = critical
;                3 = charging
;               -1 = unknown
```

```
;          cl = remaining battery life
;               0 - 100 (percentage of full charge)
;              -1 = unknown
;
; NOTE: this call is available using the APM int 15h or PM 16/32
; interface
;=============================================================================
```

Get PM event requests that AMP return its next pending APM event.

```
;=============================================================================
; Get PM event
;=============================================================================
; return the next pending PM event or indicate no PM events are
; pending.
; entry:   ax = 530Bh
; returns: cy = 0, success
;          bx = PM event code
;               0001 = system stand-by request notification
;               0002 = system suspend request notification
;               0003 = normal resume system notification
;               0004 = critical resume system notification
;               0005 = battery low notification
;          cy = 1
;          ah = error code
;               03h = interface connection not established
;               80h = no PM events pending
;
; NOTE: this call is available using the APM int 15h or PM 16/32
; interface
;=============================================================================
```

6 SERLOAD

The Ampro SERLOAD utility is a serial loader you can use to download and execute a block of executable code prior to system boot. Use a three-wire RS232 cable (Tx, Rx, and Gnd) from the host system's serial port with a CoreModule™ or Little Board™ COM1 port as the target. Use SETUP to enable the serial loader option on the target Ampro CPU product.

The host can be any PC or AT compatible with an RS232 port. Run the SERLOAD program on the host. The file must be a binary, executable file, with its origin at 0000h and a maximum size of 64 Kbytes. SERLOAD converts the file to ASCII and adds various control characters and sequences required by the serial loader function in the target Ampro CPU.

After the file is downloaded, it is executed by the target CPU. This is done by a FAR CALL. If the code terminates in a FAR RETURN, the target CPU boots. If the code does not terminate in this manner, it retains control of the target CPU and the boot process is not followed.

Examples of downloadable code are:

- A Monitor/Debugger program
- A bootstrap loader which takes control and downloads additional downloadable code
- A driver which intercepts INT13 BIOS services and uses the target's COM1 serial port to emulate a floppy or hard disk drive

6.1 Operation

To run the SERLOAD utility on the host, use the command:

```
C>SERLOAD filename.ext
```

This downloads the file *filename.ext* from the host to the target CPU. As the code is downloaded, the characters are echoed to the screen. The following characters have special meaning:

? SERLOAD is waiting for the target's serial loader function to become ready.
SERLOAD has sent a "break" character.

You may press the escape (Esc) key to exit the program during polling or download.

7 SERPROG

The purpose of the Ampro Serial SSD Programmer software, SERPROG, is to program Flash EPROM, NOVRAM, or battery-backed SRAM SSD devices directly within embedded systems that do not have disk drives. Using SERPROG, SSD devices within the embedded system (the "target") are programmed under control of a serially-connected remote PC (the "host"). This process is useful for initial programming of blank SSD devices during system manufacturing, or later on, when software updates are required.

SERPROG depends on the unique Ampro "serial loader" BIOS function for its operation, and may only be used on target systems based on Ampro CPUs that implement this function. The host system, used to remotely program the target embedded system, can be any PC-compatible computer that has a standard COM1 or COM2 serial port, including desktop, portable, laptop, and hand-held systems.

SERPROG actually consists of two pieces. The first piece, SERPROG.EXE, is a DOS program (compatible with MS-DOS, PC-DOS, and DR DOS) that runs on the host system. The other piece, SOCKETRW.BIN, is "downloaded" by SERPROG to the target system and performs the actual SSD programming process within the target system. SOCKETRW.BIN is not a DOS program. It is standalone code which runs in the target system before it boots, following either powerup or reset.

The Serial SSD Programmer software writes a simple binary image of the desired SSD into the target system's SSD device. Because of this, and because this process occurs prior to target boot, this software has two additional applications: (1) to program 55AAh "ROM-BIOS extensions" into SSD devices; and (2) to program SSD devices with SSD images for operating systems other than DOS.

7.1 Preparation

Before using the Serial SSD Programmer software, make the following software and hardware preparations:

- **Target system SETUP preparation**—enable the "Serial Loader" function within the target CPU, using SETUP. Refer to the Ampro CPU module's technical manual for information on how to do this.

Note

Depending on the particular Ampro CPU product BIOS version used, a jumper between RTS and RI on the serial port connector may be available which automatically enables both the serial loader and serial console features of the BIOS. Consult your module's technical documentation or call Ampro Technical Support for further information on this feature.

- **Host system software preparation**—copy SERPROG.EXE, SOCKETRW.BIN, and the required SSD image file(s) to a single subdirectory within the host system.

- **Serial connection**—run a serial cable between the COM1 port of the target system and either the COM1 or COM2 port of the host system. Only three wires are required: transmit data, receive data, and ground.

7.2 Command Line Parameters

The SERPROG utility, which runs on the host, has the following command line syntax:

```
SERPROG [[port] [options] [operation]
        socket=filename] [@cmdfile]
```

(The command line example appears here as two lines, but it is a single line.)

The command line parameters are:

[*port*]

COM1 or COM2 (COM1 is the default)

[*options*]

h, ? Displays a help screen listing the command line syntax with brief descriptions of each option.

d This option is used alone on the command line. It will indicate the types of SSD memory devices that SOCKETRW can program. When -d is specified, SERPROG downloads SOCKETRW to the target system, commands it to report all device types that it knows, and then displays a list of the supported device types.

[*operation*]

-w Write the contents of the SSD image in *filename*, located on the host, to the SSD device in *socket* on the target.

-r Read the contents of the target SSD device located in *socket*, and write its image to *filename* on the host.

-v Read the contents of the target SSD device located in *socket*, and compare it with the contents of the file, *filename*, on the host. Display information about the comparison.

-i Read the ID of the device in *socket* and display the device's ID, its size, and its type (5V or 12V). (This function is used with Flash EPROMs, only.)

socket

Enter a socket code to specify which socket is to be programmed. Socket codes are S0, S1, S2, S3, SX, Xna or Yna. To determine what socket code is appropriate, refer to your target board's technical manual.

For each *socket=filename* term on the command line or in a command file, SERPROG performs the most recently specified operation (or -w if no operation has been specified).

filename

Name of file containing SSD device image. This can be a full pathname for files not in the current directory or on the current drive.

[*@cmdfile*]

An optional command file, specified by *@cmdfile*, can be used to pass a series of commands to SERPROG instead of placing the commands directly on the SERPROG command line. Command files can contain any valid SERPROG commands except *@cmdfile*.

All command terms in square brackets, [ ], are options and may be omitted. Where defaults are provided, they are used when a corresponding parameter is left off the command.

All command terms except [*port*] may appear more than once on the command line.

SERPROG does not need to be told what type of device it will program. It checks an ID byte stored within the device to see if it is an SRAM (or NOVRAM) or one of several kinds of Flash EPROM, and then programs the device accordingly.

7.3 Operation

To program a memory device installed in a byte-wide socket, follow these three steps:

1. Start SERPROG on the host system, using an appropriate command (described below).
2. Powerup or reset the target system.
3. Wait for SERPROG to program the target SSD device(s) and complete its operation.

When SERPROG begins its operation, it waits indefinitely for the target to respond to its attempt to begin operation. If you wish to terminate SERPROG before it completes its operation, press the ESC key on the host system keyboard. Please note that there may be some brief periods of time during which the ESC key has no effect.

After SERPROG completes all requested operations, it sends the target a command to perform a system reset. At this point, the host system returns to DOS and the target system resumes its normal boot sequence.

The Ampro SSD Support Software includes a special utility, ADDFLASH, which allows the user to add support for Flash EPROM devices that are not included in the existing programs. When ADDFLASH is used, it creates a file, FLASH.ID, which contains the additional support. SERPROG supports this mechanism, thereby allowing you to use it with new devices that may not be currently available. In order to take advantage of this capability, the desired FLASH.ID file(s) must be present in the subdirectory from which SERPROG is run. For further information on ADDFLASH, refer to it description earlier in this manual.

7.4 Examples

Example #1: Basic use.

The simplest SERPROG command that might be used to program an SSD device in byte-wide socket S0 on any Ampro CPU module would be:

```
C>SERPROG S0=SSD.S0
```

In this case, the binary SSD image to be programmed into the SSD device in socket S0 on the target CPU board is located in the file SSD.S0. The particular file name used in this example, SSD.S0, could have been any legal DOS file name. Since no [port] is specified, SERPROG will use the default, COM1, on the host. Similarly, since no operation is specified, SERPROG assumes the user means to program SSD.S0 into the SSD in socket S0 (i.e. the -w command).

Example #2: Programming multiple devices.

Multiple SSD devices can be programmed in one command, as follows:

```
C>SERPROG S0=SSD.S0 S1=SSD.S1 S2=SSD.S2
```

In this example, it is possible that the three target SSD sockets contain different types of memory devices. For example, S0 might contain a battery-backed SRAM while S1 and S2 contain Flash EPROMs. The software automatically determines which devices are present, and programs them accordingly.

Example #3: Programming devices on an SSD expansion board.

You can also use SERPROG to write to devices on the Ampro MiniModule™/SSD expansion module. A typical command, which would program all four sockets, might be:

```
C>SERPROG X1A=SSD.X1A X1B=SSD.X1B X1E=SSD.X1E
          X1F=SSD.X1F
```

Two of the above commands might be included in a single command line such as:

```
C>SERPROG S0=SSD.S0 X1A=SSD.X1A X1B=SSD.X1B X1E=SSD.X1E
          X1F=SSD.X1F
```

This command would program SSDs on both a CoreModule CPU and a MiniModule/SSD expansion module.

Example #4: Verifying that a device was programmed correctly.

This example illustrates how to verify that the target SSD devices have been programmed accurately.

```
C>SERPROG S0=SSD.S0 -v S0=SSD.S0
```

First, this operation writes the contents of the file SSD.S0 to S0. Then, it verifies the contents of the SSD located in socket S0 on the target against the contents of the file SSD.S0 on the host.

Example #5: Using a command file.

This example substitutes a command file for the command line parameters. (In this example, the command line from Example #3 above is used.)

First, create the command file using an ASCII text editor such as DOS's EDIT.EXE. Enter the command line as it would be written on the command line, but do not invoke the SERPROG program:

```
S0=SSD.S0 X1A=SSD.X1A X1B=SSD.X1B X1E=SSD.X1E X1F=SSD.X1F
```

Save the file, giving it a filename, for instance, COMMAND.FIL.

To use the command file, execute the following command line:

```
C>SERPROG @COMMAND.FIL
```

8 SETSPEED

You can use the Ampro SETSPEED utility to set the Ampro CPU speed to either normal or slow speed. Refer to your CPU's technical documentation to ascertain whether your particular CPU supports dual-speed operation.

You can run the program automatically from a batch file, or manually from the DOS command line.

8.1 Operation

To use the SETSPEED utility, either type the program's name, along with the desired option (listed below), on the DOS command line, or enter a similar command in a batch file and execute the batch file.

The command choices are:

```
C>SETSPEED P=LO     ; sets CPU to slow speed
C>SETSPEED P=HI     ; sets CPU to normal speed
C>SETSPEED ?        ; displays a help screen
C>SETSPEED          ; displays current speed status
```

Changing the speed of the CPU does not change the expansion bus speed. The expansion bus is fixed, and is typically set to emulate a standard 8 MHz PC/AT bus.

9 SETUP

The Ampro SETUP function resides in the ROM BIOS. The Ampro SETUP.COM utility provides access to the SETUP function from the DOS command line. Use it to initialize the nonvolatile Configuration Memory on an Ampro CPU module or board. System configuration parameters, used by the ROM BIOS to establish the system setup at boot time (powerup or reset), are stored in two non-volatile memory components on the Ampro CPU:

- AT compatible battery-backed CMOS RAM within the clock device
- Ampro-unique EEPROM memory

The SETUP related contents of the CMOS RAM are automatically backed up in the EEPROM to provide for battery-free operation. One utility, SETUP, initializes and modifies parameters in both memory devices. These two memories are called the Configuration Memory.

The SETUP function is organized into pages. Generally the first page is devoted to standard settings, the settings normally found on a desktop PC. Additional pages are added for extended settings, created by Ampro to enhance the CPU's usefulness and configurability in embedded system applications. Each CPU has a slightly different set of SETUP parameters. Refer to your CPU's technical manual for details about setting up its configuration memory.

Note

Changes made using the SETUP function do not take effect until the next time the system boots.

There are two ways to access the SETUP function. It can be accessed from its interactive menu-based user interface, or it can be directed to automatically take its inputs from a file. The first of these methods is the normal way you initialize and modify a system's configuration. The file-based mode of operation allows you to automate the process—for example, in volume system production.

9.1 Interactive Mode of Operation

There are two ways to invoke SETUP's interactive mode of operation. You can invoke SETUP at system powerup or reset time by holding down the following hot key combination:

```
CTRL-ALT-ESC
```

When you power up or reset the system, a message at the bottom of the screen tells you when you can use the hot key entry into SETUP.

You can also invoke SETUP during system operation, with the Ampro SETUP.COM program, by typing the program's name at the DOS command line, as follows:

```
A>SETUP
```

9.2 Configuration File Mode of Operation

The Ampro SETUP utility, SETUP.COM, offers the option of saving and loading the contents of the configuration memory from a disk file. The following command line parameters are available with SETUP.COM:

```
SETUP [-switches] [ @file.ext | Wfile.ext ]
```

The meanings of the command line parameters are:

-?	Display a usage help screen.
-T	Set the NOVRAM time and date to the DOS time and date.
-O	Allows entry to SETUP even if the Extended BIOS services have been disabled.
-0	Set the EEPROM write count to zero.
@*file.ext*	Writes the specified file to the board's Configuration EEPROM. The drive and path are optional in the file name.
W*file.ext*	Write EEPROM contents to the file specified. The file name may contain an optional drive and path.

You can save a copy of the current contents of the board's Configuration Memory to a disk file by using the W switch. You can load into Configuration Memory the contents of a file using the @ switch. Configuration changes made using the SETUP *@file.ext* command do not take effect until the next reset. Also, using the W switch to write the configuration to a file will not reflect any changes made since the last reset.

The data saved in a configuration file includes the entire contents of the nonvolatile configuration EEPROM, except the current time and date. The first 512 bits are the SETUP information, the last 512 bits are available for OEM storage (See Ampro Application Note AAN-8805). The file you create with this menu option can be used as a source for programming the Configuration Memory of your Ampro CPU at a later time.

Following is an example of a typical file saved by SETUP:

```
NOVRAM 00:  24 00 35 00 08 00 07 05 - 12 89 26 02 50 80 00 00
NOVRAM 10:  12 00 00 00 41 80 02 80 - 01 00 00 00 00 00 00 00
NOVRAM 20:  00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 01 56
NOVRAM 30:  80 01 19 80 00 00 00 00 - 00 00 00 00 39 00 38 E8
EEPROM 00:  EB 33 10 19 19 1A A8 BF - F4 FF FF FF FF FF 3E 00
EEPROM 10:  81 E1 FF FA FF FF FF FF - FF FF FF FF FF FF FF FF
EEPROM 20:  FF FF FF FF FF FF FF FF - FF FF FF FF FF FF FF FF
EEPROM 30:  FF FF FF FF FF FF FF FF - FF FF FF FF FF FF FF FF
EEPROM 40:  FF FF FF FF FF FF FF FF - FF FF FF FF FF FF FF FF
EEPROM 50:  FF FF FF FF FF FF FF FF - FF FF FF FF FF FF FF FF
EEPROM 60:  FF FF FF FF FF FF FF FF - FF FF FF FF FF FF FF FF
EEPROM 70:  FF FF FF FF FF FF FF FF - FF FF FF FF FF FF FF 00
```

As an example, the following command initializes the EEPROM values with a previously saved configuration:

```
C>SETUP  @SYSTEM.CFG
```

Assuming you created the file SYSTEM.CFG with SETUP's write option, SETUP will initialize the EEPROM Configuration Memory using the contents of that file.

Note

Addresses not specifically named are not changed. For example, if you have a file with an entry for EEPROM 20, only the entry at that location is changed. All other contents of Configuration Memory remain the same.

Using SETUP with the write and read parameters can be useful when many boards must be initialized automatically. Another use for this SETUP mode might be to change between several predefined system configurations.

10 SCSI-ID

The Ampro SCSI-ID utility reports the system's SCSI Initiator ID to the console, and also sets the DOS ERRORLEVEL so it may be tested in a batch file. One use of the SCSI-ID utility is that it allows you to easily verify the SCSI-ID setting of your Ampro CPU. An interesting use of this program is to allow multiple CPUs to boot from a single SCSI device, yet automatically begin execution of unique applications based on each board's SCSI ID.

10.1 Command Line Operation

To use the program, simply enter the program's name on the DOS command line or in a batch file. For example, if your system's SCSI ID is 7 and you use the following command line:

```
C>SCSI-ID
```

SCSI-ID will display:

```
My SCSI ID is 7
```

10.2 Batch File Operation

In addition, SCSI-ID sets the DOS ERRORLEVEL to a number corresponding to the SCSI-ID setting, for testing within batch files. Using the program in this manner, it is possible to boot multiple Ampro CPUs from a single SCSI drive under PC-DOS, yet have each board come up running a different application.

To accomplish this, the AUTOEXEC.BAT file on the boot must test for the DOS ERRORLEVEL and branch to a unique batch file appropriate for the specific CPU. Then, each CPU is set for a different SCSI ID, between 1 and 7, and the boot device is jumpered for ID 0.

Ampro application note AAN-8803 discusses the use of SCSI-ID in a DOS system composed of multiple Ampro CPUs.

11 SCSI-VER

This utility reports the AMPRO SCSI/BIOS version to the console, and also sets the DOS ERRORLEVEL so that it may be tested in a batch file. To use this program, simply enter its name on the DOS command line. For example, if the currently installed SCSI/BIOS is version 1.03 and the following line is used:

```
C>SCSI-VER
```

SCSI-VER will display:

```
My SCSI version is 1.03
```

It will also set the DOS ERRORLEVEL to the integer value of the version multiplied by 10, which in the case of version 1.03 would be "10".

12 SCSICOMP

The Ampro SCSICOMP utility allows you to compare the contents of two SCSI direct access devices (e.g. hard disk drives). Although the program must be run from DOS, SCSICOMP does a binary comparison of blocks of data. It does not matter what (if any) operating system was used to write the data to the SCSI devices.

You can specify any two SCSI controller ID's, drive Logical Unit Numbers (LUN's), SCSI block range to be compared.

12.1 Operation

To run the program, type its name at the DOS command line:

```
A>SCSICOMP
```

The program will display a sign-on message and will then prompt you to press the <Enter> key to continue. After you press the <Enter> key, the program will prompt you for information required, including the first SCSI ID, first logical unit number, second SCSI ID, second logical unit number, starting block number, and number of blocks to compare. Then, the program will compare the specified segments of the two SCSI devices and report on any errors or differences that occur during the comparison.

The required parameters are defined as follows:

- SCSI ID—Each device's SCSI ID is determined by the jumpering of the device's SCSI controller and is in the range 0 through 7.
- Logical Unit Number (LUN)—Each device's logical unit number is based on how the device is connected to the controller and may also depend on the setting of jumpers on the device; it is generally 0 or 1.
- Block Number—A SCSI "block" is 512 bytes of data. Blocks are numbered starting from 0.

The current version of SCSICOMP requires that the block starting numbers on the two drives be the same, so only one starting block number is requested by the program.

12.2 Example

In this example, 2000 blocks (10,240,000 bytes) of data are compared, between LUN 0 on SCSI ID 0 and LUN 1 on SCSI ID 1.

```
What is the first SCSI ID and logical unit number?
SCSI ID (0-7): 0
Logical unit number (0-3): 0

What is the second SCSI ID and logical unit number?
SCSI ID (0-7): 1
Logical unit number (0-3): 1

Starting block number: 0
Number of blocks: 2000
```

```
You are about to compare:

From: SCSI ID: 0          To:  SCSI ID: 1
          LUN: 0                    LUN: 1

Starting with block 0, for 2000 blocks.

Is this correct (Y/N)? Y
Press the <Enter> key to begin or the <ESC> key to start over:<Enter>

Compare completed, 0 errors.

Compare another (Y/N)? N
A>
```

13 SCSICOPY

The Ampro SCSICOPY utility allows you to copy a block of data between two SCSI direct access devices (e.g. hard disk drives). Although the program must be run from DOS, SCSICOMP does a binary comparison of blocks of data. It does not matter what (if any) operating system was used to write the data to the SCSI devices.

You can specify any two SCSI controller ID's, drive Logical Unit Numbers (LUN's), and any block range.

Warning!

(1) SCSICOPY will destroy data on the destination device within the specified block range. Use with extreme care.

(2) Copying less than the full drive may leave part of the destination device unusable.

13.1 Operation

To run the program, type its name at the DOS command line:

```
A>SCSICOPY
```

The program will display a sign-on message and will then prompt you to press the <Enter> key to continue. After you press the <Enter> key, the program will prompt you for information required, including the first SCSI ID, first logical unit number, second SCSI ID, second logical unit number, starting block number, and number of blocks to copy. Then, the program will copy the specified segment between the two SCSI devices and report on any errors that occur during the process.

The required parameters are defined as follows:

- SCSI ID—Each device's SCSI ID is determined by the jumpering of the device's SCSI controller and is in the range 0 through 7.
- Logical Unit Number (LUN)—Each device's logical unit number is based on how the device is connected to the controller and may also depend on the setting of jumpers on the device; it is generally 0 or 1.
- Block Number—A SCSI "block" is 512 bytes of data. Blocks are numbered starting from 0.

The current version of SCSICOPY requires that the block starting numbers on the two drives be the same, so only one starting block number is requested by the program.

13.2 Example

In this example, 2000 blocks (10,240,000 bytes) of data are copied from LUN 0 on SCSI ID 0 to LUN 1 on SCSI ID 1.

```
What is the first SCSI ID and logical unit number?
SCSI ID (0-7): 0
Logical unit number (0-3): 0
```

```
What is the second SCSI ID and logical unit number?
SCSI ID (0-7): 1
Logical unit number (0-3): 1

Starting block number: 0
Number of blocks: 2000

You are about to copy:

From:SCSI ID: 0    To:  SCSI ID: 1
          LUN: 0   LUN: 1

Starting with block 0, for 2000 blocks.

Is this correct (Y/N)? Y
Press the <Enter> key to begin or the <ESC> key to start over:<Enter>

Copy completed, 0 errors.

Copy another (Y/N)? N
A>
```

14 SCSIFMT

SCSIFMT is the Ampro SCSI hard disk formatter utility. It is used to perform the low-level format for SCSI hard disk drives, prior to final preparation of the drive using the standard drive preparation utilities offered by your operating system.

SCSIFMT supports the SCSI Common Command Set (CCS) direct access devices (typically hard disk drives). Consult the specific SCSI hard disk controller manual for information as to compatibility with the SCSI CCS.

SCSIFMT can be used to completely reformat the target drive, and to map out bad blocks. The list of bad blocks is appended to the on-disk bad block table, which is furnished by the disk manufacturer.

Warning!

If you format your drive, all data on the drive will be destroyed.

14.1 Operation

The general format of SCSIFMT is as follows:

```
SCSIFMT Ii [Ll] [Zz] [F] [M[m]] [Y]
```

Parameters may be entered in any order and are case insensitive. The parameters are:

i The SCSI ID number, 0 through 7, of the SCSI device that is to be accessed. This parameter is required.

l The Logical Unit Number (LUN) of the SCSI device that is to be accessed. If not supplied, the default value is 0.

z The disk interleave factor, which only has meaning when used with the F (format) option. If not specified (or set to 0), the interleave factor is left unchanged from the drive's current setting. A typical value is 2; most fast drives when used with Ampro's SCSI/BIOS can support an interleave factor of 1.

F The "format" option. It must be specified if you wish to format the drive (and optionally change the interleave factor).

M The "map out" option. When selected, any bad blocks (blocks with read errors, write errors, or both) are mapped out of the drive's block allocation table. If data exists in the bad block, an attempt is made to safely copy the data to another block.

m The number of clean passes for the bad-block map out. Specifies how many passes must be made with no bad blocks found. If not specified, defaults to 1. The maximum value is 255. A new pass begins when a bad block is found in the current pass.

Y Confirm bypass option. If used, SCSIFMT does not ask for à confirming "Y" keystroke before continuing with the format. USE WITH CAUTION!

After the format is complete, you still must prepare the drive in the standard manner for access by your operating system. Refer to your CPU board's technical manual for detailed step-by-step SCSI hard disk installation procedures.

14.2 Examples

Example #1: Bad block search.

```
A:>SCSIFMT I4 L0 M5
```

This example searches the SCSI device with SCSI ID 4, LUN 0, for bad blocks and maps out any bad blocks that are discovered. Five clean passes are performed before the command completes.

Example #2: Format a SCSI drive.

```
A:>SCSIFMT I0 F Z1
```

This example formats the SCSI device with SCSI ID 0, using an interleave factor of 1.

Example #3: Format a SCSI drive, not confirmation.

```
A:>SCSIFMT I2 F Y
```

This example formats the SCSI device with SCSI ID 2, *without asking for confirmation.*

15 SCSIPARK

The Ampro SCSIPARK utility is used to position the read/write head(s) of a SCSI drive to a predefined safety zone on the disk surface, to guard against accidental data loss due to either power on/off glitches in the drive electronics or media damage due to mechanical shock.

This is a "legacy" utility, as most modern SCSI drives automatically park their heads in a safe landing zone during power down. You would only use this utility on drives that do not provide automatic parking. There is no harm in using this utility on drives that automatically park their heads—it's just not needed.

The current version of SCSIPARK supports the following SCSI controllers and drives:

- Little Board-based SCSI controllers
- MiniModule-based SCSI controllers
- Adaptec ACB-4000 and ACB-5000 family controllers
- Seagate ST225N SCSI drive
- Most "Common Command Set" SCSI drives

15.1 Operation

To use SCSIPARK to park a drive's heads, type the program's name followed by one or more SCSI ID(s) of the controller(s) to which the drive(s) are connected. All drives on each SCSI controller will be parked by the single command. For example:

```
C>SCSIPARK 0 1
```

will park all drives detected on SCSI controllers with ID's of 0 and 1. You can also use an asterisk (*), to have all possible SCSI ID's parked. That is,

```
C>SCSIPARK *
```

will park drives on all SCSI controllers detected. "SCSIPARK *" is equivalent to "SCSIPARK 01234567".

Note

Be sure not to access a hard disk drive after it is parked, as this will move the drive's heads off the landing zone.

You can simplify the use of the SCSIPARK utility by creating a DOS "batch" file containing the required command line.

16 SCSITOOL

The Ampro SCSITOOL utility is a powerful and flexible SCSI debugging tool which allows you to demonstrate, test, and debug SCSI commands and devices. Options are provided which allow you to create any desired SCSI command, and write or read desired data patterns to or from any type of SCSI device. Although the program must be run from DOS, SCSITOOL pays no attention to the contents of the SCSI devices accessed, and does not care what (if any) operating system has been used to write data to them.

Warning!

SCSITOOL can destroy data on the destination SCSI device. Use this utility with extreme care!

16.1 Operation

The SCSITOOL utility is very easy to use, but you need to understand SCSI command principles in general, as well as the specific SCSI command set of the SCSI target devices you will be accessing. Refer to the technical information supplied by the SCSI target devices' manufacturers, and to the ANSI X3.131 SCSI specification for a full functional specification of the SCSI interface protocols.

To begin program operation, type the program's name at the DOS command line:

```
A>SCSITOOL
```

The program will display a brief sign-on message, followed by the SCSITOOL command prompt:

```
Command (C,D,E,R,S,?,<ESC> to quit): __
```

Entering a "?" results in the display of SCSITOOL's command menu:

```
Commands available:

      C - Enter command buffer
      D - Enter data buffer
      E - Execute SCSI command
      R - Reset SCSI bus
      S - Set SCSI ID
      ? - Display this help message

Command (C,D,E,R,S,?,<ESC> to quit):
```

When you use either the C (Enter command buffer) or D (Enter data buffer) options, the program will display the current contents of the SCSI command or data buffer in one area of your screen, and will position the cursor over the first byte of data on the command or data buffer entry line.

For example:

```
Command (C,D,E,R,S,?,<ESC> to quit): C

The Command Buffer currently contains:

0  1  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18 19 ...
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- ...
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ...

Enter the new values in hexadecimal.  The <Spacebar> leaves a
digit unchanged, while the <Enter> key stops editing and returns
to the list of commands.
```

You can use the <Spacebar> key to move through the buffer to the bytes you wish to modify. Enter the data using the keyboard. Press the <Enter> key to terminate the editing process.

Similarly, you view and optionally edit data in the SCSI data buffer with the D (Enter data buffer) option, if data is to be sent or received from the SCSI device. The data buffer size is actually 5K bytes, but only the first 512 bytes are available through this program's D command. Since the entire 512 bytes of the data buffer display do not fit entirely on one screen, use <CTRL-S> to pause the display when necessary (and any other key to resume display).

Do not execute a SCSI command with the E (Execute SCSI command) option until you have first used the S (Set SCSI ID) option to specify the SCSI ID of the device to which the command will be issued.

16.2 Examples

Example #1: Test Unit Ready command.

Enter all 0's in the command buffer with the C command; set the appropriate SCSI controller ID with the S command. Then use the E command to perform the Test Unit Ready function. (This example assumes LUN 0.)

Example #2: Rezero Unit command.

Enter 01 in the first command byte, and all 0's in the rest of the command buffer, with the C command; set the appropriate SCSI controller ID with the S command. Then use the E command to perform the Test Unit Ready function. (This example assumes LUN 0.)

17 TVTERM

TVTERM is a terminal emulator program with partial emulation of the Televideo 900 series (and compatible) terminals. TVTERM allows a PC-compatible computer system to act as a remote console for Ampro CPU products configured for serial console mode. This feature is enabled in the Ampro BIOS extended SETUP menu, and is available on all Ampro CPU products.

Note

Depending on the particular Ampro CPU product BIOS version used, a jumper between RTS and RI on the serial port connector may be available which automatically enables the serial console feature of the Ampro BIOS. Consult your module's technical documentation or call Ampro Technical Support for further information about this feature.

TVTERM supports IBM CGA, MONO, EGA, and VGA compatible video adapters in 80 X 25 text modes. Serial data rates from 110 through 115,200 baud are selectable. The serial port used must be a PC-compatible 8250- (16C450-) type UART, addressed as either "COM1" or "COM2". Note that not all systems can support the higher speeds above 9600 baud. Hardware handshake via DTR/RTS is used to control communications. XON/XOFF handshaking is not supported. TVTERM will continue to process serial data interrupts and process data when running in background mode.

The Televideo "no wrap" option is supported. That is, if the cursor is in column 80, the next character will not cause the cursor to advance to the first position of the next line. Emulation of Televideo function keys and arrow keys are not supported. Bright and half intensity are supported.

17.1 TVTERM Hot Keys

TVTERM is a "terminate-stay-resident" (TSR) type program. Like most TSRs, when you run TVTERM, it loads itself into memory and then waits for a "hot-key" to activate it.

Enter and Exit Terminal Mode

<Alt-RightShift-P> is used to enter terminal mode.

<Alt-RightShift-L> is used to return to normal display mode.

These are the primary hot-keys. When you exit terminal mode, the screen is restored to what was there before entering.

Status Mode

<Alt-1> enables "Status Mode". This will display the status of the handshake lines and communication errors on line 25 of your screen.

Monitor Mode

<Alt-2> enables "Monitor Mode". Monitor Mode may only be enabled if TVTERM is already in "Status Mode." In this mode, the remainder of line 25 is used to display characters as they are received from the serial port. Characters are displayed with the half intensity reverse attribute. Control characters (Hex 00 to Hex 1F) are shown in Hex, with the half intensity attribute. Characters received with an error condition (parity, framing, and overrun) are displayed with normal background hi-intensity attribute.

Monitor Mode may drop characters at higher baud rates, depending on the device you are communicating with.

17.2 Operation

TVTERM is installed by typing it's name and optional parameters at the DOS prompt, or including a similar text line in a batch file (possibly in AUTOEXEC.BAT). The general form of the command is:

```
C>TVTERM COMx Bb Ss Dd Pp
```

The command line parameters are used to select and initialize the serial port used for TVTERM. The following command line options may be used, in any order. Parameters are case insensitive.

COM*x* Communication port:
x = 1: use COM1 port
x = 2: use COM2 port (default)

B*b* Baud rate:
b = 110, 150, 300, 600, 1200, 2400, 4800, 9600 (default), 19200, 38400, 57600, or 115200 BAUD.

S*s* Stop bits:
s = 1: 1 stop bit (default)
s = 2: 2 stop bits

D*d* Word length:

d = 5: 5 data bits
d = 6: 6 data bits
d = 7: 7 data bits
d = 8: 8 data bits (default)

P*p* Parity:

p = N: no parity (default)
p = E: even parity
p = O: odd parity
p = S: space parity
p = M: mark parity

TVTERM does not protect you from illegal or unusual data lengths or combinations of parameters.

18 WATCHDOG

The Ampro WATCHDOG utility is used to start, stop, or retrigger an Ampro CPU board's watchdog timer from the command line or from within a batch file.

18.1 Operation

To use the WATCHDOG utility, simply type the program's name, along with the desired option (listed below), on the DOS command line. Or, you may enter a similar command in an appropriate batch file.

The command choices are:

```
C>WATCHDOG OFF
```

which turns the timer off. To set a specific watchdog time interval, use this syntax:

```
C>WATCHDOG ON=secs
```

This command starts or retriggers the watchdog timer with a time-out of *secs* seconds. **xxx** has a range of 1 to 255 seconds.

```
C>WATCHDOG ?
```

displays a help screen.

Generally, the watchdog timer should be retriggered from within your application program, rather than with the WATCHDOG utility program. See your Ampro CPU technical manual for information about the ROM-BIOS function provided for this purpose.

19 XTCLK

XTCLK is a clock utility for the CoreModule/*XTPlus*. (It is only used for this product.) It is used to initialize the DOS clock from the battery-backed real-time clock at power up time. It can also be used to set the real-time clock to the current DOS time.

XT-class computers did not have a real-time clock, and as a result, BIOSes do not typically support a real-time clock like they do on AT-class computers. This utility is executed at boot time, typically from the AUTOEXEC.BAT file, and synchronizes the DOS time and date values to the time and date values it finds in the batter-backed real-time clock.

When setting the DOS time and date values from the real-time clock, there are no command line arguments to set. Simply include XTCLK on a line in your AUTOEXEC.BAT file, or run it from the command line prompt.

To set the hardware real-time clock to a new date and time, set the desired date and time using the DOS DATE and TIME commands. Once the DOS date and time values are correct, enter the following command to set the real-time clock to match the DOS date and time:

```
C:>XTCLK /S
```

The /S (case insensitive) switch tells XTCLK to set the real-time clock to match the DOS clock.

To get a help screen, type the following command:

```
C:>XTCLK /H
```

The /H (case insensitive) causes a brief help message to be displayed on the console.

20 XTPEMM

XTPEMM is an Expanded Memory Specification driver (EMS), used to access onboard RAM above the 640K boundary in conformance with the LIM 4.0 standard. (It is only used for the CoreModule/XT*Plus* product.) This is a DOS device driver, intended to be loaded at boot time by the CONFIG.SYS file.

Internal control logic can convert onboard DRAM beyond 640K into expanded memory (EMS). With the EMS option enabled, 640K of the module's DRAM is present in the normal V30 system address space, and the balance of the memory installed on the module is accessed as EMS memory by page addressing. This EMS memory can be accessed by a wide variety of application and utility programs, including both disk cache and RAM-disk drivers. Special programming techniques must be used when writing software which accesses expanded memory. Refer to the LIM 4.0 specification for details.

Selection of the page frame memory address is done with a command line parameter in the CONFIG.SYS file. It is important to configure your system such that the memory space specified for EMS does not conflict with other functions. A memory map for the CoreModule/XT*Plus*, including possible memory assignments for the EMS 64K page frame, is included in Chapter 4 of the CoreModule/XT*Plus* Technical Manual.

To use XTPEMM.SYS, add a line to the CONFIG.SYS file as follows:

```
DEVICE=XTPEMM.SYS
```

With this command, the EMS driver will be loaded with a set of default values for the I/O addresses that control the EMS hardware on the CoreModule and the page address used for the EMS 64K page frame buffer. The system can then be used to support expanded memory.

Normally, you do not need to alter the default values that are provided in the program. There will not be an I/O address conflict because the I/O port addresses used by the module's paging hardware is located in the range of 00h to Ffh, which are reserved port addresses not used by any standard peripherals. However, the page frame address of the memory buffer can be used by other devices in the system. Therefore, there is provision for changing this value on the CONFIG.SYS command line. The default value is D0000h. To change the page frame address assignment, add the new assignment on the command line as follows:

```
DEVICE=XTPEMM.SYS [page address]
```

Where [*page address*] can be C0000, D0000, or E0000. (Do not include the brackets.) For example, to set the EMS page frame address to E0000, use the following line:

```
DEVICE=XTPEMM.SYS E0000
```

This command sets the EMS page frame to E0000h to EFFFFh.

5.2 Ampro Application Handbook

The Ampro Application Handbook contains several application specific papers and notes, which aid the user both in the configuration of hardware and the development of software. Additional application notes and white papers can be found at

http://www.ampro.com/university/index.htm.

The Ampro Application Handbook is included in the Quick Start Kit – CM2-SXI-K-00, which can be purchased from Ampro for $186.

Application Handbook

Technical Manual

P/N: 5000121

Revision: B

Ampro Computers, Incorporated
4757 Hellyer Avenue ■ San Jose, CA 95138
Tel (408) 360-0200 ■ FAX (408) 360-0220
WEBSITE: www.ampro.com

Application Note Handbook

Table of Contents

Application Note Index

APPLICATION NOTE

APPLICATION NOTE INDEX

November 5, 1996

The following application notes are currently available from the Ampro Technical Services Department :

AAN-8702A -- ROM-based system operation.

AAN-8804A -- Ampro SCSI/BIOS Interface Specification.

AAN-8805C -- Using the onboard configuration memory EEPROM for custom applications.

AAN-9002A -- Third party support software for serial port and modem operation.

AAN-9003A -- Using the Ampro ROM-BIOS "OEM hooks".

AAN-9005C -- Third party sources of enclosures for Little Board based systems.

AAN-9006B -- A plug-in "watch dog timer" for the Ampro Little Board products.

AAN-9009A -- Multi-tasking and real-time operating systems for embedded PC architecture applications.

AAN-9101B -- Using expansion cards and passive backplanes with Ampro Little Board and CoreModule products.

AAN-9201B -- Design Issues for Embedded PCs.

AAN-9202B -- Object-Oriented Hardware Design for Embedded Systems.

AAN-9204A -- Third Party Sources of PC Keyboard Simulators.

AAN-9205A -- 4-Point Mounting of MiniModule Stacks.

AAN-9206A -- Third Party Sources of Monitors.

AAN-9207A -- Third Party Sources of Touch Input Devices.

AAN-9208A -- Third Party Sources of Real Time Executives.

AAN-9209A -- Third Party Sources of TCP/IP and NFS Support Software.

AAN-9210A -- Ampro Extended BIOS Interface Specification.

AAN-9401D -- A Mean Time Between Failure analysis of various Ampro products.

AAN-9402A -- Building CoreModule Systems.

AAN-9403C -- "Remote hosted" use of the CoreModule/PC.

AMPRO®
AMPRO COMPUTERS, INCORPORATED
990 Almanor Avenue, Sunnyvale, California 94086
TEL (408) 522-2100 - FAX (408) 720-1305

Application Notes

APPLICATION NOTE

Number: #AAN-8702 Date: September 3, 1987

Title: ROM-based operation of the Little Board/PC.

Product(s): Little Board/PC

Abstract: A discussion of two alternative methods of loading software from EPROM, including an example of how to generate a "ROM-BIOS extension".

INTRODUCTION

The features of the AMPRO CMOS Little Board/PC (TM) make it ideally suited for use in rugged or harsh operating environments, or as a controller "embedded" within some device which is not perceived as a computer. In such applications the use of magnetic media such as floppy and hard disk drives to hold the operating system and application programs is often unacceptable. Instead, it is desired in these cases to substitute semiconductor memory (EPROM and/or nonvolatile RAM) for the disk drives normally required to boot and run the system.

There are two approaches to ROM-based Little Board/PC operation:

(1) The AMPRO Solid State Disk (SSD) Drive Support Software, allows you to configure a Little Board/PC system to boot, operate, and even store data using one or two EPROM and/or RAM drives, under control of the DOS (PC-DOS or MS-DOS) operating system. In this case, the board's two spare byte-wide memory sockets (U15 and U26) can be populated with either EPROM's or nonvolatile RAM devices. The AMPRO SSD utilities are used to format and access the devices in these sockets as DOS disk drives, and no special software development effort is required. In addition, since system operation is based on PC-DOS, floppy and hard disk support are always available if required for data storage or loading by your application program.

(2) If your application does not require the services of DOS (PC-DOS or MS-DOS), and if the use of floppy or other disk drives for data storage or loading is also not required, you can use the Little Board/PC's two spare byte-wide memory sockets to load your application program directly. This is done by means of what is known as a "BIOS Extension", which automatically executes upon system powerup or reset.

The remainder of this application note provides the technical information a programmer needs to produce a BIOS Extension capable of automatic powerup

AMPRO COMPUTERS, INCORPORATED
990 Almanor Avenue, Sunnyvale, California 94086
Telephone: (408) 522-2100 - FAX (408) 720-1305

execution without the use of DOS.

HOW A BIOS EXTENSION WORKS

As mentioned above, the Little Board/PC's ROM-BIOS contains a mechanism for loading programs directly from EPROM prior to booting DOS. These programs are called "BIOS Extensions", and can be contained in one or both of the board's two spare byte-wide memory sockets (U15 and U26).

Following system poweron or reset, the ROM-BIOS initializes the board's hardware and performs a Power-On-Self-Test (POST). Following the POST, the ROM-BIOS scans upper memory to find any "BIOS Extensions" that may exist. This is the point at which the BIOS Extension can gain control.

Because a BIOS Extension executes before DOS is loaded, it cannot use DOS services, nor can it depend on any device driver specified in a CONFIG.SYS file or resident programs that might be loaded by an AUTOEXEC.BAT. All ROM-BIOS services, however, are available. These should be adequate for most applications.

STRUCTURE

The BIOS Extension has a well defined structure. It begins with a three-byte header which consists of a two-byte ID pattern (55h, AAh) followed by a SIZE byte. The SIZE byte indicates the number of 512 byte blocks to be included in the checksum calculation (described below).

When the ROM-BIOS locates a properly formatted BIOS Extension, it makes a FAR CALL to the fourth byte in the BIOS Extension (address 0003 in the EPROM). This is the "entry point" of the EPROM. The remainder of the BIOS Extension contains the program itself. (One additional constraint is that the checksum of the blocks referenced by the SIZE byte must be zero, as described below.)

LOCATION

When the Little Board/PC ROM-BIOS scans system memory for BIOS Extensions, it begins at address C8000h and checks for the 55AAh header on every 2K boundary up to address F4000h. If a BIOS Extension header is not found within that region of memory, the ROM-BIOS proceeds with the normal system boot sequence. Consequently, if a properly formatted BIOS Extension is present on a 2K boundary between C8000h and F4000h in memory, it can seize control of the system before any attempt to boot DOS is made.

CHECKSUM

Prior to transferring control to a BIOS Extension, the ROM-BIOS performs a checksum calculation on the number of blocks of the BIOS Extension indicated by the SIZE byte, and verifies that the result is zero. The checksum process consists of a simple additive sum (modulo 100 hex), beginning with the "55AA" ID bytes and including all other locations in memory defined by the SIZE byte. If the result is non-zero, a checksum error message is displayed, the BIOS Extension is ignored, and the boot sequence continues. Consequently, when you create a BIOS Extension, you must calculate the checksum of the memory image and then alter a byte value in the program to cause the checksum calculation to

become zero.

RAM USAGE

The AMPRO ROM-BIOS itself uses RAM from 00000 to 00500h. Your BIOS Extension can use any RAM above this, up to the limit of the available memory in the system. It is recommended that you begin your program's data areas at 00600h, to allow for future BIOS RAM area expansion. The ROM-BIOS provides a service for determining the amount of RAM available in the system. Executing an INT 12H will return the available memory size, in Kbytes, in register AX.

A PROGRAMMING EXAMPLE

The following example shows how to set up an assembly language program as a BIOS Extension.

```
;=============================================================
;   This initialization code is for a "tiny model", with code, data
;   and stack in the same segment.
;=============================================================

BIOS_END equ     00060h                    ; end of RAM used by BIOS

DGROUP           GROUP PROG
PROG             SEGMENT BYTE PUBLIC 'PROG'
                 ASSUME  CS:PROG, DS:PROG
page
;=============================================================
; Initialize the run time code
;
; Places stack in RAM, all else is in ROM

Init             PROC    FAR
        db       055h, 0aah          ; ID pattern
Blocks  db       1                          ; # of 512 blocks code & data
;
; Entry point:
;
Entry:
        int      012h                       ; get number of Kbytes in AX
        mov      cl, 6                      ; divide by 64
        shl      ax, cl                     ;   to get paragraph address
        sub      ax, 01000h          ; 64 Kbytes for stack
        mov      bx, ss                     ; get callers FAR return
        mov      cx, sp                     ;   address
        mov      ss, ax                     ; now setup
        mov      sp, 0ffffh          ;   the real stack
        push     cx                         ; save the callers return
        push     bx                         ;   address on my stack
;
        call     MainCode            ; execute your program
;
```

```
; If your program exits (via a FAR return), control is given back
;    to the BIOS.
;
            pop         bx                          ;  get the BIOS
            pop         cx                          ;    stack back
            mov         ss, bx                      ;  restore
            mov         sp, cx                      ;    them
            ret                                     ;  & ret to BIOS & boot
Init                    ENDP

;
; MainCode is a simple sample program.  It signs on and returns.
;
MainCode    PROC        NEAR
            lea         si, Message         ; point to the message
MainCodeLoop:
            cld                                     ; insure forward direction
            lodsb                                   ;  get the next chacter
            or          al, al                      ;  is it the end?
            jz          MainCodeExit                ;    YES, exit
            mov         ah, 14                      ;    NO, get TTY output parm
            mov         bx, 7                       ;          screen attribute
            int         010h                        ;          output the character
            jmp         SHORT MainCodeLoop;         & loop for more
MainCodeExit:
            ret                                     ; return to BIOS

Message db 'Main Code', 13, 10, 0  ; signon message

MainCode    ENDP

PROG                    ENDS
                        END
```

APPLICATION NOTE

Number: #AAN-8804 Date: Sep 14, 1988

Title: Ampro SCSI/BIOS Interface Specification.

Product: LB/186, LB/PC, LB/286

Abstract: Ampro has provided a ROM-BIOS enhancement which allows programmers to write hardware-independent software for SCSI bus interfacing. This ap note documents Ampro SCSI/BIOS functions.

1.0 INTRODUCTION

The Ampro SCSI/BIOS adds an additional layer of standardization to SCSI, by providing a high level interface for software which eliminates the need for direct programming of the SCSI bus interface hardware. This standard software interface is accomplished through an extension to the interrupt 13h support normally provided in a PC or AT ROM-BIOS for hard disk access. SCSI drivers and utilities written around the Ampro SCSI/BIOS interrupt 13h extensions will run on any system in which the interrupt 13h extensions, conforming to these standards, are available.

AMPRO COMPUTERS, INCORPORATED
990 Almanor Avenue, Sunnyvale, California 940
Telephone: (408) 522-2100 - FAX (408) 720-130

2.0 STANDARD INTERRUPT 13H BIOS SERVICES

On a standard PC the hard disk operation is performed by requesting disk services using the BIOS software interrupt 13h. To emulate these services with a SCSI drive, the Ampro BIOS extension will service the requests using a SCSI drive. The interface specifications for these services are the same as on a standard PC. For a complete description of the required inputs and returns see the IBM Technical Reference or similar manual. The standard interrupt 13h services are listed here only for completeness.

NOTE: These services emulate a standard hard disk interface as used by DOS and should not be used as an alternate to direct SCSI services.

AH = 00h Reset disk.
AH = 01h Read status of last operation.
AH = 02h Read sectors.
AH = 03h Write sectors.
AH = 04h Verify sectors.
AH = 05h Format track.
AH = 06h Format bad track.
Not implemented, returns bad command status.
AH = 07h Format drive at track.
Not implemented, returns good status.
AH = 08h Return current drive parameters.
AH = 09h Initialize drive characteristics.
Not implemented, returns good status.
AH = 0Ah Read long.
Not implemented, returns bad command status.
AH = 0Bh Write long.
Not implemented, returns bad command status.
AH = 0Ch Seek.
AH = 0Dh Alternate disk reset.
AH = 0Eh Read sector buffer.
Not implemented, returns bad command status.
AH = 0Fh Write sector buffer.
Not implemented, returns bad command status.
AH = 10h Test drive ready.
AH = 11h Recalibrate.
AH = 12h Controller RAM diagnostic.
Not implemented, returns good status.
AH = 13h Drive diagnostic.
Not implemented, returns good status.
AH = 14h Controller internal diagnostic.
Not implemented, returns good status.
AH = 15h Read DASD type.
Not implemented, returns bad command status.

3.0 AMPRO INTERRUPT 13H SCSI/BIOS EXTENSIONS

The AMPRO SCSI/BIOS extensions are accessed in much the same manner as the standard hard disk interface using interrupt 13h. These extended services are requested according to the service request parameter in the AH register.

3.1 Direct SCSI Service Requests

Direct SCSI service requests insulate the application software from the actual hardware of the SCSI bus. This service requires that the caller create a data structure with data and pointers that will be required by the SCSI operation.

Included in this data structure will be the Initiator and Target Identification (ID) Number. Each ID is bit mapped into a byte with a one bit set. To determine the bit to set, use the ID number as the bit number. Bits are numbered 0 through 7, right to left.

Examples:

ID		76543210		hexadecimal		decimal
0	=	00000001	=	01h	=	1
4	=	00010000	=	10h	=	16
7	=	10000000	=	80h	=	128

With SCSI devices that support multiple units, the Logical Unit Number (LUN) is mapped, as a binary value, into the top three bits of a byte. To calculate a LUN byte, take the LUN number (0 through 7) and do a binary shift 5 places to the left with zero padding on the right. One way to do this is to multiply the LUN by 32.

Examples:

LUN		76543210		hexadecimal		decimal
0	=	00000000	=	00h	=	0
3	=	01100000	=	60h	=	96
7	=	11100000	=	E0h	=	224

The direct SCSI service request takes the following form:

AH = (C1h) Execute SCSI command.
Input registers:
ES:BX = Pointer to the SCSI command table.

Return registers:
AX = Status of SCSI operation. 0000h, good ending status, non zero if an error occurred.
CY = Reset if good ending, set if an error occurred.

SCSI command table format is:

Offset	Size	Use
00	byte	Initiator ID (bit mapped).
01	byte	Target ID (bit mapped).
02	byte	Phase map.
03	byte	Vendor unique.
04	dword	(off,seg) pointer to Command.
08	dword	(off,seg) pointer to Data.
0C	dword	(off,seg) pointer to Status.
10	dword	(off,seg) pointer to Message Out.
14	dword	(off,seg) pointer to Message In.

NOTE: The size and content of Command, Data, Message Out and Message In blocks are defined by the Programmers Manual of the SCSI device being accessed. In some services the messages are not used, but the pointer double words must still be present.

Example:

With an Initiator ID of 7, test if the unit at ID 3, LUN 2 is ready

```
DATA            SEGMENT   WORD PUBLIC 'DATA'
SCSIcmd         db   00h                  ; Test Unit Ready
                db   2 SHL 5              ;   LUN 2, in top 3 bits
                db   0, 0, 0, 0           ;   reserved
DataBuf         equ  $                    ; not used by
                                          ;   Test Unit Ready
Status          db   0, 0                 ; returned status
MsgOut          db   0, 0                 ; message out buffer
MsgIn           db   0, 0                 ; message in buffer

CmdTable        equ  $                    ; SCSI command table
Initiator       db   1 SHL 7              ; my ID 7, bit mapped
                db   1 SHL 3              ; Target ID 3, bit mapped
                db   0                    ; Phase map
                db   0                    ; Vendor unique
                dd   SCSIcmd              ; pointer to Command
                dd   DataBuf              ; pointer to Data
                dd   Status               ; pointer to Status
                dd   MsgOut               ; pointer to Message out
                dd   MsgIn                ; pointer to Message in
DATA            ENDS

CODE            SEGMENT   WORD PUBLIC 'CODE'
TestUnitReady   PROC NEAR
     lea        bx, CmdTable              ; offset in BX
     push       es                        ; save 'cause I need
     push       ds                        ; copy DS
     pop        es                        ;   into ES
     mov        ah, 0C1h                  ; execute SCSI command
```

```
        int     013h                    ; request the service
        pop     es                      ; restore ES
        jc      Error                   ; if there was an error
        ...                             ; else continue
TestUnitReady   ENDP
CODE            ENDS
```

3.2 Extended Read/Write Operations

Extended read/write operations are provided for disk operating systems and/or device drivers that support hard disks with more that 1023 cylinders. Although the calling parameters specify head, cylinder and sector number, this service will only function with a SCSI drive.

AH = (C2h) Extended sector(s) read.
Input registers:
DL = Drive number (80h-87h).
DH = Head number (0-15).
SI = Cylinder number (0-65535).
CL = Sector number (1-17).
AL = Number of sectors (01h-80h).
ES:BX Points to buffer.

Return registers:
AH = Error code if CY = 1.
CY = Reset is good ending.
CY = Set if error.

Example:

Read 1 sector from the first hard disk starting at head 0, cylinder 0, sector 1.

```
        mov     dl, 080h                ; 1st hard disk
        mov     dh, 0                   ; head 0
        mov     si, 0                   ; cylinder 0
        mov     cl, 1                   ; sector 1
        mov     al, 1                   ; number of sectors
        lea     bx, Buffer              ; offset of buffer
        push    es                      ; save 'cause I need
        push    ds                      ;   copy DS
        pop     es                      ;     into ES
        mov     ah, 0C2h                ; extended read
        int     013h                    ; request the service
        pop     es                      ; restore
        jc      Error                   ; if there was an error
        ...                             ; else continue
```

AH = (C3h) Write sector(s).
Input registers:
DL = Drive number (80h-87h).
DH = Head number (0-15).
SI = Cylinder number (0-65535).

CL = Sector number (1-17).
AL = Number of sectors (01h-80h).
ES:BX Points to buffer.

Return registers:
AH = Error code if CY = 1.
CY = Reset if good ending.
CY = Set if error ending.

Example:

Write 1 sector to the first hard disk starting at head 0, cylinder

```
mov     dl, 080h                ; 1st hard disk
mov     dh, 0                   ; head 0
mov     si, 0                   ; cylinder 0
mov     cl, 1                   ; sector 1
mov     al, 1                   ; number of sectors
lea     bx, Buffer              ; offset of buffer
push    es                      ; save 'cause I need
push    ds                      ;   copy DS
pop     es                      ;     into ES
mov     ah, 0C3h                ; extended write
int     013h                    ; request the service
pop     es                      ; restore
jc      Error                   ; if there was an error
...                             ; else continue
```

3.3 SCSI Maintenance Services

The following interrupt 13h extensions are used to verify that the SCSI is operational, get the SCSI/BIOS version, get pointers to its disk mapping tables and reset the SCSI bus.

3.3.1 SCSI Check

To determine if the SCSI bus is operational, the version of the AMPRO SCSI/BIOS extension and the Initiator ID (SCSI ID of the system), the following service request is used:

AH = (C0h) Check if SCSI is valid and return the SCSI/BIOS version and Initiator ID.

Input registers:
None.

Return registers:
AX = FFFFh if SCSI is operational.
CY = Set.
If AX = FFFFh:
CH = Minor version number in binary.
CL = Major version number in binary.
DL = Initiator ID in bit mapped format.

Example:

```
mov     ah, 0C0h                ; SCSI check
int     013h                    ; request the service
cmp     ax, 0FFFFh              ; is SCSI valid ?
jz      NoScsi                  ;   NO
mov     [ ScsiVer ], cx         ;   YES, store SCSI Ver
mov     [ MyId ], dl            ;     & my ID
...                             ; continue
```

3.3.2 Return Pointer to SCSI Data

This service returns a pointer to the Disk Physical Table (DPT) and a pointer to the SCSI Device Map Table (DMT). The DPT is terminated by a NULL word. The second DMT contains the IDs and LUNs of the disks described in the DPT. Both tables will contain the same number of entries. The first entry in both tables corresponds to the first SCSI drive, the second if present, to the second SCSI drive. This in formation is provided for software, such as device drivers.

AH = (CEh) Return pointers to the address of the DPT and DMT.

Input registers:

None.

Return registers:

DS:DI = A pointer to double word (off,seg) address of the DPT.

DS:SI = A pointer to double word (off,seg) address of the DMT.

DPT format is:

Offset	Size	Use.
		1st drive
0000	word	Number of heads.
0002	word	Sectors per track.
0004	word	Number of cylinders.
		2nd drive if present
		3rd, ...
xxxx	word	(0000) terminator.

DMT format is :

Offset	Size	Use
		1st drive
0000	byte	Drives SCSI ID (bit mapped)
0001	byte	Drives SCSI LUN (in the top 3 bits)
		2nd drive if present
		3rd, ...

Example:

```
mov     ah, 0CEh                ; return SCSI pointers
int     013h                    ; request the service
...                             ; DS:DI -> DPT
...                             ; DS:SI -> DMT
```

3.3.3 SCSI Reset

This service resets the SCSI bus and SCSI devices connected to it.

AH = (CFh) Reset the SCSI bus.

Input registers:

None.

Return registers:

Nothing.

Example:

```
mov     ah, 0CFh                ; reset SCSI
int     013h                    ; request the service
...                             ; SCSI has been reset
                                ;  if present
```

APPLICATION NOTE

NUMBER: AAN-8805 REVISION: C DATE: 2/12/92

TITLE: Using the onboard configuration memory EEPROM for custom applications.

PRODUCT: All Ampro CoreModules and LittleBoards except the LB/PC.

SUMMARY: An EEPROM device on Ampro CPU's is used to store configuration information used during powerup or system reset. Space has also been reserved within the EEPROM for OEM use. This application note discusses the data structure within the EEPROM, and the ROM-BIOS functions that should be used to access or modify the EEPROM data from custom software.

INTRODUCTION

A standard PC/AT uses battery backed RAM within the onboard real time clock device to store system configuration information in addition to the time and date. This information is used by the ROM-BIOS during system startup to determine how to initialize the hardware. Ampro CPU products provide additional nonvolatile information storage in the form of an EEPROM (Electrical Erasable Programmable Read Only Memory). In addition to the contents of the standard CMOS nonvolatile RAM (NOVRAM), the stored EEPROM information is used by the ROM-BIOS during startup for Ampro-specific system initialization.

The contents of both the standard AT NOVRAM (within the clock) and the non-standard EEPROM device can be edited "manually" using the SETUP.COM utility, which is included on the Utilities diskette provided with all Ampro CPU products. The EEPROM may be written up to 10,000 times.

Some custom applications need to read and/or modify the parameters in the EEPROM. In addition, space in the EEPROM is available for the storage of data, parameters, serial numbers, passwords, etc., for use by the OEM's application. The ROM-BIOS functions that are used to access and reprogram the EEPROM contents have been standardized for all CPU products, and are documented here.

Please note the following:

- At system startup, data is read from the EEPROM and stored in RAM by the ROM-BIOS.
- Any writes to the EEPROM that change the system startup parameters (used by the ROM-BIOS) will not take effect until the system has been reset (power on/off cycle, hardware reset, Ctrl-Alt_Del).
- The Ampro EEPROM read/write ROM-BIOS services (discussed below) maintain an accurate EEPROM write count and an EEPROM data checksum. It is therefore essential that the ROM-BIOS services, or an appropriate Ampro utility (e.g., SETUP.COM), be used to alter the EEPROM contents.
- The format of the portion of the EEPROM that is used by the ROM-BIOS may change in the future.

AMPRO®
AMPRO COMPUTERS, INCORPORATED
990 Almanor Avenue, Sunnyvale, California 94086
Telephone: (408) 522-2100 - FAX (408) 720-1305

ROM-BIOS FUNCTIONS FOR EEPROM READ/WRITE

Ampro has provided an extension to the int 13h service routines to provide read/write access to the EEPROM. There are two sets of services. The first accesses the EEPROM relative to the start of the User EEPROM data area, starting at word 20h. The second accesses the entire EEPROM from offset 0h to offset 3fh. This second service is intended for use by Ampro to perform maintenance on the EEPROM (such as with the utility SETUP.COM).

BIOS Interface to User EEPROM Data Area

ROM-BIOS service functions 00h, 02h and 03h, and their calling formats are intended for access to the user portion of the EEPROM. When a service is performed, these functions will use the word address as an offset into the User EEPROM Data Area. Thirty-two 16-Bit words, at offsets 0 through 1Fh, are available for user data. The User Data Area starts at word 20h in the EEPROM.

```
FUNCTION 00H --- READ USER WORD

Entry:          AX = cc00h
                DX = address (0-1fh)
Returns:        cy set    AH = error code
                cy reset  AX = word from User EEPROM


Example:
    mov         ax, 0cc00h      ; read word
    mov         dx, 5           ;   offset of word
    int         013h            ;   request service

Result:
    Word 5 of the user data area would be returned in AX.
    This word is at EEPROM offset 32 + 5 = 37 (25h).
```

```
FUNCTION 02H --- USER READ WORDS

Entry:          AX = cc02h
                CX = word count (0-20h)
                DX = starting address (0-1fh)
                ES:BX = buffer address
Returns:        cy set    AH = error code
                cy reset  count words from USER EEPROM in buffer

Example:
    mov         ax, 0cc02h      ; read words
    mov         cx, WORD_COUNT  ;   count of words to read
    mov         dx, 5           ;   starting offset of read
    push        es              ;   get
    mov         bx, ds          ;     the data seg
    mov         es, bx          ;     into ES
    lea         bx, Buffer      ;   ES:BX -> read buffer address
    int         013h            ;   request service
    pop         es              ; restore ES
WORD_COUNT      equ  16
Buffer          db   WORD_COUNT DUP (?)
Result:
     16 words, starting at word number 5 of the 'User Data
     Area', would be read into the buffer.  These would be
     words starting at EEPROM offset 32 + 5 = 37 (25h).
```

```
FUNCTION 03H --- WRITE USER WORDS, UPDATE CHECKSUM

Entry:      AX = cc03h
            CX = word count (0-20h)
            DX = starting address (0-1fh)
            ES:BX = write buffer address

Returns:    cy set    AH = error code
            cy reset  count words written to User EEPROM

Example:
    mov     ax, 0cc03h      ; user write words
    mov     cx, WORD_COUNT  ;   count of words to write
    mov     dx, 5           ;   starting offset of write
    push    es              ;   get
    mov     bx, ds          ;     the data seg
    mov     es, bx          ;     into ES
    lea     bx, Buffer      ;   ES:BX -> write buffer address
    int     013h            ; request service

Result:
     The 16 words in the write buffer would be written to
     the User Data Area, starting at word number 5.  These
     would be words starting at EEPROM offset 32 + 5 = 37 (25h).
```

BIOS Interface to All EEPROM Data Areas

Functions 0Ah, 0Ch, 0Eh, and 0Fh, and their calling formats, are used to access all data areas within the EEPROM. When a service is performed, these functions will use the word address as an offset from the start of the EEPROM data area. A service specifying word 5 would operate on the word at offset 5. NOTE: The first 32 words -- i.e., word 0 through 31 (1Fh) -- are reserved for Ampro BIOS and utilities use.

```
FUNCTION 0AH --- READ WORD

Entry:      AX = cc0ah
            DX = address (0-3fh)
Returns:    cy set    AH = error code
            cy reset  AX = word from EEPROM

Example:
    mov     ax, 0cc0ah      ; read word
    mov     dx, 5           ;   offset of word
    int     013h            ;   request service

Result:
    Word number 5 of the EEPROM would be returned in AX.
```

```
FUNCTION 0CH --- READ WORDS

Entry:      AX = cc0ch
            CX = word count (0-40h)
            DX = starting address
            ES:BX = buffer address
Returns:    x words in buffer

Example:
    mov     ax, 0cc0ch      ; read words
    mov     cx, WORD_COUNT  ;   count of words to read
    mov     dx, 5           ;   starting offset of read
    push    es              ;   get
    mov     bx, ds          ;     the data seg
    mov     es, bx          ;     into ES
    lea     bx, Buffer      ;   ES:BX -> read buffer address
    int     013h            ;   request service
```

```
    pop       es              ; restore ES
WORD_COUNT    equ  16
Buffer        db   WORD_COUNT DUP (?)

Result:
    16 words of the EEPROM, starting at word 5, would be
    read into the buffer.
```

```
FUNCTION 0EH --- WRITE WORDS, UPDATE CHECKSUM
```

```
Entry:        AX = cc0eh
              CX = word count (0-40h)
              DX = starting address
              ES:BX = write buffer address
Returns:      cy set    AH = error code
              cy reset  count words written to EEPROM

Example:
    mov       ax, 0cc0eh      ; write words
    mov       cx, WORD_COUNT  ;   count of words to write
    mov       dx, 5           ;   starting offset of write
    push      es              ;   get
    mov       bx, ds          ;     the data seg
    mov       es, bx          ;     into ES
    lea       bx, Buffer      ;   ES:BX -> write buffer address
    int       013h            ; request service

WORD_COUNT    equ  16
Buffer        db   WORD_COUNT DUP (?)

Result:
    The 16 words in the write buffer would be written
    to the EEPROM, starting at word number 5.
```

```
FUNCTION 0FH --- CHECK CHECKSUM
```

```
Entry:        AX = cc0fh
Returns:      cy set    AH = error code
              cy reset  CHECKSUM OK

Example:
    mov       ax, 0cc0fh      ; write words
    int       013h            ; request service
Result:
    cy set    Checksum Bad
    cy reset  Checksum Good
```

Error Codes

Upon the completion of an Ampro ROM-BIOS EEPROM service request, the carry flag will be set if there was an error. If the carry is set, an error code will be returned in AH.

```
Returned Error Codes
     AH =     1, illegal address requested.
     AH =     2, bad word count, would access more than max
              address.
     AH =     3, error accessing EEPROM.
     AH =     4, checksum error.
     AH =     -1, bad function call.
```

EEPROM DATA FORMAT

The device used for the EEPROM is a National NMC9346 (or equivalent) serial EEPROM. This EEPROM contains 1024 bits of writable data space. It is organized as 64 16-bit words. Half of these, the first 32 words, have been reserved by Ampro for system use. The second 32 words are available for the user. These data words are defined for Ampro products in general. Refer to the technical manual for each product for additional information. Specific products may not use some of the bits.

WORD 0 -- CHECKSUM, SCSI ID, ENABLE BIT MAP FOR PORT 372H:

```
bit definition:
f e d c b a 9 8 7 6 5 4 3 2 1 0
                            | 0 - enable byte wide socket 0
                            | 1 - disable byte wide socket 0
                          0 --- enable byte wide socket 1
                          1 --- disable byte wide socket 1
                        0 ----- enable serial port 1
                        1 ----- disable serial port 1
                      0 ------- enable serial port 2
                      1 ------- disable serial port 2
                    0 --------- enable parallel port
                    1 --------- disable parallel port
                  x x x ------- SCSI ID 000 to 111
x x x x x x x x ----------------- 8 bit checksum
```

WORD 1 -- SCSI TARGET ID AND LUN FOR 2 DEVICES:

```
bit definition:
f e d c b a 9 8 7 6 5 4 3 2 1 0
                        | | x x x - target id first device
                        | 0 ------- first device defined
                        x --------- undefined/reserved AMPRO
                  x x x ----------- lun id for first device
              x x x --------------- target id second device
            0 ----------------------- second device defined
          x ------------------------- undefined/reserved AMPRO
x x x ------------------------------- lun id for second device
```

WORD 2 -- SCSI TARGET ID AND LUN FOR 2 DEVICES:

```
bit definition:
f e d c b a 9 8 7 6 5 4 3 2 1 0
                        | | x x x - target id third device
                        | 0 ------- third device defined
                        x --------- undefined/reserved AMPRO
                  x x x ----------- lun id for third device
              x x x --------------- target id fourth device
            0 ----------------------- fourth device defined
          x ------------------------- undefined/reserved AMPRO
x x x ------------------------------- lun id for fourth device
```

WORD 3 -- HARD DRIVE MAP:

```
bit definition:
f e d c b a 9 8 7 6 5 4 3 2 1 0
| | | | | | | | | | | | x x x - first drive number 0 - 7
| | | | | | | | | | | | 0 ------- hd controller device
| | | | | | | | | | | | 1 ------- scsi device
| | | | | | | | | | x x x---------- second drive number 0 - 7
| | | | | | | | | | 0 -------------- hd controller device
| | | | | | | | | | 1 -------------- scsi device
| | | | | | | x x x ---------------- third drive number 0 - 7
| | | | | | | 0 ----------------------hd controller device
| | | | | | | 1 ---------------------- scsi device
| | | | x x x ------------------------ fourth drive number 0 - 7
| | | | 0 ---------------------------- hd controller device
| | | | 1 ---------------------------- scsi device
```

If controller is an AT HDC device then drive number is 0 or 1 only.

WORD 4 -- FLAGS 1:

```
bit definition:
f e d c b a 9 8 7 6 5 4 3 2 1 0
| | | | | | | | | | | | | 0 0 - floppy 0      = boot device
| | | | | | | | | | | | | 0 1 - ssd           = boot device
| | | | | | | | | | | | | 1 0 - hard device 0 = boot device
| | | | | | | | | | | | | 1 1 - undefined/reserved AMPRO
| | | | | | | | | | | | 0 ------- bus speed fast, cpu speed
| | | | | | | | | | | | 1 ------- bus speed slow, half cpu speed
| | | | | | | | | | | 1 --------- SCSI disk interface disable
| | | | | | | | | | | 0 --------- SCSI disk interface enable
| | | | | | | | | | 0 ----------- floppy slow mode, cpu is set to half speed during i/o
| | | | | | | | | | 1 ----------- floppy fast mode, cpu speed is not changed
| | | | | | | | | 0 ------------- Byte Wide 2 enabled
| | | | | | | | | 1 ------------- Byte Wide 2 disabled
| | | | | | | | 0 --------------- Byte Wide 3 enabled
| | | | | | | | 1 --------------- Byte Wide 3 disabled
| | | | | | | 1 1 ----------------- Console out = video, standard
| | | | | | | 0 0 ----------------- Console out = serial 1
| | | | | | | 0 1 ----------------- Console out = serial 2
| | | | | | | 1 0 ----------------- Console out = none, int 10h will point to iret
| | | | | | 1 1 ------------------- Console in  = keyboard, standard
| | | | | | 0 0 ------------------- Console in  = serial 1
| | | | | | 0 1 ------------------- Console in  = serial 2
| | | | | | 1 0 ------------------- Console in  = none, int 16h will return no char available
| | | | | |                                       irq1 will service the keyboard controller
| | | | | |                                       but no character will be put in keyboard
| | | | | |                                       buffer.
| | | | 0 ------------------------- Serial 1 extended baud rate, to support 19200 thru 57600 baud
| | | 0 --------------------------- Serial 2 extended baud rate, to support 19200 thru 57600 baud
| | 1 ----------------------------- Serial 1 Deny Award bios or int 14h access to hardware
| 1 ------------------------------- Serial 2 Deny Award bios or int 14h access to hardware
0 --------------------------------- Serial console output hardware handshake enabled
```

WORD 5 -- FLAGS 2:

```
bit definition:
f e d c b a 9 8 7 6 5 4 3 2 1 0
| | | | | | | | | | | | | | 1 - Enable Hot key access to bios setup
| | | | | | | | | | | | | | 0 - Disable Hot key access to bios setup
x x x x x x x x x x x x x x --- undefined/reserved AMPRO
```

WORD 6 -- FLAGS 3:

```
bit definition:
f e d c b a 9 8 7 6 5 4 3 2 1 0
x x x x x x x x x x x x x x x x - undefined/reserved AMPRO
```

WORD 7 -- WRITE COUNT:

```
bit definition:
f e d c b a 9 8 7 6 5 4 3 2 1 0
x x x x x x x x x x x x x x x x - count 0 - 65535
```

Each time the EEPROM is written, the count is incremented.

WORD 8 -- SERIAL CONFIG:

```
bit definition:
f e d c b a 9 8 7 6 5 4 3 2 1 0
|   |   |   |   |   |   |   x x x x x x x x - com1 parameters
|   |   |   |   |   |   |   |   |   |   |   0 0 - 5 bit data length
|   |   |   |   |   |   |   |   |   |   |   0 1 - 6 bit data length
|   |   |   |   |   |   |   |   |   |   |   1 0 - 7 bit data length
|   |   |   |   |   |   |   |   |   |   |   1 1 - 8 bit data length
|   |   |   |   |   |   |   |   |   |   0 ----- 1 stop bit
|   |   |   |   |   |   |   |   |   |   1 ----- 2 stop bit
|   |   |   |   |   |   |   |   |   x 0 ------- no parity
|   |   |   |   |   |   |   |   |   0 1 ------- odd parity
|   |   |   |   |   |   |   |   |   1 1 ------- even parity
|   |   |   |   |   |   |   0 0 0 ----------- 110 baud
|   |   |   |   |   |   |   0 0 1 ----------- 150 baud
|   |   |   |   |   |   |   0 1 0 ----------- 300 baud
|   |   |   |   |   |   |   0 1 1 ----------- 600 baud
|   |   |   |   |   |   |   1 0 0 ----------- 1200 baud
|   |   |   |   |   |   |   1 0 1 ----------- 2400 baud
|   |   |   |   |   |   |   1 1 0 ----------- 4800 baud
|   |   |   |   |   |   |   1 1 1 ----------- 9600 baud
x x x x x x x x ----------------- com2 parameters
|   |   |   |   |   | 0 0 ----------------- 5 bit data length
|   |   |   |   |   | 0 1 ----------------- 6 bit data length
|   |   |   |   |   | 1 0 ----------------- 7 bit data length
|   |   |   |   |   | 1 1 ----------------- 8 bit data length
|   |   |   |   | 0 ------------------- 1 stop bit
|   |   |   |   | 1 ------------------- 2 stop bit
|   |   |   | x 0 --------------------- no parity
|   |   |   | 0 1 --------------------- odd parity
|   |   |   | 1 1 --------------------- even parity
0 0 0 ------------------------- 110 baud
0 0 1 ------------------------- 150 baud
0 1 0 ------------------------- 300 baud
0 1 1 ------------------------- 600 baud
1 0 0 ------------------------- 1200 baud
1 0 1 ------------------------- 2400 baud
1 1 0 ------------------------- 4800 baud
1 1 1 ------------------------- 9600 baud
```

WORD 9 -- SETUP.COM VERSION NUMBER, SCSI READ/WRITE RETRIES:

bit definition:

```
f e d c b a 9 8 7 6 5 4 3 2 1 0
| | | | | | | | x x x x x x x x - setup version number
| | | | x x x x ---------------- read/write retries
x x x x ------------------------ undefined/reserved AMPRO
```

WORD 10 -- SCSI TARGET ID AND LUN FOR 2 DEVICES:

bit definition:

```
f e d c b a 9 8 7 6 5 4 3 2 1 0
| | | | | | | | | | | | x x x - target id fifth device
| | | | | | | | | | | 0 ------- fifth device defined
| | | | | | | | | | x --------- undefined/reserved AMPRO
| | | | | | | | x x x ---------- lun id for fifth device
| | | | x x x ---------------- target id sixth device
| | | 0 ---------------------- sixth device defined
| | x ------------------------ undefined/reserved AMPRO
x x x -------------------------- lun id for sixth device
```

WORD 11 -- SCSI TARGET ID AND LUN FOR 2 DEVICES:

bit definition:

```
f e d c b a 9 8 7 6 5 4 3 2 1 0
| | | | | | | | | | | | x x x - target id seventh device
| | | | | | | | | | | 0 ------- seventh device defined
| | | | | | | | | | x --------- undefined/reserved AMPRO
| | | | | | | | x x x ---------- lun id for seventh device
| | | | x x x ---------------- target id eighth device
| | | 0 ---------------------- eight device defined
| | x ------------------------ undefined/reserved AMPRO
x x x -------------------------- lun id for eighth device
```

WORD 12 -- HARD DRIVE MAP:

bit definition:

```
f e d c b a 9 8 7 6 5 4 3 2 1 0
| | | | | | | | | | | | x x x - fifth drive number 0 - 7
| | | | | | | | | | | 0 ------- hd controller device
| | | | | | | | | | | 1 ------- scsi device
| | | | | | | | | x x x---------- sixth drive number 0 - 7
| | | | | | | | 0 ---------- hd controller device
| | | | | | | | 1 ---------- scsi device
| | | | | x x x ---------------- seventh drive number 0 - 7
| | | | 0 ---------------------- hd controller device
| | | | 1 ---------------------- scsi device
| | | x x x ------------------------ eighth drive number 0 - 7
| | 0 ---------------------------- hd controller device
| | 1 ---------------------------- scsi device
```

If controller is an AT HDC device then drive number is 0 or 1 only.

WORDS 13-31 -- RESERVED FOR FUTURE AMPRO USE

APPLICATION NOTE

NUMBER: AAN-9002 **REVISION:** A DATE: 1/05/90

TITLE: Third party support software for serial port and modem operation.

PRODUCT: LB/PC, LB/286, LB/386, MM/SSP, MM/Modem

SUMMARY: A discussion of several third party programs, utilities, and libraries which support PC compatible serial port and modem operation.

TYPES AND FEATURES OF SERIAL SUPPORT SOFTWARE

This application note discusses several third party programs, utilities, and function libraries which can be used to control the PC compatible serial ports of a Little Board. In addition, control functions for use with the MiniModule/Modem are included.

Serial and modem support software packages generally fall into one of three categories:

- **Memory resident drivers** -- An installable device driver or TSR ("terminate-stay-resident") program is used to load serial and modem control functions into system memory, for use by programs which are subsequently loaded. These drivers are usually accessed by the application program through a software interrupt, typically an extension of the ROM-BIOS interrupt 14h services. As a result, these drivers are *language independent*. "Bindings" (for various languages) which allow the application program to access the functions provided by the memory resident driver are generally included (in source form) with these driver packages.

- **Linkable function libraries** -- The serial and modem control functions are in the form of a library of modules written for a particular language (or with language binders) which must be linked into your application program at compile time. Because these must be linked into a program written in a particular high level language (e.g. C, Basic, Pascal), this type of software support is *language specific*. In fact, only one or two specific compilers are usually supported by this type of library of functions; however, source code is often included, allowing the libraries to be modified for compilers not supported.

- **Communications programs** -- There are many sources of general purpose, ready-to-run communications programs capable of transferring files and data over serial ports or modems. Script language support, available with many communications programs, makes it possible to use these programs to perform all required serial or modem control functions in certain applications. These script languages process commands in simple text files, which generally resemble the DOS batch language in their style and usage; they often have BASIC-like extensions for testing results, waiting for specific byte strings, looping, incrementing counters, inputting data, formatting the video display, etc.

Regardless of type, the serial and modem support packages offer features such as: interrupt driven or polled buffered input and output data; XMODEM (and other) protocol data transfers; support for multiple simultaneous serial or modem channels; interrupt sharing in both "OR" (as on the MiniModule/SSP) and standard PC "XOR" style; error checking and handling; port initialization; both software (XON/XOFF) and hardware (DTR/DSR) handshaking; arbitrary serial port I/O base address selection; arbitrary serial port interrupt (IRQ) selection.

AMPRO®
AMPRO COMPUTERS, INCORPORATED
990 Almanor Avenue, Sunnyvale, California 94086
Telephone: (408) 522-2100 - FAX (408) 720-1305

The remainder of this application note briefly discusses several specific third party software packages, relative to the above overview of the general types and features of serial and modem support software. They can be used to program the functions of the PC compatible serial ports on the Ampro Little Boards and MiniModule/SSP, and can be used to control the modem functions of the MiniModule/Modem. The information shown is extracted from the software vendors' technical manuals and is believed to be accurate. Please contact the specific software vendor for further details.

MID and ASYNC EXECUTIVE

Multiline Interrupt Driver (MID) and Async Executive are two examples of *memory resident drivers* which provide a full set of the serial port (and modem) control and data transfer features listed above. In terms of features, MID and Async are quite similar. Both offer fully interrupt driven, buffered input and output for multiple PC compatible serial ports. Both are intended to provide fast, efficient multiport serial I/O. Both drivers are implemented as extensions to the ROM-BIOS interrupt 14h serial support functions, and therefore can be used with standard ROM-BIOS programming methods. As with all *memory resident drivers*, both of these drivers can be accessed directly from assembly language, or from high level languages via a binding. Both MID and Async Executive support multiple simultaneous serial port interrupt driven I/O. Both allow the use of any I/O base address, and any IRQ choice. In addition, both support true interrupt sharing (OR-ing), as used on the Ampro MiniModule/SSP.

Features

- Fully buffered and interrupt driven on both input and output
- Optimized for low overhead and highly efficient operation
- Supports multiple serial ports (8) per interrupt, with OR-ing
- Supports multiple serial ports on multiple interrupts
- Simultaneous bidirectional I/O on all ports
- Character I/O or block I/O
- Character I/O emulates standard ROM-BIOS support (interrupt 14h)
- Block reads and writes provide high data throughput
- Software (XON/XOFF) or hardware (DTR/DSR) handshaking
- Async Executive provides XMODEM file transfer support

For further information, contact:

MID
Parasoft
441 South Park Street
Springfield, OR 97477
Phone: (503) 741-1600

Async Executive
Cirrus Software
PO Box 51924
Palo Alto, CA 94303-1924
Phone: (415) 949-1470

GREENLEAF COMM LIBRARY

The Greenleaf Comm Library as a *linkable function library* which offers an extensive and complete set of serial port control and communication functions that can be integrated into application programs written in the C programming language. All of the usual serial port control and support functions (described above) are included, as indicated in the following features list. Greanleaf currently supports Lattice C and Microsoft C, but source code is included so the libraries can be modified for other C compilers. Support for multiple serial ports per interrupt (interrupt sharing) is included, but it is based on the presence of a hardware register in which the port(s) interrupting can be read by interrupt service software. The interrupt service routine would require modification to allow its use with interrupt sharing as implemented on the MiniModule/SSP.

Features:

- Supports up to 32 serial ports
- Hardware interrupt services
- Circular transmit and receive buffers
- Hardware (RTS/CTS) and software (XON/XOFF) handshaking
- Port initialization
- Hayes compatible modem support
- XMODEM, YMODEM, and KERMIT file transfer support

For further information, contact:

Greenleaf Software
16479 Dallas Parkway, Suite 570
Dallas, TX 75248
Phone: (214) 248-2561

MIRROR

Mirror is a *communications program* with extensive support for PC compatible serial port and modem based communications and data transfer. Like most such programs, Mirror can be operated from a set of menus (by a user); or it can be operated automatically using its script language processor. All of the common file transfer protocols are supported, including XMODEM, YMODEM, Crosstalk, Kermit, and CompuServe, resulting in error free data transfer. Terminal emulations are provided for DEC, Televideo, Wyse, and others.

Mirror supports operation of only one single serial port (or modem) at a time. Any I/O port base address, and any hardware interrupt (IRQ) can be specified. During operation, the active port to be controlled by Mirror can be changed, but only one port is supported at a time.

MIRROR's script-based programming language (including the PRISM script extension option) makes MIRROR useful in certain types of dedicated and embedded applications. The script language can be used to design custom user interfaces or automate complex data communication tasks. Among other functions, the script language provides looping mechanisms (FOR/NEXT, etc.), graphics display support, prompting for inputs, waiting for specified events, password protection functions, and context sensitive help.

Another option available for use with MIRROR is an MNP 5 protocol package. The MNP 5 option emulates MNP 5 hardware protocol in software, thereby allowing a modem such as the Ampro MiniModule/Modem (without MNP 5 support) to communicate with a modem having MNP 5 capability. The MNP 5 protocol also provides data compression, resulting in faster transmissions and lower phone costs.

For further information, contact:

SoftKlone
327 Office Plaza Drive
Tallahassee, FL 32301
Phone: (904) 877-9763

APPLICATION NOTE

NUMBER: AAN-9003 **REVISION:** A DATE: 1/5/90

TITLE: Using the Ampro ROM-BIOS "OEM Hooks".

PRODUCT: LB/PC, LB/286, SB/286, LB/386

SUMMARY: Description and definition of the "OEM Hooks" which are present in the Ampro ROM-BIOS, and how to generate "OEM extensions" and "OEM patches" based on these hooks.

INTRODUCTION

To allow a number of powerful and flexible changes to be made in system operation by OEMs, Ampro has extended the ability of the ROM-BIOS to execute OEM supplied code prior to system booting. However, the added "OEM hooks" allow control over functions which would otherwise not be possible without modifying the ROM-BIOS itself.

At various points during the system initialization and the power on self test (POST) process, which occurs on system powerup or reset, the Ampro ROM-BIOS checks for the presence of either of two types of OEM modifications: *OEM extensions* and *OEM patches*, which are defined (briefly) as follows:

- OEM EXTENSIONS -- As mentioned above, at specific points during the initialization and powerup self test sequence, the ROM-BIOS scans for the presence of code containing OEM ROM-BIOS extensions. The technique is similar to that of the PC compatible 55AAh ROM-BIOS extensions, as described in Ampro application note AAN-8702. The area from C8000h to the base of the ROM-BIOS (F0000h for the LB/286, SB/286, and LB/386; F8000h for the LB/PC) is scanned on 1K boundaries. The formatting of the OEM extension is similar to that defined for normal PC compatible 55AAh ROM-BIOS extensions, including length, checksum, and initialization entry points. The only difference between these OEM extensions and standard ROM-BIOS extensions is the 2-byte ID. IDs in the range of 55A0h through 55A9h are used instead of the normal 55AAh. The unique 2-byte OEM extension ID is used to indicate at what point in the ROM-BIOS powerup/reset sequence the particular OEM extension is to be executed.

  These OEM extensions can be placed in several locations in the system: in the on-board byte-wide sockets; in socket "E" of the Expansion SSD Board; or on a board plugged into the PC bus. Since the on-board sockets on the LB/286, SB/286, and LB/386, and the entire SSD Expansion Board, can be enabled and disabled under software control, there is a scan priority based on where the OEM extension is located. The priority is (highest to lowest): on the bus; on the SSD Expansion Board (socket E); in the on-board sockets. Also, if an *OEM patch* is present (described below) along with an OEM extension (55Axh), the OEM extension rather than the patch code is executed; i.e., OEM extensions have priority over OEM patches.

AMPRO®
AMPRO COMPUTERS, INCORPORATED
990 Almanor Avenue, Sunnyvale, California 94086
Telephone: (408) 522-2100 - FAX (408) 720-1305

- **OEM PATCHES** -- A second means to modifying the powerup/reset startup sequence is based on a table of 16 pointers beginning at F000:30h, in the beginning of the ROM-BIOS EPROM (except on the LB/PC). For each OEM hook, the ROM-BIOS will execute code located at the memory address patched into the ROM-BIOS EPROM at the location corresponding to the particular OEM hook, if the default value in the table (FFFFh) has been altered. This allows the OEM code to be located directly within the ROM-BIOS EPROM. Note that if the corresponding *OEM extension* (described above) is present, then the corresponding *OEM patch* is not used; i.e., OEM extensions have priority over OEM patches. Refer to the section on "Patching the BIOS EPROMs", for details on how the patching is done. Note that OEM patches are not available on the LB/PC.

OEM HOOK DEFINITIONS

The specific OEM hooks provided in the Ampro ROM-BIOS, and their possible uses, are discussed below. They are listed in order of occurrence. Refer to the section "Patching the BIOS ROM" for further details on installing OEM patches, and to Ampro application note AAN-8702 for further details on the structure of ROM-BIOS extensions. Examples of OEM extensions are available from Ampro Technical Support on diskette. Remember, OEM patches are not available on the LB/PC.

- **Early Reset** Extension ID: none Patch location: F000:30h
  Description: Occurs very close to reset time, prior to POST. Hardware has not been initialized. Memory may not have been tested. The only valid registers are CS:IP, no hardware assumptions can be made. Note that this hook is available as an OEM patch only. If present, the vector will be entered by JMP NEAR with the return address in BX, OEM code must reside in the same code segment, no CALLS can be executed from OEM code. Use JMP BX to return to the ROM-BIOS. A distinction between processor reset (shutdown mode, i.e. protected mode, reset) and system reset is made.

- **Late Reset** Extension ID: 55A0h Patch location: F000:32h
  Description: Occurs at late reset time. Video has not been initialized, only the first 64k of memory has been tested and initialized, and some default interrupt vectors have been installed. Typical use would be to initialize custom hardware functions (e.g. a watch dog timer).
  LB/286, SB/286, LB/386: POST tests 0 through 21 have been completed, SS:SP are valid, interrupt vectors have not been installed.
  LB/PC: This routine is called before the equipment scan and memory test, so it's early in the power up sequence. Don't change any interrupt vectors or make any software interrupt calls to the ROM-BIOS (i.e. video, keyboard, serial, or printer).

- **Video Initialization** Extension ID: 55A1h Patch location: F000:34h (not for LB/PC)
  Description: Typical use would be to configure special video cards to emulate MONO or CGA video cards.
  LB/286, SB/286, LB/386: Occurs at the beginning of POST video initialization, just prior to POST 23 (see table below), prior to video initialization. Default interrupt vectors have been installed. After this scan, the ROM-BIOS tests for MONO video memory; if none is found, CGA is assumed; mode 7 is set if MONO card is present, or mode 3 if CGA. If there is video BIOS present in the EGA/VGA video BIOS ROM area from C0000h to C8000h, INT 10h is patched to point to an IRET, the video BIOS is tested, and a far call is made to the initialization entry.
  LB/PC: This routine is called *after* all display devices (i.e. MONO/CGA/EGA/VGA) are initialized and installed.

- **OEM Signon** Extension ID: 55A4h Patch location: F000:3Ah
  Description: OEM signon display. The behavior of this OEM hook differs, depending on whether it is implemented as an extension or as a patch. In either case, if this hook is utilized, this information will display instead of the normal Ampro signon message.
  OEM extensions: When provided as an extension, can contain code to generate custom video interface routines or display custom signon messages. The ROM-BIOS INT 10h services may be used to display desired messages.
  OEM patch: When provided as a patch, this entry is a pointer to an OEM signon message string, unlike the other OEM patches which contain vectors to code. The first byte of the string is the length of the string to be displayed.
  LB/286, SB/286, LB/386: Video has been initialized and POST test 23 completed. A short single beep has been sounded to indicate the first phase of post has been completed.
  LB/PC: Video has been initialized.

- **Pre-55AA** Extension ID: 55A2h Patch location: F000:36h
  Description: Occurs after POST video initialization, but prior to scanning for standard PC 55AAh ROM-BIOS extensions. Floppy, AT hard disk (not for LB/PC), serial cards, parallel printer ports, and joy stick ports have been initialized and have been logged into the equipment flags. Default interrupt vectors have been installed and will not be modified by the ROM-BIOS, however 55AAh ROM-BIOS extensions (if present) may change them.

- **Pre-Boot** Extension ID: 55A3h Patch location: F000:38h
  Description: Occurs following scan for 55AAh ROM-BIOS extensions, and just prior to operating system boot. Anything may be changed, and changes should be expected to remain until boot code is executed. Anything changed with previous extensions can be restored at this time.

PATCHING THE BIOS EPROMS *(Not LB/PC)*

There are approximately 24,000 bytes of free ROM space in the current LB/286, SB/286, and LB/386 ROM-BIOS chips. This ROM space is uninitialized (filled with bytes of FFh) and may be reprogrammed for OEM use. An OEM interface is provided by Ampro to this uninitialized ROM area to facilitate custom system implementations. Remember, OEM patches are not available on the LB/PC.

At the beginning of the ROM-BIOS, at memory address F000:0000h, there is a 16 entry three byte jump table where the Award BIOS interfaces to the Ampro BIOS. The last entry in this table points to a return instruction that is the last physical byte in the Ampro BIOS. The area between this point and the beginning of the Award BIOS, which occupies the top of the ROM-BIOS EPROMs may be used for OEM patch code. The unused region is filled with FFs, as is the patch table, allowing the standard ROM-BIOS EPROM to be modified by an EPROM programmer.

```
org     F000:0000 hex    ; org 0 in rom
jmp     ampro_reserved   ; 0, reserved
jmp     ampro_reserved   ; 1, reserved
jmp     ampro_reserved   ; 2, reserved
jmp     ampro_reserved   ; 3, reserved
jmp     ampro_reserved   ; 4, reserved
jmp     ampro_reserved   ; 5, reserved
jmp     ampro_reserved   ; 6, reserved
jmp     ampro_reserved   ; 7, reserved
jmp     ampro_reserved   ; 8, reserved
jmp     ampro_reserved   ; 9, reserved
jmp     ampro_reserved   ;10, reserved
jmp     ampro_reserved   ;11, reserved
jmp     ampro_reserved   ;12, reserved
jmp     ampro_reserved   ;13, reserved
jmp     ampro_reserved   ;14, reserved
jmp     ampro_end        ;15, end of ampro BIOS code
```

Following this jump table is a table of 16 unprogrammed words. Each entry in this table is tested during power on self test (POST). If an entry has been changed from the default (0ffffh), a near call to the offset address indicated is made for execution of the OEM patch code.

```
org     0F000:0030h      ; org 30h in rom
dw      0ffffh           ; 0, early reset
dw      0ffffh           ; 1, late reset
dw      0ffffh           ; 2, pre video
dw      0ffffh           ; 3, pre 55aa
dw      0ffffh           ; 4, pre boot
dw      0ffffh           ; 5, oem signon
dw      0ffffh           ; 6-15, reserved
```

INTERRUPT CONSIDERATIONS

Interrupt driven code residing in OEM extensions cannot coexist with SSD on an SSD Expansion Board in systems based on a LB/286, SB/286, or LB/386, or cannot reside in socket "E" of an SSD Expansion Board if SSD software is in use. This is a result of the fact that the on-board sockets on these three boards, and the entire SSD Expansion Board, can be enabled and disabled under software control. When an SSD disk access is requested by the operating system or an application, on-board sockets on the LB/286, SB/286, or LB/386, or the entire SSD Expansion Board, may be temporally disabled. Also, during OEM extension ROM scans, sockets are disabled with interrupts cleared (CLI) while searches for ROMs in other devices are being made.

LB/286, SB/286, LB/386 POST TEST SEQUENCES

For reference purposes, the power-on self test procedures used in the LB/286, SB/286, and LB/386 are shown in the following table:

```
POST  1 - Processor test #1, status verification
POST  2 - Determine type of post test
POST  3 - Clear 8042 interface
POST  4 - Reset 8042
POST  5 - Get 8042 manufacturing status, normal or manuf
POST  6 - Init chips (dma, 8259's, cmos)
POST  7 - Processor test #2
POST  8 - Initialize cmos chip
POST  9 - Eprom checksum for 32 kbytes
POST 10 - Initialize video interface
POST 11 - Test 8254 channel 0
POST 12 - Test 8254 channel 1
POST 13 - Test 8254 channel 2
POST 14 - Test cmos date and timer
POST 15 - Test cmos shutdown byte
POST 16 - Test dma channel 0
POST 17 - Test dma channel 1
POST 18 - Test dma page registers
POST 19 - Test 8741 keyboard controller
POST 20 - Test memory refresh toggle circuits
POST 21 - Test 1st 64k bytes of system memory
POST 22 - Setup interrupt vector table
POST 23 - Setup video i/o operation
POST 24 - Test video memory
POST 25 - Test 8259 channel 1 mask bits
POST 26 - Test 8259 channel 2 mask bits
POST 27 - Test cmos battery level
POST 28 - Test cmos checksum
POST 29 - Setup configuration byte from cmos
POST 30 - Sizing system memory & compare w/cmos
POST 31 - Test found system memory
POST 32 - Test stuck 8259's interrupt bits
POST 33 - Test stuck nmi (parity/io chk) bits
POST 34 - Test 8259 interrupt functionality
POST 35 - Test protected mode and a20 gate
POST 36 - Sizing extended memory above 1mb
POST 37 - Test found system/extended memory
POST 38 - Test exceptions in protected mode
POST 39 - Reserved
POST 40 - Reserved
```

APPLICATION NOTE

NUMBER: AAN-9005 **REVISION:** C **DATE:** 10/16/90

TITLE: Third party sources of enclosures for Little Board based systems.

PRODUCT: LB/386, LB/286, LB/PC, LB/186

SUMMARY: Information regarding a variety of third party sources for enclosures which may be useful in packaging Little Board based systems.

This application note provides reference information regarding several sources of enclosures which may prove useful in integrating Ampro Little Board based systems. Please note that the information below is meant to assist Ampro OEM's in locating appropriate accessories for use with the Ampro Little Board single board systems, and is not intended as an endorsement of any particular company or product; nor have the products offered by companies listed been tested by Ampro. The suitability of these devices will be dependent on the requirements of the specific OEM application.

The following company has taken over the manufacturing and sales of the discontinued Ampro Little Board enclosures, and is also developing a number of additional enclosures specifically intended for Ampro OEM's:

- ENCLOSURE TECHNOLOGIES - Ypsilanti, MI - (313) 481-2200
  Product: a wide variety of Ampro-specific table top and rack mount enclosures. Ampro-specific mounting and cabling kits are available. Custom versions and system assembly available.

Some of the following sources offer enclosures which are specifically targeted to Little Board packaging, while others offer disk drive enclosures which can be adapted to accommodate Little Board based OEM systems:

- COBOTYX CORPORATION - Danbury, CT - (203) 748-0095 x116, Attn: Bill Strait.
  Product: a 15.5" x 9.5" x 5.75" enclosure for the LB/PC. Includes a 5 slot backplane, 60 watt power supply, and mounting and cabling for LB/PC and two 3.5" disk drives. Available with LB/PC and disk drives (720K floppy and 20 or 40 MB hard disk) installed. FCC Class A and UL 478 rated.

- ELMA ELECTRONICS - Fremont, CA - (415) 656-3400.
  Product: a wide variety of modular enclosures for portable and rack mount applications. Mounting kits available for monitors and disk drives. Portable cases include mounting for PC or AT compatible keyboards. Custom enclosures and bracketry available.

- EVEREST ELECTRONIC EQUIPMENT - Anaheim, CA - (714) 634-2200.
  Product: several disk drive enclosures (DB-5 Series), which can be adapted for Little Board packaging. Removable front panel drive cover plates and removable rear panels. Custom versions and custom connector panels available.

- INTEGRAND RESEARCH - Visalia, CA - (209) 651-1203.
  Product: a wide variety of disk drive and single board computer enclosures. Removable front panel drive cover plates and removable rear panels. Custom versions and custom connector panels available.

- JMR ELECTRONICS - Northridge, CA - (818) 993-4801.
  Product: a wide variety of disk drive enclosures which can be adapted for Little Board packaging. Custom versions available.

- RELAX TECHNOLOGY - Union City, CA - (415) 471-6112.
  Product: a variety of disk drive enclosures which can be adapted for Little Board packaging. Products available include both compact enclosures (the size of a single disk drive) and "tower" enclosures.

AMPRO

Ampro Computers, Incorporated
990 Almanor Avenue, Sunnyvale, CA 94086
(408) 522-2100 FAX (408) 720-1305

APPLICATION NOTE

NUMBER: AAN-9006 **REVISION:** B **DATE:** 10/16/90

TITLE: A "watch dog timer" for the Ampro single board systems.

PRODUCT: LB/PC, LB/286, LB/386, SB/286, SB/386.

SUMMARY: In most embedded applications it is important to protect the system against software lockup and system crashes. This application note explains how to use an available "watch dog timer" module to provide this function.

INTRODUCTION

During the operation of systems containing an embedded microcomputer, unexpected events can sometimes cause the application software to lock up or crash due to either software or hardware problems. A common problem is improper system resetting (and hence restarting) due to power browouts (i.e. brief period of low or interrupted power) in which the loss of power supply voltages is insufficient to trigger a proper powerup reset pulse.

For maximum embedded system reliability, a mechanism must be provided to restart the system in such cases. A common way to do this is by means of a "watch dog timer" (WDT), which resets the system (causing it to reboot and restart the application) if the WDT is not serviced within a specified time interval. Thus, if the software hangs up in an infinite loop, or if the system "crashes" completely, a reset will soon occur to restart the system. If the WDT is designed such that it "remembers" its previous timeout settings on powerup, then the same function can to serve to guarantee that the system will always start properly following a power down -- or brownout -- condition. The DS1286 WDT discussed in this application note provides all of these features.

SOFTWARE ISSUES

There are three basic functions usually associated with the use of a watch dog timer:

- Enable -- start the watch dog timer and initialize its timeout time period.
- Tickle -- access the watch dog timer by a read or write, to prevent the system from being reset due to timeout of the WDT counter.
- Disable -- stop the watch dog timer, so no timeouts (and resets) will occur.

Typically, the WDT is enabled during the embedded system production setup and test process. Once this is done, the WDT must be prevented from timing out or the system will reset periodically. During system operation, the application software must "tickle" the WDT often enough to prevent the periodic resets from occuring. For the WDT to be effective, the "tickling" function should not be done from a routine that is triggered by an interrupt such as the DOS timer tick, because many software lockup or crash conditions do not prevent interrupts from being serviced. (For example, crashed systems can often be restarted from the keyboard using Ctrl-Alt-Del, which indicates that interrupt service continues to work in spite of the crash.) Most embedded application programs are structured with some type of main control loop, performing and servicing various system functions. If these functions can be guaranteed not to

AMPRO
Ampro Computers, Incorporated
990 Almanor Avenue, Sunnyvale, CA 94086
(408) 522-2100 FAX (408) 720-1305

take longer than the WDT timeout value, calling the "tickle" routine just before calling each of the functions in the main loop would be the most reliable means of using the WDT. For example:

```
MainLoop
        Tickle
        Routine 1
        Tickle
        Routine 2
        Tickle
        Routine 3
Repeat MainLoop
```

If one of the routines can take longer than the WDT timout, it would periodically need to call the tickle routine and could be in the same format as "MainLoop". This might occur, for example, during disk I/O. In such cases, "Tickle" should be called before each access to a disk (this includes SSD).

Software examples for the Enable, Tickle, and Disable WDT functions are given later.

HARDWARE ISSUES AND INSTALLATION

A watch dog timer module which can be added to the Ampro single board systems is the Dallas DS1286. The DS1286 is a combination clock, alarm, and WDT, but only the WDT function is considered here. For further information on the DS1286, contact Dallas Semiconductor, at (214) 450-0400. The DS1286 WDT can be plugged directly into one of the onboard byte-wide memory sockets. The configuration requirements are slightly different on each of the Ampro single board systems, as described below.

It should be noted that the DS1286 WDT's control registers are located in memory space. Consequently, a runaway program might accidentally alter the state of operation and stop the WDT function from working properly. For this reason, on the LB/286, SB/286, LB/386, and SB/386, the byte-wide socket is left disabled when not being accessed; it thus cannot be accidently accessed. The LB/PC, however, does not have a software controllable byte-wide socket disable function. For the LB/PC, it is therefore useful to disconnect the write signal from the byte-wide socket after the WDT is initialized and enabled, as indicated in the LB/PC installation procedure.

Caution

When installing the WDT on a LB/PC, LB/286, or SB/286, a wire must be added to the board (described below), to connect the required WDT RESET output signal. You can add this wire between an existing jumper pin and the board's RESET cable harness using wire-wrap wire, or you can solder the wire directly to the bottom of the board. **Note, however, that board damage caused by inproper soldering or installation of the RESET signal wire will void the warranty.**

WDT Installation on LB/PC (Model 4B)

1. The board's powerup reset components must be changed to meet the DS1286's current driving capacity. To do this, change C31 from 33 mf to 3.3 mf and R34 from 10k to 100k. Note that the required components may already be present in newer boards.

2. Pins 23 and 26 on the DS1286 must be cut off, as they contain signals which conflict with signals on the board. Then install the DS1286 in socket U26 on the board.

3. Remove any jumpers at W35 and W45. Install jumpers in W36 and W37.

4. Run a wire from W45 pin 4 to J4 pin 4 (RESET). This may be soldered on the bottom of the board.

5. Be sure there is no other system memory in the range E0000-E7FFFh, as this is occupied by the DS1286.

6. To enable writing to the DS1286 -- which is required for enabling and disabling, but not for tickling the watch dog timer -- install a jumper between W38 pins 2 and 3. To prevent writing, remove the jumper; this is useful to protect the WDT function once it is in operation.

WDT Installation on LB/286 and SB/286

1. The board's powerup reset components must be changed to meet the DS1286's current driving capacity. To do this, change C31 from 33 mf to 3.3 mf and R34 from 10k to 100k. Note that the required components may already be present in newer boards.

2. Pins 23 and 26 on the DS1286 must be cut off, as they contain signals which conflict with signals on the board. Then install the DS1286 in socket U33 on the board.

3. Remove any jumpers from W20. Run a wire from W20 pin 2 to J4 pin 4 (RESET). This may be soldered on the bottom of the board.

4. Remove the jumper from W21 pins 5 and 6, if present. To enable writing to the DS1286 -- which is required for enabling and disabling, but not for tickling the WDT -- install a jumper between W21 pins 3 and 4. To prevent writing, install a jumper between W21 pins 1 and 2 instead; however, this may not be required if the byte-wide socket is left disabled by the Tickle function.

5. Be sure there is no other system memory in the range D8000-DFFFFh, as this is occupied by the DS1286.

6. Use SETUP to disable socket U33 (it is enabled when necessary, by the access software).

WDT Installation on LB/386 and SB/386

1. Pins 23 and 26 on the DS1286 must be cut off, as they contain signals which conflict with signals on the board. Then install the DS1286 in socket U33 on the board.

2. Remove any jumpers from W5. The LB/386 and SB/386 have a jumper (W8) which can be used to connect U33 pin 1 to the system RESET input. Install a jumper in W8.

3. To enable writing to the DS1286 -- which is required for enabling and disabling, but not for tickling the WDT -- install a jumper between W5 pins 4 and 7. To prevent writing, leave pins 4 and 7 unshorted; however, this may not be required if the byte-wide socket is left disabled by the Tickle function.

4. Be sure there is no other system memory in the range D8000-DFFFFh, as this is occupied by the DS1286.

5. Use SETUP to disable socket U33 (it is enabled when necessary, by the access software). Also use SETUP to select "32K bytes" as the byte-wide device socket memory size.

SOFTWARE EXAMPLES FOR WATCH DOG TIMER CONTROL

The following pages contain sample code which illustrates how you can provide the three functions required for using the Dallas DS1286 watch dog timer: Enable, Tickle, and Disable.

All constants are in the WDTINCLS.INC file which, by an INCLUDE statement, is used by all the other source files. Also in this file are the code and data segment and class names and the group name. This was done so that changing them in the included file would also change them in all files.

The constant COM_FILE, when defined, structures the source files for the creation of DOS-executable (.COM) programs. If the statement defining it is commented out, the resulting .OBJ file should be suitable for linking to other .OBJ files that you have generated for your application.

NOTE: Depending on the compiler you use, you may need to modify the names of the code, data or group names, or classes. The code which appears below can be assembled with MASM 5.1.

```
;=============================================================================
;   AMPRO COMPUTERS, INC.
;   Copyright 1990, All right reserved.
;       WDTINCLS.INC - Include file for the Watch Dog Timer source files
;
;   Mod Record:
;       04/03/90    01.00   DcMc    New Code
;=============================================================================

        ; comment out if linked to other than stand alone .COM file
COM_FILE        equ     1               ; asm as stand alone .COM file

CR              equ     0dh             ; carriage return
LF              equ     0ah             ; line feed
EOM             equ     0               ; msg string terminator
MAX_TYPE        equ     3               ; max type # of CPU boards known
GET_BOARD_TYPE  equ     0cd00h          ; get what type AMPRO board it is

        ; Table of segment and socket numbers
SEGTBL          STRUC
Segment         dw      ?               ; Segment that WDT is at
Socket          db      ?               ; Socket that WDT is in
                ENDS

        ; Watch Dog Timer registers
WDTREGS         STRUC
SecsLow         db      ?
SecsHigh        db      ?
Minutes         db      ?
MinutesAlarm    db      ?
Hours           db      ?
HoursAlarm      db      ?
Days            db      ?
DaysAlarm       db      ?
Date            db      ?
Months          db      ?
Years           db      ?
CmdRegs         db      ?
WdtSecLow       db      ?
WdtSecHigh      db      ?
WDTREGS         ENDS

; Command Register 0B
; TrnE | TD/WD = A/B (0)| B sor| A pulse (1) | WD msk (1) | TD Msk| WDF | TDF

        ; Definition for segment, class, and group names
        ; If changed here, they will be changed in all files
        ; This allows easy modification for other compiliers, memory models
CODE_PROLOGUE   MACRO
_TEXT                SEGMENT BYTE PUBLIC 'CODE'

IFDEF   COM_FILE
        ORG     100H
ENDIF
                ENDM

CODE_EPILOGUE   MACRO
_TEXT           ENDS
                ENDM

DATA_PROLOGUE   MACRO
_DATA                SEGMENT WORD PUBLIC 'DATA'
                ENDM

DATA_EPILOGUE   MACRO
_DATA                ENDS
                ENDM

Display         MACRO   text
        push    si                      ; save
        lea     si, text
        call    _DisplayText            ; call the display string function
        pop     si
        ENDM
```

```
_TEXT                   SEGMENT BYTE PUBLIC 'CODE'
_TEXT                   ENDS

_DATA                   SEGMENT WORD PUBLIC 'DATA'
_DATA                   ENDS

IFDEF   COM_FILE
                        DGROUP              GROUP         _TEXT, _DATA
                ASSUME  CS:_TEXT, DS:DGROUP
DOEND           MACRO   start
        END     start
                ENDM
ELSE
                ASSUME  CS:_TEXT, DS:_DATA
DOEND           MACRO
        END
                ENDM
ENDIF
;============================ End of file ====================================

;=============================================================================
;   AMPRO COMPUTERS, INC.
;   Copyright 1988, All right reserved.
;       WDTSUBS.ASM - Subroutines for Watch Dog Timer routines
;
;   Mod Record:
;       04/26/90    00.01   DcMc    New Code
;=============================================================================

                PUBLIC  _DisplayText, _GetWdt, _WdtAbort
                PUBLIC  _EnableSocket, _DisableSocket

                INCLUDE wdtincls.inc

                CODE_PROLOGUE

;=============================================================================
;   Display text
;       INPUTS:         SI points to NULL terminated text to display
;       OUTPUTS:        The message, SI advance to point to terminating NULL

_DisplayText    PROC    NEAR
        pushf
        push    ax                      ; save these
        push    bx                      ;
        push    di                      ;
DisplayLoop:
        mov     al, [ si ]              ; get the next character
        inc     si                      ;   & advance the pointer
        or      al, al                  ; is the character the terminater, 0?
        jz      DisplayX                ;   YES, we're done
        push    si                      ;   NO, save the pointer
        mov     ah, 0eh                 ;     write char in TTY mode command
        xor     bx, bx                  ;     BL=foreground color, BH=page
        int     10h                     ;     BIOS video services
        pop     si                      ;     restore the pointer
        jmp     SHORT DisplayLoop       ;     & loop for more
DisplayX:
        pop     di                      ; restore
        pop     bx                      ;   these
        pop     ax                      ;     registers
        popf
        ret
_DisplayText    ENDP

;=============================================================================
;   Get the segment an socket number of the Watch Dog Timer
;       INPUTS:         None
;       OUTPUTS:        ES = segment of WDT
;                       AL = socket sub-function number

_GetWdt         PROC    NEAR
        mov     si, [ SegTblPointer ]   ; get the entry pointer
        or      si, si                  ; has it been initialized ?
        jnz     GetWdtDone              ;   YES, get the data
```

```
        mov     ax, GET_BOARD_TYPE      ;   NO, get the AMPRO board type
        int     013h                    ;     service request
        cmp     cx, 08000h              ;     is the service there ?
        jz      GetWdtAmpro             ;       YES
        Display BadBios                 ;       NO, report error
        jmp     _WdtAbort               ;         & abort
GetWdtAmpro:
        and     bx, 0000fh              ;     keep only the product code
        cmp     bx, MAX_TYPE + 1        ;     do we know this type ?
        jc      GetWdtType              ;       YES
        push    si
        Display BadBios                 ;       NO, report error
        jmp     _WdtAbort               ;          & abort
GetWdtType:
        lea     si, SegTable            ;     point segment/socket table
        or      bx, bx                  ;     get flags back
GetWdtLoop:
        jz      GetWdtStore
        add     si, SIZE SEGTBL         ;     adv to next entry
        dec     bx                      ;     dec the index
        jmp     GetWdtLoop              ;       & loop for the next one
GetWdtStore:
        mov     [ SegTblPointer ], si   ;   save the pointer to table entry
GetWdtDone:
        mov     ax, [ si.Segment ]      ; get the segment
        mov     es, ax                  ;   in ES
        mov     al, [ si.Socket ]       ; get socket ID in AL
        ret
_GetWdt         ENDP

;=============================================================================
;   Enable the Watch Dog Timer socket
;       INPUTS:         AL = socket number to enable
;       OUTPUTS:        None

_EnableSocket   PROC    NEAR
        mov     ah, 0cdh                ; AMPRO function code
        mov     bx, 1                   ; enable it
        int     013h
        ret
_EnableSocket   ENDP

;=============================================================================
;   Disable the Watch Dog Timer socket
;       INPUTS:         AL = socket number to disable
;       OUTPUTS:        None

_DisableSocket  PROC    NEAR
        mov     ah, 0cdh                ; AMPRO function code
        mov     bx, 1                   ; disable it
        int     013h
        ret
_DisableSocket  ENDP

;=============================================================================
;   Watch Dog Code abort routine
;       INPUTS:         None
;       OUTPUTS:        exits to DOS with errorlevel 255

_WdtAbort       PROC    NEAR
        call    _GetWdt                 ; get the socket number
        call    _DisableSocket          ; disable the WDT socket
        mov     ax, 4cffh               ; DOS terminate process
        int     021h                    ;   service request
_WdtAbort       ENDP
                CODE_EPILOGUE

;=============================================================================
;   Data area

                DATA_PROLOGUE

BadBios         db      'ABORTED - Wrong version of BIOS for this utility.'             ;       NO, report error
                db      CR, LF, 0
```

```
; WDT seg & socket  LB/PC        LB/286        SB/286        LB/386        SB/386
SegTable SEGTBL < 0e000h, 3 >, < 0d800h, 4 >, < 0d800h, 4 >, < 0d800h, 4 >, < 0d800h, 4 >   ; WDT segment, socket

SegTblPointer   dw      0               ; pointer to table entry to use

                DATA_EPILOGUE

                END
;========================== End of file ==================================

;=============================================================================
;   AMPRO COMPUTERS, INC.
;   Copyright 1990, All right reserved.
;       ENABLE.ASM - Enable the watchdog timer in Dallas
;                    Semiconductor DS1286
;
;   Mod Record:
;       04/03/90    01.00   CT  New Code
;=============================================================================

                EXTRN   _DisplayText:NEAR, _GetWdt:NEAR, _WdtAbort:NEAR
                EXTRN   _EnableSocket:NEAR, _DisableSocket:NEAR

                PUBLIC  _Enable

                INCLUDE WDTINCLS.INC

                CODE_PROLOGUE


;=============================================================================
;   Enable the Watch Dog Timer

_Enable         PROC    NEAR
IFNDEF          COM_FILE
        push    si
        push    di
ENDIF
        call    _GetWdt                 ; get WDT segment in ES, socket in AL
        call    _EnableSocket           ; enable the WDT socket
        and     ES:[ Months ], 07fh     ; enable the osillator
        mov     al, ES:[ CmdRegs ]      ; get command byte
        and     al, 058h                ; ck bits 3, 4 & 6 in cmd reg
        cmp     al, 010h                ; is it enabled ?
        jnz     NotEnabled              ;   NO
        mov     al, ES:[ WdtSecHigh ]   ; read seconds register
        cmp     al, 099h                ; already enabled?
        jnz     NotEnabled              ;   NO
        Display AlreadyEnabled          ; report problem
        jmp     _WdtAbort
NotEnabled:
        mov     BYTE PTR ES:[ WdtSecHigh ], 099h        ;   & set it up
        cmp     al, 099h                ; did it take ?
        jnz     Failed                  ;   NO
        mov     al, ES:[ CmdRegs ]          ;   & get it
        and     al, 0a7h                ; WD pulsed out on INTA, (low 3 ms)
        or      al, 010h                ;   keep only bit 4
        mov     ES:[ CmdRegs ], al          ;   put it back
        mov     al, ES:[ CmdRegs ]          ;   & read it back
        and     al, 058h                ;   mask it
        xor     al, 010h                ; cmd reg set right ?
        jz      Done                    ;   YES
Failed:
        Display EnableFailed            ;   NO, report error
        jmp     _WdtAbort               ;   & abort
Done:
        call    _DisableSocket          ; disable the WDT socket
IFDEF           COM_FILE
        mov     ah, 04Ch                ; DOS to terminate process
        int     021h
ELSE
        pop     di
        pop     si
        ret
ENDIF
```

```
_Enable         ENDP

                CODE_EPILOGUE

;===============================================================================
;   Data area

                DATA_PROLOGUE

AlreadyEnabled  db      'Watchdog timer was already enabled.', CR, LF, EOM
EnableFailed    db      'Failed to enable the watchdog timer.', CR, LF, EOM

                DATA_EPILOGUE

                DOEND   _Enable
;============================== End of file ====================================


;===============================================================================
;   AMPRO COMPUTERS, INC.
;   Copyright 1990, All right reserved.
;       TICKLE.ASM - Tickle the watchdog timer
;
;   Mod Record:
;       04/03/90    01.00   CT  New Code
;===============================================================================

                EXTRN   _DisplayText:NEAR, _GetWdt:NEAR, _WdtAbort:NEAR
                EXTRN   _EnableSocket:NEAR, _DisableSocket:NEAR

                PUBLIC  _Tickle

                INCLUDE WDTINCLS.INC

                CODE_PROLOGUE

;===============================================================================
;   Tickle the Watch Dog Timer

_Tickle         PROC    NEAR
IFNDEF          COM_FILE
        push    si
        push    di
ENDIF
        call    _GetWdt                 ; get WDT segment in ES, socket in AL
        call    _EnableSocket           ; enable the WDT socket
        mov     al, ES:[ WdtSecHigh ]   ; read seconds register
        call    _DisableSocket          ; disable the WDT socket
IFDEF           COM_FILE
        mov     ah, 04Ch                ; DOS to terminate process
        int     021h
ELSE
        pop     di
        pop     si
        ret
ENDIF
_Tickle         ENDP

                CODE_EPILOGUE

                DOEND   _Tickle
;============================== End of file ====================================

;===============================================================================
;   AMPRO COMPUTERS, INC.
;   Copyright 1990, All right reserved.
;       DISABLE.ASM - Disable the watchdog timer in Dallas
;                     Semiconductor DS1286
;
;   Mod Record:
;       04/03/90    01.00   CT  New Code
;===============================================================================

                EXTRN   _DisplayText:NEAR, _GetWdt:NEAR, _WdtAbort:NEAR
                EXTRN   _EnableSocket:NEAR, _DisableSocket:NEAR
```

```
                PUBLIC  _Disable

                INCLUDE WDTINCLS.INC

                CODE_PROLOGUE

;===============================================================================
;   Disable the Watch Dog Timer

_Disable        PROC    NEAR
IFNDEF          COM_FILE
        push    si
        push    di
ENDIF
        call    _GetWdt                 ; get WDT segment in ES, socket in AL
        call    _EnableSocket           ; enable the WDT socket
        mov     al, ES:[ CmdRegs ]      ; get command byte
        and     al, 058h                ; keep bits 3, 4 & 6 in the cmd reg
        cmp     al, 018h                ; see if already disabled
        jnz     NotDisabled
        cmp     BYTE PTR ES:[ WdtSecLow ], 0
        jnz     NotDisabled             ; see if WDT interval is set to 0
        cmp     BYTE PTR ES:[ WdtSecHigh ], 0
        jnz     NotDisabled
        Display AlreadyDisabled
        jmp     _WdtAbort
NotDisabled:
        or      ES:[ Months ], 080h     ; disable osillator, save battery
        mov     al, ES:[ CmdRegs ]
        and     al, 0a7h
        or      al, 018h                ; mask off the watchdog timer
        mov     ES:[ CmdRegs ], al
        mov     al, ES:[ CmdRegs ]
        and     al, 058h
        xor     al, 018h                ; see if cmd reg is set right
        jnz     Failed
        mov     ES:[ WdtSecLow ], al
        or      al, ES:[ WdtSecLow ]
        jnz     Failed
        mov     ES:[ WdtSecHigh ], al
        or      al, ES:[ WdtSecHigh ]
        jz      Done                    ; if ok, errorlevel = 0
Failed:
        Display FailedDisabled
        call    _WdtAbort
Done:
        call    _DisableSocket          ; disable the WDT socket
IFDEF           COM_FILE
        mov     ah, 04ch                ;tell DOS to terminate process
        int     021h
ELSE
        pop     di
        pop     si
        ret
ENDIF
_Disable        ENDP

                CODE_EPILOGUE

;===============================================================================
;   Data area

                DATA_PROLOGUE

AlreadyDisabled db      'Watchdog timer was already disabled.', CR, LF, EOM
FailedDisabled  db      'Failed to disable the watchdog timer.', CR, LF, EOM

                DATA_EPILOGUE

                DOEND   _Disable

;============================ End of file ======================================
```

APPLICATION NOTE

NUMBER: AAN-9009 **REVISION:** A **DATE:** 10/16/90

TITLE: Multi-tasking and real-time operating systems for embedded PC architecture applications.

PRODUCT: LB/PC, LB/286, LB/386, SB/286, SB/386

SUMMARY: Information regarding a variety of third party multi-tasking and real-time operating systems for Little Board or Slot Board based applications.

INTRODUCTION

The PC and PC/AT compatible Ampro Little Board and Slot Board *single board systems* are ideally suited for embedded DOS (PC-DOS, MS-DOS, DR DOS) based applications. However, the real time requirements of many embedded microcomputer applications often require operating system services and performance which extend beyond those offered by DOS. In many cases, the system design requires the use of multi-tasking and/or real time operating systems.

This document describes several third party operating systems which may provide desirable operating environments for Little Board or Slot Board based applications, and indicates restrictions or special requirements that are known to apply. Only products which provide a full set of operating system services -- including a file system -- have been included.

Please be aware of the following notes:

Note

Please note that the information provided below is meant to assist Ampro OEM's in locating and evaluating appropriate operating systems for use with the Ampro single board system products, and is not intended as an endorsement of any particular company or product. Products are listed alphabetically, by product name.

Due to the nature and complexity of the operating systems listed below, Ampro cannot assume responsibility for proper operation in a particular OEM system configuration. Although all operating systems referenced have been demonstrated to function in specific tested configurations, it is the OEM's responsibility to determine if current versions of a particular operating systems are usable with an intended system configuration.

Ampro Computers, Incorporated
990 Almanor Avenue, Sunnyvale, CA 94086
(408) 522-2100 FAX (408) 720-1305

Note

Multi-tasking operating systems do not generally use ROM-BIOS functions for hard disk interface control, and so cannot make use the Ampro SCSI ROM-BIOS and hardware functions without the addition of custom SCSI drivers. Therefore, the following hard disk, BIOS, and SETUP issues apply to hard disk support:

- Unless SCSI support is indicated, either a standard bus plug-in hard disk controller and drive, or an "IDE" intelligent drive, must be used. IDE drives can be connected to a LB/286 or LB/386 with an Ampro MiniModule/ATDisk IDE interface; the SB/286 and SB/386 have built-in IDE interfaces. IDE is not available for the LB/PC, so an XT compatible hard disk controller would be required.

- Unless SCSI support is used, on the LB/286, LB/386, SB/286, and SB/386, it is generally recommended that you use the SETUP option to disable Ampro BIOS extensions. This requires the presence of Ampro ROM-BIOS Version 3.02 (or later) for the LB/286 and SB/286, and Version 1.06 (or later) for the LB/386 and SB/386. Contact Ampro Technical Support for ROM-BIOS update information.

OPERATING SYSTEM SUMMARIES -- MULTI-TASKING, REAL-TIME

- **FlexOS** -- FlexOS is a modular real-time, multi-tasking operating system which offers interrupt latencies on the order of 10's of microseconds. Key features are: preemptive, event-driven dispatcher; unlimited number of processes; modular architecture with dynamically loadable and unloadable device drivers; protected-mode support. FlexOS also features an enhanced, DOS-compatible file system and a DOS application environment with DOS 3.x function call support. In addition, a library is available which provides broad POSIX function support. The X/GEM graphics interface option supports multi-tasking graphics applications. FlexOS can be used on the LB/286, SB/286, LB/386, or SB/386, provided an IDE interfaced hard disk drive is used. Use of SCSI would require custom drivers. Subsets of the full capability are available, and the kernel and drivers are ROMable. For further information, contact: Digital Research, Inc. -- Monterey, CA -- Phone: (408) 649-3896.

- **OS-9000** -- OS-9000 is a modular real-time, multi-tasking operating system which provides many Unix-like system functions, including a C compiler, C source-level debugger, a shell, hierarchical directory/file structure, and over 70 utility programs for the development environment. The modular design of OS-9000 allows each OEM to modify and configure the system for the specific needs of the application. Drivers are included for the standard PC/AT serial, parallel, keyboard, and floppy interfaces. OS-9000 can be used on the LB/386 or SB/386, provided an IDE interfaced hard disk drive is used. Use of SCSI would require custom drivers. Subsets of the full capability are available, and the kernel, drivers, and all code are ROMable. For further information, contact: Microware Systems Corp. -- Des Moines, Iowa -- Phone: (515) 224-1929.

- **QNX** -- QNX (pronounced "Q-NIX") is a real-time, multi-tasking operating system for 8088, 80286, and 80386 based PC and PC/AT compatible computers, offering UNIX-like system services, but requiring minimal memory and disk resources. QNX provides response times of 75 microseconds (context switch time) on a 16 MHz 80286, utilizing preemptive prioritized task scheduling. Message-passing, distributed processing is fundamental to the QNX architecture. QNX supports the PC and PC/AT architectures, including drivers for the standard serial, parallel, keyboard, and floppy interfaces. A QNX option allows DOS and DOS-based programs to be run as tasks under QNX, and allows QNX programs to read and write DOS files transparently. QNX is usable on the Ampro LB/PC, LB/286, LB/386, SB/286, and SB/386. Ampro "Solid State Disk" (SSD) support for QNX is currently in beta test, and will be production released shortly. Full hard disk functionality is available on the Ampro '286 and '386 boards using

5.1 Ampro Common Utilities (Card Specific Software)

Information regarding the embedded-PC BIOS and the software utilities related to that BIOS follow. This information was downloaded from the Ampro Internet site at:

http://www.ampro.com/products/software/sw-bios.pdf.

Several software programs are provided by Ampro to access the CPU hardware and various memory locations. The technical manual for these common utilities was downloaded from the Ampro Internet site at:

http://www.ampro.com/techman/5000931b.pdf.

IDE. It should also be possible to utilize the Ampro SCSI interface for QNX hard disk access, by installing the real mode QNX option and using the QNX BIOS hard disk driver. Protected mode operation (on the Ampro and '386 boards) requires IDE interfaced hard disks; use of SCSI in protected mode would require custom drivers. For further information, contact: Quantum Software Systems, Ltd. -- Kanata, Ontario, Canada -- Phone: (613) 591-0931.

- VENIX -- This is AT&T UNIX System V/386 with added real time functionality, durability, and embedded capability. VENIX's real time performance is specified as: 350 microseconds worst case response time; 300 microseconds typical response time; and 50 microsecond worst case interrupt latency. "Advertised" real-time features include: bounded response times, priority scheduling, biased scheduling, high resolution timers, asynchronous and unbuffered I/O, contiguous file system, memory preallocation and locking, physical memory and I/O bus access, and user level driver support. VENIX can be used on the LB/386 or SB/386, provided an IDE interfaced hard disk drive is used. Use of SCSI would require custom drivers. For further information, contact: VenturCom Inc. -- Cambridge, MA -- Phone: (617) 661-1230.

- VRTX -- VRTX is a modular real time multi-tasking executive, with ready-to-use device drivers for PC and PC/AT compatible I/O interfaces and peripherals (serial ports, printer port, graphics, and disk drives). "VRTX-PC" includes the basic VRTX kernel along with a DOS compatible file system and DOS function call support, and allows execution of most DOS application software. Using VRTX-PC, DOS and BIOS functions can be accessed from a real time VRTX application. VRTX-PC is usable on the Ampro LB/PC, LB/286, LB/386, SB/286, and SB/386. VRTX-PC uses ROM-BIOS functions for disk interfacing, so it can use either SCSI or IDE hard disk drives. VRTX is targeted to ROM based applications, and provides the support required to load the system directly from EPROM. However, because VRTX-PC uses ROM-BIOS functions for disk interfacing, VRTX-PC based applications should also be bootable from a Solid State Disk (SSD) created with the Ampro SSD/DOS Support Software. Note that the LB/PC's "V40 serial port" is not supported, and would require a custom driver for use under VRTX. Subsets of the full VRTX capability are available, and the kernel and drivers are ROMable. For further information, contact: Ready Systems -- Sunnyvale, CA -- Phone: (408) 736-2600.

OPERATING SYSTEM SUMMARIES -- MULTI-TASKING, NON-REAL-TIME

- Interactive UNIX -- This 80386-only port of AT&T UNIX Version 3.2 can be used on the LB/386 or SB/386, provided an IDE interfaced hard disk drive is used. Use of SCSI would require custom drivers. Note that this is a "standard" UNIX implementation, and is therefore not a real time operating system. For further information, contact: Interactive Systems Corp. -- Phone: (213) 453-8649.

- OS/2 -- Operating System/2 ("OS/2") is a multi-tasking operating system offering, among other things, the capability of running multiple DOS applications simultaneously. The DOS emulator function offers partial downward DOS compatibility. OS/2. OS/2 Version 1.2, as available from IBM, can be used on the LB/286, SB/286, LB/386, or SB/386, provided an IDE interfaced hard disk drive is used. Use of SCSI would require custom drivers. Note that OS/2 is not a real time operating system. For further information, contact IBM.

- Xenix System V (UNIX) -- This port of AT&T UNIX System V to the PC/AT environment can be used on the LB/286, SB/286, LB/386, or SB/386, provided an IDE interfaced hard disk drive is used. Use of SCSI would require custom drivers. Note that this is a "standard" UNIX implementation, and is therefore not a real time operating system. For further information, contact: Santa Cruz Operation -- Santa Cruz, CA -- Phone: (408) 425-7222.

APPLICATION NOTE

NUMBER: AAN-9101 **REVISION:** B **DATE:** 4/23/93

TITLE: Using expansion cards and passive backplanes with Little Board ™and CoreModule ™ products.

PRODUCTS: LB/386, LB/286, LB/PC, CM/XT, MB/AT, MB/PC

SUMMARY: A discussion of various options and issues regarding interface between the Little Board and CoreModule CPU products and PC and AT bus plug-in cards and passive backplanes. Information is included on both Ampro and third-party products.

The Little Board and CoreModule products each provide an expansion bus interface in the form of header connectors, allowing external PC or AT bus expansion. Several options, available from Ampro and others, allow connection between either of these CPU modules and standard PC and AT bus plug-in cards. A number of considerations apply to the length and quality of the cable used to connect these options. This application note provides information helpful in selecting suitable options and in ensuring that they work reliably.

The following topics are covered:

- **Ampro Expansion Options** -- An overview of the plug-in card interface adapters available from Ampro, including two-slot passive backplanes and a unique plug-in card carrier board. These are designed specifically for use with the Ampro Little Board and CoreModule products, and provide maximum flexibility, reliability, and ease of use.

- **Direct Plug-in Card Connection** -- A discussion of how to construct a cable which can be used for direct connection between the edgecard connectors of standard PC or AT bus plug-in cards and the bus expansion header connectors of the Little Board and CoreModule products.

- **Using Third-Party Passive Backplanes** -- A discussion of how to construct a cable which can be used for direct connection between the edgecard slot of non-Ampro PC or AT bus passive backplanes and the bus expansion header connectors of the Little Board and CoreModule products. This is useful when more than two plug-in card slots are required.

- **Bus Expansion Guidelines** -- A discussion of several technical issues which influence system reliability when ribbon cable is used for bus expansion of the Ampro Little Board and CoreModule products, including cable length and quality, backplane quality, and bus termination.

AMPRO®
AMPRO COMPUTERS, INCORPORATED
990 Almanor Avenue, Sunnyvale, California 94086
TEL (408) 522-2100 - FAX (408) 720-1305

AMPRO EXPANSION OPTIONS

- **StackPlane™ /PC** -- This member of the StackPlane form-factor family of expansion cards (5.75" x 8") provides a single card slot for an 8-bit (i.e. PC bus) industry standard expansion card. The StackPlane/PC is normally stacked above or below a Little Board CPU module using four spacers. The PC bus plug-in card mounts flat on the StackPlane, plugging into a right angle edgecard connector. Multiple StackPlane expansion boards can be combined with a Little Board to produce a uniquely compact, stacked system assembly. The StackPlane/PC can also be used to add a PC expansion plug-in card slot to a CoreModule.

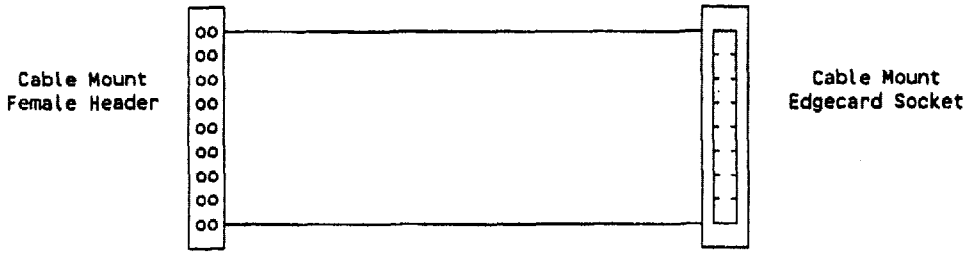
- **MiniBackplane™ /PC** -- This compact passive backplane provides two 8-bit (PC bus) card slots for use with a Little Board/PC or CoreModule/XT. It connects to the CPU module's bus expansion header with a short ribbon cable. An onboard option allows OEM addition of a -12V DC converter (not included), which can be used to eliminate a -12V system power supply in some applications.

- **MiniBackplane/AT** -- This compact passive backplane provides two 16-bit (AT bus) card slots for use with a Little Board/286 or /386. It connects to the CPU board's bus expansion headers with a pair of ribbon cables. Onboard sockets allow OEM addition of +12V (25 mA), -12V (25 mA), and -5V (50 mA) DC-to-DC converters, making +5-volt-only system operation possible in some applications. (The option of up to three onboard DC converters makes this backplane desirable in some Little Board/PC or CoreModule/XT applications as well, to reduce the required number of power supply voltages.) This backplane also includes an option which allows OEM addition of onboard "AC" bus termination (described later).

DIRECT PLUG-IN CARD CONNECTION

It is possible to directly connect one or more PC or AT bus plug-in cards to a Little Board or CoreModule CPU module, using an Expansion Card Adapter Cable (see Figure 1) which can be easily constructed from available components. The cable must have a female header connector at one end for connection to the CPU module's bus interface header, and have one or more cable mounted edgecard socket connectors at the other end for connection to the PC or AT bus plug-in cards.

Figure 1. Expansion Card Adapter Cable



The following connectors can be used to construct this type of cable:

- **Cable Mount Header Connectors** -- The end of the cable that connects to the CPU module requires a 64-contact cable mount female header connector for mating with the 64-pin header bus expansion connector on the CPU module. An additional cable with a 40-contact header connector is required when 16-bit (AT bus) expansion cards are involved. These connectors, called "IDC Female Ribbon Cable Header" connectors, are readily available from several vendors. The following 3M connectors can be used:

  3M 7964-6500EC -- 64-contact cable mount header
  3M 3417-2040 -- 40-contact cable mount header

- **Cable Mount Edgecard Sockets** -- The end of the cable that connects to the bus plug-in card requires a 62-contact cable mount edgecard socket connector. An additional cable with a 26-contact socket connector is also required in the case of 16-bit (AT bus) expansion cards. Not all cable mount edgecard connectors provide the required relationship of edgecard signals relative to the signals bus cable. Be careful not to use connectors which reverse the signal arrangement, swapping alternate signals. The required cable mount edgecard socket connectors are available from PCD Connectors, Peabody, MA 01960. Eastern Sales (508) 532-8800, Western Sales (805) 371-0437. The following PCD connectors can be used:

  RF18-2852-5 -- 36-contact cable mount edgecard socket
  RF31-2852-5 -- 62-contact cable mount edgecard socket

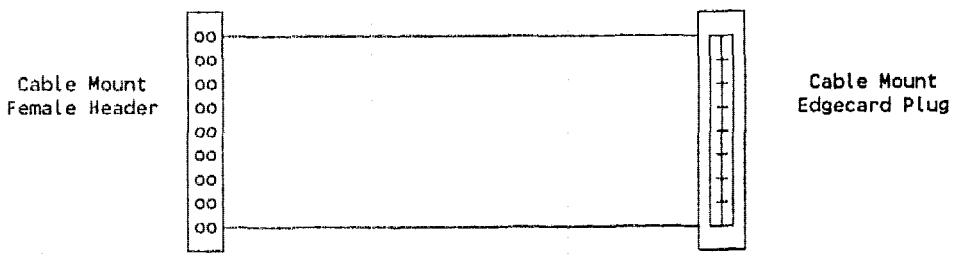
- **Standard Ribbon Cable** -- this is covered in the next section, on bus expansion guidelines.

The CPU module's bus expansion header connectors provide several extra ground connections which will not be used in a cable having these components, because the edgecard connector only has 62 signal contacts whereas the bus header connector has 64. Specifically, the last two conductors of the 64-pin expansion bus header (pins A32 and B32) should be left unconnected, and both the first and last two conductors of the 40-pin bus expansion header (pins C0,C19,B0,B19) should be left unconnected.

USING THIRD-PARTY PASSIVE BACKPLANES

The Little Board and CoreModule CPU modules can also be used with third-party PC and AT bus passive backplanes, using a cable which connects between the CPU module's bus expansion headers and one of the backplane's expansion card slots. To accomplish this, a Standard Backplane Adapter Cable with a unique male cable mount edgecard plug connector on one end must be constructed as described below.

Figure 2. Standard Backplane Adapter Cable



The following components can be used to construct this cable:

- **Cable Mount Header Connectors** -- The end of the cable that connects to the CPU module requires a 64-contact cable mount female header connector for mating with the 64-pin header bus expansion connector on the CPU module. An additional cable with a 40-contact header connector is required when 16-bit (AT bus) expansion cards are involved. These connectors are called "IDC Female Ribbon Cable Header" connectors, and are readily available from several vendors. The following 3M connectors can be used:

  3M 7964-6500EC -- 64-contact cable mount header
  3M 3417-2040 -- 40-contact cable mount header

- **Cable Mount Edgecard Plugs** -- The end of the cable that connects to the passive backplane card slot requires a 62-contact cable mount edgecard plug connector. An additional cable with a similar 26-contact plug connector is also required in the case of 16-bit (AT bus) expansion cards. Not all cable mount edgecard connectors provide the required relationship of edgecard signals relative to the signals bus cable. Be careful not to use connectors which reverse the signal arrangement, swapping alternate signals. The required cable mount edgecard socket connectors are available from PCD Connectors, Winchester MA, (617) 721-1280. The following PCD connectors can be used:

  PF1828520 -- 36-contact cable mount edgecard plug
  PF3128520 -- 62-contact cable mount edgecard plug

- **Standard Ribbon Cable** -- this is covered in the next section, on bus expansion guidelines.

The CPU module's bus expansion header connectors provide several extra ground connections which will not be used in a cable having these components, because the edgecard connector only has 62 signal contacts whereas the bus header connector has 64. Specifically, the last two conductors of the 64-pin expansion bus header (pins A32 and B32) should be left unconnected, and both the first and last two conductors of the 40-pin bus expansion header (pins C0,C19,B0,B19) should be left unconnected.

BUS EXPANSION GUIDELINES

Several guidelines should be observed in the design of systems that have ribbon cable bus connections between Little Board or CoreModule CPU modules and passive backplanes or other expansion cards. Specifically:

- **Cable Length and Quality** -- In general, keep the bus expansion cable as short as possible. Excessively long cables will result in reduced system reliability.

  -- For cable lengths up to 6 inches, use a high quality standard cable, such as 3M 3365/64 (64 conductor) and 3365/40 (40 conductor).

  -- For cable lengths between 6 and 12 inches, use a high quality *ground plane* cable, such as 3M part number 3353/64 (64 conductor) and 3353/40 (40 conductor).

  -- Cable lengths in excess of 12 inches are not recommended.

- **Backplane Quality** -- Be sure to use a high quality backplane having minimal signal crosstalk. Use of power and ground planes, and guard traces between bus signals will maximize system reliability.

- **Reset and TC Deglitching** -- Some PC and AT bus expansion cards have asynchronous TTL logic inputs that are susceptable to signal noise and crosstalk. The active high RESET and TC bus lines are especially vulnerable. The reliability of these signals can be increased by adding a 200 pF to 500 pF capacitor between the signal and ground to prevent false triggering by filtering noise on the signals. These RESET

and TC deglitching capacitors are included on most Ampro MiniBackplanes and StackPlane expansion products.

- **Bus Termination** -- Many backplanes include bus termination to improve system reliability by lowering backplane signal impedance to match the impedance of the rest of the system. The IEEE-P996 draft specification for the AT expansion bus recommends the use of "AC" termination rather than resistive termination. The recommended "AC" termination consists of a 50 to 100 pF capacitor, in series with a 50 to 100 ohm resistor, between each signal and ground. Ampro provides positions for OEM addition of AC termination on most bus expansion products, including the MiniBackplane/AT.

CAUTION

Do not use resistive bus termination, or system reliability may be degraded! Use AC termination only, if termination is required.

The AC terminators used on the Ampro MiniBackplane/AT consist of eight signal terminators in a 9-pin single-in-line package (SIP). Pin 1 of the 9-pin SIP AC terminator package is connected to a common ground, and the remaining eight pins are for the bus signals. Typical manufacturer part numbers for 9-pin, eight-terminator devices containing 100 pF capacitors in series with 100 ohm resistors are:

-- Dale part number CSRC-09C30-101J-101M

-- Bourns part number 4609H-701-101/101

The actual requirements for signal termination depend on the system configuration, on the CPU and bus clock speeds, on the interconnecting bus cable, and on the number and type of expansion cards used. In all cases, it is the system engineer's responsibility to determine the need for termination by observing the signal quality in the system.

APPLICATION NOTE

NUMBER: AAN-9201 **REVISION:** B **DATE:** 7/15/93

TITLE: Design Issues for Embedded PCs.

PRODUCT: Non-specific

SUMMARY: An overview of technical issues regarding the use of the PC architecture in embedded applications. Discusses many of the hardware and software enhancements that are implemented in Ampro Little Board, CoreModule, and MiniModule products.

See attached 13 page document

AMPRO®

AMPRO COMPUTERS, INCORPORATED
990 Almanor Avenue, Sunnyvale, California 94086
TEL (408) 522-2100 - FAX (408) 720-1305

990 Almanor Ave. ▪ Sunnyvale, CA 94086 ▪ Phone 408-522-2100 ▪ Fax 408-720-1305

USING THE PC ARCHITECTURE IN EMBEDDED APPLICATIONS

Rick Lehrbaum
Executive Vice President, Strategic Development
Ampro Computers, Inc.
990 Almanor Avenue
Sunnyvale, CA 94086

ABSTRACT

Originally intended for office desktops, the IBM PC (and PC/AT) architecture has become an extremely popular platform for a myriad of applications never dreamed of by its originators. These include communications devices, vending machines, test equipment, data acquisition and control products, and medical instruments. There are many reasons why embedded system designers would like to use the PC architecture embedded within their systems, including widespread availability of PC-compatible hardware, software, peripheral devices and development tools. Despite these advantages, however, normal desktop PC designs are not reliable enough for many kinds of embedded applications. Fortunately, a number of PC enhancements and extensions have been developed which solve these problems. This makes the PC architecture an important option for use in embedded system applications.

THE NEED FOR EMBEDDED SYSTEM STANDARDS

Companies that embed microcomputers as controllers within their products have long been frustrated by the lack of consistent microcomputer architecture standards. Without a consistent architecture standard, each microprocessor-based embedded system design starts from ground zero, with the most basic architectural decisions:

- Select a microprocessor
- Design or select a system bus
- Design or select an operating system

Frequently, a significant portion of the product development design cycle is taken up by these basic tasks associated with defining the system architecture. Also, when the architecture varies from project to project, much additional time and energy are wasted locating and learning to use the development tools unique to each project.

Another problem is that without consistent embedded system standards, the hardware engineer, in addition to worrying about providing the system-unique interfaces and packaging, must also be a computer architect. Similarly, software engineers, in addition to providing the application specific control software, often must write the equivalent of a small operating system or multi-

tasking executive.

GROWING POPULARITY OF THE PC ARCHITECTURE

Over the last decade, the IBM PC architecture has become an increasingly popular microcomputer system standard. Although originally intended for office desktops, the PC has become so pervasive — and inexpensive — that it has found its way into a myriad of non-desktop, *embedded* applications never dreamed of by its designers. Typical embedded applications for PCs now include: communications subsystems, vending machines, test equipment, data acquisition and control, and various medical devices.

A combination of factors have contributed to the PC architecture's growing popularity in embedded applications. These include:

- **Cost** — The vast installed base of PCs — over a hundred million — and the large number of PC-compatible product vendors (thousands), have produced dramatic reductions in the price of components, peripherals, and software.

- **Familiarity** — The universal presence of PCs in both the workplace and at home has resulted in an exceedingly high level of PC architecture familiarity among microcomputer system specifiers, designers, and programmers.

- **Development software** — The PC's well known system-level standard has given rise to an extremely diverse toolset of application development support software, including operating systems, device drivers, function libraries, compilers, and debuggers.

- **Performance evolution** — The continual evolution of both the "80x86" CPU family and the MS-DOS operating system have enabled the PC architecture to evolve in performance and sophistication, without sacrificing previously developed applications and technologies.

In short, the PC architecture's acceptance has become so widespread, and its applications so broad, that it has been dubbed the *Industry Standard Architecture* (ISA).

As a result of all these factors, more and more companies developing microcomputer-based products in a diverse range of markets, weary of endlessly reinventing the wheel, are now investigating the possible use of the PC architecture — the Industry Standard Architecture — as the basis of their future embedded system designs.

DEFINING THE PC "STANDARD"

A relatively recent effort at standardizing the PC and PC/AT-bus is seen in the activities of the IEEE's P996 committee. The P996 (draft) specification includes timing diagrams, bus drive, signal functions, and mechanical dimensions for the PC (8-bit) and PC/AT (16-bit) buses.

But the *PC architecture* represents much more than what could be included within the P996 spec. The "standardized" use of a choice of CPUs, that range from the 8088 to the 80486 (and new "Pentium"), and of "PC compatible" controllers for DMA, interrupts, timing, serial ports, graphics, LAN interfaces, etc., are the results of to designers copying what works and

becomes popular — they are not based on any written standard. Software conventions such as BIOS functions, device drivers, communications protocols, and memory partitioning practices are also important parts of this unwritten — but extremely comprehensive — PC *system standard*.

With millions of PCs now installed in a wide range of non-desktop and embedded applications, an entire industry has emerged based on providing application-specific hardware and software products targeted to the use of PCs in a wide range of dedicated and embedded applications. Numerous publications, catalogs, and companies specialize in providing components, peripherals, software, and technical references for development of systems based on the PC architecture.

There is so much PC architecture-based development in process, with so many products available to support it, that the *PC architecture* might more appropriately be termed the *PC phenomenon*.

PC HARDWARE OPTIONS

In addition to the expected array of desktop-oriented devices such as disk drives, CRT monitors, modems, printers, keyboards, and pointing devices, a broad assortment of application-specific expansion cards and peripherals targeted to dedicated and embedded uses of PCs is available. Most are offered with plug-and-play interface cards and software drivers that greatly ease the task of system integration. Such devices include:

- **Bus Plug-in Cards** — FAX transmission and reception; analog and digital I/O; speech and video processing; specialized communications interfaces including RF, AC, infrared, and serial LANs.

- **Alternative Display Technologies** — Compact, ruggedized CRTs; various flat panel displays, including liquid crystal displays (LCDs), electroluminescent (EL) displays, and plasma displays; and even a tiny display that can be worn directly on the user's head.

- **User Input Alternatives** — Specialized keyboards and keypads; touch input sensors; speech recognition devices; and keyboard simulators that connect to customized switch matrices.

PC SOFTWARE OPTIONS

Software development — rather than hardware development — increasingly dominates new product design cycles. It is no surprise, then, that one of the most common reasons why system designers choose the PC architecture is for its rich and cost-effective software toolset. This includes operating systems, device drivers, libraries, languages, and debugging tools. Some of the application-specific software options available are:

- **Alternative Operating Systems** — Numerous real-time, multi-tasking operating systems, executives, and DOS extensions support real-world interface and control applications. Most of these include ready-to-use drivers for standard or popular PC interfaces (DMA, interrupts, timers, keyboard, serial, modem, printer, disk, networking, and display).

- **Languages and debugging tools** — Virtually all programming languages are offered for application development and debug directly on the PC. Powerful, yet inexpensive, debugging tools are also readily available.

- **Libraries and Drivers** — A wide array of pre-developed driver and utility routines simplify development tasks. In many cases, optional source code allows effortless customization. Functions available include: memory management, graphics, windowing, communications, FAX, networking, terminal emulation, and numerous application-oriented functions.

SHORTCOMINGS OF STANDARD PCs

Is the PC ideal for *all* embedded applications? Some embedded systems have unique needs which dictate highly specialized system architectures. They may have critical I/O, memory, computational, or other requirements which cannot be satisfied by the PC (or AT) system architecture or performance capabilities.

Because the *desktop* market, for which the PC was created, has very different priorities from those of the *embedded system* markets, there are a number of issues which must be considered carefully when designing PC technology into embedded systems.

A brief discussion of some of the potential limitations and key issues of standard PC (and AT) technology, relative to embedded applications, follows.

BUS BANDWIDTH

The original IBM PC/AT was based on a 6 MHz 80286 CPU and had a 6 MHz AT-bus. Compaq's 8 MHz AT implementation soon became the *de facto* standard, which evolved into the IEEE P996 (draft) bus specification. A further bandwidth limitation results from implementing "AT-compatible" DMA. Unfortunately, the IBM PC/AT's DMA logic, which set the standard, was implemented in a way that limits I/O-to-memory DMA transfers to around .5 MByte/sec. on 8-bit, and 1 MByte/sec. on 16-bit, bus transfers.

To circumvent this, programmed I/O "block moves" can be used to achieve higher burst data transfer rates, approaching 2 MByte/sec. This is done in the case of standard AT hard disk controllers (including IDE interfaces). Another solution is to use "third-party DMA", based on the AT-bus "bus master" function (described later).

INTERRUPTS

The PC provides eight hardware interrupts, while the AT increases this to fifteen. Table 1 shows the way interrupts are normally allocated in PC and PC/AT systems. Of the PC's eight interrupts, two are reserved for motherboard functions, so only six are present on the PC-bus. Of these, three are typically occupied by the COM1 serial port, LPT1 parallel port, and the floppy controller, leaving just three channels typically unoccupied.

Although the AT's 16-bit bus provides seven additional channels, most off-the-shelf expansion cards are 8-bit cards, making the additional AT-bus interrupts generally unusable. As a result, interrupts can be in short supply in PC-based systems.

Table 1. Typical PC and PC/AT Interrupt Allocation

INTERRUPT	FUNCTION
	Interrupts present in PC system:
(IRQ0)	ROM-BIOS clock tick. Not on bus.
(IRQ1)	Keyboard controller. Not on bus.
IRQ2	EGA or VGA controller
IRQ3	Secondary serial port
IRQ4	Primary serial port
IRQ5	Secondary parallel printer (seldom used)
IRQ6	Floppy controller
IRQ7	Primary parallel printer (seldom used)
	Additional interrupts in AT system:
IRQ8	Battery-backed clock alarm. Not on bus.
IRQ9 *	Substitutes for IRQ2, which is unavailable.
IRQ10	Available
IRQ11	Available
IRQ12	Available
(IRQ13)	80287 Math Coprocessor. Not on bus.
IRQ14	Hard disk controller
IRQ15	Available
* Corresponds to IRQ2 on the PC's expansion bus.	

Table 2. Typical PC and PC/AT I/O Address Allocation

I/O Address	Function
03F8 - 03FFh	Primary serial port
03F0 - 03F7h	Floppy disk controller
03D0 - 03DFh	CGA display adapter
03C0 - 03CFh	EGA display adapter
03B0 - 03BFh	Monochrome display adapter
03A0 - 03AFh	Reserved for primary bisync port
0390 - 039Fh	Available
0380 - 038Fh	Reserved for secondary bisync port
0378 - 037Fh	Primary parallel printer port
0370 - 0377h	Available
0360 - 036Fh	Reserved
0300 - 031Fh	Reserved for prototype card
02F8 - 02FFh	Secondary serial port
0278 - 027Fh	Secondary parallel printer port
0200 - 0207h	Reserved for game port
01F0 - 01F7h	Hard disk controller
00F0 - 00FFh	Reserved
00C0 - 00DFh	DMA controller 2 (8237 equiv.)
00A0 - 00BFh	Interrupt controller 2 (8359 equiv.)
0080 - 009Fh	DMA page registers (74LS612 equiv.)
0070 - 007Fh	Real time clock and NMI mask
0060 - 006Fh	Keyboard controller (8042 equiv.)
0040 - 005Fh	Programmable timer (8254 equiv.)
0020 - 003Fh	Interrupt controller 1 (8359 equiv.)
0000 - 001Fh	DMA controller 1 (8237 equiv.)

I/O ADDRESS SPACE

Although the 8088 CPU on which the original PC-bus was based has a *sixteen bit* I/O address space, only *ten bits* of I/O addressability were designed into the IBM PC's expansion bus. While 1,024 I/O port addresses may seem sufficient, the situation is worsened by the common practice of minimally decoding I/O addresses in the interest of keeping component costs down. As a result, the PC architecture's I/O space has become relatively congested, with few regions available for nonstandard devices (see Table 2).

MEMORY ARCHITECTURE

The earliest PCs came with 256 Kbytes of RAM memory. Even when fully loaded, the 8-bit PC-bus only supports 1 megabyte of directly accessed memory. This is not surprising, since the 8088 CPU around which the PC-bus was designed has a 1 megabyte address space. While there are certainly many embedded applications that can run within a megabyte of memory, dramatic reductions in the cost of memory — and increases in its density — have precipitated rapid increases in the memory requirements of programs.

Although the AT-bus extends the system's memory address space to 16 megabytes, which is adequate for most (though not all) embedded applications, another architectural issue limits the usefulness of this so-called *Extended Address Space*. The problem is that accessing memory above the first megabyte by an 80286 CPU involves switching in and out of 80286 "protected mode", which, in the AT-architecture, requires the awkward (and time consuming) process of resetting the CPU through the keyboard microcontroller.

As a result of both the PC-bus memory space limit and the 80286 protected-mode problem, memory paging schemes have been used to allow access to additional system memory from within the lower megabyte. Memory accessed in this manner is known as *Expanded Memory*, and this technique has become standardized in the form of the *de facto* Expanded Memory Standard (EMS). Unfortunately, the constraints of this sort of memory paging have limited the use of EMS memory to functions such as RAM disk or hard disk caching, not program execution. Together, these factors prevented DOS from developing facilities for programs to use more than 640 Kbytes of system memory.

The sophisticated memory management logic of the 80386 (and later) CPUs theoretically solves the high memory access problem. Until recently, however, most PC operating systems — which were designed around the 8088, and which shunned the complexities of 80286 protected mode — have not supported this capability. Sophisticated PC operating systems such as Unix — and DOS-extensions such as Windows — now support larger amounts of memory via the advanced memory management capabilities of the 80386 (or later) CPUs. Applications requiring efficient access to large amounts of memory (greater than 1 megabyte) therefore demand 80386 and 80486 CPUs, and are increasingly using operating systems other than DOS.

MULTI-MASTER SUPPORT

The 8-bit PC-bus does not support multiple bus masters. The 16-bit AT-bus, on the other hand, has a simple "bus mastering" function that allows appropriately designed bus devices to perform direct DMA transfers to and from other cards on the bus. This feature is rarely used, however, and is not even implemented in some AT logic chipsets and AT system designs. Fortunately, few embedded applications require multi-processing or multiple bus masters, so this shortcoming of the AT-bus is not generally a problem.

SETUP VOLATILITY

In PCs, a group of configuration switches are used to define the number of floppy drives, type of video controller, and other such system parameters. ATs, on the other hand, store system such configuration information (called the system "SETUP") in a battery-backed CMOS static RAM located inside the real-time clock chip. Unfortunately, when the SETUP backup battery goes bad (and they all do, eventually), most ATs won't boot. This situation is undesirable in most embedded applications, but can easily be catastrophic in some! Another problem is that

batteries are not allowed in some applications, due to their caustic and potentially explosive chemicals.

BUS TERMINATION

Bus termination was not designed into the original IBM PC- and PC/AT-buses, since it was not perceived to be needed for their intended application — desktop computers, with limited expandability. As a result, the use of bus termination has never been common in the design of PC- and AT-compatible systems. Unfortunately, many of the latest high-speed PC and AT chipsets, used to generate the bus signals, do not have adequate signal slew rate control. This can result in excessive signal noise and crosstalk, causing spurious resets, inaccurate timings, and faulty data under certain conditions. The problem is not severe in small systems with few bus devices and short signal traces, but can compromise the reliability of larger or higher speed systems. It is also more likely to cause problems in embedded systems than in desktop systems, due to wider operating temperature ranges, larger numbers of I/O devices, electrically noisy environments, or the need for round-the-clock non-stop system operation.

MECHANICAL AND ENVIRONMENTAL CONCERNS

Desktop market cost constraints limit the levels of ruggedness and reliability that are designed into the PC and AT systems. Unlike desktop systems, however, most *embedded* systems are generally expected to run without operator intervention, continuously, and without failing. These considerations apply not only to the computer electronics, but to the disk drives, displays, and other system peripherals as well. Some mechanical and environmental shortcomings of standard PCs, relative to the requirements of embedded systems, are:

- **Size** — Unlike factory floors, where industrialized standard PCs are common, space is extremely precious in many embedded system environments. Consequently, standard form-factors (motherboard, bus cards, and associated mechanical components) are often too bulky for many embedded systems.

- **Shock and Vibration** — Vibration and shock can be significant factors in embedded systems destined for portable or vehicular use. In standard PC motherboard-based systems, the electronics cards are not well secured; the long, narrow bus cards (13.5 x 4.8 in.) are only fastened at one end. Attempts to solve this problem with various mechanical means have been frustrated by two factors: (1) board heights vary, due to the dimensional differences between the PC (8-bit) and AT (16-bit) buses; (2) board lengths are also inconsistent, because there is no standard whatsoever for "short" cards, which can be (and are!) any desired length.

- **Operating Temperature** — The electronics within embedded systems may be required to work reliably over a wide temperature range due to restricted internal airflow or because of extremes in their ambient environment. For the desktop ambient system environment, which is generally held in the neighborhood of 20 $\pm$5°C, an operating temperature rating of 0 to 55°C for the internal electronics is considered more than adequate. Embedded system environmental specs usually demand control electronics to be rated for *at least* a range of 0 to 70°C, with many needing -20 or -25°C "extended temperature" operation, at the low end (e.g. for outdoor, unheated operation).

- **EMI** — Another important issue is radiation of — and susceptibility to — electro-magnetic interference. High levels of electro-magnetic interference are present in many

embedded environments; conversely, electro-magnetic emissions may need to be kept to an absolute minimum.

QUALITY ISSUES

The desktop and embedded system markets have very different quality requirements. What may be annoying on the desktop can be disastrous in the operating room. "Zero defects" costs money — and PC manufacturers constantly cut corners wherever possible, to produce products that can compete effectively in the highly price-sensitive personal computer marketplace.

A further issue, especially significant to manufacturers of many kinds of embedded systems, is the need for product consistency. It is imperative that embedded system manufacturers receive the *exact same* board (and BIOS) version and configuration every time — or the production line gets stopped. Most embedded system manufacturers demand that their suppliers implement strict quality and configuration management control systems, and typically require six-month to one-year change notification. Needless to say, these requirements run counter to the business practices and philosophies of virtually all desktop-oriented PC motherboard and expansion card manufacturers — and even chipset vendors — who are driven far more by cost than quality.

FUNCTIONAL ENHANCEMENTS FOR EMBEDDED APPLICATIONS

A number of hardware and software extensions and enhancements to the PC architecture have been developed, in the course of successfully adapting it to a wide range of embedded applications. A brief discussion of several of these follows.

SOLID STATE DISK (SSD)

In most embedded applications, the system is expected to behave like an *appliance*, instantly performing its single predefined function when it is turned on. In these applications, the embedded PC needs to act like a programmed microcontroller. Booting such a system from a disk drive is generally undesirable.

An additional disadvantage of running an embedded system from a disk drive is the susceptibility of disk drives and their magnetic media to mechanical shock and vibration, environmental contaminants, electro-magnetic interference, and thermal factors. Magnetic drives and media are simply not reliable enough for the non-stop, error-free operation required of most embedded systems.

"Solid State Disk" (SSD) offers an excellent alternative. SSD substitutes EPROM, Flash EPROM, or nonvolatile RAM (NOVRAM) memory devices for normal disk drives and media. An SSD has no moving parts to break down, generates minimal heat, is not subject to soft or hard data errors, and provides practically instantaneous data access.

SSD operation is accomplished by means of driver software which intercepts and extends the floppy or hard disk services within the system's BIOS software. From the perspective of the operating system-and application software, the SSD appears to be a normal disk drive, so no modifications to software are needed. In effect, SSD provides the equivalent of ROM-based operation, but without requiring special programming techniques.

SSDs can be read-only, using EPROMs, or read-write using nonvolatile RAMs (NOVRAMs). Flash EPROM is becoming increasingly popular, and can be implemented as either read-only or read-write media.

PCMCIA

An important emerging standard for SSD in the PC architecture is the PCMCIA memory card standard, which was developed for use with credit card sized EPROM, Flash EPROM, or NOVRAM memory card SSDs. PCMCIA memory cards can be interfaced to the system in a number of ways, including via a PC-bus card or by connection of an adapter to one of the PC's existing interfaces such as a serial port, printer port, or IDE interface.

PCMCIA has evolved beyond its original purpose as a memory card interface, into a more generalized interface. The PCMCIA standard now supports I/O expansion as well, allowing addition of such functions modems, faxes, magnetic-media hard disks, and network adapters. As PCMCIA grows in popularity in the laptop and palmtop mass markets, PCMCIA I/O devices will become an increasingly important part of the embedded-PC toolset.

SCSI

The Small Computer System Interface (SCSI) was developed to allow simple interfacing of microcomputer systems with a diverse range of peripherals. Although originally developed as a disk drive controller interface, SCSI is actually a highly generalized and flexible peripheral interface. Disk and tape drives, CD-ROMs, scanners, printers, network interfaces, and computer-to-computer connections are a few typical SCSI-based functions.

Unfortunately, little of SCSI's potential has been realized in the PC environment. This is due to the absence of a major PC vendor incorporating SCSI into a popular product and thereby setting a specific implementation standard. Without such a standard-setter, each system's SCSI interface implementation requires unique hardware-specific drivers.

Only recently has a software driver convention been developed to standardize the use of SCSI in PC systems. Called the Common Access Method (CAM), the new standard defines a driver structure that may allow SCSI to one day achieve its goal of providing plug-and-play system peripheral expansion. In the mean time, SCSI is mainly being used for connecting CD-ROMs and tape drives, but requires care in the selection of driver software.

WATCHDOG TIMER

A "watchdog timer" is used to restart the system in case normal operation is interrupted by unexpected events such as software or hardware failures, power brownouts, etc. Non-stop operation is required in critical applications such as medical instruments and security systems, but is also desirable in less critical applications, like vending machines, test equipment, and communications devices.

Watchdog timers are generally based on a timing device which must periodically be retriggered by the application software. When a timeout condition occurs, the watchdog timer logic re-starts the system by generating a non-maskable interrupt or hardware reset (the latter is preferred).

The AT architecture contains a battery-backed real time clock device which includes a little-

used *alarm* output. With minimal added components, this alarm function can be made to trigger a hardware reset, resulting in a simple but effective watchdog timer. To simplify software development, BIOS support should be included.

BATTERY-FREE SETUP

As discussed earlier, loss of the system's battery-backed "SETUP" configuration data, due to failure of the real time clock backup battery, will prevent most AT-compatible systems from booting. Fortunately, there is a simple solution to this problem. A copy of the SETUP data can be saved in an electrically reprogrammable EPROM (EEPROM) device which does not require a battery to retain its data in the absence of system power. Also, this approach allows the use of embedded AT-compatible systems that do not even contain a battery (which may be an environmental constraint). Appropriate BIOS support is required, to make this work in a fully "compatible" manner.

IMPROVED BUS TERMINATION

The most common (and least expensive) forms of bus termination are purely resistive "DC termination", using either "pull-up" resistors or resistive dividers. However, the bus drive requirement of properly designed DC termination exceeds the drive capabilities of many PC-bus cards and chipsets. Therefore, simple DC termination is not practical in PC-compatible systems.

The IEEE-P996 PC-bus specification recommends using a different type of termination, called "AC termination". This consists of a series resistor and capacitor between each signal line and ground. (The recommended values are 50-100 pF in series with 50-100 ohms.) AC bus termination has several important advantages: (1) it draws current only during signal transitions, so it consumes minimal power; (2) logic levels are not degraded by excessive DC loading; (3) it filters out the signal "spikes" associated with logic level transitions. These factors combine to improve bus noise immunity and reduce EMI radiation.

INTERRUPT SHARING

In some applications, there may not be enough free interrupt channels on the PC-bus. Although the 16-bit AT-bus offers seven additional interrupt channels, space or cost constraints often dictate using the smaller PC-bus. In this case, a simple method of "interrupt sharing" can be implemented to allow the presence of additional interrupting devices.

If the bus interrupt signals were generated by *active low*, open collector drivers (as is the practice with many microcomputer buses), multiple device interrupts could be connected to each bus interrupt line. However, since the bus interrupt signals are active high, "wire-oring" of interrupts is not so easily accomplished. To circumvent this limitation, the IEEE-P996 draft specification includes a method of sharing bus interrupt lines based on using a tri-state gate to drive the shared bus interrupt signal, with a pull-down resistor on its output. A configuration jumper between the pull-down resistor and the bus interrupt request line is used to connect the pull-down resistor on only one card driving each interrupt request.

Since multiple devices now generate a single bus interrupt, the interrupt service routine must poll all the interrupting devices corresponding to each shared interrupt, to see which one is the cause of the interrupt. This process is straightforward and requires minimal additional software overhead. Device drivers for some PC interfaces (particularly serial) commonly support

this method of interrupt sharing.

REPACKAGING THE PC FOR EMBEDDED SYSTEMS

As discussed earlier, standard desktop PC motherboards and bus expansion cards are generally not suitable for embedded applications, due to considerations of reliability, ruggedness, and quality. As a result, a growing number of manufacturers have developed various PC-compatible alternatives, specifically targeting the requirements of embedded applications.

PASSIVE BACKPLANE SYSTEMS

The simplest (and most common) packaging improvement — frequently used by companies offering rackmount industrial PC's — is known as the "passive backplane" PC- or AT-bus. In this approach, a simple connector-only ("passive") backplane is substituted for the usual motherboard; the functions normally on the motherboard are placed on a card that plugs into the passive backplane and has the normal PC expansion card form-factor.

This improves system "mean time to repair" (MTTR), since the motherboard function can now be quickly replaced, and also makes the system more modular and upgradable. To alleviate problems of bus cards loosening from their slots in the backplane due to shock and vibration, the cards are usually anchored in place using adjustable hold-down brackets.

The main disadvantage of passive backplane systems is their size. Many embedded systems simply cannot afford the bulk of backplanes, card cages, and multiple plug-in cards.

HYBRID BUSES

Another approach focuses mainly on gaining the advantages of PC *software*-compatibility, by adapting the PC's functional architecture to one of the popular industrial backplane-buses, such VME or STD bus.

In this case, the system's bus conforms to the specific industrial backplane-bus specification (i.e. VME or STD), while the key hardware functions (CPU, memory, interrupts, DMA, timers, keyboard interface, etc.) and software protocols (e.g. BIOS) conform as closely as possible to those of a normal PC system. This results in an STD- or VME-based system which can run PC compatible operating systems, drivers, and applications without modification (hopefully), but which retains the ability to use all of the industrially-oriented expansion cards available for the particular backplane-bus.

Such "hybrid bus" systems satisfy the ruggedness, reliability, and quality requirements of most embedded applications. STD and 3U VME, moreover, represent compact form-factors that can fit within the tight space constraints of many embedded systems. However, full "PC-compatibility" has not always been achieved in practice, due to the hybrid nature of these system architectures. A further disadvantage is the inability to fully leverage the PC architecture by directly using readily-available PC- or AT-bus expansion cards and their associated drivers and support software.

"BUSLESS" SINGLE-BOARD COMPUTERS

One way to create compact, reliable, fully compatible, and cost effective embedded-PC's is to incorporate all the essential ingredients of a complete PC-compatible *system* on a single board. This eliminates much of the bulk, weight, and costs associated with most multiple-board solutions. Although these products generally contain all of the standard "computer I/O" functions, such as serial, parallel, keyboard, and disk controller interfaces, they also typically provide a means for adding application-specific interfaces or functions, such as a mezzanine bus.

While the size of a busless single-board computer is entirely arbitrary, there is growing convergence on two form-factors: 5.75" x 8", which is based on the 5.25" disk drive footprint; and 100 x 160 mm, which corresponds to the 3U Eurocard format.

The main advantages of busless single-board computers are reduction in the number of boards and elimination of electrical and mechanical interface "glue" (e.g. backplanes and card cages). These products are most desirable when they closely match the application's requirements. On the other hand, they diminish in usefulness when a lot of additional functionality must be added externally, or when they contain too many unwanted functions.

COMPACT EMBEDDED-PC MODULES

An emerging standard that is extremely well suited to embedded applications is the miniaturized version of the PC- or AT-bus known as PC/104. Basically, PC/104 only changes the dimensions; all other PC or AT system features and functions are retained. The result is compact, component-like, and highly modular.

Named for its 64-pin (P1) plus 40-pin (P2) bus connectors, the PC/104 standard is being proposed as an extension to the IEEE-P996 specification (P996.1). The major differences from the normal PC- and AT-buses are: (1) highly compact form factor, 3.6 by 3.8 inches; (2) self-stacking bus, allowing modules to be stacked directly together without backplanes, card cages, or interconnecting adapters; and (3) relaxed bus drive requirement (6 mA), which lowers power consumption and reduces component count.

At about ten square inches in footprint, PC/104 modules might still be too large for the more space-sensitive embedded applications. Also, when large amounts of I/O are required, either a passive backplane PC or a hybrid industrial bus may offer greater ease of expandability.

SUMMARY AND CONCLUSIONS

Through their extremely widespread use and availability, the IBM PC and PC/AT (and compatibles) have established the PC architecture as the most popular and well-supported microcomputer system standard. The use of this popular hardware/software architecture has become attractive to designers of non-desktop "embedded" microprocessor-based equipment, partly because of the attractiveness of low cost components, peripherals, and development tools, but also as a means to avoid the need to "reinvent the wheel".

The PC architecture is not ideal for all embedded applications. Its expansion bus speed and interrupt structure may be inadequate for some high bandwidth I/O intensive applications, but will not be a problem in most cases. On the other hand, PC CPU horsepower spans over two

orders of magnitude, with continual evolutionary improvements promised for the foreseeable future.

Although standard desktop PCs are ill-suited for the reliability, size, quality, and other requirements of embedded applications, a number of excellent repackaged PC-compatible alternatives are now available. These range from passive backplane versions which match the PC-bus plug-in card form factor, to compact, component-like modules specifically intended for embedded-PC applications. Included within some of the board- and module-level PCs intended for embedded applications, are a wide range of useful system enhancements. Examples include solid state disk (SSD), BIOS improvements, and watchdog timer functions.

The result is a well stocked tool chest of building blocks which offer the benefits of using the PC standard along with the flexibility, reliability, and quality needed in embedded applications.

Although not all embedded applications are suited to the PC architecture, when the fit is right, the benefits are compelling: reduced development costs; minimized technical risks; and faster time-to-market.

BIBLIOGRAPHY

The following publications contain additional information about PC- and PC/AT-compatible hardware, software, development tools, and peripheral devices that may be helpful in designing systems based on embedded-PC technology:

IEEE P996 Draft Specification — IEEE Publications, 908-981-1393.

The XT/AT Handbook — Annabooks, 619-673-0870.

ISA and EISA Theory and Operation — Annabooks, 619-673-0870.

PC/104 Specification — PC/104 Consortium, 408-245-9348.

The Connection (software buyer's guide), 800-336-1166.

The Programmer's Shop (software buyer's guide), 800-421-8006.

Personal Computing Tools (hardware buyer's guide), 800-767-6728.

Personal Engineering (hardware magazine), 603-427-1377.

The Computer Applications Journal (hardware magazine), 203-875-2751.

Embedded Systems Programming (software magazine), 415-905-2200.

Dr. Dobb's Journal (software magazine), 415-358-9500.

APPLICATION NOTE

NUMBER: AAN-9202 REVISION: B DATE: 7/29/93

TITLE: Object-Oriented Hardware Design for Embedded Systems

PRODUCT: Non-specific

SUMMARY: A discussion of how Ampro's embedded microcomputer products facilitate modular, object-oriented system architectures, and the benefits of using such an approach.

See attached 6 page document

AMPRO®

AMPRO COMPUTERS, INCORPORATED
990 Almanor Avenue, Sunnyvale, California 94086
TEL (408) 522-2100 - FAX (408) 720-1305

990 Almanor Ave. ▪ Sunnyvale, CA 94086 ▪ Phone 408-522-2100 ▪ Fax 408-720-1305

OBJECT-ORIENTED *HARDWARE* DESIGN FOR EMBEDDED SYSTEMS

Rick Lehrbaum
Executive Vice President, Strategic Development
Ampro Computers, Inc.

In designing microcomputer-based systems, size, power, cost, ruggedness, reliability, and other constraints often preclude using off-the-shelf bus-based microcomputer boards, resulting in the need for chip-based custom designs. This, in turn, means costly development cycles, including significant expenses for PC board design, BIOS development, debug equipment, system qualification, and manufacturing tooling. And perhaps most importantly, with the complex development cycles of chip-based designs also come inevitable schedule slips and delayed product introductions.

During the past several years, *object-oriented software* development technologies and methods have achieved widespread acceptance. It is now universally acknowledged that significant development time is saved — and greater functionality achieved — when software design takes advantage of high-level-languages and object-oriented, structured programming techniques.

Hardware development, on other hand, has lagged in the use of such methodologies, particularly for embedded applications. Factors like size, power consumption, ruggedness, reliability, and quality usually prevent embedded system designers from using off-the-shelf options. Instead, chip-based custom designs, with their costly and time-consuming product development cycles, have been required. In addition, these complex custom development programs have carried substantial risks of schedule slips and missed market opportunities.

Thanks to highly integrated components and increasingly dense manufacturing technologies, this situation is beginning to change. Modular approaches to hardware integration, which can substantially simplify development efforts, are now becoming practical.

THE HARDWARE-SOFTWARE ANALOGY

During a period of about two decades, *software technology* has evolved from binary coding, to assembly language, to first generation high-level languages like Fortran and Basic, to second generation *object-oriented* languages like Pascal, C, and C++.

Over the same span of time, embedded system *hardware technology* has undergone a comparable evolution: from transistor-based design, to small-scale and medium-scale integration (SSI and MSI) single-function ICs, to large-scale and very-large-scale integration (LSI and VLSI) multiple-function ICs, to complex modules that can be used as high-level *object-oriented* embedded system building blocks.

In this analogy, using transistors is like writing software in binary; each "1" and "0" is like an individual transistor. This represents the lowest-level technology option. Like assembly language, SSI and MSI ICs are relatively simple building blocks. LSI and VLSI ICs offer higher levels of functionality and greatly simplify the design and development process, as do the early programming languages; but they do not offer direct plug-and-play interconnectivity, from either a functional or mechanical perspective.

Highly integrated hardware *modules* can provide an *object-oriented*, building-block approach to embedded hardware design, much like the structured-programming techniques used with today's object-oriented programming languages. They can be used like electronic legos, to rapidly design and construct application-specific embedded systems. When custom modules are needed, the module family's well defined functional and mechanical specifications can greatly simplify the required design tasks. Also, concurrent development — of both hardware and software — is facilitated by the inherent modularity of this object-oriented approach.

AN EXAMPLE OF OBJECT-ORIENTED HARDWARE: PC/104

A new modular embedded-microcomputer standard, called "PC/104", takes advantage of object-oriented hardware techniques. The PC/104 standard defines "snap-together" modules intended to be used as system-level building-blocks for designing embedded systems (see Figure 1).

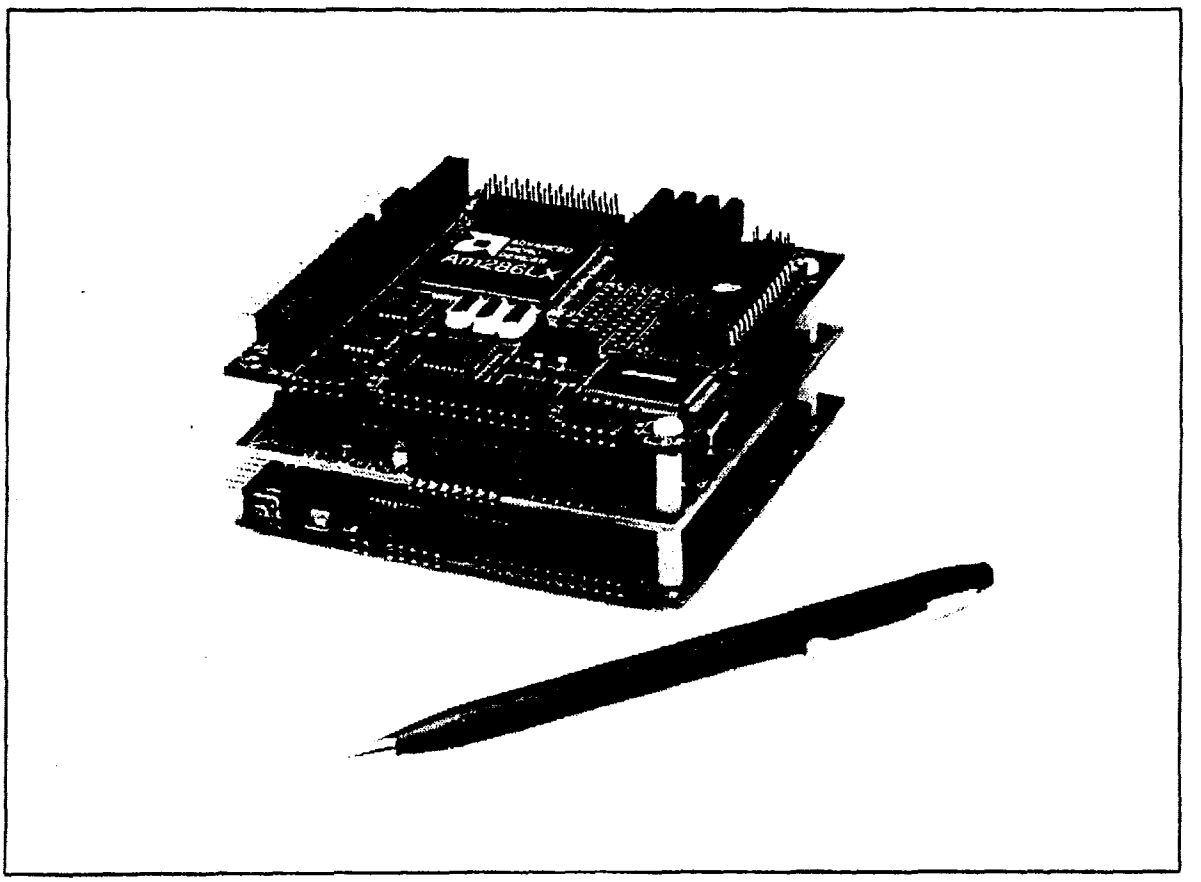


Figure 1. Object-oriented PC/104 Modules Provide a Building-block approach to Embedded System Design

To satisfy the size, reliability, and environmental demands of embedded applications, PC/104 modules conform to a compact form-factor (3.6 by 3.8 inches), stack directly without card-cages or backplanes, and minimize power consumption through extensive use of low power CMOS devices.

Because PC/104 modules are based on the functional and electrical specifications of the IEEE-P996 (PC and PC/AT Bus) draft specification, designs based on them benefit from the enormous base of products and support associated with this most familiar and widely supported microcomputer hardware/software architecture. Furthermore, the PC/104 spec itself is in the process of being standardized by the IEEE P996.1 Working Group.

PC/104 modules can be integrated into systems in two ways: they can be plugged into custom carrier boards, like macrocomponents; or they can be stacked together and mounted directly within the target system. Either way, using PC/104 modules greatly simplifies and accelerates the tasks of system definition and development, due to the resulting object-oriented approach.

BENEFITS OF OBJECT-ORIENTED HARDWARE

Some of the important benefits that an object-oriented hardware approach brings to embedded system product development and support are:

- **Shortened and Simplified Development Cycles** — With an object-oriented approach, much of the hardware development portion of the project may be reduced to selection and configuration of off-the-shelf modules, peripheral selection and interfacing, and mechanical design. This obviously results in big savings of development time and money, and reduced technical risks.

- **Enhanced Product Sophistication** — By using a building-block approach, more features can be incorporated into the product than might otherwise have been developed. Also, once the need to "reinvent the wheel" is eliminated, more of the development budget is often applied to product-specific *software* development. Together, these factors result in a more feature-rich, powerful, and competitive — and hence more profitable — end product.

- **Availability of Product Options and Upgrades** — With a module-based system architecture, the manufacturer can offer a spectrum of performance or features, simply by reconfiguring the base system with a variety of CPU and I/O modules. For example, the same basic product design could be fitted with either an 80286- or 80386-based CPU, allowing faster product operation with similar features, or enhancing features while maintaining acceptable performance. The capability to install a modem module or a LAN module in the same location might allow the product to be offered with a range of communications capabilities.

- **Protection from Component Obsolescence** — A critical problem facing all manufacturers is what to do when, inevitably, key components are obsoleted by their producers. This issue is exacerbated when using PC-compatible ICs, due to the rapid evolution of the desktop PC. Components for the desktop PC market are sometimes obsoleted in as little as 18 to 24 months, particularly by IC companies whose only business is desktop-oriented. Given the length of product development cycles (typically 12 to 18 months), a company using PC-compatible ICs shouldn't be surprised to receive

a last-buy notification for a key component just as a new product incorporating that component is being released for production!

Since the modules within an object-oriented product architecture are essentially "black boxes", specified by their "external" physical, functional, and electrical specifications, alternate modules having different internal components but identical external characteristics can be substituted. It becomes the module manufacturer's responsibility to ensure uninterrupted availability of functionally compatible modules — a commitment that suppliers of modules to the OEM marketplace must be prepared to make.

An object-oriented product architecture thus provides a much-needed insurance policy against the ever-present nightmare of component obsolescence.

- **Lengthened Product Life Expectancy** — Finally, products based on object-oriented design methodologies can be upgraded during their life cycle, to keep them competitive and maintain acceptable price-performance ratios. Possible upgrades include increased computing power, additional or improved I/O functions or other system resources, software enhancements, and cost reductions. In this manner, a product based on an object-oriented architecture can continue to evolve over time, resulting in an extended product life cycle and, of course, increased revenues.

FINANCIAL ISSUES

In comparing the chip-based and module-based approaches from a financial perspective, both methodologies are seen to have advantages and disadvantages. Either approach may be best, depending on the specific application and situation.

The three main factors to evaluate when choosing an approach are material costs, development expenses, and time-to-market.

- **Material Costs** — Off-the-shelf modules generally do not provide an exact match to system requirements, and therefore contain excess components that add some amount of unnecessary cost. Also, using multiple modules instead of a single custom assembly containing all of the system's exact requirements increases costs due to added components, interconnections, and interfaces.

  The significance of these added costs depends on how closely the available modules match the application's requirements, and also on the annual quantity level at which the product will be manufactured. However, when annual quantities are not high, the module supplier's superior purchasing power can actually make the cost of buying modules lower than that of buying the individual components for a chip-based custom design, in spite of extra costs inherent in the modular approach.

- **Development and Time-to-Market Costs** — For both of these important factors, the advantage is clearly on the side of the module-based approach. Using modules, the hardware portion of the embedded system's design can be as simple as selecting, configuring, and plugging together the modules, and integrating them with peripherals and packaging. When application-specific modules must be developed, that process is also greatly simplified, thanks to the modules' well defined system specifications. Component-based designs, on the other hand, require lengthy and detailed development and

debug cycles, with accompanying high costs and risks.

SOLVING THE EQUATION

Obviously, all three factors — material costs, development expenses, and time-to-market — must be considered when determining which architecture to use. Although the factors and circumstances surrounding each company's business environment are unique, time-to-market is increasingly seen as the dominant factor.

Figure 2 shows a comparison of the relative impact of these three cost factors, at three different annual quantity levels. From examining the graphs, it is clear that each methodology tends to be best suited to a specific volume range.

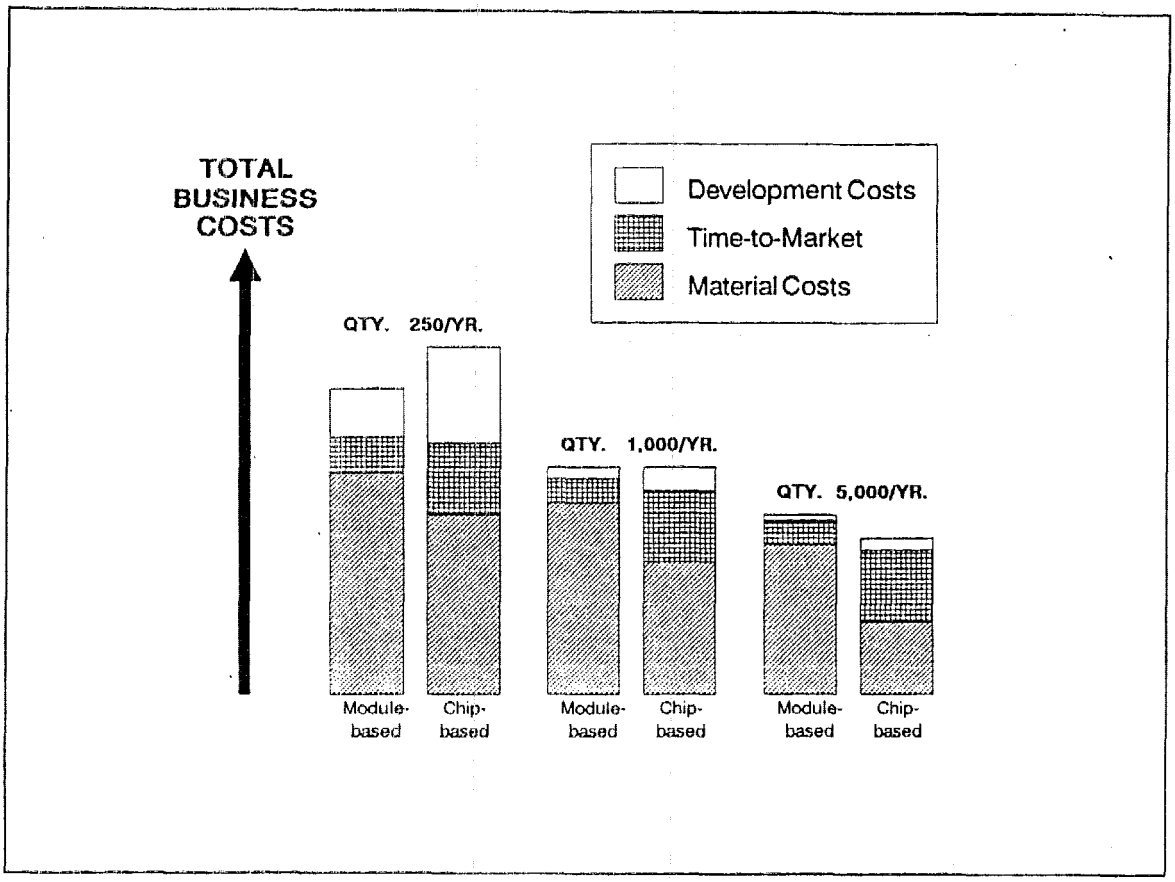


Figure 2. Relative Impact of Key Cost Factors

- **Low volumes** — At annual volumes below 250 units per year, the module-based approach generally wins, mainly due to the availability of off-the-shelf options which eliminate much of the cost and time of embedded electronics development. In these low volumes, it is difficult to justify a custom development project based on chips — unless modules just can't do the job.

- **High Volumes** — At the other end of the spectrum, for volumes in excess of 5,000 units annually, component-based designs provide rock bottom material costs. At these

high volumes, even a few dollars of cost savings can significantly increase corporate profits. Still, the potential penalties of extended development time and increased technical risks must not be overlooked.

- **Medium Volumes** — When annual volumes are around 1,000 units, the decision is not as clear-cut. At these quantities, the module-based approach has the advantage of significantly reducing the development expenses that must be amortized over the product's life, while both getting the product to market sooner and keeping it there longer. On the other hand, the component-based approach may offer lower material costs.

CONCLUSION

By combining widely accepted object-oriented *software* methodologies with newly available object-oriented *hardware* options, embedded system designers of the 90's can complete development projects faster and with significantly reduced costs and risks. Also, the lengthened product life cycle that results from both earlier market introduction and continuing product evolution (due to enhancements and upgrades) means revenues and profits realized by the product during its lifetime will be significantly increased.

In deciding whether to use a chip-based or module-based approach, it is important that companies weigh all key factors thoroughly, being sure to include a careful consideration of the anticipated annual volume requirements of the product. Not surprisingly, today's increased emphasis on maximizing profits and minimizing risks has shifted the break-even point in the make-or-buy decision towards object-oriented, module-based approaches, even for high volumes in excess of 5,000 per year.

The advent of *object-oriented hardware* therefore represents a significant advancement in the art of embedded system design.

APPLICATION NOTE

NUMBER: AAN-9204 **REVISION:** A **DATE:** 12/2/92

TITLE: Third Party Sources of PC Keyboard Simulators

PRODUCT: Non-specific

The following companies offer keyboard "simulators" which are intended to allow scanned pushbuttons to emulate a standard PC keyboard:

- USAR Systems -- New York, NY -- 212-222-5438
- VG Controls -- Hewitt, NJ -- 201-853-4600
- Vetra Systems -- Plainview, NY -- 516-454-6469

In addition, the following publication may be of interest:

- **PC Keyboard Design**, by Gary J. Konzak. Available from Annabooks -- 800-462-1042.

Please Note

This information is provided as a service to Ampro OEMs, and is not intended as an endorsement of any particular company or product. Ampro has not necessarily tested the products offered by the companies listed. It is the OEM's responsibility to determine the quality, suitability, reliability, and compatibility of any particular product according to the requirements of the specific intended application.

AMPRO®
AMPRO COMPUTERS, INCORPORATED
990 Almanor Avenue, Sunnyvale, California 94086
TEL (408) 522-2100 - FAX (408) 720-1305

APPLICATION NOTE

NUMBER: AAN-9205 **REVISION:** A **DATE:** 12/2/92

TITLE: 4-Point Mounting of MiniModule Stacks

PRODUCT: Non-specific

CoreModule CPU modules were provided with four mounting holes to facilitate reliable installation of CoreModule/MiniModule stacks. In some applications, however, it may be preferred to install the CoreModule at the top of the stack rather than in the bottom position. Also, there are applications in which multiple MiniModule stacks -- without a CoreModule in the stack -- may be required.

A recommended method for 4-point mounting of MiniModules is to use edge-mount standoffs, such as those manufactured by Richco. For reliable 4-point mounting, you can combine a pair of edge-mount standoffs (positioned near the bus edge of the module) along with two ordinary hole-mount standoffs (used in the module's mounting holes).

Matched pairs (edge-mount, and hole-mount) are listed in the following table.

LENGTH (IN)	RICHCO EDGE-MOUNT STANDOFF	RAF HOLE-MOUNT STANDOFF
1/4	TCEHCBS-4-01	1633-440-N
3/8	TCEHCBS-6-01	1635-440-N
1/2	TCEHCBS-8-01	1637-440-N
5/8	TCEHCBS-10-01	1639-440-N
3/4	TCEHCBS-12-01	1641-440-N
7/8	TCEHCBS-14-01	1643-440-N

The spacers listed in the above table are available from the following manufacturers:

- RICHCO Plastic Company -- Chicago, IL -- 312-539-4060
- RAF Electronic Hardware -- Seymour, CT -- 203-888-2133

Please Note

This information is provided as a service to Ampro OEMs, and is not intended as an endorsement of any particular company or product. Ampro has not necessarily tested the products offered by the companies listed. It is the OEM's responsibility to determine the quality, suitability, reliability, and compatibility of any particular product according to the requirements of the specific intended application.

AMPRO®
AMPRO COMPUTERS, INCORPORATED
990 Almanor Avenue, Sunnyvale, California 94086
TEL (408) 522-2100 - FAX (408) 720-1305

APPLICATION NOTE

NUMBER: AAN-9206 **REVISION:** A **DATE:** 12/2/92

TITLE: Third Party Sources of Monitors

PRODUCT: Non-specific

The following companies offer CRT displays that may be useful in CoreModule and Little Board based OEM applications:

- Data Ray Corp. -- Westminster, CO -- (303) 451-1300
- Display Technologies, Inc. -- Elgin, IL -- (708) 931-2100
- Dotronix -- New Brighton, MN -- (612) 633-1065
- Kristel Corp. -- West Chicago, IL -- (708) 293-1255
- Microvitec, Inc. -- College Park, GA -- (404) 991-2246
- Modgraph -- Burlington, MA -- (617) 229-4800
- Omni Vision, Inc. -- Glendale Heights, IL -- (708) 893-1720
- Trans 2000 Inc. -- Whittier, CA -- (310) 908-6814
- Wells-Gardner Electronics -- Chicago, IL -- (312) 252-8220

Please Note

This information is provided as a service to Ampro OEMs, and is not intended as an endorsement of any particular company or product. Ampro has not necessarily tested the products offered by the companies listed. It is the OEM's responsibility to determine the quality, suitability, reliability, and compatibility of any particular product according to the requirements of the specific intended application.

AMPRO®
AMPRO COMPUTERS, INCORPORATED
990 Almanor Avenue, Sunnyvale, California 94086
TEL (408) 522-2100 - FAX (408) 720-1305

APPLICATION NOTE

NUMBER: AAN-9207 **REVISION:** A **DATE:** 12/2/92

TITLE: Third Party Sources of Touch Input Devices

PRODUCT: Non-specific

The following companies offer touch input interfaces that might be useful in CoreModule and Little Board based applications:

- Carroll Touch -- Round Rock, TX -- (512) 244-7040
- Dale Electronics, Inc. -- Columbus, NE -- (402) 563-6413
- Elographics -- Oak Ridge, TN -- (615) 481-6301
- The Graphics Technology Co. -- Austin, TX -- (512) 328-9284
- Keytec, Inc. -- Richardson, TX -- (214) 234-8617
- Memtron Technologies -- Frankenmuth, MI -- (517) 652-2656
- Microtouch Systems -- Woburn, MA -- (617) 935-0080
- Transparent Devices -- Newbury Park, CA -- (805) 499-5000

Please Note

This information is provided as a service to Ampro OEMs, and is not intended as an endorsement of any particular company or product. Ampro has not necessarily tested the products offered by the companies listed. It is the OEM's responsibility to determine the quality, suitability, reliability, and compatibility of any particular product according to the requirements of the specific intended application.

AMPRO®

AMPRO COMPUTERS, INCORPORATED
990 Almanor Avenue, Sunnyvale, California 94086
TEL (408) 522-2100 - FAX (408) 720-1305

APPLICATION NOTE

NUMBER: AAN-9208 **REVISION:** A **DATE:** 12/2/92

TITLE: Third Party Sources of Real Time Executives

PRODUCT: Non-specific

There are a wide range of real-time operating systems and executives that may be useful in developing embedded systems based on the Ampro CoreModule and Little Board products. Ampro Application Note AAN-9009 lists and discusses several real-time operating systems. This writeup lists real-time executives.

The main difference between an *executive* and an *operating system* is generally the complexity of resources provided. This impacts their overall software size and therefore the amount of RAM and disk (or program) storage required. Executives typically do not provide mass storage or networking support; these are usually features of operating systems. Also, because of the limited functionality offered by executives, they are generally less expensive than operating systems and may even be offered for a one-time payment. In addition, source code is sometimes available for executives.

The following real-time multi-tasking executives support 80X86 processors and provide driver (or other) support for the PC and PC/AT environment. Most include drivers for the standard serial, parallel, keyboard, and speaker functions. Some include disk drive support. Others provide a mechanism that allows DOS to run as a background task, which allows use of standard DOS functions (including the file system, disk drives, etc.).

- **AMX:** Kadak Products Ltd. -- Vancouver, BC, Canada -- 604-734-2796
- **OS-9000:** Microware Systems Corp. -- Des Moines, IA -- 515-224-1929
- **Multitask! 80x86:** US Software -- Portland, OR -- 503-641-8446
- **Nucleus RTX:** Accelerated Technology -- Mobile, AL -- 205-450-0707
- **Precise/MPX:** Precise Software Technologies -- Nepean, ON, Canada -- 613-596-2251
- **QNX:** Quantum Software -- Kanata, ON, Canada -- 613-591-0931
- **RTXC:** A.T.Barrett & Assoc. -- Houston, TX -- 713-728-9688
- **SMX:** Micro Digital -- Cypress, CA -- 714-891-2363
- **TICS Realtime Kernel:** TICS Realtime -- Sunnyvale, CA -- 408-739-2100
- **VRTX:** Ready Systems -- Sunnyvale, CA -- 408-736-2600

Please Note

This information is provided as a service to Ampro OEMs, and is not intended as an endorsement of any particular company or product. Ampro has not necessarily tested the products offered by the companies listed. It is the OEM's responsibility to determine the quality, suitability, reliability, and compatibility of any particular product according to the requirements of the specific intended application.

AMPRO®

AMPRO COMPUTERS, INCORPORATED
990 Almanor Avenue, Sunnyvale, California 94086
TEL (408) 522-2100 - FAX (408) 720-1305

APPLICATION NOTE

NUMBER: AAN-9209 **REVISION:** A **DATE:** 12/2/92

TITLE: Third Party Sources of TCP/IP and NFS Support Software

PRODUCT: MiniModule/Ethernet, MiniModule/Ethernet-TP

The following companies offer NFS and TCP/IP support for PC and AT compatible systems that can be used with Novell NE1000 compatible Ethernet LAN adapters. This implies that their software will run on both the Thin-cable and Twisted-pair versions of the Ampro MiniModule/Ethernet network adapters.

- **"BW-NFS" and "BW-TCP"**
  Beame & Whiteside Software Ltd.
  Dundas, ON Canada
  (416) 648-6556
- **"PC/TCP"**
  FTP Software Inc.
  Wakefield, MA USA
  (617) 246-0900
- **"PC-NFS"**
  Sun Microsystems
  (800) 872-4786

NOTES

1. The "standard" pricing normally quoted by these companies is intended for *end-user desktop PC* applications. Larger discounts for *OEM embedded applications* are usually available.
2. In most cases, the NE1000 hardware adapter compatibility is accomplished via a public domain NE1000 "NDIS" (or packet) driver, available from the software vendors.
3. The diskless boot ROMs provided by these companies are not typically compatible with NE1000 hardware. Use Ampro Solid State Disk instead.

Please Note

This information is provided as a service to Ampro OEMs, and is not intended as an endorsement of any particular company or product. Ampro has not necessarily tested the products offered by the companies listed. It is the OEM's responsibility to determine the quality, suitability, reliability, and compatibility of any particular product according to the requirements of the specific intended application.

AMPRO®
AMPRO COMPUTERS, INCORPORATED
990 Almanor Avenue, Sunnyvale, California 94086
TEL (408) 522-2100 - FAX (408) 720-1305

APPLICATION NOTE

NUMBER: AAN-9210 **REVISION:** A **DATE:** 12/2/92

TITLE: Ampro Extended BIOS Interface Specification

PRODUCT: Non-specific

SUMMARY: The Ampro extended BIOS services provide interfaces to the unique features and hardware present on Ampro CPU products. These features generally are enhancements to the original architectures and do not exist in standard IBM/PC/AT systems or compatibles. The BIOS services are all available via interrupt 13h. This interface is not available when the extended BIOS has been disabled in SETUP.

Interrupt 13h
Function 0C0h, Return SCSI Driver Status
Use: Return SCSI Initiator ID and SCSI bios revision.
Board Applicability: All

This function returns the level of SCSI Firmware but more importantly it is a means of identifying the boards SCSI Initiator ID from setup information. The SCSI Initiator ID value can be used to uniquely identify the board by application software. It is also needed to fill the SCSI parameter block structure defined in function 0C1h below.

```
Entry:    AH   = 0C0h

Returns:  AX   = 0FFFFh
          CX   = SCSI driver version
          CL   = Major version number
          CH   = Minor version number
          DL   = SCSI Initiator ID
          CY   = 1
```

NOTE: The processor carry flag will be set upon return.

AMPRO®
AMPRO COMPUTERS, INCORPORATED
990 Almanor Avenue, Sunnyvale, California 94086
TEL (408) 522-2100 - FAX (408) 720-1305

Interrupt 13h
Function 0C1h, Direct SCSI Command
Use: SCSI interface for device drivers and utilities.
Board Applicability: All

This function negates the requirement for embedding chip level hardware specific code in a SCSI utility or device driver. It is used by all Ampro developed SCSI utilities and device drivers. Custom device drivers to communicate with any SCSI device (TAPE,CD-ROM,WORM) can be developed using this function.

```
Entry:    AH   = 0C1h
          ES   = Segment of SCSI Parameter Structure
          BX   = Offset of SCSI Parameter Structure
                 (See Structure definition below)

Returns:  AL   = 0
          AH   = Status
          CF   = 0, command completed
          CF   = 1, command error
```

Status Codes returned in AH:

```
          00h  = Command completed
          FFh  = failed to complete all SCSI phases, lost busy
          FEh  = target did not respond
          FDh  = illegal SCSI phase
          FCh  = Arbitration timeout
```

NOTE: Application should also check status_ptr, msg_in_ptr for errors. SCSI hardware interface initializes msg_in_ptr to -1 prior to SCSI select sequence.

The LB/PC and LB/186 using the 5380 SCSI controller chip return with error status AH = FFh when SCSI fails.

SCSI parameter block structure definition.

```
spb             struc
initiator_id    db   ?    ; SCSI initiator ID
target_id       db   ?    ; SCSI target ID
                db   ?    ; Unused
                db   ?    ; Unused
command_ptr     dd   ?    ; SCSI command pointer
data_ptr        dd   ?    ; SCSI data pointer
status_ptr      dd   ?    ; SCSI status pointer
msg_in_ptr      dd   ?    ; SCSI message in pointer
msg_out_ptr     dd   ?    ; SCSI message out pointer
spb             ends
```

NOTE: Initiator_id and target_id are binary (ie. 80h = SCSI ID 7, 01h = SCSI ID 0).

```
Example:  mov  spb.initiator_id,my_iniator_id
          mov  spb.target_id,my_target_id

          mov  ax,cs
          mov  word ptr spb.command_ptr,offset my_scsi_cmd
          mov  word ptr spb.command_ptr+2,ax

          mov  word ptr spb.data_ptr,offset my_data_buffer
          mov  word ptr spb.data_ptr+2,ax

          mov  word ptr spb.status_ptr,offset my_status
          mov  word ptr spb.status_ptr+2,ax

          mov  word ptr spb.msg_in_ptr,offset my_msg_in
          mov  word ptr spb.msg_in_ptr+2,ax

          mov  word ptr spb.msg_out_ptr,offset my_msg_out
          mov  word ptr spb.msg_out_ptr+2,ax

          mov  es,ax            ; es:bx = spb
          mov  bx,offset spb

          mov  ah,0c1h          ; make function call
          int  13h

          test my_status,-1     ; see if error
          jnz  error
```

Interrupt 13h
Function 0C2h, Extended Read Logical Block
Use: SCSI interface for device drivers and utilities.
Board Applicability: All

This function call was developed for custom operating systems using large hard disk. Can be considered obsolete.

```
Entry:    AH   = 0C2h
          AL   = Sector count, 1 - 0ffh
          CX   = Sector, 1 - 03Fh
          DH   = Head, 0 - 0ffh
          SI   = Cylinder, 0 - 0ffffh
          ES   = Buffer Segment
          BX   = Buffer offset

Returns:  CY   = 0, command completed
          AH   = 0
          CY   = 1, error
          AH   = Status
```

Status: Same as returned by bios compatible disk interface.

NOTE: There is no range checking for overflow beyond logical block address 1FFFFFh.

Interrupt 13h
Function 0C3h, Extended Write Logical Block
Use: SCSI interface for device drivers and utilities.
Board Applicability: All

This function call was developed for custom operating systems using large hard disk. Can be considered obsolete.

```
Entry:    AH   = 0C3h
          AL   = Sector count, 1 - 0ffh
          CX   = Sector, 1 - 03fh
          DH   = Head, 0 - 0ffh
          SI   = Cylinder, 0 - 0ffffh
          ES   = Buffer Segment
          BX   = Buffer offset

Returns:  AH   = Status
          CY   = 0, command completed
          CY   = 1, error
```

Status: Same as returned by compatible bios disk interface.

NOTE: There is no range checking for overflow beyond logical block address 1FFFFFh.

Interrupt 13h
Function 0C4h - 0CBh, Future expansion
Use: N/A
Board Applicability: All

```
Entry:    AH   = 0C4h - 0CBh

Returns:  AH   = 1
          CY   = 1
```

Interrupt 13h
Function 0CCh, Ampro Serial EEPROM interface.
Use: Seep interface
Board Applicability: All except LB/PC

Most Ampro boards include a Serial EEPROM for storage of setup and bios configuration information. All systems allocate 512 bits of storage for OEM information. Functions 0, 2, and 3 are used to read and write to the OEM area. The other functions allow access to the entire serial EEPROM area. Writing to areas other than the OEM area can cause the board to fail initialization.

NOTE: Boards with 2k serial EEPROMS have a maximum address of 7Fh and a word count 0f 80h when using functions 0Ah, 0Ch, and 0Dh. OEM allocation remains 512 bits.

Function 00h, Return word from seep user area

```
Entry:    AH   = 0CCh
          AL   = 00h
          DX   = Address, 00h - 1Fh

Returns:  CY   = 0, no error
          AX   = Word
          CY   = 1
          AH   = error code, see below
```

Function 2, Read user seep into buffer

```
Entry:    AH   = 0CCh
          AL   = 02h
          CX   = Word count, 1 - 20h
          DX   = Starting address, 00h - 1Fh
          ES   = Buffer segment
          BX   = Buffer offset

Returns:  CY   = 0, no error
          CY   = 1, error
          AH   = error code, see below
```

Function 3, Write user seep from buffer

```
Entry:    AH   = 0CCh
          AL   = 03h
          CX   = Word count, 1 - 20h
          DX   = Starting address, 00h - 1Fh
          ES   = Buffer segment
          BX   = Buffer offset

Returns:  CY   = 0, no error
          CY   = 1, error
          AH   = error code, see below
```

Function 0Ah, Read word from seep

```
Entry:    AH   = 0CCh
          AL   = 0Ah
          DX   = Address, 00h - 3Fh

Returns:  CY   = 0, no error
          AX   = Word
          CY   = 1
          AH   = error code, see below
```

Function 0Ch, Read seep into buffer

```
Entry:    AH   = 0CCh
          AL   = 0Ch
          CX   = Word count, 1 - 40h
          DX   = Starting address, 00h - 3Fh
          ES   = Buffer segment
          BX   = Buffer offset

Returns:  CY   = 0, no error
          CY   = 1, error
          AH   = error code, see below
```

Function 0Dh, Write seep from buffer

```
Entry:    AH   = 0CCh
          AL   = 0Dh
          CX   = Word count, 1 - 40h
          DX   = Starting address, 00h - 3Fh
          ES   = Buffer segment
          BX   = Buffer offset

Returns:  CY   = 0, no error
          CY   = 1, error
          AH   = error code, see below
```

Function 0Fh, Check seep checksum

```
Entry:    AH   = 0CCh
          AL   = 0Fh

Returns:  CY   = 0, good checksum
          CY   = 1, error
          AH   = error code, see below
```

Error codes for function 0CCh, seep interface

```
          AH   = 1, illegal address requested
          AH   = 2, illegal word count
          AH   = 3, error accessing seep
          AH   = 4, checksum error
          AH   = -1, bad function call
```

Interrupt 13h
Function 0CDh, Ampro hardware interface.
Use: Device drivers, application software
Board Applicability: All

This interface was developed to allow application software to manipulate hardware features as available on Ampro boards. These function calls provide a non-hardware specific interface to the Ampro features.

Function 00h, Return board type
Board Applicability: All

```
Entry:    AH   = 0CDh
          AL   = 00h

Returns:  AH   = 0
          CY   = 0
          BX   = product, see below
          CX   = 8000h
```

Product codes returned in BX

```
0000h = LB/PC
1001h = LB/286
1002h = SB/286
2003h = LB/386
2004h = SB/386
0005h = CM/XT
1006h = CM/286
2008h = LB/386SX
2009h = CM/386SX
300Ah = LB/486
```

```
Example:  mov  ah,0cdh
          mov  al,00h          ; function CD00
          mov  cx,0            ; precondition cx
          int  13h
          jc   not_supported   ; error
          test cx,8000h        ; cx msb set ?
          jz   not_supported
```

Function 01h, Set LB/PC SCSI dma halt flag
Board Applicability: All

```
Entry:    AH   = 0CDh
          AL   = 01h

Returns:  AH   = 0
          CY   = 0

Example:  mov  ah,0cdh
          mov  al,01h          ; function CD01
          int  13h
          jc   not_supported   ; error
```

NOTE: The halt flag is only used on LB/PC. When this flag is set the processor is halted during SCSI dma providing maximum throughput. Proper operation depends on the jumpers W1 and W6 being installed.

Function 02h, Clear LB/PC SCSI dma halt flag
Board Applicability: All

```
Entry:    AH   = 0CDh
          AL   = 02h

Returns:  AH   = 0
          CY   = 0

Example:  mov  ah,0cdh
          mov  al,02h          ; function CD02
          int  13h
          jc   not_supported   ; error
```

NOTE: The halt flag is only used on LB/PC. When this flag is cleared the processor will poll the SCSI and dma chips for completion.

Function 03h, Byte wide socket 0 enable/disable
Board Applicability: All except LB/PC

```
Entry:    AH   = 0CDh
          AL   = 03h
          BL   = 0, disable socket
          BL   = 1, enable socket
          BH   = xxxx0000b, where xxxx = socket page

Returns:  CY   = 0
```

```
Example1: mov   ah,0cdh
          mov   al,03h          ; function CD03
          mov   bx,0001h        ; enable socket 0 (s0)
          int   13h
          jc    not_supported   ; error

Example2: mov   ah,0cdh
          mov   al,03h          ; function CD03
          mov   bx,0000h        ; disable socket 0 (s0)
          int   13h
          jc    not_supported   ; error
```

NOTE: Not applicable to LB/PC. Socket paging of devices larger than 64k is only supported by those boards with 32 pin byte wide EPROM sockets.

Function 04h, Byte wide socket 1 enable/disable
Board Applicability: LB/SB/286, LB/SB/386, CM/XT

```
Entry:    AH    = 0CDh
          AL    = 04h
          BL    = 0, disable socket
          BL    = 1, enable socket
          BH    = xxxx0000b, where xxxx = socket page

Returns:  CY    = 0

Example1: mov   ah,0cdh
          mov   al,04h          ; function CD04
          mov   bx,0001h        ; enable socket 1 (s1)
          int   13h
          jc    not_supported   ; error

Example2: mov   ah,0cdh
          mov   al,04h          ; function CD04
          mov   bx,0000h        ; disable socket 1 (s1)
          int   13h
          jc    not_supported   ; error
```

NOTE: Not applicable to LB/PC or CM/286. Socket paging of devices larger than 64k is only supported by those boards with 32 pin byte wide EPROM sockets.

Function 05h, Serial port 0 enable/disable
Board Applicability: LB/SB/286, LB/SB/386

```
Entry:    AH    = 0CDh
          AL    = 05h
          BL    = 0, disable serial port
          BL    = 1, enable serial port

Returns:  CY    = 0
```

```
Example1: mov   ah,0cdh
          mov   al,05h          ; function CD05
          mov   bx,0001h        ; enable serial port 0
          int   13h
          jc    not_supported   ; error

Example2: mov   ah,0cdh
          mov   al,05h          ; function CD05
          mov   bx,0000h        ; disable serial port 0
          int   13h
          jc    not_supported   ; error
```

Function 06h, Serial port 1 enable/disable
Board Applicability: LB/SB/286, LB/SB/386

```
Entry:    AH    = 0CDh
          AL    = 06h
          BL    = 0, disable serial port
          BL    = 1, enable serial port

Returns:  CY    = 0

Example1: mov   ah,0cdh
          mov   al,06h          ; function CD06
          mov   bx,0001h        ; enable serial port 1
          int   13h
          jc    not_supported   ; error

Example2: mov   ah,0cdh
          mov   al,06h          ; function CD06
          mov   bx,0001h        ; disable serial port 1
          int   13h
          jc    not_supported   ; error
```

Function 07h, Parallel port enable/disable
Board Applicability: LB/SB/286, LB/SB/386

```
Entry:    AH    = 0CDh
          AL    = 07h
          BL    = 0, disable parallel port
          BL    = 1, enable parallel port

Returns:  CY    = 0
```

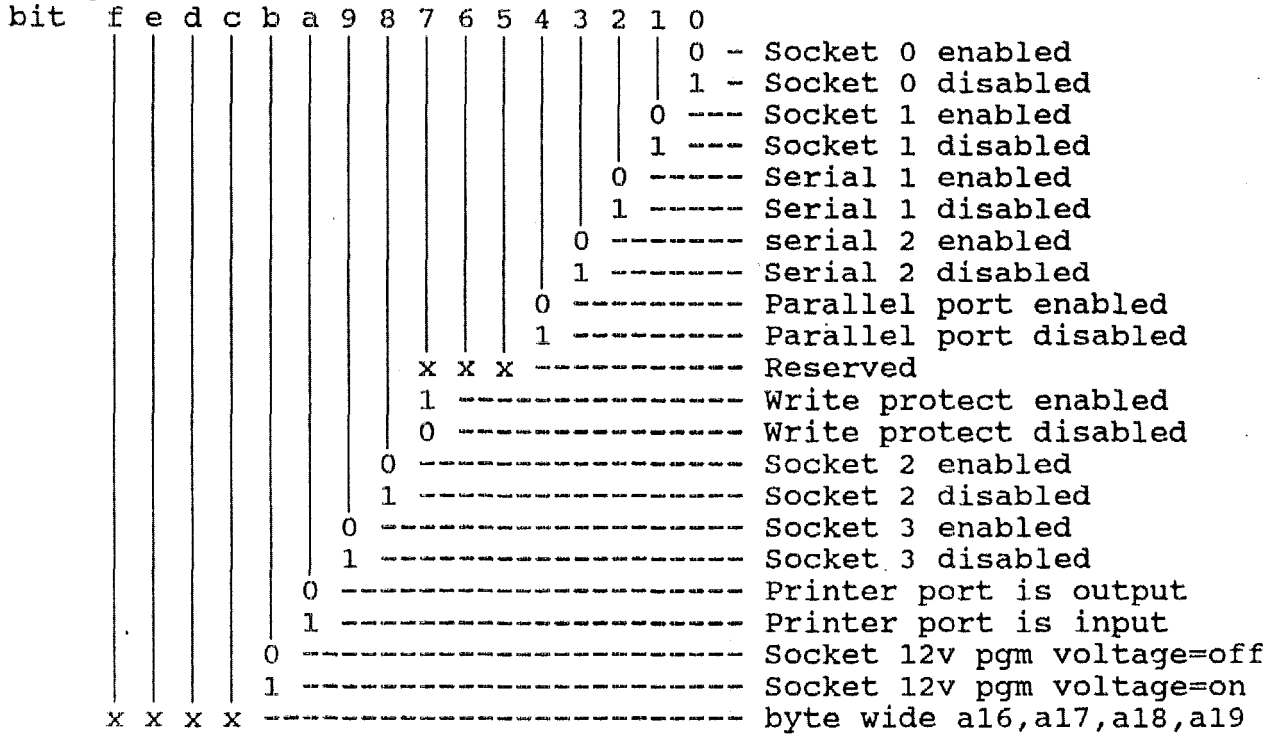
```
Example1: mov   ah,0cdh
          mov   al,07h          ; function CD07
          mov   bx,0001h        ; enable parallel port
          int   13h
          jc    not_supported   ; error

Example2: mov   ah,0cdh
          mov   al,07h          ; function CD07
          mov   bx,0000h        ; disable parallel port
          int   13h
          jc    not_supported   ; error
```

Function 08h, Return Peripheral Status
Board Applicability: LB/SB/286, LB/SB/386, CM/XT, CM/286

Entry: AH = 0CDh
AL = 08h

Returns: AX = Peripheral Status

Peripheral Status:



```
Example:  mov   ah,0cdh
          mov   al,08h          ; function CD08
          int   13h
          jc    not_supported   ; error
```

Function 09h, Socket 12 volt programing control
Board Applicability: CM/XT, CM/286

```
Entry:    AH    = 0CDh
          AL    = 09h
          BL    = 0, 12v disabled
          BL    = 1, 12v enabled

Returns:  CY    = 0

Example1: mov   ah,0cdh
          mov   al,09h          ; function CD09
          mov   bx,0001h        ; turn on 12v socket vpp
          int   13h
          jc    not_supported   ; error

Example2: mov   ah,0cdh
          mov   al,09h          ; function CD09
          mov   bx,0000h        ; turn off 12v socket vpp
          int   13h
          jc    not_supported   ; error
```

Function 0Ah, Byte wide socket 2 enable/disable
Board Applicability: SB/286, LB/486

```
Entry:    AH    = 0CDh
          AL    = 0Ah
          BL    = 0, socket 2 disabled
          BL    = 1, socket 2 enabled

Returns:  CY    = 0

Example1: mov   ah,0cdh
          mov   al,0ah          ; function CD0A
          mov   bx,0001h        ; enable socket 2 (s2)
          int   13h
          jc    not_supported   ; error

Example2: mov   ah,0cdh
          mov   al,0ah          ; function CD0A
          mov   bx,0000h        ; disable socket 2 (s2)
          int   13h
          jc    not_supported   ; error
```

```
Function 0Bh, Byte wide socket 3 enable/disable
Board Applicability: SB/286

Entry:     AH    = 0CDh
           AL    = 0Bh
           BL    = 0, socket 3 disabled
           BL    = 1, socket 3 enabled

Returns:   CY    = 0

Example1:  mov   ah,0cdh
           mov   al,0bh          ; function CD0B
           mov   bx,0001h        ; enable socket 3 (s3)
           int   13h
           jc    not_supported   ; error

Example2:  mov   ah,0cdh
           mov   al,0bh          ; function CD0B
           mov   bx,0000h        ; disable socket 3 (s3)
           int   13h
           jc    not_supported   ; error
```

```
Function 0Ch, Parallel port direction control
Board Applicability: SB/286, SB/386, CM/XT, CM/286

Entry:     AH    = 0CDh
           AL    = 0Ch
           BL    = 0, parallel port is output
           BL    = 1, parallel port is input

Returns:   CY    = 0

Example1:  mov   ah,0cdh
           mov   al,0ch          ; function CD0C
           mov   bx,0000h        ; set parallel port to output
           int   13h
           jc    not_supported   ; error

Example2:  mov   ah,0cdh
           mov   al,0ch          ; function CD0C
           mov   bx,0001h        ; set parallel port to input
           int   13h
           jc    not_supported   ; error
```

```
Function 0Dh, Get Socket Window and Size
Board Applicability: CM/286, CM/386SX, LB/386SX, LB/486

Entry:    AH   = 0CDh
          AL   = 0Dh
          BX   = Socket number, 0,1,2,3,etc.

Returns:  CY   = 0
          BX   = Socket Segment Address
          CX   = Socket Window Size in Kbytes

Example:  mov  ah,0cdh
          mov  al,0dh          ; function CD0D
          mov  bx,0            ; get socket 0
          int  13h
          jc   not_supported   ; error
```

```
Function 0Eh, Socket write protect
Board Applicability: CM/XT,CM/286,CM/386SX,LB/386SX,LB/486

Entry:    AH   = 0CDh
          AL   = 0Eh
          BL   = 0, disable write protect
          BL   = 1, enable write protect

Returns:  CY   = 0

Example1: mov  ah,0cdh
          mov  al,0eh          ; function CD0E
          mov  bx,0001h        ; write protect sockets
          int  13h
          jc   not_supported   ; error

Example2: mov  ah,0cdh
          mov  al,0eh          ; function CD0E
          mov  bx,0000h        ; disable write protect
          int  13h
          jc   not_supported   ; error
```

```
Function 0Fh, Enter bios setup
Board Applicability: LB/SB/286, LB/SB/386, CM/XT, CM/286

Entry:    AH   = 0CDh
          AL   = 0Fh

Returns:  AX   = 0
          CY   = 0

Example:  mov  ah,0cdh
          mov  al,0fh          ; function CD0F
          int  13h             ; bios setup
```

```
Error codes for Function 0CDh
     Status     CY    = 0, command completed
                CY    = 1, error
                AH    = -1, error bad command
```

```
Interrupt 13h
Function 0CEh, Return SCSI pointers
Use: Device drivers, application software
Board Applicability: All

Obsolete. This function was used by early versions of setup and
remains for compatibility.

Entry:     AH    = 0CEh

Returns:   DI    = offset of pointer to scsi_phy_tab
           SI    = offset of pointer to scsi_device_map
           DS    = pointer segment
           AH    = 0

scsi_phy_tab points to a structure of physical drive characteristics
for each hard disk drive in system. See hdp below.

hdp        struc
drv_heads  dw    ?          ; number of heads drive 0
drv_sect   dw    ?          ; number of sectors
drv_cyl    dw    ?          ; number of cylinders
           dw    3 dup (?)  ; drive 1
           dw    3 dup (?)  ; drive 2
           dw    3 dup (?)  ; drive 3
           dw    3 dup (?)  ; drive 4
           dw    3 dup (?)  ; drive 5
           dw    3 dup (?)  ; drive 6
           dw    3 dup (?)  ; drive 7
hdp        ends

scsi_device_map points to a structure of SCSI ID's and LUN's for each
hard disk device in system. See below.

           db    scsi_id,lun     ; drive 0
           db    scsi_id,lun
           db    scsi_id,lun
           db    scsi_id,lun
           db    scsi_id,lun
           db    scsi_id,lun
           db    scsi_id,lun
           db    scsi_id,lun     ; drive 7
```

Interrupt 13h
Function 0CEh, Issue SCSI reset
Use: Device drivers, application software
Board Applicability: All

This function provides a non-hardware specific means of issuing a SCSI bus reset.

Entry: AH = 0CFh

Returns: AH = 0

— End —

APPLICATION NOTE

NUMBER: AAN-9401 **REVISION:** D **DATE:** 11/28/95

TITLE: A Mean Time Between Failure analysis of selected Ampro products.

PRODUCT: Various

SUMMARY: This document contains a summary of Mean Time Between Failure (MTBF) values computed for the products covered, at three temperatures (25°, 55°, and 70° C), and for the four most appropriate environments defined by the MIL 217F handbook.

INTRODUCTION

Predicted component reliability is often characterized by a specification known as Mean Time Between Failure (MTBF). The MTBF value for a particular device or component indicates the expected operating time before some part of the device or component fails, requiring repair or service.

It is important to understand that MTBF values from various manufacturers are not consistently determined, even though everyone refers to the US Government standard relating to MTBF: MIL-HDBK-217F. Unfortunately, this results in MTBF specifications being *relative*, rather than *absolute* parameters. It is therefore difficult to compare Ampro's product MTBF specifications with those of a competitor, since each company uses its own assumptions and techniques to produce MTBF values.

We have endeavored to use as accurate a means as possible to generate our predicted MTBF values. A wide range of data points based on a variety of temperature and environmental alternatives allows engineers to use the data as an aid in system design.

The MTBF figures presented here have been updated from a previous Application Note. The differences in the figures are due to updated information on components and a change in the US Government standard, MIL-HDBK-217F. Figures in the previous Application Note were based on an older version of the handbook, MIL-HDBK-217E.

AMPRO®

AMPRO COMPUTERS, INCORPORATED
990 Almanor Avenue, Sunnyvale, California 940E
TEL (408) 522-2100 - FAX (408) 720-1305

MEANS OF OBTAINING THE FIGURES

The MTBF figures presented in the tables which follow are based upon a combination of the most current information available from component manufacturers, data obtained from Ampro Technical Service, and a commercially available component reliability library. A computer program, "RelCalc2" from T-Cubed Systems, was used to process the information into a report on each product. The RelCalc2 program uses the *component stress analysis prediction procedure* set forth in MIL-HDBK-217F, 2 December, 1991, "Reliability Prediction of Electronic Equipment".

Several assumptions were made in using this method of calculation:

- **Correction Factors** -- *Non-military* qualified parts are typically given as little as 1/80th of the life expectancy of the top rated *military* qualified parts (MIL-M-38510, Class S). This was found to result in the generation of MTBF numbers that fell far short of real world data, both from the point of view of Ampro product service history, and from our suppliers' projected and empirical data. Therefore, to make the MTBF computations consistent with component manufacturers' data, we divided the component manufacturer's MTBF numbers by the computed MTBF's (at the appropriate temperature) to obtain "correction factors". These correction factors were entered into the program data base and the MTBF numbers were recalculated. In some cases, adjustment of the correction factor was insufficient to bring the MTBF in line with the manufacturer's published figures. For these devices, the "part quality factor" was increased by as many levels as necessary to produce a component Failure Rate which corresponded to that supplied by the manufacturer of the component. Both "correction factors" and "part quality factors" are variables in the equations specified by MIL-HDBK-217F.

- **Component Approximations** -- Many IC manufacturers were unable to supply MTBF values for their components, or could not even provide the data required by RelCalc2 to perform an MTBF computation. In those cases, we made assumptions for certain variables such as die complexity and package thermal characteristics. The results of the calculations for these parts were compared to parts with known MTBF as a means of error checking.

Ampro is engaged in a continuing effort to gather accurate data from IC and component manufacturers. As new data is acquired it will be integrated into the MTBF tabulations, which will be updated periodically.

MIL-HDBK-217F ENVIRONMENTS

Ampro products are used in many diverse applications. Most MTBF figures quoted in the industry are calculated for controlled, "benign" environments. This produces an attractive, but not very realistic figure. In order to make the Failure Rate data more useful, a spectrum of MTBF estimates encompassing a range of temperature and environmental conditions were calculated. There are many "environments" defined by MIL-HDBK-217F, of which the following four are the most likely to be seen by Ampro products:

- **Ground, Benign:** *Nonmobile, temperature and humidity controlled environments readily accessible to maintenance; includes laboratory instruments and test equipment, medical electronic equipment, business and scientific computer complexes, and missiles and support equipment in ground silos.*

- **Ground, Fixed:** *Moderately controlled environments such as installation in permanent racks with adequate cooling air and possible installation in unheated buildings; includes permanent installation of air traffic control radar and communications facilities.*

- **Ground, Mobile:** *Equipment installed in wheeled or tracked vehicles and equipment manually transported; includes tactical missile ground support equipment, mobile communication equipment, tactical fire direction systems, handheld communications equipment, laser designations and range finders.*

- **Airborne, Inhabited, Fighter:** *Equipment installed on high performance aircraft such as fighters and interceptors. Typical conditions in compartments which can be occupied by an aircrew. Environment extremes of pressure, temperature, shock and vibration are minimal. Examples include the F15, F16, F111, F/A 18, and A10 aircraft.*

TABULATED MTBF DATA

The tables below summarize the MTBF predictions for various Ampro products at three different ambient temperatures, and in the four MIL-HDBK-217F environments mentioned above. The MTBF numbers in each temperature column represent *Power-On-Hours* (POH).

It is important to note that the tabulated MTBF values are for *continuous operation*. A continuous year of operation is 8,760 hours; on the other hand, an 8-hour work day, at 5 days per week, 50 weeks per year, results in a 2000 hour year. *Clearly the actual MTBF -- in calendar time -- will depend on the specific type of application.*

As seen in the tables, *temperature of operation is a critical factor affecting MTBF.* Normal office environments are 20° to 25° C. In an office environment, the temperature to which an Ampro board might be subjected will tend to exceed the ambient conditions by 5° to 10° C, depending on how the system is constructed and cooled. On the other hand, temperatures in industrial environments might exceed the office environment temperatures by 20° C or more. It is important to base an MTBF estimate on the thermal characteristics of the system into which Ampro's board is being integrated. From the data in the tables, it can be seen that *adequate airflow across the electronics is of great importance.*

Finally, other environmental factors such as shock, vibration, corrosive atmosphere, and thermal stress, have a significant effect on MTBF. By selecting MTBF values for the appropriate MIL-HDBK-217F defined operating environment, the system designer can most accurately predict the resulting system MTBF.

LITTLE BOARD™/486*i* MTBF DATA
09/18/95

GROUND, BENIGN ENVIRONMENT

PRODUCT	25°C	55°C	70°C
LB/486*i* (0 MB)	134,769	71,942	45,805
LB/486*i* (4 MB)	123,510	58,942	34,474
LB/486*i* (8 MB)	113,986	49,921	27,637
LB/486*i* (16 MB)	107,453	40,239	20,652
LB/486*i* (32 MB)	95,185	31,409	15,296
LB/486*i* (64 MB)	72,851	19,880	9,083

GROUND, FIXED ENVIRONMENT

PRODUCT	25°C	55°C	70°C
LB/486*i* (0 MB)	56,018	38,866	28,457
LB/486*i* (4 MB)	51,149	33,536	23,074
LB/486*i* (8 MB)	47,059	29,492	19,403
LB/486*i* (16 MB)	46,141	25,896	15,707
LB/486*i* (32 MB)	41,474	21,348	12,216
LB/486*i* (64 MB)	32,530	14,553	7,693

GROUND, MOBILE ENVIRONMENT

PRODUCT	25°C	55°C	70°C
LB/486*i* (0 MB)	19,871	14,932	11,702
LB/486*i* (4 MB)	18,712	13,798	10,518
LB/486*i* (8 MB)	17,681	12,823	9,552
LB/486*i* (16 MB)	17,605	12,119	8,573
LB/486*i* (32 MB)	16,422	10,823	7,327
LB/486*i* (64 MB)	13,861	8,412	5,285

AIRBORNE, INHABITED, FIGHTER ENVIRONMENT

PRODUCT	25°C	55°C	70°C
LB/486*i* (0 MB)	19,894	15,256	12,144
LB/486*i* (4 MB)	18,495	13,939	10,794
LB/486*i* (8 MB)	17,279	12,831	9,713
LB/486*i* (16 MB)	17,229	12,137	8,709
LB/486*i* (32 MB)	15,883	10,742	7,380
LB/486*i* (64 MB)	13,074	8,206	5,249

LITTLE BOARD/486-II MTBF DATA
2/13/95

GROUND, BENIGN ENVIRONMENT

PRODUCT	25°C	55°C	70°C
LB/486-II (1 MB)	138,641	80,711	53,760
LB/486-II (4 MB)	125,752	70,193	44,678
LB/486-II (16 MB)	107,788	43,117	22,494
LB/486-II (32 MB)	81,522	23,033	10,743

GROUND, FIXED ENVIRONMENT

PRODUCT	25°C	55°C	70°C
LB/486-II (1 MB)	56,555	39,291	28,979
LB/486-II (4 MB)	49,416	34,337	24,935
LB/486-II (16 MB)	44,580	25,681	15,861
LB/486-II (32 MB)	37,676	16,588	8,865

GROUND, MOBILE ENVIRONMENT

PRODUCT	25°C	55°C	70°C
LB/486-II (1 MB)	21,151	13,537	9,597
LB/486-II (4 MB)	19,173	12,521	8,919
LB/486-II (16 MB)	18,014	11,008	7,341
LB/486-II (32 MB)	16,289	8,776	5,327

AIRBORNE, INHABITED, FIGHTER ENVIRONMENT

PRODUCT	25°C	55°C	70°C
LB/486-II (1 MB)	19,325	13,390	9,952
LB/486-II (4 MB)	17,293	12,213	9,124
LB/486-II (16 MB)	16,191	10,703	7,447
LB/486-II (32 MB)	14,617	8,524	5,359

Little Board™/486SLC MTBF DATA
7/14/93

GROUND, BENIGN ENVIRONMENT

PRODUCT	25°C	55°C	70°C
LB/486SLC (512K)	197,910	133,273	96,233
LB/486SLC (1 MB)	191,096	126,607	89,633
LB/486SLC (2 MB)	184,419	118,600	81,420
LB/486SLC (4 MB)	167,442	102,511	66,944
LB/486SLC (8 MB)	164,336	77,490	42,884
LB/486SLC (16 MB)	137,032	53,472	27,019

GROUND, FIXED ENVIRONMENT

PRODUCT	25°C	55°C	70°C
LB/486SLC (512K)	82,061	60,826	46,828
LB/486SLC (1 MB)	78,171	57,932	44,353
LB/486SLC (2 MB)	74,275	54,715	41,402
LB/486SLC (4 MB)	65,158	47,770	35,533
LB/486SLC (8 MB)	68,677	43,132	28,070
LB/486SLC (16 MB)	57,005	32,520	19,575

GROUND, MOBILE ENVIRONMENT

PRODUCT	25°C	55°C	70°C
LB/486SLC (512K)	29,685	19,651	14,113
LB/486SLC (1 MB)	28,644	19,111	13,762
LB/486SLC (2 MB)	27,681	18,558	13,366
LB/486SLC (4 MB)	25,133	17,146	12,410
LB/486SLC (8 MB)	26,453	16,842	11,512
LB/486SLC (16 MB)	23,179	14,430	9,552

AIRBORNE, INHABITED, FIGHTER ENVIRONMENT

PRODUCT	25°C	55°C	70°C
LB/486SLC (512K)	27,566	19,628	14,776
LB/486SLC (1 MB)	26,463	18,985	14,332
LB/486SLC (2 MB)	25,435	18,334	13,843
LB/486SLC (4 MB)	22,795	16,703	12,675
LB/486SLC (8 MB)	24,223	16,578	11,823
LB/486SLC (16 MB)	20,919	14,001	9,655

Little Board™/386SX MTBF DATA
6/6/94

GROUND, BENIGN ENVIRONMENT

PRODUCT	25°C	55°C	70°C
LB/386SX (512K)	198,050	145,655	96,375
LB/386SX (1 MB)	191,227	139,032	89,756
LB/386SX (2 MB)	184,541	131,285	81,521
LB/386SX (4 MB)	167,543	115,001	67,012
LB/386SX (8 MB)	164,433	91,876	42,912
LB/386SX (16 MB)	137,100	65,660	20,030

GROUND, FIXED ENVIRONMENT

PRODUCT	25°C	55°C	70°C
LB/386SX (512K)	82,080	65,131	46,860
LB/386SX (1 MB)	78,188	62,085	44,382
LB/386SX (2 MB)	74,290	58,779	41,427
LB/386SX (4 MB)	65,170	51,479	35,551
LB/386SX (8 MB)	68,690	48,274	28,082
LB/386SX (16 MB)	57,014	37,272	19,580

GROUND, MOBILE ENVIRONMENT

PRODUCT	25°C	55°C	70°C
LB/386SX (512K)	29,684	21,495	14,115
LB/386SX (1 MB)	28,643	20,880	13,764
LB/386SX (2 MB)	27,680	20,266	13,368
LB/386SX (4 MB)	25,132	18,679	12,411
LB/386SX (8 MB)	26,452	18,647	11,513
LB/386SX (16 MB)	23,178	16,103	9,553

AIRBORNE, INHABITED, FIGHTER ENVIRONMENT

PRODUCT	25°C	55°C	70°C
LB/386SX (512K)	27,565	21,162	14,778
LB/386SX (1 MB)	26,462	20,444	14,334
LB/386SX (2 MB)	25,434	19,733	13,845
LB/386SX (4 MB)	22,795	17,935	12,677
LB/386SX (8 MB)	24,222	18,100	11,825
LB/386SX (16 MB)	20,919	15,408	9,656

CoreModule™/486-II MTBF DATA
9/13/94

GROUND, BENIGN ENVIRONMENT

PRODUCT	25°C	55°C	70°C
CM/486-II (2 MB)	298,481	171,438	88,796
CM/486-II (4 MB)	273,717	144,520	68,097
CM/486-II (8 MB)	283,415	147,179	68,682
CM/486-II (16 MB)	249,401	113,093	46,990

GROUND, FIXED ENVIRONMENT

PRODUCT	25°C	55°C	70°C
CM/486-II (2 MB)	112,436	81,781	51,979
CM/486-II (4 MB)	103,742	72,693	43,283
CM/486-II (8 MB)	109,418	75,435	44,240
CM/486-II (16 MB)	98,718	63,236	33,518

GROUND, MOBILE ENVIRONMENT

PRODUCT	25°C	55°C	70°C
CM/486-II (2 MB)	42,723	30,246	19,177
CM/486-II (4 MB)	40,418	28,425	17,667
CM/486-II (8 MB)	42,120	29,256	17,985
CM/486-II (16 MB)	39,352	26,726	15,745

AIRBORNE, INHABITED, FIGHTER ENVIRONMENT

PRODUCT	25°C	55°C	70°C
CM/486-II (2 MB)	38,252	28,989	19,651
CM/486-II (4 MB)	36,008	27,093	17,973
CM/486-II (8 MB)	37,705	28,043	18,386
CM/486-II (16 MB)	35,049	25,487	15,965

CoreModule™/386-II MTBF DATA
9/13/94

GROUND, BENIGN ENVIRONMENT

PRODUCT	25°C	55°C	70°C
CM/386-II (2 MB)	323,812	191,028	98,482
CM/386-II (4 MB)	294,870	158,196	73,653
CM/386-II (8 MB)	306,155	161,388	74,337
CM/386-II (16 MB)	266,842	121,299	49,570

GROUND, FIXED ENVIRONMENT

PRODUCT	25°C	55°C	70°C
CM/386-II (2 MB)	117,583	86,938	55,544
CM/386-II (4 MB)	108,108	76,740	45,727
CM/386-II (8 MB)	114,286	79,802	46,797
CM/386-II (16 MB)	102,663	66,276	34,965

GROUND, MOBILE ENVIRONMENT

PRODUCT	25°C	55°C	70°C
CM/386-II (2 MB)	43,768	31,088	19,707
CM/386-II (4 MB)	41,352	29,167	18,117
CM/386-II (8 MB)	43,136	30,043	18,451
CM/386-II (16 MB)	40,237	27,381	16,101

AIRBORNE, INHABITED, FIGHTER ENVIRONMENT

PRODUCT	25°C	55°C	70°C
CM/386-II (2 MB)	39,218	29,836	20,243
CM/386-II (4 MB)	36,862	27,832	18,467
CM/386-II (8 MB)	38,643	28,835	18,904
CM/386-II (16 MB)	35,858	26,140	16,354

CoreModule™/XT *Plus* MTBF DATA
2/23/94

GROUND, BENIGN ENVIRONMENT

PRODUCT	25°C	55°C	70°C
CM/XT *Plus* (2 MB)	306,121	137,500	79,354

GROUND, FIXED ENVIRONMENT

PRODUCT	25°C	55°C	70°C
CM/XT *Plus* (2 MB)	116,259	72,440	48,846

GROUND, MOBILE ENVIRONMENT

PRODUCT	25°C	55°C	70°C
CM/XT *Plus* (2 MB)	45,591	27,931	19,058

AIRBORNE, INHABITED, FIGHTER ENVIRONMENT

PRODUCT	25°C	55°C	70°C
CM/XT *Plus* (2 MB)	40,373	27,038	19,399

CoreModule™/PC MTBF DATA
5/31/94

GROUND, BENIGN ENVIRONMENT

PRODUCT	25°C	55°C	70°C
CM/PC	395,611	252,308	152,833

GROUND, FIXED ENVIRONMENT

PRODUCT	25°C	55°C	70°C
CM/PC	149,860	112,106	76,960

GROUND, MOBILE ENVIRONMENT

PRODUCT	25°C	55°C	70°C
CM/PC	53,966	37,880	24,222

AIRBORNE, INHABITED, FIGHTER ENVIRONMENT

PRODUCT	25°C	55°C	70°C
CM/PC	48,664	36,423	24,885

MiniModule™/SVG-II MTBF DATA
2/10/95

GROUND, BENIGN ENVIRONMENT

PRODUCT	25°C	55°C	70°C
MM/SVG-II (512K)	718,494	341,292	195,048
MM/SVG-II (1M)	680,958	304,721	168,827

GROUND, FIXED ENVIRONMENT

PRODUCT	25°C	55°C	70°C
MM/SVG-II (512K)	284,322	167,841	107,495
MM/SVG-II (1M)	270,570	155,964	98,029

GROUND, MOBILE ENVIRONMENT

PRODUCT	25°C	55°C	70°C
MM/SVG-II (512K)	87,414	48,785	30,795
MM/SVG-II (1M)	85,073	47,421	29,844

AIRBORNE, INHABITED, FIGHTER ENVIRONMENT

PRODUCT	25°C	55°C	70°C
MM/SVG-II (512K)	85,963	51,915	34,204
MM/SVG-II (1M)	83,224	50,201	32,962

MiniModule™/VGA-FP MTBF DATA
1/24/94

GROUND, BENIGN ENVIRONMENT

PRODUCT	25°C	55°C	70°C
MM/VGA-FP	476,740	270,556	170,124

GROUND, FIXED ENVIRONMENT

PRODUCT	25°C	55°C	70°C
MM/VGA-FP	153,263	114,266	85,730

GROUND, MOBILE ENVIRONMENT

PRODUCT	25°C	55°C	70°C
MM/VGA-FP	63,260	44,015	32,145

AIRBORNE, INHABITED, FIGHTER ENVIRONMENT

PRODUCT	25°C	55°C	70°C
MM/VGA-FP	54,836	41,493	32,064

MiniModule™/CGA MTBF DATA
6/23/94

GROUND, BENIGN ENVIRONMENT

PRODUCT	25°C	55°C	70°C
MM/CGA	1,005,297	755,548	591,280

GROUND, FIXED ENVIRONMENT

PRODUCT	25°C	55°C	70°C
MM/CGA	507,617	387,231	306,094

GROUND, MOBILE ENVIRONMENT

PRODUCT	25°C	55°C	70°C
MM/CGA	224,308	140,341	98,330

AIRBORNE, INHABITED, FIGHTER ENVIRONMENT

PRODUCT	25°C	55°C	70°C
MM/CGA	209,013	142,664	105,061

MiniModule™/LCD MTBF DATA
6/23/94

GROUND, BENIGN ENVIRONMENT

PRODUCT	25°C	55°C	70°C
MM/LCD (64KB DRAM)	1,032,617	706,776	505,260
MM/LCD (128KB DRAM)	939,165	619,926	425,911

GROUND, FIXED ENVIRONMENT

PRODUCT	25°C	55°C	70°C
MM/LCD (64KB DRAM)	511,033	367,648	274,492
MM/LCD (128KB DRAM)	435,132	316,137	235,138

GROUND, MOBILE ENVIRONMENT

PRODUCT	25°C	55°C	70°C
MM/LCD (64KB DRAM)	242,736	144,656	98,224
MM/LCD (128KB DRAM)	208,870	130,161	89,919

AIRBORNE, INHABITED, FIGHTER ENVIRONMENT

PRODUCT	25°C	55°C	70°C
MM/LCD (64KB DRAM)	223,822	147,963	106,028
MM/LCD (128KB DRAM)	188,710	130,011	94,920

MiniModule™/FI MTBF DATA
10/14/94

GROUND, BENIGN ENVIRONMENT

PRODUCT	25°C	55°C	70°C
MM/FI	870,407	589,051	425,852

GROUND, FIXED ENVIRONMENT

PRODUCT	25°C	55°C	70°C
MM/FI	309,936	237,231	185,001

GROUND, MOBILE ENVIRONMENT

PRODUCT	25°C	55°C	70°C
MM/FI	107,609	74,066	53,258

AIRBORNE, INHABITED, FIGHTER ENVIRONMENT

PRODUCT	25°C	55°C	70°C
MM/FI	97,427	72,488	55,111

MiniModule™/FSS MTBF DATA
6/23/94

GROUND, BENIGN ENVIRONMENT

PRODUCT	25°C	55°C	70°C
MM/FSS	1,415,556	823,156	554,515

GROUND, FIXED ENVIRONMENT

PRODUCT	25°C	55°C	70°C
MM/FSS	500,581	336,616	243,937

GROUND, MOBILE ENVIRONMENT

PRODUCT	25°C	55°C	70°C
MM/FSS	166,366	98,464	66,086

AIRBORNE, INHABITED, FIGHTER ENVIRONMENT

PRODUCT	25°C	55°C	70°C
MM/FSS	141,028	94,020	67,541

MiniModule™/SSD MTBF DATA
6/23/94

GROUND, BENIGN ENVIRONMENT

PRODUCT	25°C	55°C	70°C
MM/SSD	1,170,212	623,211	398,783

GROUND, FIXED ENVIRONMENT

PRODUCT	25°C	55°C	70°C
MM/SSD	466,783	297,823	209,393

GROUND, MOBILE ENVIRONMENT

PRODUCT	25°C	55°C	70°C
MM/SSD	157,079	92,358	62,596

AIRBORNE, INHABITED, FIGHTER ENVIRONMENT

PRODUCT	25°C	55°C	70°C
MM/SSD	149,026	95,127	67,417

MiniModule™/SPL MTBF DATA
3/14/94

GROUND, BENIGN ENVIRONMENT

PRODUCT	25°C	55°C	70°C
MM/SPL	1,048,023	642,641	444,788

GROUND, FIXED ENVIRONMENT

PRODUCT	25°C	55°C	70°C
MM/SPL	390,149	267,396	196,274

GROUND, MOBILE ENVIRONMENT

PRODUCT	25°C	55°C	70°C
MM/SPL	122,781	73,210	49,684

AIRBORNE, INHABITED, FIGHTER ENVIRONMENT

PRODUCT	25°C	55°C	70°C
MM/SPL	115,605	75,696	54,075

RS-485 ADAPTER MTBF DATA
5/17/94

GROUND, BENIGN ENVIRONMENT

PRODUCT	25°C	55°C	70°C
RS-485 Adapter	14,387,502	5,160,164	1,866,862

GROUND, FIXED ENVIRONMENT

PRODUCT	25°C	55°C	70°C
RS-485 Adapter	4,602,933	2,350,572	1,125,999

GROUND, MOBILE ENVIRONMENT

PRODUCT	25°C	55°C	70°C
RS-485 Adapter	2,096,645	1,099,365	565,914

AIRBORNE, INHABITED, FIGHTER ENVIRONMENT

PRODUCT	25°C	55°C	70°C
RS-485 Adapter	1,406,597	812,653	469,989

MiniModule/ENET-II MTBF DATA
3/7/95

GROUND, BENIGN ENVIRONMENT

PRODUCT	25°C	55°C	70°C
MM/ENET-II (Thin Net)	588,922	307,388	207,093
MM/ENET-II (AUI / TP)	601,495	310,341	207,624

GROUND, FIXED ENVIRONMENT

PRODUCT	25°C	55°C	70°C
MM/ENET-II (Thin Net)	355,470	209,020	145,583
MM/ENET-II (AUI / TP)	356,756	206,570	140,262

GROUND, MOBILE ENVIRONMENT

PRODUCT	25°C	55°C	70°C
MM/ENET-II (Thin Net)	164,923	93,655	63,360
MM/ENET-II (AUI / TP)	161,571	91,093	60,515

AIRBORNE, INHABITED, FIGHTER ENVIRONMENT

PRODUCT	25°C	55°C	70°C
MM/ENET-II (Thin Net)	159,995	97,426	68,423
MM/ENET-II (AUI / TP)	158,518	95,534	66,100

MiniModule™/ENET MTBF DATA
6/23/94

GROUND, BENIGN ENVIRONMENT

PRODUCT	25°C	55°C	70°C
MM/ENET	794,263	502,285	342,392

GROUND, FIXED ENVIRONMENT

PRODUCT	25°C	55°C	70°C
MM/ENET	372,296	253,938	181,389

GROUND, MOBILE ENVIRONMENT

PRODUCT	25°C	55°C	70°C
MM/ENET	154,491	91,867	61,715

AIRBORNE, INHABITED, FIGHTER ENVIRONMENT

PRODUCT	25°C	55°C	70°C
MM/ENET	141,028	93,652	66,784

MiniModule™/PCMCIA MTBF DATA
12/12/94

GROUND, BENIGN ENVIRONMENT

PRODUCT	25°C	55°C	70°C
MM/PCMCIA	441,218	228,440	134,815

GROUND, FIXED ENVIRONMENT

PRODUCT	25°C	55°C	70°C
MM/PCMCIA	170,816	104,117	66,931

GROUND, MOBILE ENVIRONMENT

PRODUCT	25°C	55°C	70°C
MM/PCMCIA	51,227	27,986	17,159

AIRBORNE, INHABITED, FIGHTER ENVIRONMENT

PRODUCT	25°C	55°C	70°C
MM/PCMCIA	53,253	31,176	19,861

Memory Card Adapter MTBF DATA
3/14/94

GROUND, BENIGN ENVIRONMENT

PRODUCT	25°C	55°C	70°C
Memory Card Adapter	697,287	364,998	219,990

GROUND, FIXED ENVIRONMENT

PRODUCT	25°C	55°C	70°C
Memory Card Adapter	300,133	171,933	108,821

GROUND, MOBILE ENVIRONMENT

PRODUCT	25°C	55°C	70°C
Memory Card Adapter	79,015	42,624	25,995

AIRBORNE, INHABITED, FIGHTER ENVIRONMENT

PRODUCT	25°C	55°C	70°C
Memory Card Adapter	84,956	47,546	29,804

APPLICATION NOTE

NUMBER: AAN- 9402 **REVISION:** A **DATE:** 7/18/95

TITLE: Building CoreModule™ systems

PRODUCT: All CoreModule CPUs and MiniModule™ expansion board products featuring PC/104 Specification, Version 2.1 Stackthrough and Double-Stackthrough bus header connectors.

SUMMARY: A discussion of various options and issues regarding CoreModule stacks and interface between CoreModule CPUs and PC and AT passive backplanes and plug-in cards.

CoreModule CPU and MiniModule expansion products provide an expansion bus interface in the form of PC/104 Specification, Version 2.1 stackthrough and double-stackthrough bus header connectors. There are several options for CoreModule system configurations, including CoreModule stacks, embedded OEM engines and bus expansion using ribbon cables. This application note provides information that is helpful when working with these options.

The following topics are covered:

- **CoreModule Stacks** -- An overview of CoreModule stacks using PC/104 Specification, Version 2.1 stackthrough and double-stackthrough bus header connectors. Embedded OEM Engines are also discussed.

- **Ampro Expansion Options** -- An overview of the plug-in card backplane interface adapters available from Ampro. These are designed specifically for use with Ampro CoreModule CPUs, and provide maximum flexibility, reliability, and ease of use.

- **Direct Plug-in Card Connection** -- A discussion of how to construct a cable which can be used for direct connection between the edgecard connectors of standard PC or AT bus plug-in cards and the bus expansion header connectors of the CoreModule and MiniModule products.

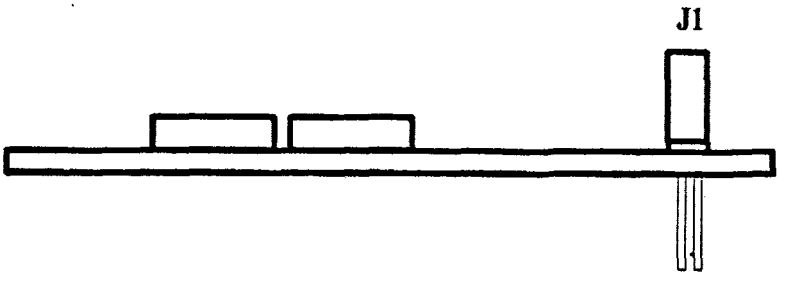
- **Bus Expansion Guidelines** -- A discussion of several technical issues which influence system reliability when ribbon cable is used for bus expansion of the Ampro CoreModule and MiniModule products, including cable length and quality, backplane quality, and bus termination.

AMPRO®

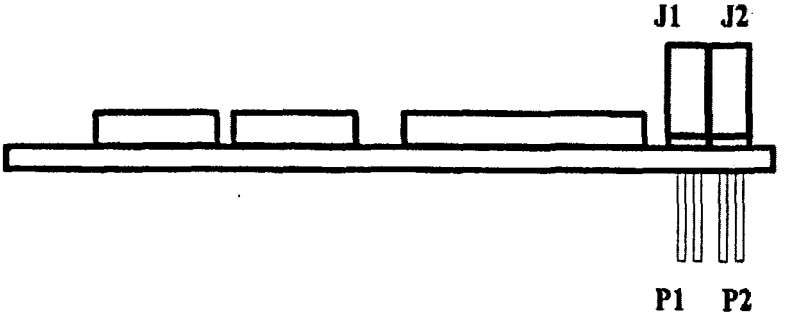
AMPRO COMPUTERS, INCORPORATED
990 Almanor Avenue, Sunnyvale, California 94086
TEL (408) 522-2100 - FAX (408) 720-1305

PC/104 Specification, Version 2.1 Stackthrough 8-bit and 16-bit Bus Connectors

Ampro CoreModule CPUs and MiniModule Expansion Boards feature PC/104 Specification, Version 2.1, stackthrough 8-bit bus and double-stackthrough 16-bit bus header connectors. Stackthrough 8-bit bus and double stackthrough 16-bit bus header connector P1 and P2 pins extend directly through the printed circuit board as illustrated below.



PC/104 Version 2
Stackthrough 8-Bit Bus



PC/104 Version 2
Stackthrough 16-Bit Bus

Building CoreModule Systems

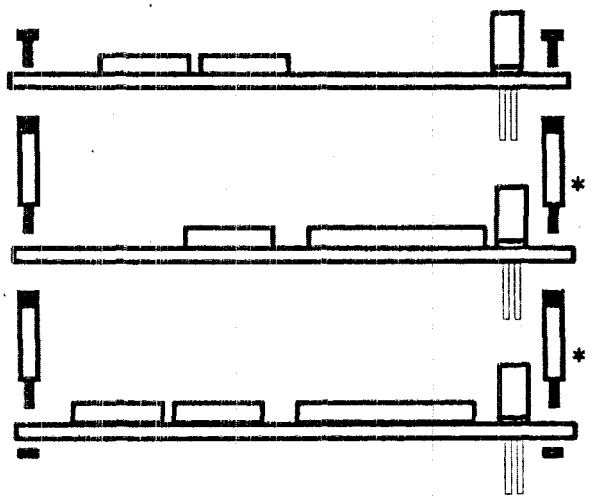
Ampro CoreModules afford a great deal of flexibility in system design. CoreModule CPUs can be stacked with several MiniModules, plugged onto an OEM proprietary application board as a component, or connected to a passive backplane using interface cables. Three basic CoreModule system configurations are discussed in this document including: Self-Stacking Systems, Embedded OEM Engines, and Bus Expansion Options using CoreModule CPU and MiniModule Products.

Self-Stacking Systems

The simplest way to expand an Ampro CoreModule CPU is with self-stacking Ampro MiniModules. MiniModules are available for disk interfaces, including floppy, SCSI, and IDE. The SCSI interface supports a variety of peripheral devices like disk, tape, and CD ROM. MiniModules also provide video interfaces from monochrome through Super VGA, including flat panel displays. Other MiniModules provide additional serial and parallel ports, modem, network, and other interfaces.

The CoreModule can be stacked directly with the MiniModules to avoid using card cages and backplanes. MiniModules mount directly on the stackthrough bus connector(s) of the CoreModule. Thus a three module system fits in a 3.6 inch by 3.8 inch space. It takes only minutes to assemble a system.

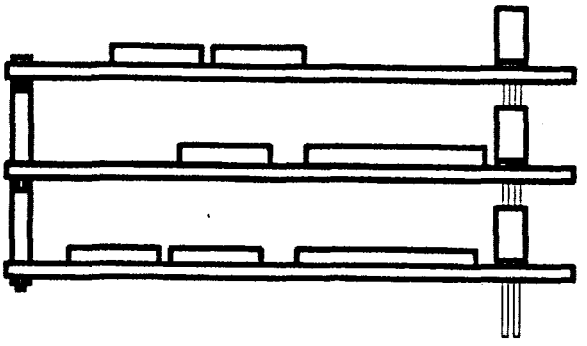
Mounting hardware is included with CoreModule CPU and MiniModule expansion products



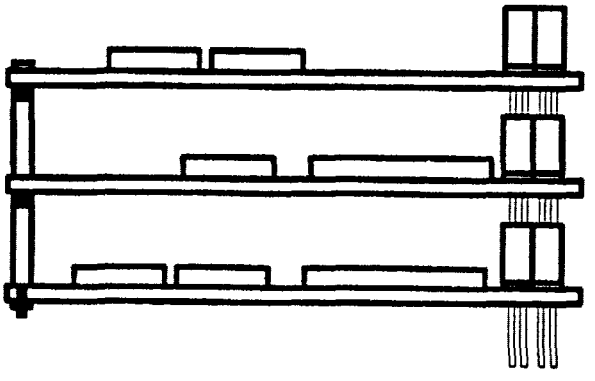
** Either two or four 0.6" long stacking spacers are used to attach modules. Older Ampro 8-bit modules do not have 4 mounting holes, as they were not required by the PC/104 Specification, Version 1. PC/104 module male-female stacking spacer, model 4000-440-N MOD L=600, are available from RAF Electronic Hardware, phone (203) 888-2133.*

Self Stacking System Components

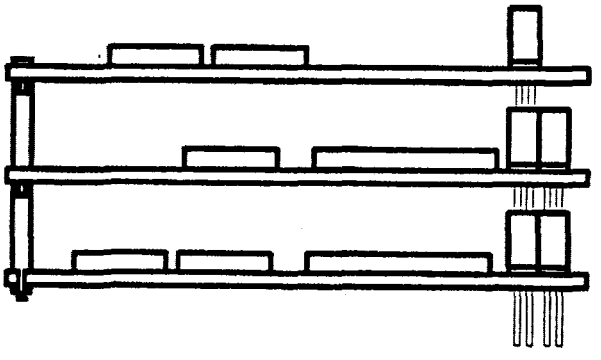
The following are examples of an 8-bit module stack, a 16-bit module stack, and a combination 8-bit / 16-bit module stack.



8-Bit Module Stack



16-Bit Module Stack

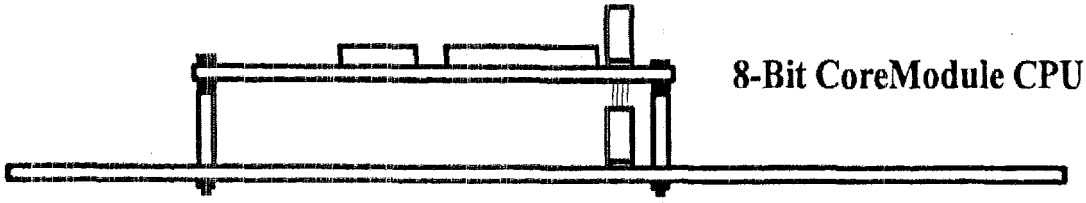


8-Bit / 16 Bit Module Stack

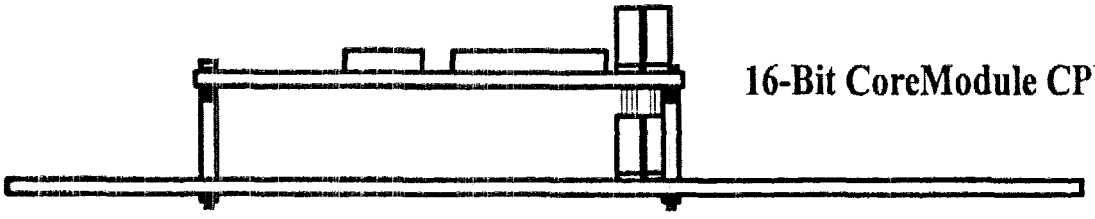
Note: 8-bit modules must not be placed between 16-bit modules.

Embedded OEM Engine

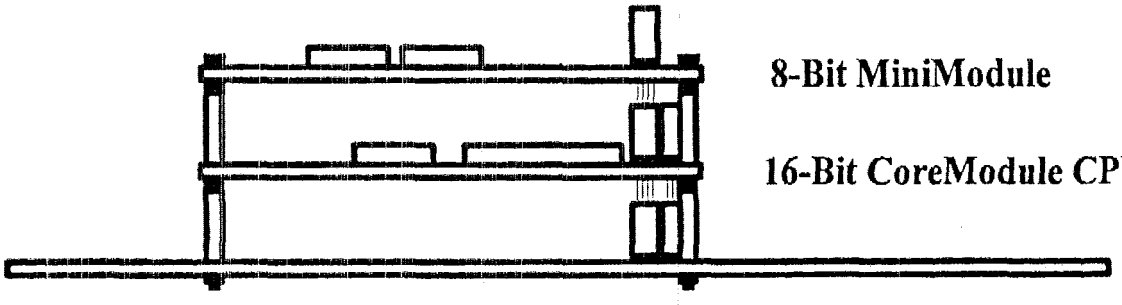
Most often, the CoreModule is used as a computing engine plugged into OEM custom logic or interface devices. The following illustrations show possible CoreModule engine and stack configurations, mounted component side up on an OEM Proprietary Application Board.



Proprietary Application Board



Proprietary Application Board

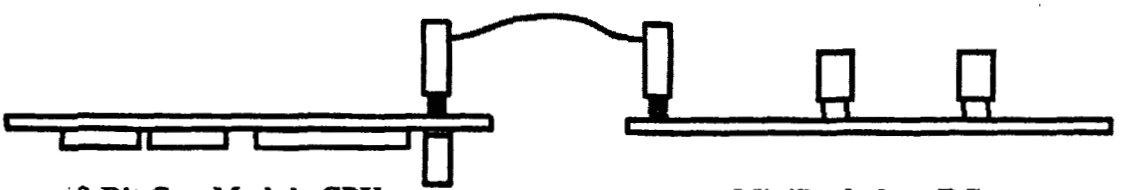


Proprietary Application Board

Ampro Expansion Options

Ampro offers the MiniBackplane™/PC and the MiniBackplane/AT adapters for PC bus and AT bus plug-in cards. These modules are designed specifically to interface conventional PC or AT bus (ISA) expansion cards with CoreModule CPU products. The easy-to-use expansion products offer flexibility and reliability.

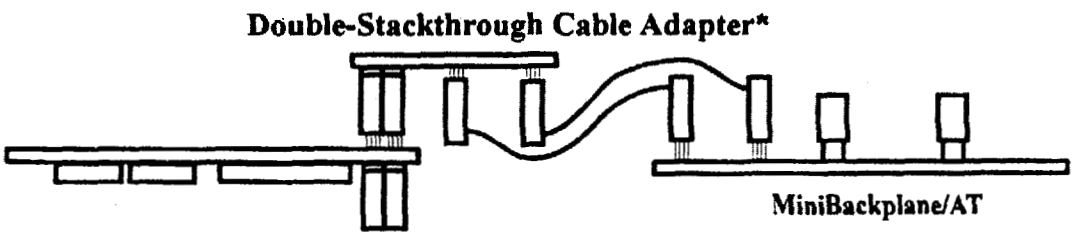
- **MiniBackplane/PC** -- The MiniBackplane/PC connects to 8- or 16-bit CoreModule CPU products with a short ribbon cable. The backplane features two 8-bit plug-in card slots and the option of OEM addition of an on board -12V DC converter. Assembly of the PC bus passive backplane system requires a single 64-conductor expansion ribbon cable and is illustrated below.



8-Bit Passive Backplane System

- **MiniBackplane/AT** -- The MiniBackplane/AT connects with 16-bit CoreModule CPU products using two bus expansion ribbon cables and a Double-Stackthrough Cable Adapter. The backplane features two 16-bit plug-in card slots and sockets which allow OEM addition of +12V DC (25mA), -12V DC (25mA), and -5V DC (50mA), DC-to-DC converters. Also, optional AC bus termination, as recommended by the PC/104 specification, may be installed directly on the backplane.

  The Double-Stackthrough (DST) Cable Adapter provides the capability to expand a 16-bit double-stackthrough PC/104 bus. The Double-Stackthrough Cable Adapter has two PC/104 female bus connectors to provide easy installation on the PC/104 bus and two male connectors to attach interface cables to the MiniBackplane/AT.

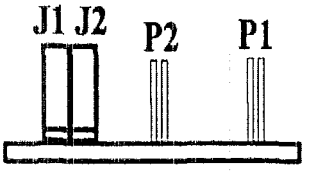


16-Bit Bus Passive Backplane System

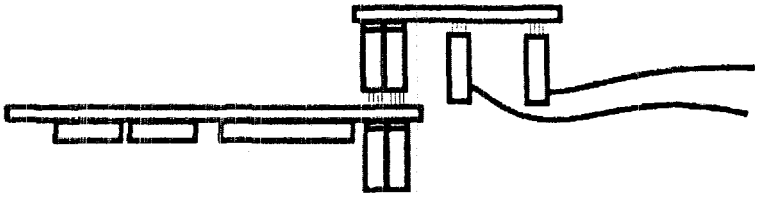
** Note: Double-Stackthrough bus expansion requires the Double-Stackthrough Cable Adapter*

Double-StackThrough Cable Adapter

The Double-Stackthrough (DST) Cable Adapter provides the user with the capability to expand the PC/104 DST bus using ribbon cables and to interface DST CoreModule CPUs with the MiniBackplane/AT. The DST Cable Adapter features female PC/104 female J1 and J2, and male P1 and P2 bus header connectors.



Double-Stackthrough Cable Adapter

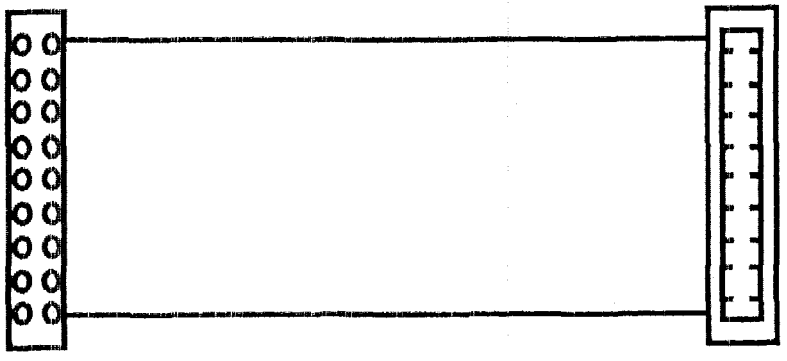


Double Stackthrough 16-Bit CoreModule CPU

Double-Stackthrough Bus Expansion Using Ribbon Cables

Direct Plug-In Card Connection

It is possible to directly connect one or more standard PC or AT bus plug-in cards to a CoreModule CPU, using an expansion card adapter cable, shown in the figure below, which can be easily constructed from available components. The cable must have a female header on one end for connection to the CoreModule CPU bus interface header, and have one or more cable mounted edgecard socket connectors at the other end for connection to the PC or AT bus plug-in cards. Expanding a double-stackthrough bus requires the Double-Stackthrough Cable Adapter described in the previous section.



The following connectors can be used to construct this type of cable:

- **Cable-Mount Header Connectors** -- The end of the cable that connects to the CPU module requires a 64-contact cable-mount female header connector for mating with the 64-pin header bus expansion connector on the CPU module. An additional cable with a 40-contact header connector is required when 16-bit AT bus expansion cards are involved. These connectors, called "IDC Female Ribbon Cable Header" connectors, are readily available from several vendors. The following 3M connectors can be used:

  3M 7964-6500EC -- 64-contact cable mount edgecard socket
  3M-3417-2040 -- 40-contact cable mount edgecard socket

- **Cable-Mount Edgecard Sockets** -- The end of the cable that connects to the bus plug-in card requires a 62-contact cable-mount edgecard socket connector. An additional cable with a 36-contact socket connector is required in the case of 16-bit AT bus expansion cards. Not all cable mount edgecard connectors provide the required relationship of edgecard signals relative to the signals cable bus. Be careful not to use connectors which reverse the signal arrangement, swapping alternate signals. The required cable-mount edgecard socket connectors are available from PCD Connectors, Peabody, MA 01960. Eastern Sales (508) 532-8800, Western Sales (805) 371-0437. The following PCD connectors can be used:

  RF18-2852-5 -- 36-contact cable mount edgecard socket
  RF31-2852-5 -- 62-contact cable mount edgecard socket

The CoreModule CPU bus expansion header connectors provide several extra ground connections which will not be used in a cable having these components. The edgecard connector only has 62 signal contacts, whereas the bus header connector has 64. Specifically, the last two conductors of the 64-pin expansion bus header (pins A32 and B32) should be left unconnected, and both the first and last two conductors of the 40-pin bus expansion header (pins C0, C19, B0, B19) should be left unconnected.

Bus Expansion Guidelines

Here are several points regarding the use of bus expansion ribbon cables to connect CoreModule CPU products with a passive backplane or other expansion cards:

- **Cable Length** - if the CoreModule expansion bus is connected to expansion bus devices with a ribbon cable, keep the cable as short as possible. Expansion bus cables longer than 6 inches can reduce system reliability.
  -- For cable lengths up to 6 inches, use a high quality ribbon cable, such as 3M part number 3365/64.
  -- For cable lengths between 6 and 12 inches, use a high quality ribbon cable with a ground plane, such as 3M part number 3353/64.
  -- Cable lengths greater than 12 inches are not recommended.

- **Backplane Quality** - Be sure to use a high quality backplane having minimal signal crosstalk. Use of power and ground planes and guard traces between bus signals are recommended.

- **Termination** - If bus termination is required, use "AC" rather than "DC" (resistive) termination as recommended in the PC/104 Specification. Placing 100 ohms in series with

100 pf to 500 pf between each signal and ground is recommended. Do not use resistive termination.

- **Reset and TC Deglitching** - Some conventional PC or PC/AT bus (ISA) expansion boards have asynchronous TTL inputs especially vulnerable to noise and crosstalk. The active high RESET and TC lines are ones to watch out for. If these signals are susceptible, a 200 pf to 500 pf capacitor connected between the signal and ground can be used to prevent false resets. These RESET and TC deglitching capacitors are included on the Ampro MiniBackplanes.

APPLICATION NOTE

NUMBER: AAN-9403 **REVISION:** C **DATE:** 09-09-96

TITLE: "Remote hosted" use of the Ampro CPU Modules

PRODUCT: All active promotion CoreModule™ and LittleBoard™ products

SUMMARY: How to use the serial console and serial boot loader features of the Ampro BIOS to gain remote access to the CoreModule/PC from a serially connected host PC.

INTRODUCTION

Embedded CoreModule and LittleBoard CPU modules are often installed in "black box" type applications which do not have a user interface of any kind nor any other means of communication with the outside world. These are dedicated systems for purposes such as data acquisition, communications, and factory automation & control. They have no need for the input and output devices which are considered standard equipment for desktop PCs, such as keyboards, mice, video monitors, and floppy disk drives. These types of systems present an interesting challenge to design engineers due to their lack of accessibility. In many such applications, a means must be provided for field service personnel or non-technical end users to perform updates to resident software or otherwise regain control of the system. This must be possible without forcing the person to open the enclosure to make hardware changes, as all such servicing should be done by trained personnel who are familiar with the procedures for handling delicate electronic equipment.

In addition to regaining control of "black box" type systems, a second common scenario requiring remote access to a CoreModule or LittleBoard is in a development environment which does not include a video display adapter or disk controller. Before the board can be used, it is often necessary to run the BIOS SETUP utility to configure the various interfaces as necessary for the target system. Because SETUP is an interactive utility, a remote console for display and keyboard access would be needed to use it. Console access might also be required to test system operation at various stages of development. As new application code is being produced, there must also be a way to transfer that code to the CoreModule or LittleBoard for testing.

The CoreModules and LittleBoards include a unique set of features which allow full access to the system at any time over a standard RS232 serial port. An embedded application may take advantage of these remote access capabilities by maintaining a serial connection on the outside of the enclosure. A remote laptop or other computer system can then be attached and used either for console I/O (display & keyboard) or for performing updates to embedded program code. A system utilizing these features of the Ampro BIOS is considered "remote hosted" because all access is controlled through a second, remotely connected host computer. By contrast, a

CoreModule or LittleBoard system which includes a video adapter & monitor, keyboard interface, and removable disk drives can be called "self hosted" because all such access is available locally. The purpose of this application note is to expound upon the remote features of Ampro CPU Products and show how they may be of benefit in both initial development and target system environments.

SERIAL CONSOLE MODE

The most common requirement for accessing an embedded system over the serial port is for console I/O. This capability is crucial in many applications because it provides immediate access without forcing enclosed systems to be opened and fitted with additional hardware. The capability of console I/O extends to the editing of CoreModule and Setup pages. Devices and embedded BIOS features can be enabled and disabled It also avoids the expense of including that hardware (video adapters, keyboards) in the system by allowing console I/O through either a low cost serial terminal or a terminal emulation program running on a laptop or desktop computer.

The BIOS of the CoreModule and LittleBoard includes a serial console function which supports many ANSI-compatible serial terminals. For compatibility with the BIOS function, a serial terminal would need to support all displayable characters plus the following five "cursor positioning" characters:

ASCII character name	ASCII character number	Function
BS	08h	Backspace
LF	0Ah	Line Feed
VT	0Bh	Vertical Tab
FF	0Ch	Nondestructive Space
CR	0Dh	Carriage Return

Lack of support for any of these control characters would yield an improper display on the terminal screen.

While there are certain specific terminals that are known to be compatible with Ampro's serial console feature, such as the Televideo 925, by far the most common approach is to use a serial communications program with built-in terminal emulation capability. Bear in mind however, that the interpretation of multiple simultaneous keystrokes is limited and the usual CTRL-ALT_ESC hot-key sequence used to enter the interactive SETUP pages cannot be interpreted by the serial terminal emulator. The emulation of varous types of serial terminals is provided in most major communications packages such as Laplink and Procomm. However, these packages are usually quite large due to their extensive feature sets and would be inefficient for use in such a restricted capacity. The Ampro utility TVTERM may represent a good option for use in both development and target environments because it is very space efficient and easy to use, yet offers full compatibility with the serial console feature of the CoreModule/PC BIOS.

HIT "+++" FOR SETUP

By simply maintaining a serial port connection on the outside of the enclosure, the CoreModule PC SETUP pages can be accessed. Field adjustments to the system setup can be made. To enter the SETUP pages in serial console mode, first look for a prompt following power on, or hard reset. The prompt will be approximately the following: "Enter '+++' for setup". Upon seeing the prompt, key in as asked. The keys you hit to edit the SETUP pages will differ slightly dur to the Televideo 925 terminal emulation not being able to process the arrow keys. Following the screen prompting, at all times during editing of the SETUP pages, remote host editing of the SETUP pages can be done.

The SETUP function of the Ampro BIOS provides total flexibility in configuring the system console mode. The console input and output devices may be individually defined as either "normal" (video for output, keyboard for input), serial, or none. In serial mode, all communication parameters including BAUD rate, parity, data bits & stop bits may be set in accordance with application requirements.

THE "SERIAL BOOT LOADER"

A second major requirement for control of a CoreModule or LittleBoard over the serial interface is for downloading application code from a software development system. In a standalone configuration with no video or disk interfaces, there is no obvious means of performing the code transfer and then testing it for proper execution. The serial boot loader function of the Ampro BIOS was designed to fulfill this need.

The serial boot loader is invoked near the end of the power-on self tests (POST), shortly before boot. For a period of a few seconds, the serial port is monitored for transmissions which conform to a defined protocol. To establish a connection with the serial boot loader, a companion program executing on a remotely connected PC would need to understand the specifics of this protocol. After the requisite handshaking has been completed and the connection is made, the companion program begins to transfer an executable program to the boot loader. This program must be fully relocatable with its origin at 0, and because it executes before system boot, it must not make any calls to DOS. The serial boot loader receives this program and loads it into system RAM. At the completion of the transfer, there is some final handshaking between the boot loader and the companion program, and then the connection is broken. The final act of the boot loader, provided that all previous steps have been performed without error, is to make a far call to the executable program which now resides in local memory.

There are two examples of the "companion program" described above which are available from Ampro. These utilities, called SERLOAD and SERPROG, fulfill most requirements for porting code to a CoreModule or LittleBoard, but other similar programs which understand the serial boot loader protocol could be written to satisfy different application requirements.

SERLOAD

The SERLOAD program is a general purpose utility designed to perform the basic function of sending an executable program through the serial port from the host PC to the embedded Ampro CPU module. Its functionality exactly matches that of the hypothetical "companion program" described above. The executable program to be transferred is specified as a parameter on the SERLOAD command line.

SERPROG

This more specialized utility combines the functions of SERLOAD and the Ampro SSD Software utility FLASHWRITE to create a NOVRAM or flash EPROM programming utility which can be executed from a serially connected host system. While the primary purpose of SERPROG is to update Ampro solid state disks remotely, the program operates on *image files* whose contents have no meaning until written into their target memory devices. For example, an Ampro SSD does not become an SSD until its image file is written into the memory device, the system is reset, and installation is triggered during the 55AA ROM scan. Because of this generality, SERPROG will also work with other types of image files such as non-Ampro SSDs, other BIOS extensions, and even non-executing data files.

Other Ampro application notes contain additional information about creating and using general purpose BIOS extensions such as 55AAs.

CONFIGURATION OF REMOTE FEATURES

The configuration of various options on the CoreModule and LittleBoard such as the serial console and serial boot loader is controlled via a SETUP program which is resident in the Ampro BIOS. SETUP may be invoked in one of three possible ways: by use of a "hot key" during POST, from the DOS command prompt with the utility SETUP.COM, or by executing a special Ampro BIOS function. Of these possibilities, only the hot key option is available during the initial stages of development in a remote hosted system.

SETUP is an interactive program which requires some type of display and keyboard for user I/O. In a remote hosted system, this would require the connection of a serial console. However, this raises an interesting question: how can SETUP be invoked from a serial console if the serial console has not already been enabled in SETUP? To overcome this mutual dependency, the Ampro BIOS provides a loopback option on the serial port interface which causes the remote features to be enabled automatically. The loopback is connected between RTS and RI on the CoreModules or LittleBoards first serial port, which correspond to pins 4 & 8 on the CoreModule or LittleBoards first serial connector. The BIOS scans for this loopback on power-

up, and if found, enables both serial console mode and the serial boot loader. In this automatic mode, a standard set of default communication parameters (9600 BAUD, no parity, 8 data bits, 1 stop bit) are used.

NOTE

The automatic remote mode jumper has priority over the SETUP definitions for the console I/O and boot loader features. Consequently, the interface will always default to 9600,N,8,1 even if serial console mode is enabled in SETUP with different communication parameters. When using automatic remote mode, be sure to reset the host system's serial port to 9600,N,8,1.

Both the serial console and serial boot loader features operate over a standard RS232 null modem cable. A "hot" cable which causes a CoreModule or LittleBoard to go into remote mode automatically after the next reset or power cycle may be constructed by simply adding the RTS to RI loopback to that end of the cable.

While some embedded systems require the services of only one of the remote features of the Ampro BIOS, there may be certain applications in which both options are needed. In addition, either function might be required when doing software development for a CoreModule or LittleBoard in a remote hosted environment. However, the serial console and serial boot loader each use the serial port in their own unique way. To support the two features simultaneously, the BIOS must manage them so that both are not actively using the port at the same time. This is only of concern on the one occasion when the serial boot loader is invoked, near the end of POST. The serial console is temporarily disabled just prior to calling the boot loader code, and then is reenabled after either the boot loader timeout elapses or a program is successfully downloaded and executed to termination. This strategy supports both a serial terminal and a serial downloader program and allows the two services to be randomly interchanged. When using a serial terminal, the only evidence that both remote mode features are enabled would be a pause of several seconds near the end of POST as the Ampro BIOS checks for input to the boot loader. Because the console is disabled, no keyboard entry is possible during the boot loader scan.

USING A "REMOTE HOSTED" SYSTEM

The benefits offered by the remote features of the CoreModules and LittleBoards BIOS are clearly illustrated throughout this document. However, the best way to utilize these features in both development and target environments depends largely on the type and method of application software storage used in the embedded system. The following is a general overview of some of the options available, along with guidelines for transferring code in each configuration:

Ampro Solid State Disks

The Ampro SSD/DOS Support Software is used to convert a JEDEC compatible memory device (i.e. EPROM, flash EPROM, NOVRAM) into a bootable floppy emulation containing both DOS and application files. The memory device is installed in any socket available on a CoreModule or LittleBoard.

The procedure for creating an Ampro SSD starts with the preparation of a "master" floppy diskette, which is specially formatted by one of the SSD utility programs. The DOS system is transferred to the master disk by use of the SYS command. Application files are then copied, and a second utility is used to generate an image file the same size as the target memory device. If an EPROM is to be used for the SSD, the image file is written into the device with an EPROM burner. For flash EPROMs, a third SSD utility called FLASHWRITE may be used to burn the chip from a local disk drive, or SERPROG will do the same over the serial port from a remote system. Both FLASHWRITE and SERPROG may also be used to write SSD images into NOVRAM or static RAM devices.

Since both the formatter and image generation utilities will operate on non-Ampro desktop computers, SSD preparation for a CoreModule or LittleBoard may be done entirely offboard. During development, new application code ready for testing on the target platform can be copied to the master diskette for image file regeneration. The new image file can then be burned into an EPROM for installation on the CoreModule or LittleBoard system. When using a flash EPROM or RAM device, SERPROG can be used to write the new image over the serial port. The automatic remote mode jumper allows this to be done easily with no changes to the BIOS SETUP. If further modifications are needed to the application program after testing, the same process may be repeated.

The automatic remote mode jumper also provides an ideal means of performing field upgrades of embedded software. If external access to the serial port interface is provided on the system enclosure, a new SSD image may be downloaded to the CoreModule and Littleboard using hot cabling from a laptop or other computer running SERPROG.

Eurom DiskOnChips

This device uses a proprietary architecture to convert multiple flash EPROMs into a bootable hard disk emulation. It is completely self-contained in a single 28 pin DIP package, and plugs directly into the bytewide socket on a CoreModule or LittleBoard. The embedded firmware includes a flash file system and a sophisticated wear leveling algorithm which maximizes component life. Because it behaves like a hard disk, data reads & writes are performed in exactly the same manner as when using a magnetic hard drive. Like Ampro SSD, the DiskOnChip is DOS-based and requires that DOS be present along with all application program files.

One of the major differences between an Ampro SSD and a Eurom DiskOnChip is that the DiskOnChip is organized in a non-linear fashion and requires a proprietary algorithm for

accessing multiple pages of the device. As a consequence, SERPROG cannot be used to update a DiskOnChip in its entirety with a new image file. Instead, individual files on the "drive" must be written through DOS. Like a hard disk drive, it must also be initialized with the DOS utilities FDISK and FORMAT prior to use. Because of these requirements, it would not be feasible to use a DiskOnChip in a remote hosted system unless a floppy drive were also included. In that configuration, a serial console could still be used as the display and keyboard interface, but the method of preparing and updating DiskOnChip software would be via a floppy diskette.

ROM-BIOS extensions

A BIOS extension is essentially a piece of embedded program code which uses a "hook" to gain control of the system sometime during the POST process. The "hook" is a special signature, such as the IBM standard 55AA, which is recognized by the BIOS during its ROM scans of upper memory. When a valid signature is found, a checksum calculation is performed to verify the device, and then control is transferred to the embedded program. BIOS extensions are usually stored in an EPROM or flash EPROM device installed in the CoreModule and LittleBoard bytewide socket S0. Because BIOS extensions are executed during POST, they cannot have any dependencies on DOS or other operating systems.

A BIOS extension may be easily converted to an image file for a memory device by increasing the file size to match the capacity of the target device and filling the added space with 0's to preserve the checksum. The preparation and conversion of a BIOS extension file may be done on a third party development system, and the options for transferring the file to the CoreModule or LittleBoard would be the same as with an Ampro SSD, described above.

The structure of an embedded program used as a BIOS extension is very similar to that of a program loaded via the serial boot loader from a host system running SERLOAD. In both cases, the code is executed prior to system boot and must be totally relocatable and non-OS dependent. Because of this similarity, code which will be used as a BIOS extension may be easily tested as a serially loaded program. This method of testing is of particular benefit when only EPROM devices are available during development, as repeated erasing and rewriting of those chips can be very time consuming.

The same technique for testing newly developed software with the serial boot loader may also be used when preparing code for installation on Ampro SSDs or other embedded storage media. This is provided that all of the same criteria are followed when converting the programs for serial transfer.

Further information on BIOS extensions may be found in other Ampro application notes.

Serially loaded programs

All of the options listed above require some type of storage medium to be present to hold application software. However, this could be avoided altogether if a serial host were

connected to the CoreModule or LittleBoard at all times in the embedded system. While this configuration is not common, such systems could use the serial boot loader feature of the Ampro BIOS to download application code during each system start-up.

A note on remote debuggers

An important point which has not yet been discussed in this application note is how to debug code on the target Ampro CPU platform when developing in a remote hosted environment. There are numerous remote debugger packages available on the market which offer excellent capabilities for both the porting and debugging of test code on a target platform. These programs can be run on varying levels of Intel 80x86 CPU architectures, and support a number of popular high level language compilers. Depending on which package is used, support may be available both for installing the debugger and/or test program on a memory chip or downloading them to the target platform over the serial port. Execution and debugging of the test program can then be carried out on the target, using the host both for console I/O and for additional code transfers.

Traditionally, it has been necessary to embed a component of the remote debugger in a ROM on the target system in order to initialize communications with the host. However, this can be avoided if the Ampro BIOS's serial boot loader feature is used as a means of transferring the debugger code to the target board. By using a "hot" null modem cable with the RTS to RI jumper, this capability would allow a standalone CoreModule or LittleBoard to be accessed for immediate software development and debug. Ampro has worked with several remote debugger suppliers to develop support for the serial boot loader interface, and the following companies now have products available which include this capability:

Concurrent Sciences
Moscow, ID
Ph. 208-882-0445
Fax 208-882-9774

Paradigm Systems
Endwell, NY
Ph. 1-800-537-5043 or 607-748-5966
Fax 607-748-5968

Phar Lap Software
Cambridge, MA
Ph. 617-661-1510
Fax 617-876-2972

Systems & Software, Inc.
Irvine, CA
Ph. 714-833-1700
Fax 714-833-1900

Contact Ampro Technical Support for an updated list of remote debuggers offering support for the serial boot loader interface.

SUMMARY

Despite the full featured capabilities of embedded CPU products like Ampro CoreModules and Little Boards, the industry perception of tiny single board computers as "macrocomponents" rather than shrunken down desktop PCs is increasing steadily. This mind set carries with it a

common expectation that full access to the system should be available without including a lot of expensive additional hardware. The remote features of the Ampro BIOS have been evolving over a several year period in response to these changing demands, and now offer a comprehensive package of support for remote console and file I/O. Ampro will continue to enhance these capabilities in the future to provide the greatest level of support possible for remote system access.

Developer's Reference List

COMPUTERS, INCORPORATED
990 Almanor Avenue ■ Sunnyvale, California 94086 ■ (408) 522-2100 ■ FAX (408) 720-1305

An Embedded-PC System Developer's Reference List

- ***Building CoreModule Systems:*** application note about various options and issues pertaining to the design of systems based on Ampro CoreModules, MiniModules, and other third-party PC/104 modules. Available from Ampro (AAN-9402).

- ***Designing with PC/104 — a Tutorial***: white paper that provides an overview of the contents of the PC/104 specification, describes typical ways to use PC/104 modules, and provides guidelines for most effective and efficient use of PC/104 in embedded system design. Available from Ampro.

- ***IEEE P996 Draft Standard:*** detailed definition, timing, and electrical characteristics of the "standard" PC/AT bus. Available from IEEE Publications, (908) 981-1393.

- ***ISA and EISA Theory of Operation:*** definitive technical reference on the "standard" PC/AT bus including detailed information on bus signal definitions, timing, and electrical characteristics. Available from Annabooks, (619) 673-0870.

- ***ISA System Architecture:*** complete reference and tutorial guide to designing with the "standard" PC/AT architecture and bus including examples and diagrams. Published by MindShare Press; available from Computer Literacy Bookshops, (408) 435-0744.

- ***PC/104 Resource Guide***: extensive product listings from suppliers of PC/104 modules and other PC/104-related hardware and software. Available from the PC/104 Consortium, (415) 903-8304.

- ***PC/104 Specification***: dimensions, bus connections, and other information required to design PC/104 compatible modules. Available from the PC/104 Consortium, (415) 903-8304.

- ***"Remote hosted" operation of Ampro CoreModules:*** application note that discusses how Ampro's CoreModule CPUs allow serially connected "remote hosted" development and support. Available from Ampro (AAN-9403).

- ***Using the PC Architecture in Embedded Applications***: white paper on issues concerning the application of PC hardware and software technology to embedded systems, and what can be done to make the PC architecture work most effectively. Available from Ampro.

- ***What is PC/104?*** two-page introduction and backgrounder on the PC/104 Specification and PC/104 Consortium. Available from Ampro.

- ***What's the Best Way to ROM Your Embedded-PC Application?*** white paper describing the embedded-PC approach to "ROMing" application software via "solid state disk" (SSD), instead of using the traditional embedded software methods of proprietary microcontroller-based architectures. Available from Ampro.

PC/104 Information

PC/104 Consortium

990 Almanor Avenue ■ Sunnyvale, CA 94086 ■ Phone 408-245-9348 ■ Fax 408-720-1322

What is PC/104?

The Need for an Embedded-PC Standard

Over the past decade, the PC architecture has become an accepted platform for far more than desktop applications. Dedicated and embedded applications for PCs are beginning to be found everywhere! PCs are used as controllers within vending machines, laboratory instruments, communications devices, and medical equipment, to name a few examples.

By standardizing hardware and software around the broadly supported PC architecture, embedded system designers can substantially reduce development costs, risks, and time. This means faster time-to-market and hitting critical market windows with timely product introductions. Another important advantage of using the PC architecture is that its widely available hardware and software are significantly more economical than traditional bus architectures such as STD, VME, and Multibus. This means lower product costs.

For these reasons, companies that embed microcomputers as controllers within their products seek ways to reap the benefits of using the PC architecture. However, the standard PC bus form-factor (12.4" x 4.8") and its associated card cages and backplanes are too bulky (and expensive) for most embedded control applications.

The only practical way to embed the PC architecture in space- and power-sensitive applications has been to design a PC – chip-by-chip – directly into the product. But this runs counter to the growing trend away from "reinventing the wheel." Wherever possible, top management now encourages out-sourcing of components and technologies, to reduce development costs and accelerate product design cycles.

A need therefore arose for a more compact implementation of the PC bus, satisfying the reduced space and power constraints of embedded control applications. Yet these goals had to be realized without sacrificing full hardware and software compatibility with the popular PC bus standard. This would allow the PC's hardware, software, development tools, and system design knowledge to be fully leveraged.

PC/104 was developed in response to this need. It offers full architecture, hardware, and software compatibility with the PC bus, but in ultra-compact (3.6" x 3.8") stackable modules. PC/104 is therefore ideally suited to the unique requirements of embedded control applications.

A Proposed Extension to IEEE-P996

Although PC/104 modules have been manufactured since 1987, a formal specification was not published until 1992. Since then, interest in PC/104 has skyrocketed, with over a hundred different PC/104 modules introduced by more than three dozen manufacturers. Like the original PC bus itself, PC/104 is thus the expression of an existing *de facto* standard, rather than being the invention and design of a committee.

In 1992, the IEEE began a project to standardize a reduced form-factor implementation of the IEEE P996 (draft) specification for the

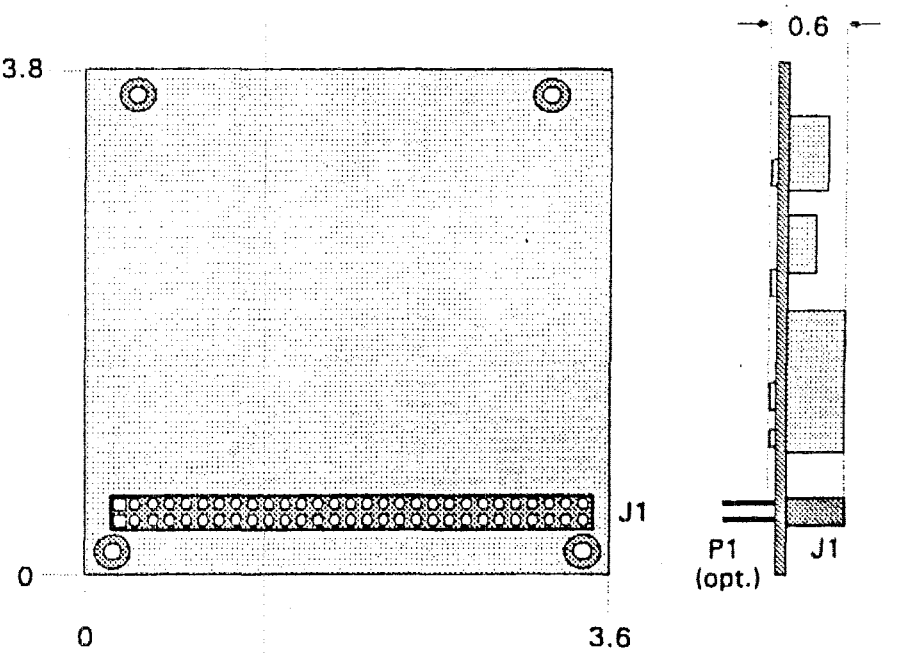


Figure 1. Basic Mechanical Dimensions (8-bit Version)

PC and PC/AT buses, for embedded applications. The *PC/104 Specification* has been adopted as the "base document" for this new IEEE draft standard, called the *P996.1 Standard for Compact Embedded-PC Modules.*

The key differences between PC/104 and the regular PC bus (IEEE P996) are:

- **Compact form-factor.** Size reduces to 3.6 by 3.8 inches.
- **Unique self-stacking bus.** Eliminates the cost and bulk of backplanes and card cages.
- **Pin-and-socket connectors.** Rugged and reliable 64- and 40-contact male/female headers replace the standard PC's edgecard connectors.
- **Relaxed bus drive (6 mA).** Lowers power consumption (to 1-2 Watts per module) and minimizes component count.

By virtue of PC/104, companies embedding PC technology in limited space applications can now benefit from a standardized system architecture complete with a wide range of multi-vendor support.

Two Ways to Use PC/104 Modules

Although configuration and application possibilities with PC/104 modules are practically limitless, there are two basic ways they tend to be used in embedded system designs:

Standalone module stacks. As shown in Figure 2, PC/104 modules are self-stacking. In this approach, the modules are used like ultra-compact bus boards, but without needing backplanes or card cages. Stacked modules are spaced 0.6 inches apart. (The three-module stack shown in Figure 2 measures just 3.6 by 3.8 by 2 inches.) Companies using PC/104 module stacks within their products frequently create one or more of their own application-specific PC/104 modules.

Component-like applications. Another way to use PC/104 modules is illustrated in Figure 3. In this configuration, the modules function as highly integrated components, plugged into custom carrier boards which contain application-specific interfaces and logic. The modules' self-stacking bus can be useful for installing multiple modules in one location. This facilitates future product upgrades or options, and allows temporary addition of modules during system debug or test.

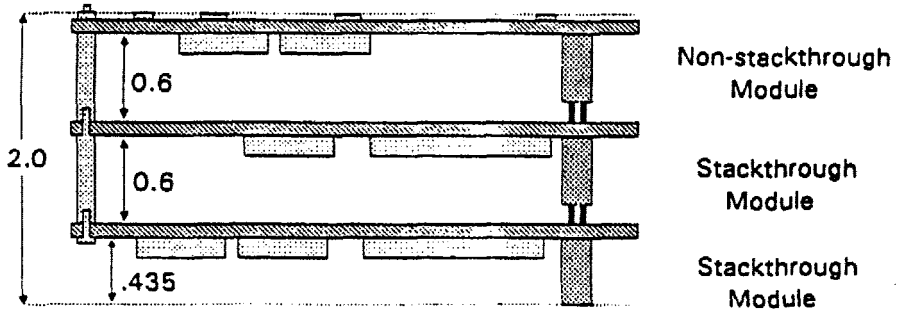


Figure 2. Standalone Module Stacks

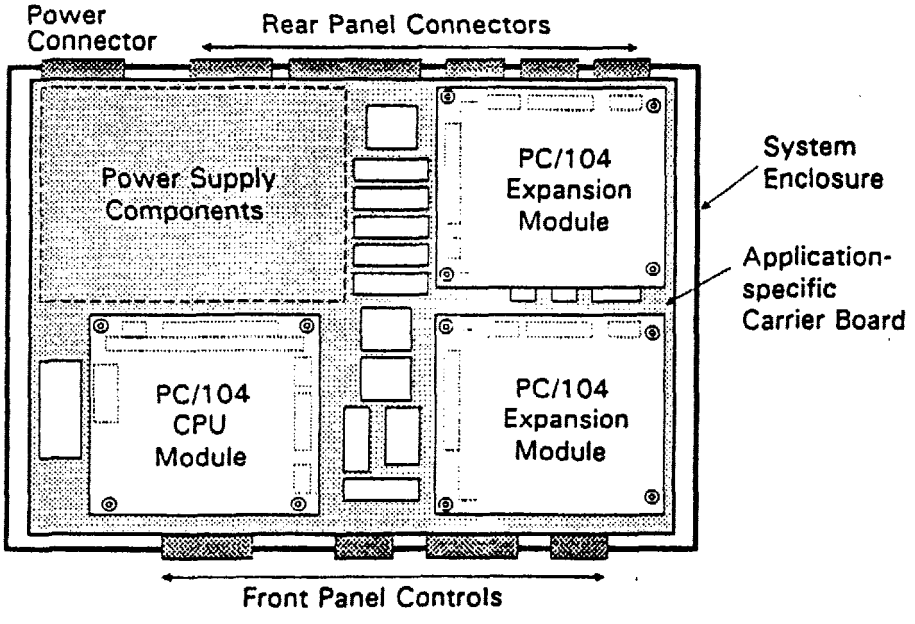


Figure 3. Component-like Applications

About the PC/104 Consortium

The purpose of the PC/104 Consortium is to establish PC/104 as a broadly supported industry standard architecture for embedded-PC applications. The PC/104 Consortium maintains and distributes the *PC/104 Specification* and other PC/104-related documents, serves as a liaison to standards bodies such as IEEE P996.1, and engages in a variety of public relations activities on behalf of PC/104. Consortium membership is open to companies who offer or use PC/104 modules as well as to companies who provide products that target PC/104 applications.

5.3 PC-104 Specifications

The computational block CPU is a PC-104 platform. This section provides the generic specifications for that class of computers. These specifications were printed from the Diamond Systems Internet site at:

http://www.diamondsystems.com/files/pc104p10.pdf.

PC/104-*Plus* Specification

Version 1.0

February 1997

Please Note

This specification is subject to change without notice. While every effort has been made to ensure the accuracy of the material contained within this document, the PC/104 Consortium shall under no circumstances be liable for incidental or consequential damages or related expenses resulting from the use of this specification. If errors are found, please notify the PC/104 Consortium.

REVISION HISTORY

Draft 0.7, November 20, 96 - Preliminary Draft

a. Formatted to meet the requirements of the PC/104 Consortium.
b. Modify the component restrictions across and to each side of the PC/104 connectors (three sides, .400" from each edge, 4.35" top clearance, .100" bottom clearance).

Draft 0.8, December 16, 1996 - Cleanup for release

a. Correct general typos
b. Correct word reference error
c. Add QuickSwitch part number and clarify Mux requirements
d. Change PCI ONLY to PCI-Only and add note
e. Correct figure 4 and Figure 5 errors
f. Correct typo in Table 3

Draft 0.9, January 10, 1997 - Cleanup for release

a. Cleanup minor grammatical errors

Version 1.0, February 1997 - Initial Release

a. Grammatical changes and cleanup per review recommendations.
b. Change Footnote 1 on Page 2 to show future support for the M66EN (66MHz Enable) signal.
c. Add signals PRSNT[1:2]* and CLKRUN* to Figure 1 to encompass all unused PCI signals.
d. Moved the Mechanical section after the Electrical section.
e. Clarified the KEY pin usage for universal modules and defined them as ground connections.
f. Clarified pin 1 for the PC/104-*Plus* connectors on Figure 4.
g. Added an example manufacturer and part No. for the PCI connector (Figure 5) and Shroud (Figure 6).
h. Modified Figure 2, Table 1, and some text under Section 2.2 to add routing recommendations for the PCI interrupt lines INTA - INTD.

TABLE OF CONTENTS

TABLE OF FIGURES

TABLE OF TABLES

PC/104-*Plus* SPECIFICATION

Version 1.0 February 1997

1. INTRODUCTION

While the PC/AT architecture is becoming increasingly popular in embedded applications, there is an increasing need for a higher performance Bus throughput. This is especially true when it comes to graphics devices as well as other high speed I/O devices such as networks.

This document supplies the mechanical and electrical specifications for the "PC/104-*Plus*" and incorporates all of the PC/104 features, with the added advantage of the high speed PCI bus. The physical size, mounting configuration and electrical interconnect portion of the PC/104 specification shall remain unchanged.

1.1 Summary of Key Differences From PC/104 Specification:

- A third connector opposite the PC/104 connectors supports the PCI bus.
- Changes to the component height requirements increase the flexibility of the module.
- Control logic added to handle the requirements for the high speed bus.

1.2 Summary of Key Differences (120-pin PCI and PCI Local Bus Specification)

- The PCI bus connector is a 4x30 (120-pin) 2mm pitch stackthrough connector as opposed to the 124-pin edge connector on standard 32-bit PCI Local Bus.
- The 120-pin PCI does not support 64-bit Extensions, JTAG, PRSNT, or CLKRUN signals.

1.3 References

This document covers the addition of the PCI functions. The following documents should be used as reference for a detailed understanding of the overall system requirements:

- PC/104 Specification Version 2.3
- PCI Local Bus Specification Revision 2.1

Contact the PCI Special Interest Group office for the latest revision of the PCI specification:

PCI Special Interest Group
P.O. Box 14070
Portland, OR 97214
800.433.5177 (U.S.) 503.797.4207 (International)

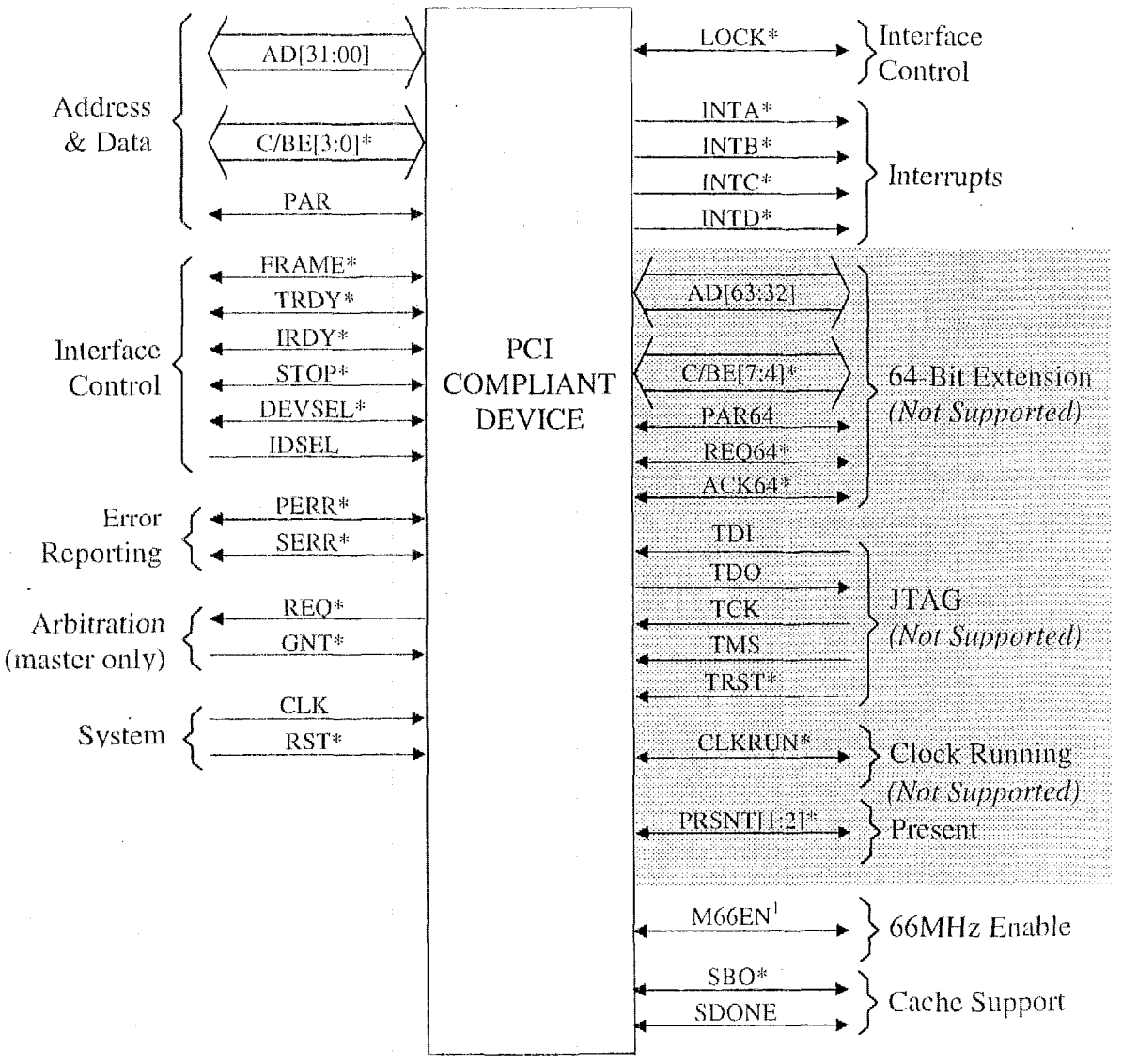
If errors are found in this document, please send a written copy of the suggested corrections to:

PC/104 Consortium
849B Independence Ave
Mountain View, CA 94043
Tel 415.903.8304
Fax 415.967.0995

2. PCI SIGNAL DEFINITION

Figure 1 shows the pins in functional groups, with the required pins on the left and the optional pins on the right side. The shaded pins on the right are unsupported features, but are included to show the entire PCI bus as defined in the PCI Revision 2.1 Specification. This version of the PCI bus is intended as a 32-bit bus running at 33MHz and therefore, 64-bit extension and 66MHz[1] are not supported at this time. Also not supported are the boundary scan features (JTAG), *Present* (PRSNR[1:2]*), and *Clock running* (CLKRUN*). The direction indication on the pins assumes a combination master/target device.

Figure 1: PCI Pin List



[1] The PCI bus has been simulated at 33MHz. For the purpose of this specification, 66MHz is not supported. To support future enhancements, the M66EN signal should be grounded on any module that cannot support 66MHz and left open for modules that can support a 66MHz clocks.

2.1 PCI Bus Signal Description

2.1.1 System

CLK	**Clock** provides timing for all transactions on the PCI bus.
RST*	**Reset** is used to bring PCI-specific registers to a known state.

2.1.2 Address and Data

AD[31:00]	**Address and Data** are multiplexed. A bus transaction consists of an address cycle followed by one or more data cycles.
C/BE[3:0]*	**Bus Command/Byte Enables** are multiplexed. During the address cycle, the command is defined. During the Data cycle, they define the byte enables.
PAR	**Parity** is even on AD[31:00] and C/BE[3:0]* and is required.

2.1.3 Interface Control Pins

FRAME*	**Frame** is driven by the current master to indicate the start of a transaction and will remain active until the final data cycle.
IRDY*	**Initiator Ready** indicates the master's ability to complete the current data cycle of the transaction.
TRDY*	**Target Ready** indicates the selected devices ability to complete the current data cycle of the transaction. Both IRDY* and TRDY* must be asserted to terminate a data cycle.
STOP*	**Stop** indicates the current selected device is requesting the master to stop the current transaction.
LOCK*	**Lock** indicates an operation that may require multiple transactions to complete.
IDSEL	**Initialization Device Select** is used as a chip-select during configuration.
DEVSEL*	**Device Select** is driven by the target device when its address is decoded.

2.1.4 Arbitration (Bus Masters Only)

REQ*	**Request** indicates to the arbitrator that this device desires use of the bus.
GNT*	**Grant** indicates to the requesting device that access has been granted.

2.1.5 Error Reporting

PERR*	**Parity Error** is for reporting data parity errors.
SERR*	**System Error** is for reporting address parity errors.

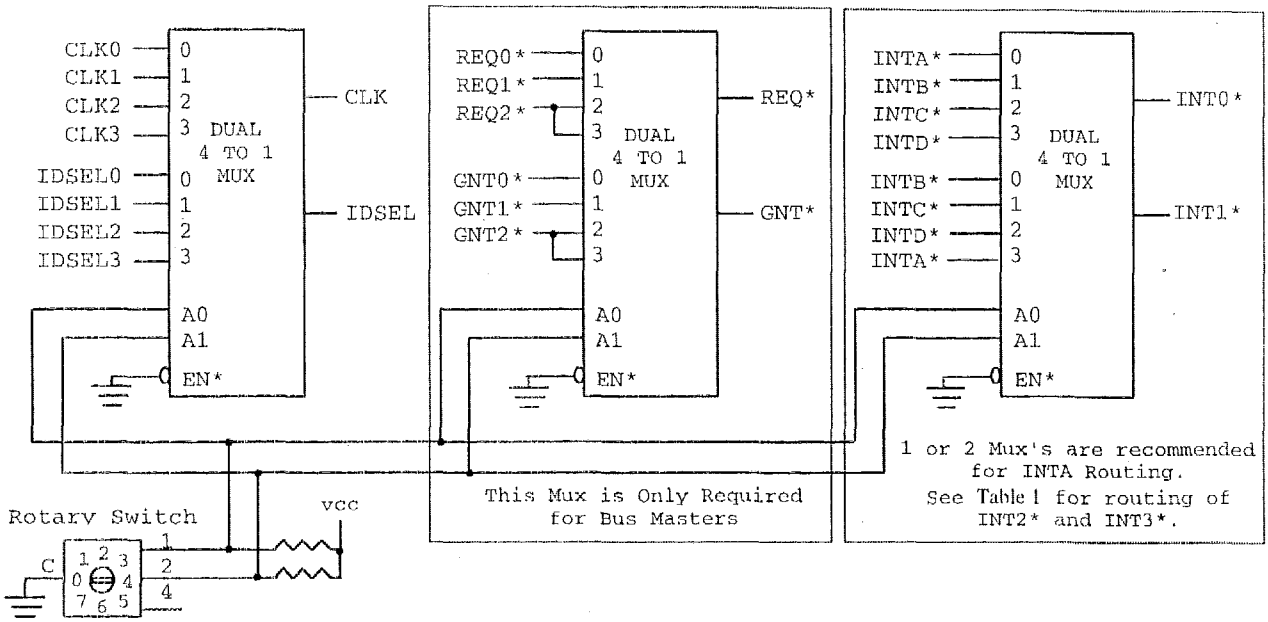
2.1.6 Interrupts

INTA*	**Interrupt A** is used to request an Interrupt.
INTB*	**Interrupt B** is used to request interrupts only for multi-function devices.
INTC*	**Interrupts C** is used to request interrupts only for multi-function devices.
INTD*	**Interrupts D** is used to request interrupts only for multi-function devices.

2.2 Signal Grouping

A means of selecting the appropriate signals must be established that will easily allow for the installation and configuration of add-in PC/104-*Plus* modules. Figure 2 shows such a method:

Figure 2: Signal Select



The multiplexer chips are Dual 4:1 Mux/Demux chips (QuickSwitch® QS3253 or equivalent). They provide a 5Ω switch that connects the input and output together. The nature of the switches, provide a bi-directional path with no signal propagation delay other than the RC delay of the on resistance of the switch and the load capacitance. This is typically 250ps at 50pF Load.

Other methods of configuring the modules are possible, but the rotary switch is the most convenient, cleanest and provides for the least possible error in configuration.

The clocks are tuned on the Host Board such that the length of CLK3 trace is ≈0.662" less than CLK2, CLK2 trace is ≈0.662" less than CLK1, and CLK1 trace is ≈0.662" less than CLK0. Therefore, the first module on the stack must select CLK0 (the longest trace), the second CLK1, etc. This provides basically no clock skew between modules. Table 1 shows the appropriate switch setting and signals used for each module in the stack. It is recommended that additional Mux chips be added to route Interrupts if required. Use one Mux for 1 to 2 interrupts or two Mux's for 3 to 4 interrupts.

Table 1: Rotary Switch Settings

Switch Position	Module Slot	REQ*	GNT*	CLK	IDSEL	ID Address	INT0*	INT1*	INT2*	INT3*
0 or 4	1	REQ0*	GNT0*	CLK0	IDSEL0	AD20	INTA*	INTD*	INTC*	INTB*
1 or 5	2	REQ1*	GNT1*	CLK1	IDSEL1	AD21	INTB*	INTA*	INTD*	INTC*
2 or 6	3	REQ2*[1]	GNT2*[1]	CLK2	IDSEL2	AD22	INTC*	INTB*	INTA*	INTD*
3 or 7	4	REQ2*[1]	GNT2*[1]	CLK3	IDSEL3	AD23	INTD*	INTC*	INTB*	INTA*

Note 1: Because module slots 3 and 4 share REQ2/GNT2, they cannot both be bus master devices.

3. ELECTRICAL SPECIFICATION

3.1 PC/104 Bus

The electrical specifications for the PC/104 bus for bus drive current, bus termination, pullup/pulldown resistors, etc. are unchanged and are defined in the PC/104 Specification. The signal assignments for the J1/P1 and J2/P2 connector are given in Appendix B, Table 4: PC/104 Bus (Reference Only).

3.2 PCI Bus

The PCI Bus mechanical interface is a stackable 30x4 header. This interface carries all of the required PCI signals per *PCI Local Bus Specification Version. 2.1.*

3.2.1 Signal Definitions

For full details on the electrical requirements for the PCI bus, reference the *PCI Local Bus Specification Version. 2.1.*

3.2.2 Signal Assignments

Signals are assigned in the same relative order as in the PCI Local Bus Specification, but transformed to the corresponding header connector pins. Because of the stackthrough nature of the bus, slot-specific signals are duplicated for each plug-in module. The system has been designed to accommodate 4 PC/104-*Plus* modules, so multiple sets of the signals have been duplicated to accommodate one signal for each module. These four signal groups include: IDSEL[3:0] - CLK[3:0] - REQ*[2:0] - GNT*[2:0]. Signal assignments for the J3/P3 connector are given in Appendix B, Table 3: PC/104-Plus Bus Signal Assignments.

3.2.3 Power And Ground Pins

The total number of power and ground signals remains the same, but the +3.3 V pins have been reduced by two and the ground pins have been increased by two. The change was the result of signal grouping on the bus and has no effect on performance or integrity.

3.2.4 Key Locations

The KEY pins are to guarantee proper module installation. Pin-A1 will be removed and the female side plugged for 5.0V I/O signals and Pin-D30 will be modified in the same manner for 3.3V I/O. Universal boards which can support either signal levels will have both key pins implemented. Universal boards must therefore be located at the top of the stack. See Appendix B, Table 3: PC/104-Plus Bus Signal Assignments.

3.2.5 AC/DC Signal Specifications

All bus timing and signal levels are identical to the *PCI Local Bus Specification Revision 2.1*.

3.3 Module Power Requirements

Table 2 specifies the voltage and maximum power requirements for each PC/104-*Plus* module. It should be noted that although the maximum requirements as specified are the same as the standard PC/104 specification, care should be used in designing PC/104-*Plus* modules to guarantee the least possible power consumption. A worst case module as specified could use almost 39 Watts of power, which would basically be unacceptable in most systems.

Table 2: Module Power Requirements

Supply	Min. Voltage	Max. Voltage	Max. Current	Max. Power
+3.3V[1]	3.00	3.60	3A	10.8W
+5V	4.75	5.25	2A	10.5W
+12V	11.4	12.6	1A	12.6W
-5V	-5.25	-4.75	0.2A	1.05W
-12V	-12.6	-11.4	0.3A	3.78W

Note 1: Host Boards implementing 5V signaling are not required to supply 3.3 volts to the modules, but must provide a bus and decoupling. If 3.3 volts is required for a module using the 5V signaling method, provisions should be made to provide its own 3.3 volts by means of an onboard regulator or some other input source. Host Boards implementing 3.3V signaling are required to supply 3.3 volts to the modules.

4. LEVELS OF CONFORMANCE

This section provides terminology intended to assist manufacturers and users of PC/104-*Plus* bus-compatible products in defining and specifying conformance with the PC/104-*Plus* Specification.

4.1 PC/104-*Plus* "Compliant"

This refers to "PC/104-*Plus* form-factor" devices that conform to all non-optional aspects of the PC/104-*Plus* Specification, including both *mechanical* and *electrical* specifications.

4.2 PC/104-*Plus* "Bus-compatible"

This refers to devices which are not "PC/104-*Plus* form-factor" (i.e., do not comply with the module dimensions of the PC/104-*Plus* Specification), but provide male or female PC/104-*Plus* bus connectors that meets both the *mechanical* and *electrical* specifications provided for the PC/104-*Plus* bus connectors.

4.3 PC/104-*Plus* "PCI-Only"[2]

Because the PC/104-*Plus* standard encompasses two different buses (i.e. PC/104 104-pin "ISA" bus and 120-pin "PCI" bus), it is possible for PC/104-*Plus* compliant or compatible modules to implement PCI only. Such modules shall have the added designation "PCI-Only" in addition to

[2] This precludes stacking standard PC/104 Modules.

the designation specified in either 4.1 and 4.2. Example: PC/104-*Plus* "Compliant", PCI-Only or PC/104-*Plus* "Bus-compatible", PCI-Only.

5. MECHANICAL SPECIFICATION

5.1 Module Dimensions

The mechanical dimensions for this module are identical to PC/104 specification with the exception of the added connector (J3), some modifications to the I/O connector area, and changes to the component height restrictions. The component height on the top side has been reduced from 0.435" to 0.345" and the bottom has been increased from 0.100" to 0.190". The component restrictions across and to each side of the PC/104 connectors (three sides, 0.400" from each edge) remains the same as the PC/104 specification. The mechanical dimensions and restrictions are given in Appendix A, Figure 4: Module Dimensions.

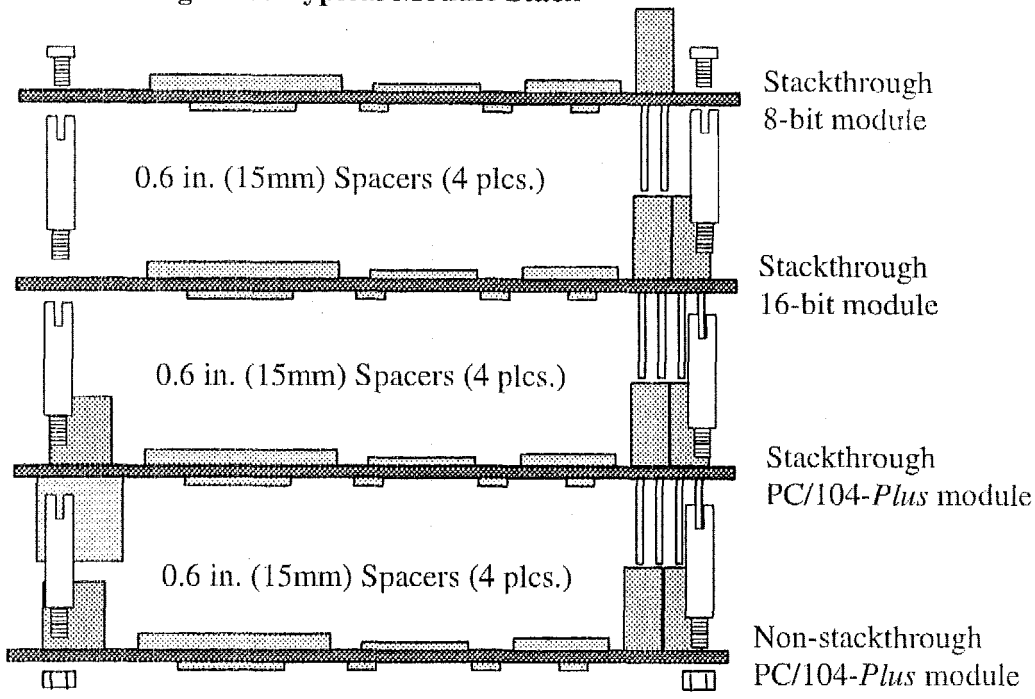
5.2 Connector And Shroud

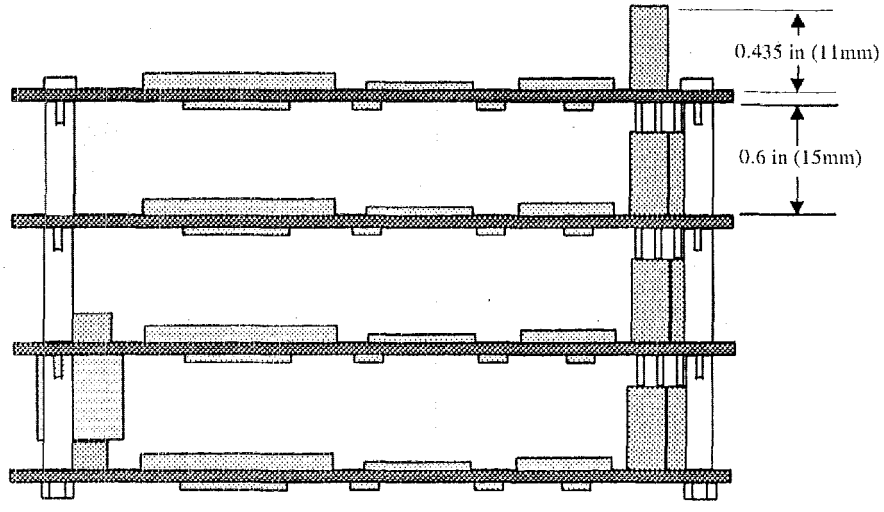
The PC/104-*Plus* connector for the PCI bus is a 4x30 (120-pin) 2mm pitch connector. The Shroud should be installed on the bottom of the PC board when a stackthrough connector is used. The mechanical dimensions and restrictions are given in Appendix A, Figure 5: PCI Connector.

6. TYPICAL MODULE STACK

Figure 3 shows a typical module stack with 2 PC/104-*Plus* modules, 1 PC/104 16-Bit module, and 1 PC/104 8-Bit module. The maximum configuration for the PCI bus of PC/104-*Plus* modules is 4 plus the Host Board. If standard PC/104 modules are used in the stack, they must be the top module(s) because they will normally not include the PCI bus.

Figure 3: Typical Module Stack





APPENDIX A

MECHANICAL DIMENSIONS

Figure 4: Module Dimensions

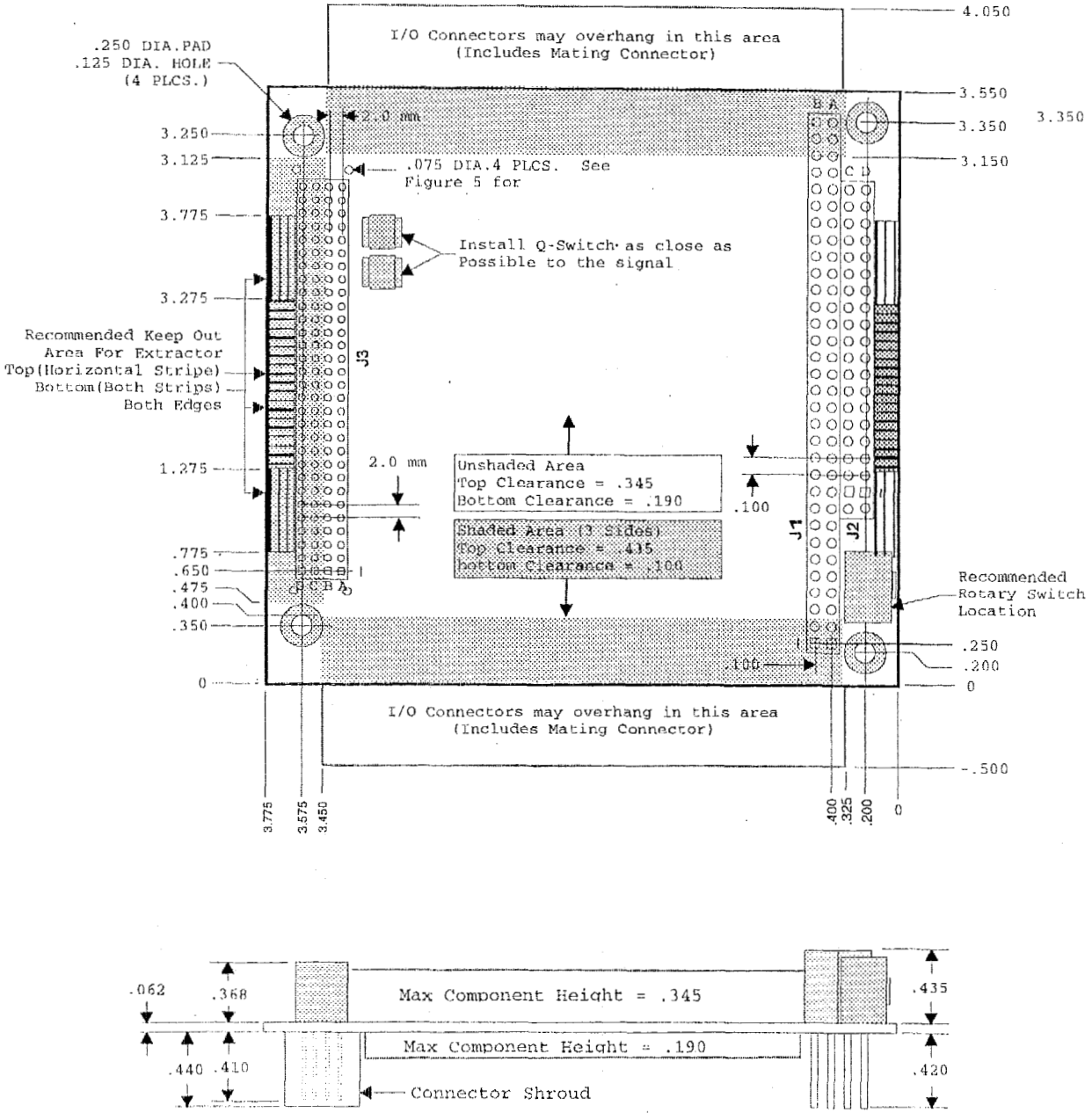


Figure 5: PCI Connector

Ⓐ shows the pin dimensions for the Stackthrough connector.
(Samtec ESQT-130-02-G-Q-368 or equivalent)

Ⓑ shows the pin dimensions for the Soldertail connector.
(Samtec ESQT-130-03-M-Q-368 or equivalent)

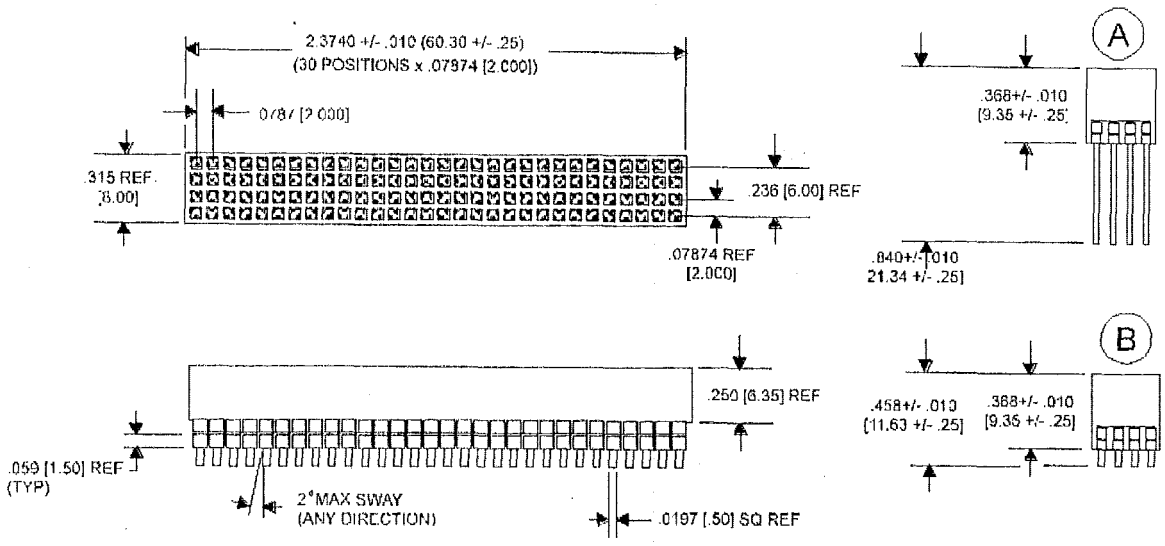
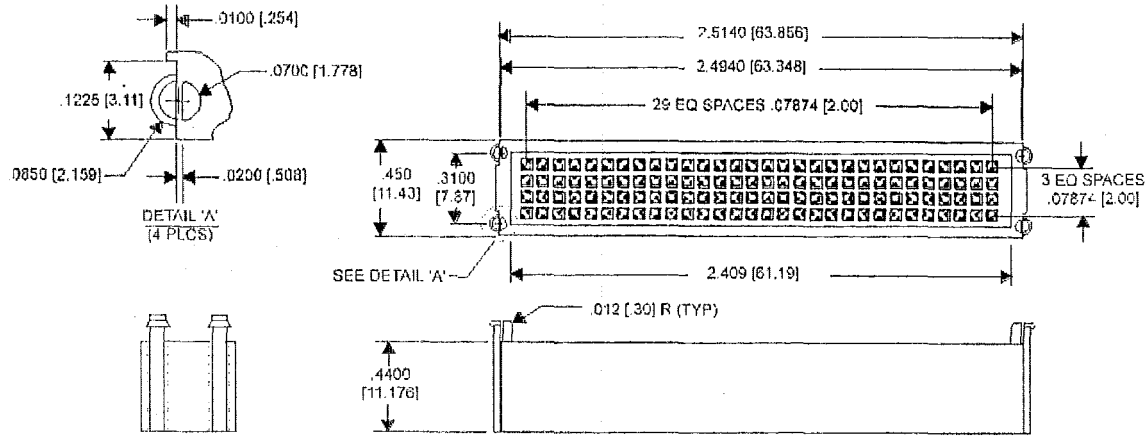


Figure 6: PCI Shroud

(Samtec TS-30-Q or equivalent)



APPENDIX B

BUS SIGNAL ASSIGNMENTS

Table 3: PC/104-*Plus* Bus Signal Assignments

J3/P3				
Pin	**A**	**B**	**C**	**D**
1	GND/5.0V KEY[2]	Reserved	+5	AD00
2	VI/O	AD02	AD01	+5V
3	AD05	GND	AD04	AD03
4	C/BE0*	AD07	GND	AD06
5	GND	AD09	AD08	GND
6	AD11	VI/O	AD10	M66EN
7	AD14	AD13	GND	AD12
8	+3.3V	C/BE1*	AD15	+3.3V
9	SERR*	GND	SB0*	PAR
10	GND	PERR*	+3.3V	SDONE
11	STOP*	+3.3V	LOCK*	GND
12	+3.3V	TRDY*	GND	DEVSEL*
13	FRAME*	GND	IRDY*	+3.3V
14	GND	AD16	+3.3V	C/BE2*
15	AD18	+3.3V	AD17	GND
16	AD21	AD20	GND	AD19
17	+3.3V	AD23	AD22	+3.3V
18	IDSEL0	GND	IDSEL1	IDSEL2
19	AD24	C/BE3*	VI/O	IDSEL3
20	GND	AD26	AD25	GND
21	AD29	+5V	AD28	AD27
22	+5V	AD30	GND	AD31
23	REQ0*	GND	REQ1*	VI/O
24	GND	REQ2*	+5V	GNT0*
25	GNT1*	VI/O	GNT2*	GND
26	+5V	CLK0	GND	CLK1
27	CLK2	+5V	CLK3	GND
28	GND	INTD*	+5V	RST*
29	+12V	INTA*	INTB*	INTC*
30	-12V	Reserved	Reserved	GND/3.3V KEY[2]

Notes:
1. The shaded area denotes power or ground signals.
2. The KEY pins are to guarantee proper module installation. Pin-A1 will be removed and the female side plugged for 5.0V I/O signals and Pin-D30 will be modified in the same manner for 3.3V I/O. It is recommended that both KEY pins (A1 and D30) be electrically connected to GND for shielding.

Table 4: PC/104 Bus (Reference Only)

J2/P2		
Pin	**D**	**C**
0	GND	GND
1	MEMCS16*	SBHE*
2	IOCS16*	LA23
3	IRQ10	LA22
4	IRQ11	LS21
5	IRQ12	LS20
6	IRQ15	LS19
7	IRQ14	LA18
8	DACK0*	LA17
9	DRQ0	MEMR*
10	DACK5*	MEMW*
11	DRQ5	SD8
12	DACK6*	SD9
13	DRQ6	SD10
14	DACK7*	SD11
15	DRQ7	SD12
16	+5V	SD13
17	MASTER*	SD14
18	GND	SD15
19	GND	GND/KEY

J1/P1		
Pin	**A**	**B**
1	IOCHCK*	GND
2	D7	RSTDRV
3	D6	+5V
4	D5	IRQ9
5	D4	-5V
6	D3	DRQ2
7	D2	-12V
8	D1	ENDXFR*
9	D0	+12V
10	IOCHRDY	GND/KEY
11	AEN	SMEMW*
12	A19	SMEMR*
13	A18	IOW*
14	A17	IOR*
15	A16	DACK3*
16	A15	DRQ3
17	A14	DACK1*
18	A13	DRQ1
19	A12	REFRESH*
20	A11	SYSCLK
21	A10	IRQ7
22	A9	IRQ6
23	A8	IRQ5
24	A7	IRQ4
25	A6	IRQ3
26	A5	DACK2*
27	A4	TC
28	A3	BALE
29	A2	+5V
30	A1	OSC
31	A0	GND
32	GND	GND

5.4 Parts List for the Overall Computational Block

The following table lists all the component parts of the computational block. The IC components on the various boards are listed in the corresponding sections.

Table of Computational Block Hardware Components

Vendor	**Part Number**	**Description**
Ampro	CM2-SXI-Q-73	3SXi 386 Processor Card
Ampro	MM2-SVG-Q-72	SVG-II VGA Video Card [1-Mbyte video memory]
Diamond Systems	Emerald-MM-DIO	Extra 4 Serial Ports & 48-bits of Digital I/O Card
Corcom	6EC1	RFI Power-Line Filter with AC connector
Condor	GSC20-5	5V 3.8A Switching Power Supply
		Push-button On/Off Switch
		Fuse
		PC104 card standoffs
		External Case
Amp		9-pin D-Sub Male Connector for Serial Port
Amp		15-pin D-Sub Female Connector for Parallel Port
Amp		25-pin D-Sub Female Connector for Serial Port
Amp		15-pin D-Sub Female Connector for Video Output

5.5 Supporting Hardware

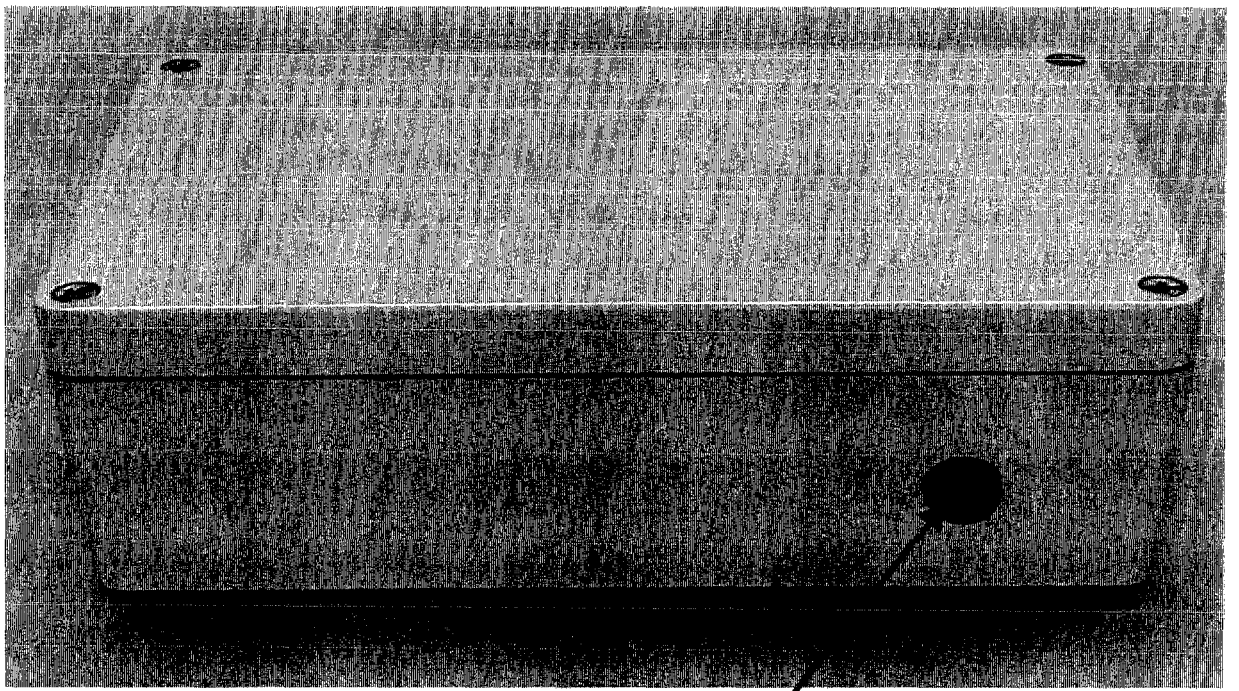
The supporting hardware in the computational block is not on the direct signal path for classified information. The following section provides documentation on supporting hardware.

5.5.1 Photographs of Supporting Hardware

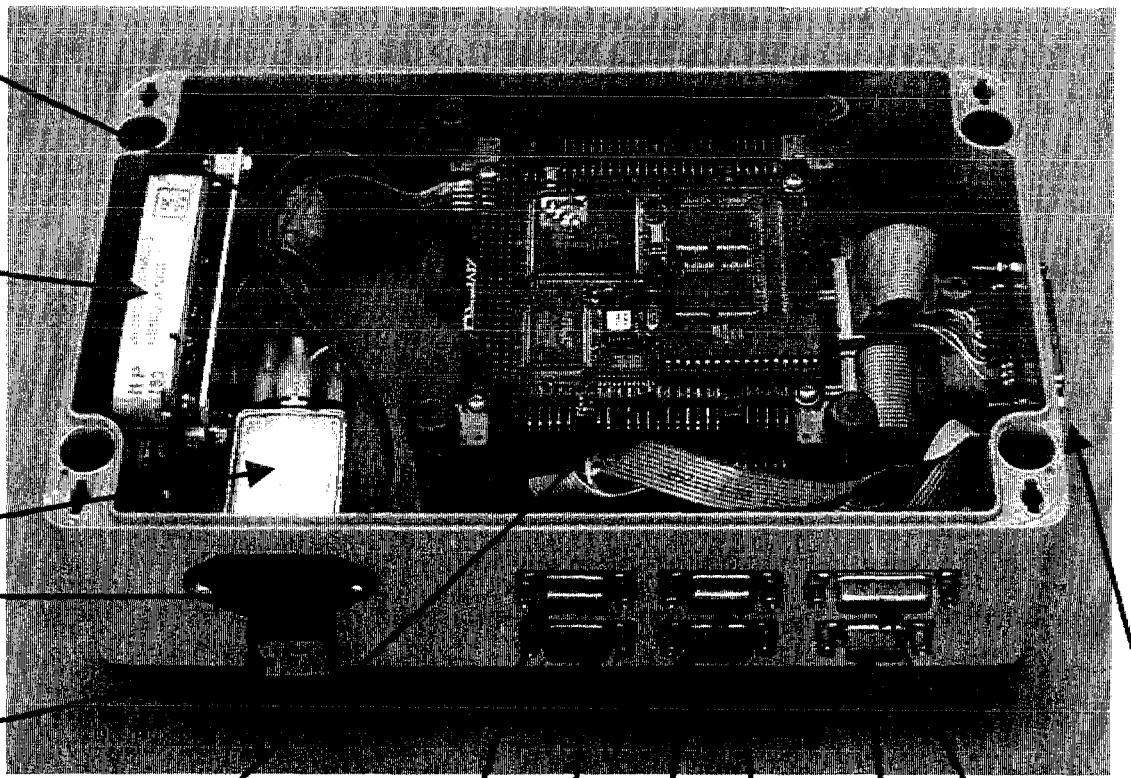
The following photographs are included to illustrate the design and layout of the computational block hardware and the supporting hardware.

- Front-view with the lid on showing the power on switch
- Inside-view with the lid off showing the internal components and the back panel connectors
- Back-view with the lid on showing the various back panel connectors
- Side-view with the lid on showing the side panel connector to the data barrier
- A view of the PC-104 single-board computer stack
- An inside-view of the computational block with the PC-104 stack removed to better show connector details
- An inside-view of the computational block showing the inner portion of the connectors and the ribbon cables used to connect to the PC-104 stack
- An inside-view of the computational block showing details of the line filter and power supply.

Computational Block
Front-View Photograph

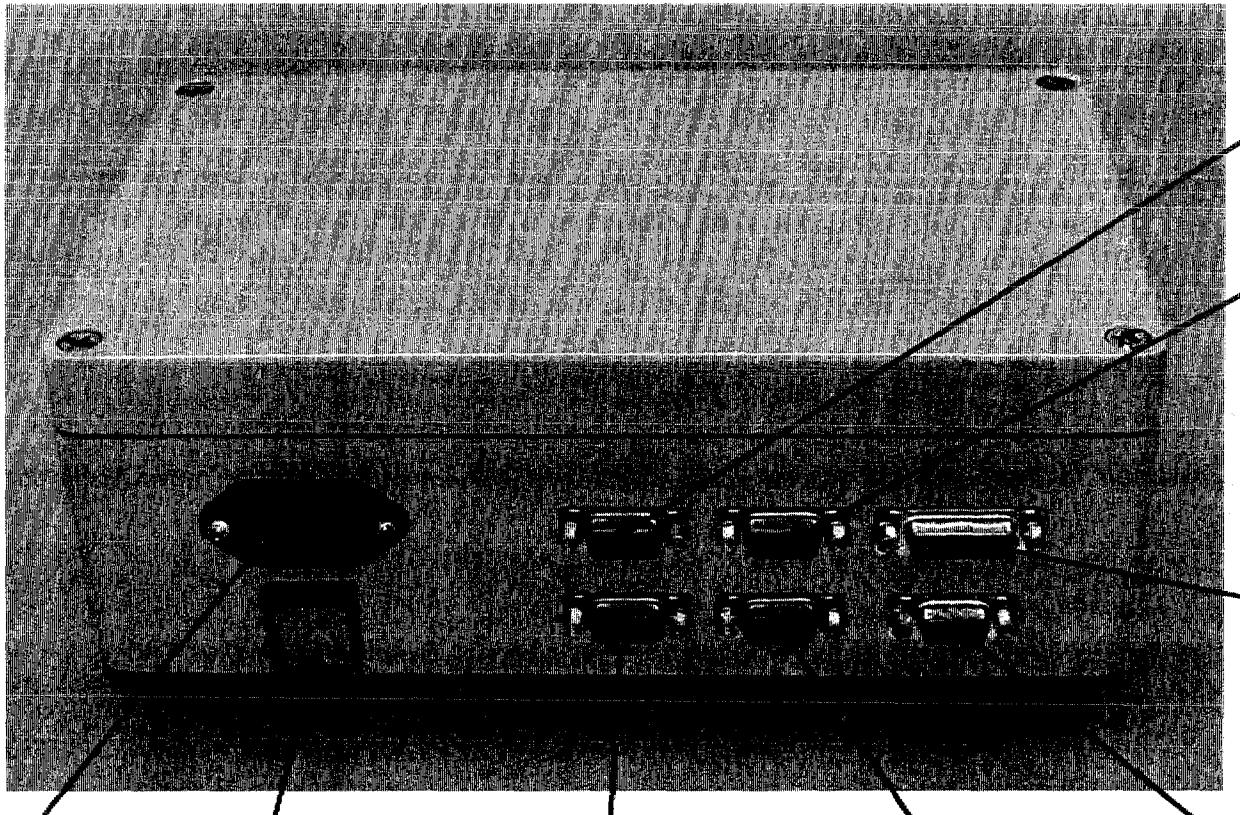


Computational Block Inside-View Photograph

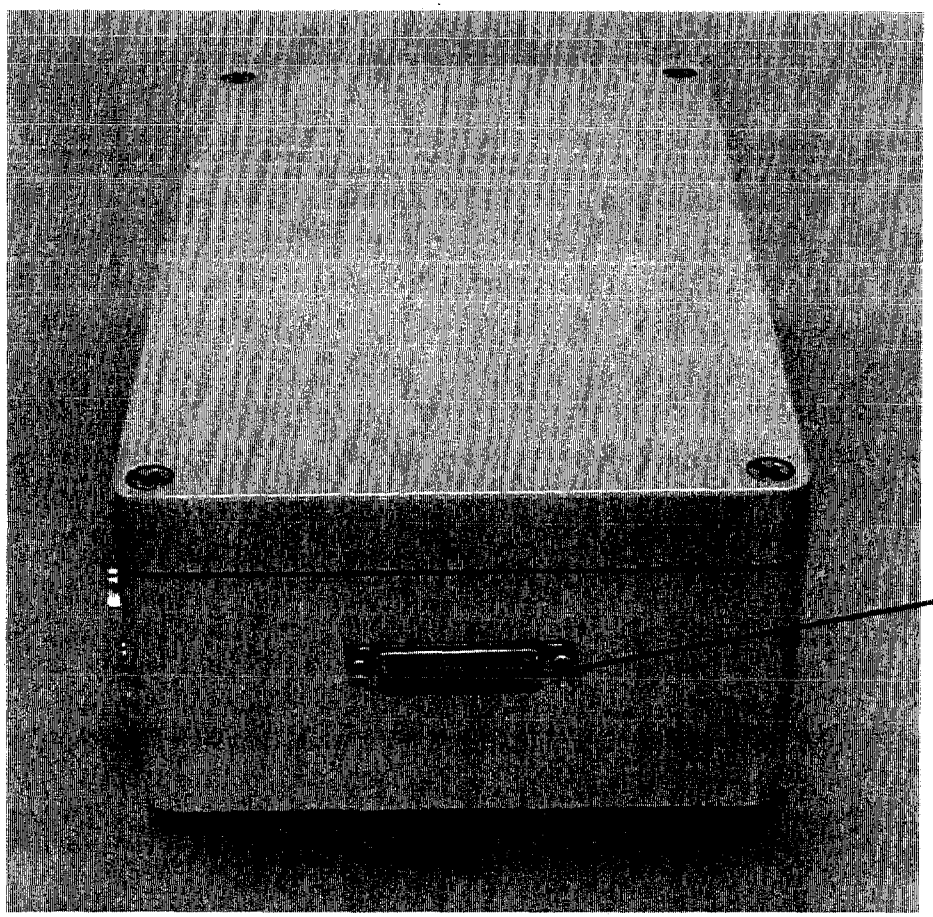


- Ampro: 3SXi processor card
- Ampro: SVG-II video card
- Diamond Systems: Emerald-MM-DIO card

Computational Block Connector-Panel Photograph

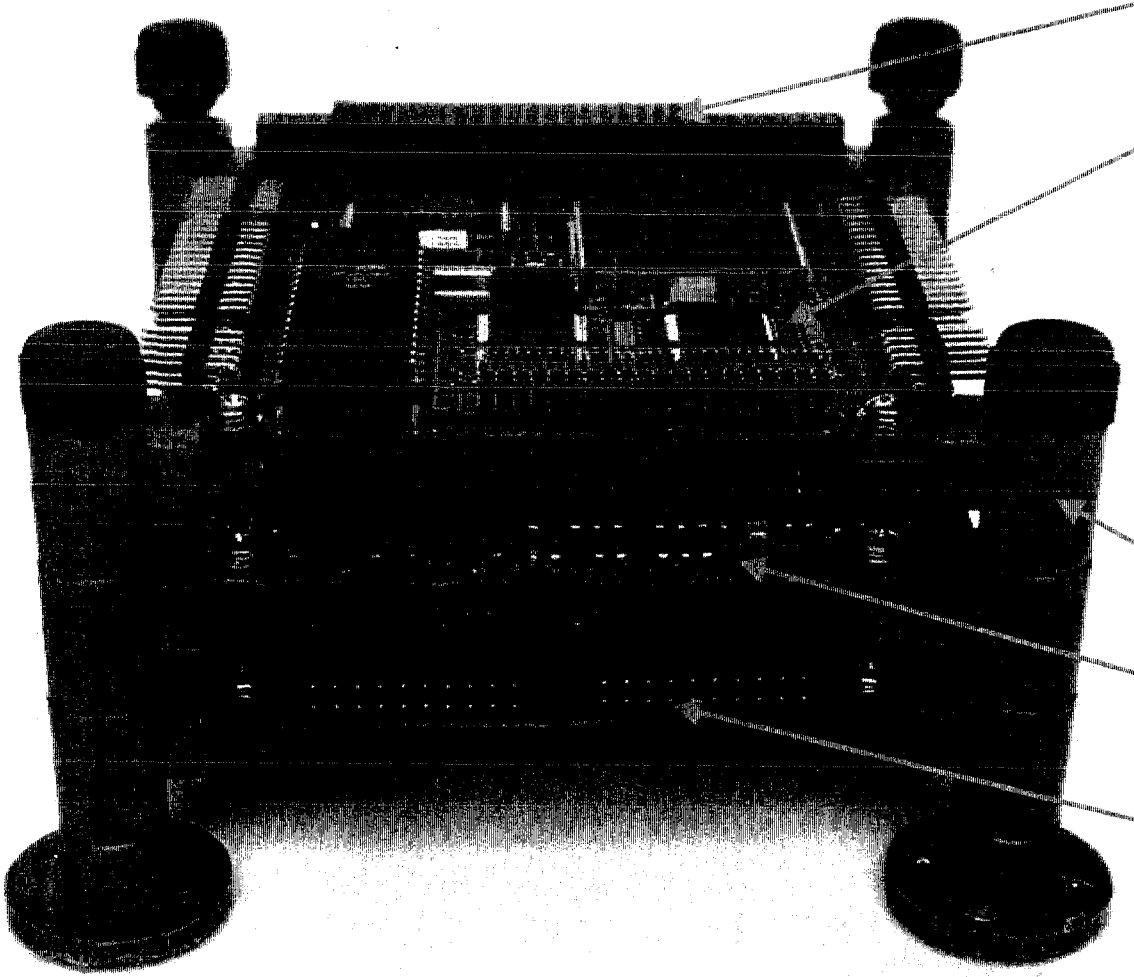


Computational Block
Connector-to-Data-Barrier Photograph

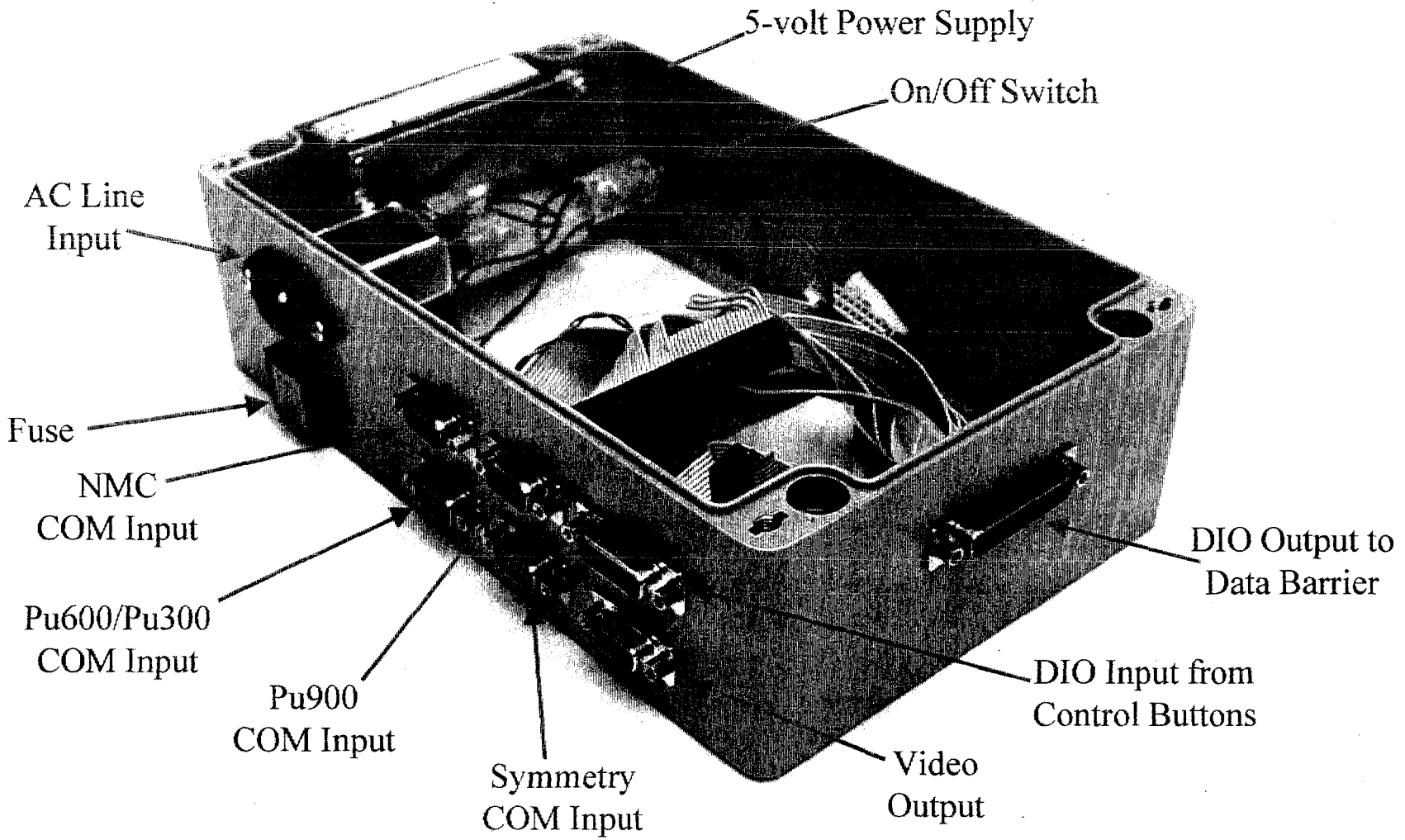


Display-Light
Data to
Data Barrier
DIO Output

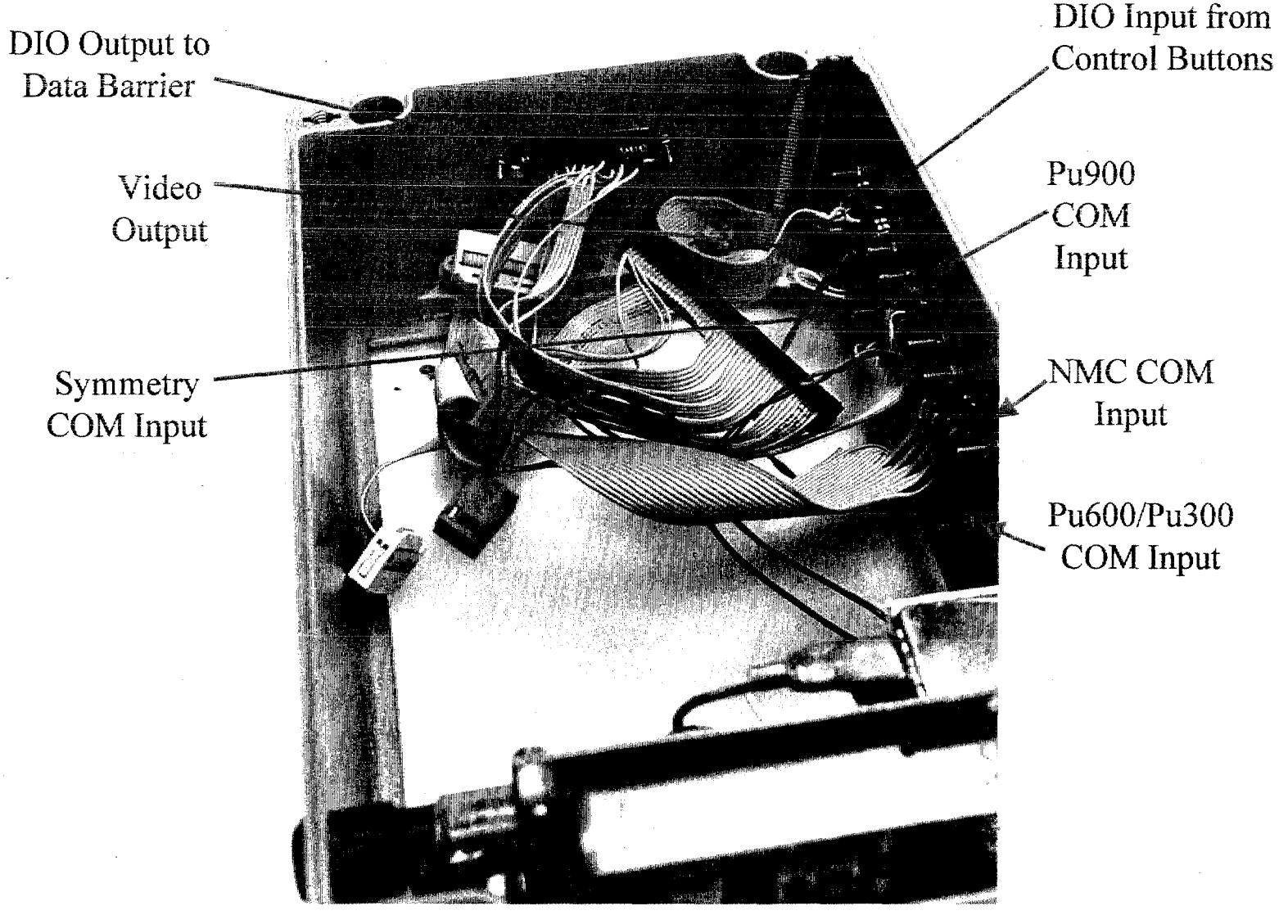
PC-104 Stack


PC-104 Bus Connectors
Ampro 3SXi
CPU Board
Standoffs
Ampro SVG-II
Video Board
Emerald-MM-DIO
Extra I/O Board

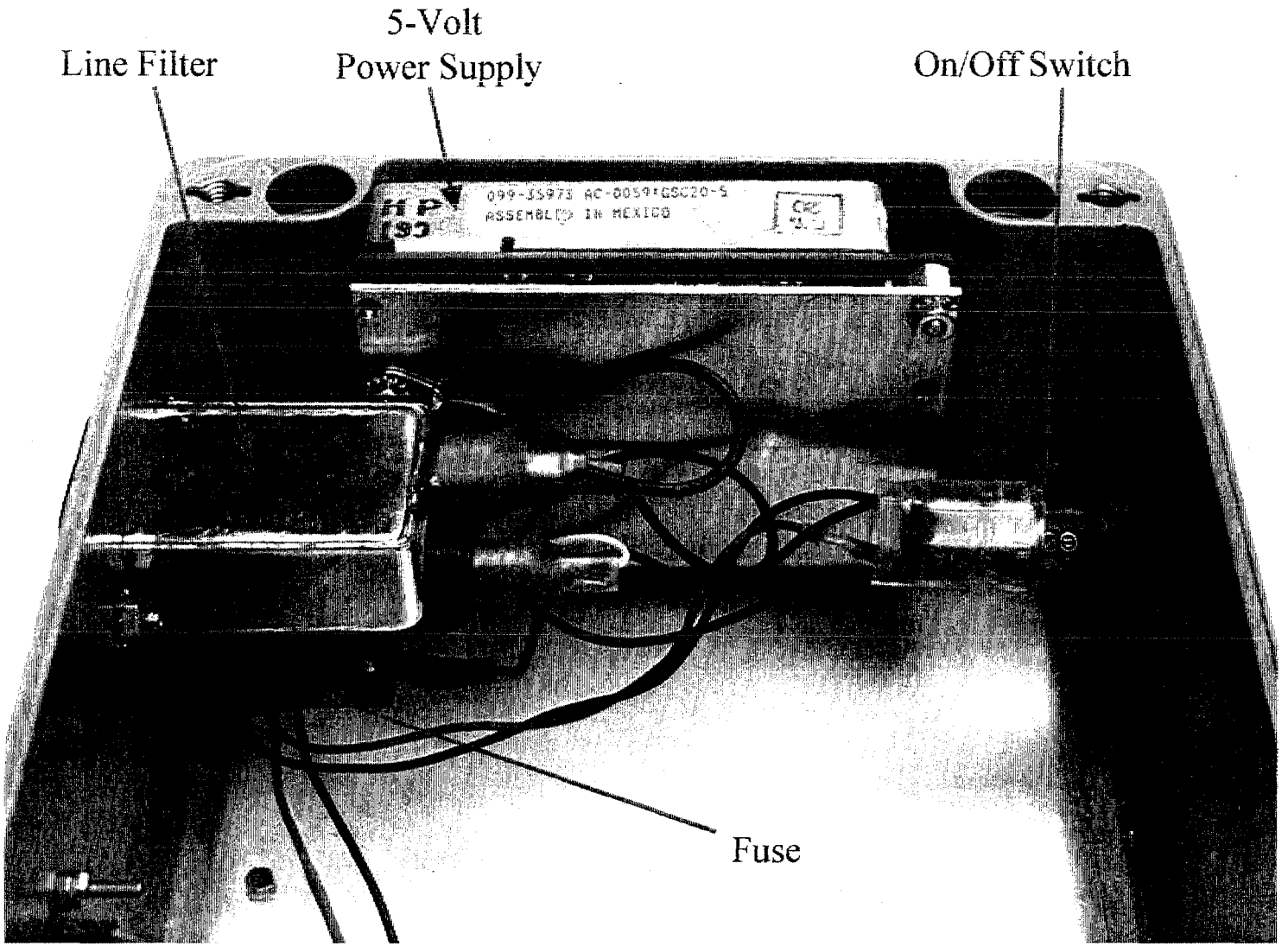
Computation Block
Connector Details



Inside of Computation Block
Connector Details



Inside of Computation Block
Line Filter and Power Supply Details



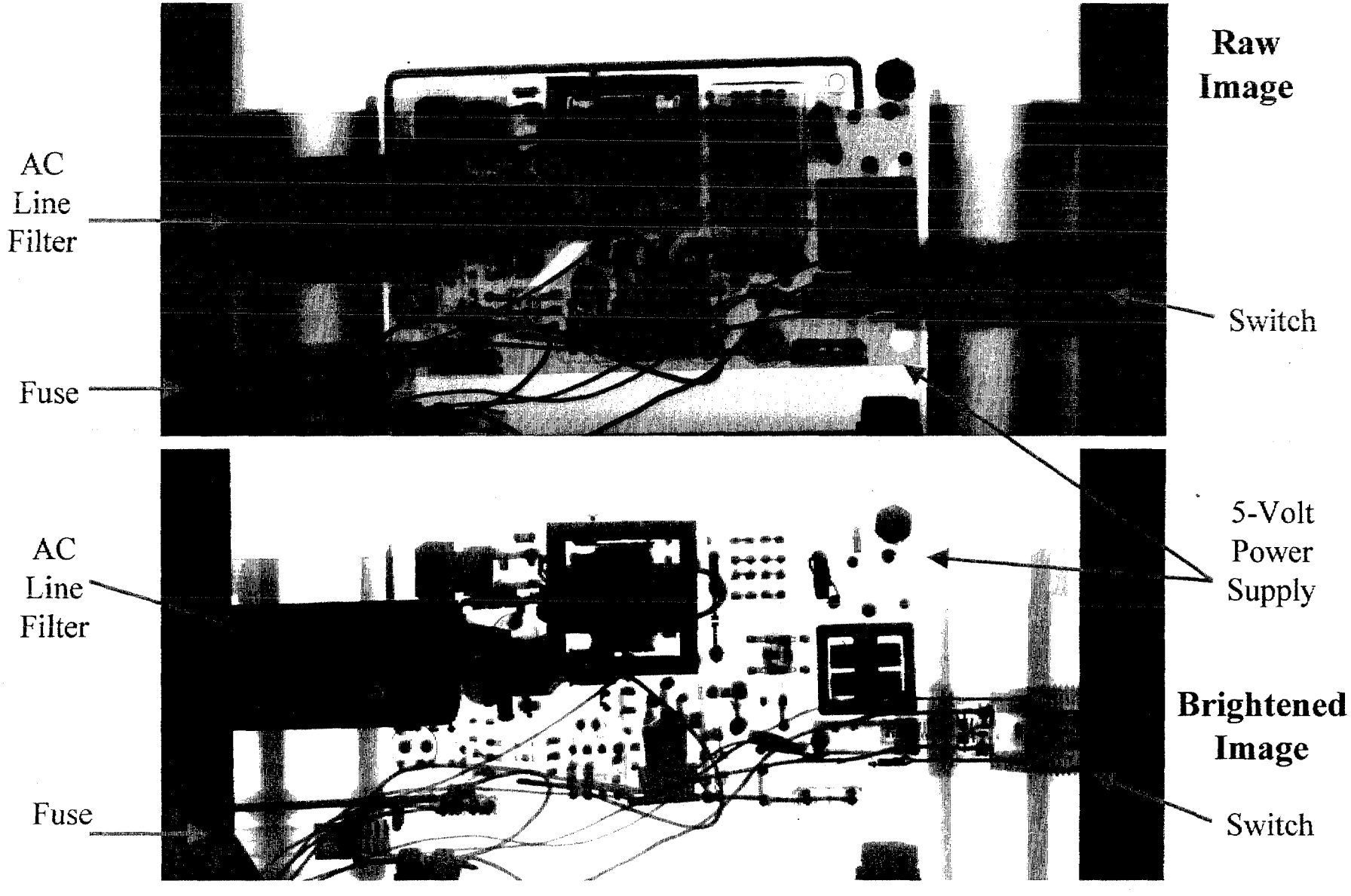
5.5.2 X-Rays of Supporting Hardware

The following x-ray is included:

- A x-ray of the computational block showing details of the line filter and power supply.

This x-ray can be compared to the last picture in section 5.5.1 of the same items. The upper image is a printout of the scanned x-ray negative. The lower image is a computer-brightened view of the same x-ray negative, which better shows the interior of the AC line filter. Altering the energy of the x-rays to be more penetrating could also be used to alter the details observed.

Computation Block – X-Ray
Line Filter and Power Supply Details



5.5.3 Specifications for the Corcom 6EC1 RFI Power-Line Filter

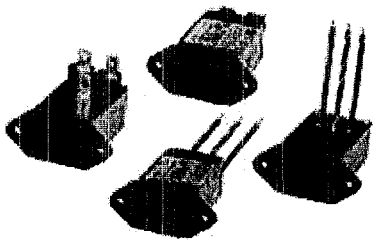
The following specifications were printed from the Corcom Internet site at:

http://www.cor.com/catalog/filters/ec/default.htm.

CII Technologies
advanced control electronic solutions
Corcom Division

EC Series

Home
Catalog
Sales Info
Tech Support
Search Tips
Europe
CII Home



EC Series

(1-10 Amp) Highest Performance Compact RFI Power Line Filters with IEC Connectors

Back To: *Catalog* OR *Selector Guide*

Description | Schematic | Specifications | Dimensions | Price List | AutoCAD | PDF File

To request a sample, use our Sample Request page.

Corcom is an ISO 9001 registered firm

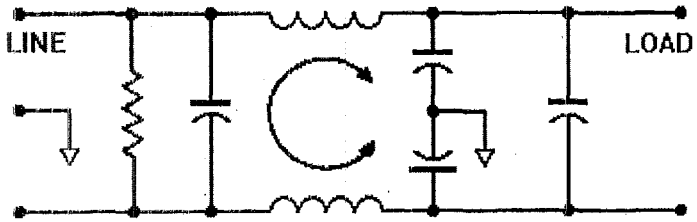
ISO 9001
A3256 A4003

EC Series RFI power line filters provide the highest attenuation of any standard filters available in a package size limited by the dimensions of the integral IEC connector. These filters combine high common-mode inductance with high differential-mode capacitance and minimal parasitic elements, for effective attenuation of line-to-ground and line-to-line noise across the frequency range.

Performance and applications are similar to those of the ED series, but with higher differential-mode performance in most applications, due to additional capacitor on the load side.

Note 1: Except all 10A and styles EC4 and EC8
Note 2: Our products have VDE approval which assists in obtaining the **CE Marking** in accordance with the EC Low Voltage Directive

Electrical Schematic



Resistor positions for reference only.

Specifications

Maximum leakage current, each line-to-ground

@120 VAC 60 Hz:	.25 mA
@250 VAC 50 Hz:	.50 mA

Hipot rating (one minute):

line-to-ground	2250 VDC
line-to-line	1450 VDC

Operating frequency: 50/60 Hz

Rated voltage: 120/250 VAC

Minimum insertion loss in dB:

Line-to-Ground in 50 ohm circuit:						
Current Rating	Frequency-MHz					
	.15	.5	1	5	10	30
1A	25	35	40	50	50	50
3A	20	30	37	47	48	50
6A	15	22	25	40	45	50
10A	7	14	20	35	39	48

Line-to-line in 50 ohm circuit:							
Current Rating	Frequency-MHz						
	.15	.5	1	5	10	20	30
EC1, EC2, EC8							
1A	5	35	50	60	60	40	40
3A	5	25	45	60	55	34	34
6A	10	10	40	65	60	40	40
10A	10	10	27	65	56	38	38
EC4							
1A	5	35	50	60	60	33	33
3A	5	30	45	60	55	34	34
6A	10	10	40	65	60	33	33

Case Dimensions

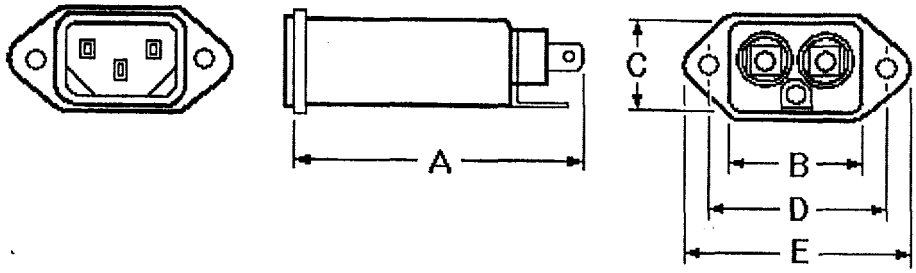
Metric shown in italics.

Part No.	A (max)	B (max)	C (max)	D +/-0.15 +/-.38	E (max)	F (ref)
1EC1	2.62 66.5	1.19 30.2	0.81 20.6	1.575 40.01	1.98 50.3	
1EC2	1.97 50.0	1.19 30.2	0.82 20.8	1.575 40.01	1.98 50.3	
1EC4	1.97 50.0	1.19 30.2	0.82 20.8	1.575 40.01	1.98 50.3	.295 7.50
1EC8	1.98 50.3	1.19 30.2	0.81 20.6	1.575 40.01	1.98 50.3	.295 7.50
3EC1	2.62 66.5	1.19 30.2	0.81 20.6	1.575 40.01	1.98 50.3	
3EC2	1.97 50.0	1.19 30.2	0.82 20.8	1.575 40.01	1.98 50.3	
3EC4	1.97 50.0	1.19 30.2	0.82 20.8	1.575 40.01	1.98 50.3	.295 7.50
3EC8	1.98 50.3	1.19 30.2	0.81 20.6	1.575 40.01	1.98 50.3	.295 7.50
6EC1	2.62 66.5	1.19 30.2	0.81 20.6	1.575 40.01	1.98 50.3	
6EC2	1.97 50.0	1.19 30.2	0.82 20.8	1.575 40.01	1.98 50.3	
6EC4	1.97 50.0	1.19 30.2	0.82 20.8	1.575 40.01	1.98 50.3	.295 7.50
6EC8	1.98 50.3	1.19 30.2	0.81 20.6	1.575 40.01	1.98 50.3	.295 7.50
10EC1	2.62 66.5	1.19 30.2	0.81 20.6	1.575 40.01	1.98 50.3	

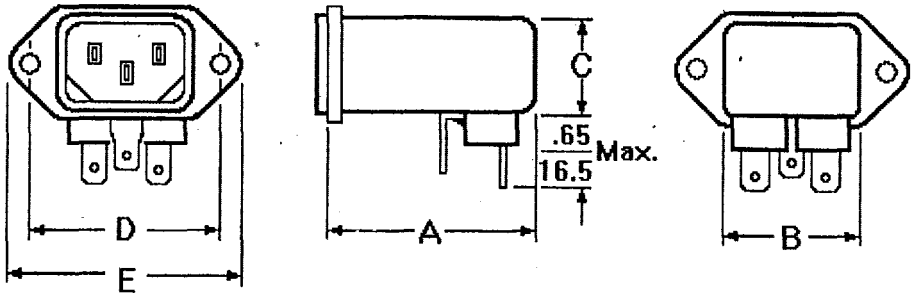
Case Styles

Metric shown in italics.

EC1



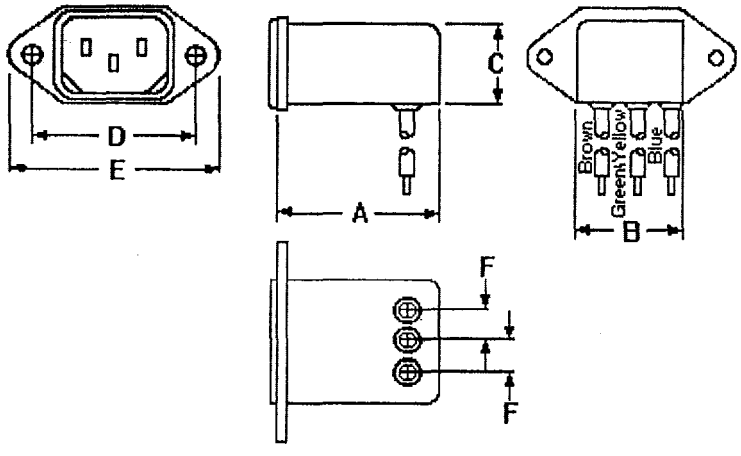
EC2



Typical dimensions

Fastons: $\frac{.250}{6.35}$ (3) **Holes** $\frac{.07}{1.8}$ Dia. **Mounting Holes:** $\frac{.132}{3.35}$ Dia. (2)

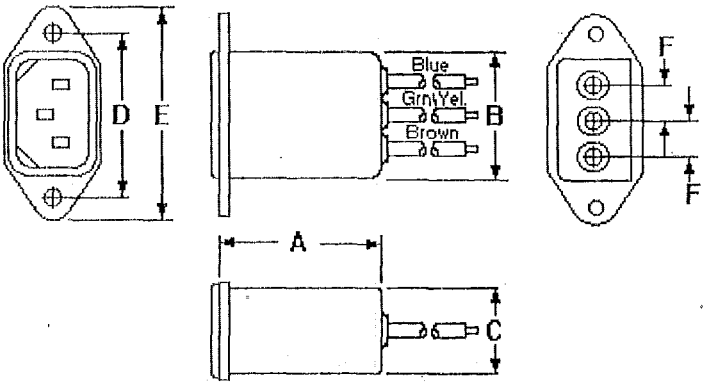
EC4



Typical dimensions

Wire leads: 4.0 / *101.6* Min. **Mounting Holes:** .132 / *3.35* Dia. (2)

EC8



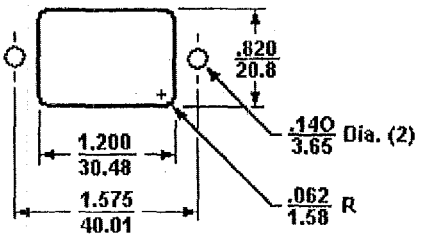
Typical dimensions

Wire leads: 4.0 / *101.6* Min. **Mounting Holes:** .132 / *3.35* Dia. (2)

Recommended Panel Cutouts

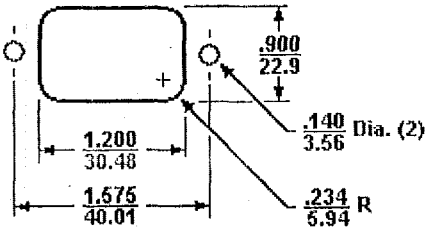
Panel cutout (Front mount)

Tolerance: +/- .005 / *0.13*



Panel cutout (Back mount)

Tolerance: +/- .005 / *0.13*



Note 1: EC1 and EC8 allow for front or back mounting.

Note 2: EC2 and EC4 allow for back mounting only.

Price List

Part No.	Unit Price	Part No.	Unit Price
1EC1	$20.70	6EC1	$20.70
1EC2	20.70	6EC2	20.70
1EC4	20.70	6EC4	20.70
1EC8	20.70	6EC8	20.70
3EC1	20.70	10EC1	21.83
3EC2	20.70		
3EC4	20.70		
3EC8	20.70		

Single piece price provided for comparison value.

Download AutoCAD® format drawing of part number:

1EC1	3EC1	6EC1	10EC1
1EC2	3EC2	6EC2	
1EC4	3EC4	6EC4	
1EC8	3EC8	6EC8	

[Home] [Contact Us]

Questions or Comments about this website? Please contact webmaster@cor.com

This page was last updated 01/21/00 02:53 PM

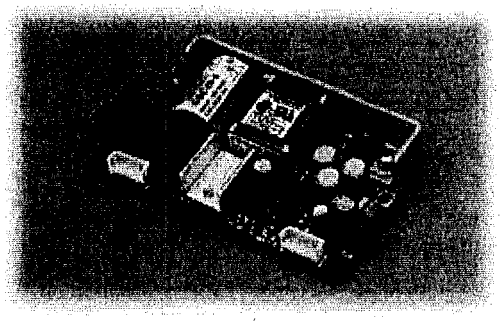
5.5.4 Specifications for the Condor GSC20-5 Switching Power Supply

The following specifications were printed from the Powerbox Internet site at:

http://www.powerbox.com.au/gsc20.htm.

GSC20 GLOBAL PERFORMANCE SWITCHERS

- Industry's smallest 20 W switcher
- Business card size (2.0" x 3.5" x 0.71")
- Approved to UL 1950, IEC 950 & CSA22.2-234 Level 3, EN60950
- EMI FCC Class B, CISPR22 B
- Overvoltage protection standard
- CE marked to LVD



SPECIFICATIONS

AC Input:	90-264 Vac, 47-63 Hz single phase.
Input Current:	Maximum input current at minimum 120 Vac, 60 Hz with full rated output load is 0.6 A.
DC Outputs:	See voltage current rating chart
Hold-up Time:	15 ms minimum from loss of AC input at full load nominal line (115 Vac).
Output Power:	Normal continuous output power is 20 W, 24 W peak for 60 s maximum duration, 10% duty cycle. Factory set to begin power limiting at approximately 28 W.
Output Regulation:	Regulation from initial setpoint measured by changing load from 5% load to 50% load or 50% load to full load in either direction. Initial setpoint tolerance is measured at 50%load.A minimum load of 5% of the output current is required to maintain proper regulation.
Overload Protection:	Fully protected against short circuit and output overload. Short circuit protection is cycling type power limit.
No Load Turn-on/ Standby:	No degradation of reliability will occur, however regulation may be affected.
Output Noise:	0.5% RMS, 1% pk-pk, 20 MHz bandwidth, differential mode. Measured with scope probe directly across output terminals of the power supply with load terminated with 0.1 uF capacitor.
Transient Response:	(Main output) 750 ms typical response time for return to within 0.5% of final value for a 50 % load step dl/dt
Reverse Voltage Protection:	All outputs protected against inadvertent application of reverse voltage up to 1 times rated current of the reversed output.
Overvoltage Protection:	Built in with firing point set per Table 1. OVP firing I reduces voltage to less than 50% of nominal voltage in 50 ms.
Adjustment:	Factory set with fixed resistors to maximize reliability.
Efficiency:	70% minimum for the 5.1 V model at full rated load, nominal input voltage. Efficiency increases as output voltage increases.
Overshoot:	Less than 2% overshoot at turn-on under nominal conditions, inversely proportional to input voltage and temperature. Less than 2% overshoot at turn-off under all conditions.
Turn-on Time:	Less than 1 second at 115 Vac, 25 °C (inversely proportionate to input voltage and thermistor temperature).
Input Protection:	Internal AC fuse provided on all units. Designed to open only if a catastrophic failure occurs in the unit- fuse does not blow on overload or short circuit.

Inrush Current:	Inrush is limited by internal thermistor. The inrush at 230 Vac, averaged over the first AC half-cycle under cold start conditions will not exceed 32 A.
Temperature Coefficient:	0.03%/°C typical on all outputs.
Temperature Range:	0 to 50 °C at full rated output power. For operation in a confined space, moving air may be required. Under all conditions, the cooling vs. load profile should be such that the heatsink temperatures do not exceed 90 °C for extended periods.
Storage:	- 40 to + 85 °C. Units should be allowed to warm-up under non-condensing conditions before application of power.
Altitude:	-500 to 10,000 ft.
Earth Leakage Current:	Leakage current measured in the Ground wire connection when measured per UL1950 or UL2601 is as follows:

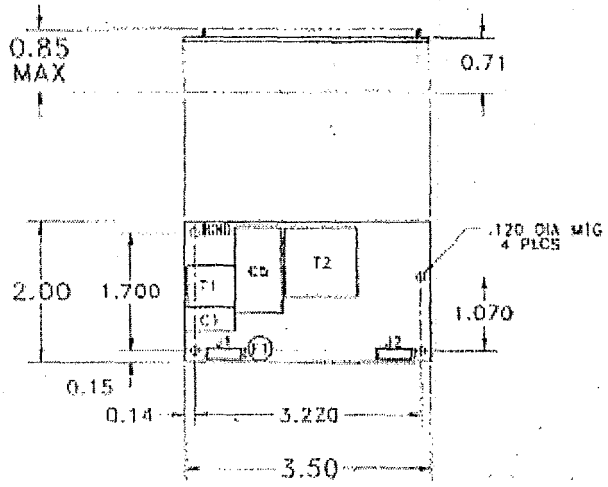
Model	Leakage Current*	Leakage Current**	Test Voltage	Test Method
GSC20	165 µA	330 µA	132 Vac/60 Hz	UL1950
GSC20	370 µA	740 µA	254 Vac/50 Hz	IEC950

*Normal Operation **Abnormal/Single fault condition

EMI Compliance:	All models include built-in EMI filtering to meet FCC Class B, CISPR 22 Class B.
Safety:	Condor D.C. Power Supplies, Inc. declares under our sole responsibility that all GSC models are in conformity with the applicable requirements of EN60950 following the provisions of the Low Voltage Directive 73/23/EEC. All GSC models are approved to UL1950, CSA22.2 No. 234 Level 3, IEC950, EN60950 and VDE 0805.
Shock and Vibration:	All models are designed to meet MIL-STD 810E, Method 514.4 Category I. This test consists of random vibration at 0.0159 ^ 2/Hz from 5 to 50 Hz declining linearly to 0.0015G ^ 2/Hz at 500 Hz in each axis for 1 Hr. All models designed to meet MIL-STD 810E, Method 516.4 Unpackaged units shall meet the bench handling requirements of Procedure VI for 4 drops per face. Packaged units shall meet the requirements of Procedure IV for transit/shipping drops.
Packaging:	Each unit individually wrapped in anti-static bag and packaged in a rugged cardboard shipping box. Unless otherwise noted, all parameters are measured at 115 VAC @ 25 °C and represent the nominal values. For limits at unusual operating conditions, consult factory.

			Load	Initial	OVP	Ripple
Model	Voltage	Current	Regulation	Setpoint	Setpoint	and Noise
				Tolerance		
GSC20-5	5.1 V	3.8A	0.75%	2.5%	6.2 +/- 0.6 V	1.4%
GSC20-12	12V	1.7 A	0.75%	2.5%	15.6 +/- 1.1 V	1%
GSC20-15	15V	1.4A	0.75%	2.5%	18.5 +/- 1.5	1%
GSC20-24	24V	0.85A	0.75%	2.5%	28 +/- 2.5	1%
GSC20-28	28V	0.7A	0.75%	2.5%	34 +/- 2.8	1%

GSC20 (WT. 0.25 LBS.)



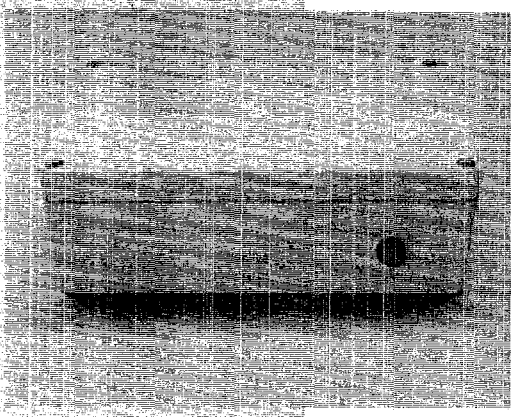
INPUT: J1 MOLEX P/N 43-8040
- PIN 1) AC LINE
- PIN 2) N/C
- PIN 3) N/C
- PIN 4) AC NEUTRAL

GND: 0.098 DIA. THRU HOLE

OUTPUT: J2 MOLEX P/N 22-43-8040
- PIN 1) OUTPUT #1
- PIN 2) OUTPUT #1
- PIN 3) COMMON
- PIN 4) COMMON

MATING CONNECTOR MOLEX P/N
HOUSING 50-37-5043
CONTACT 08-70-1040

NOTE: 3A MAXIMUM RECOMMENDED CURRENT PER CONNECTOR PIN



Attribute Measurement System Computational Block Subsystem

Design Documentation Manual Volume 2

1 Computational Block Overview

1.1 High-Level Overview of the Computational Block
1.2 Computational Block Threshold Parameters
1.3 Listing of All Major Computational Block Components and Functions
1.4 Block Diagram of Computational Block Subsystem Functions
1.5 Pictures of the Computational Block Subsystem
1.6 Explanation of Classified Data Strings Received with an Example Data Set of Each
1.7 Explanation of Threshold Calculations Required
1.8 Summary of Unclassified Output (Pass/Fail)

2 CPU Card

2.1 High-Level Description of CPU Card Operation
  2.1.1 Brief Overview Narrative
  2.1.2 Critical Specifications
2.2 Ampro, CoreModule 3SXi – 386SX Processor Card
  2.2.1 Specifications
  2.2.2 Photographs
  2.2.3 X-Ray
  2.2.4 Jumper Settings
  2.2.5 Parameter Settings Set by Software
  2.2.6 Schematics
  2.2.7 Parts List
  2.2.8 Memory Details
  2.2.9 Voltage and Current Requirements
  2.2.10 Supplementary Narrative Describing Functional Blocks and Implementation
  2.2.11 Test Point Information
  2.2.12 Technical Manual
2.3 Acer Laboratories, ALi M6117C – 386SX Embedded Microcontroller IC
  2.3.1 Specifications
  2.3.2 Functional Description
  2.3.3 Data Sheet
2.4 Atmel, AT27C080 – 1-Mbyte EPROM
  2.4.1 Photograph
  2.4.2 X-Ray
  2.4.3 Data Sheet
  2.4.4 Part Number Codes
2.5 Intel, 28F008SA – 1-Mbyte Flash Memory
  2.5.1 Data Sheet
  2.5.2 Part Number Codes

3 Video Card

4 Quad COM Port and Digital-I/O Card

5 Additional Hardware Support Documentation

5.1 Ampro Common Utilities (Card Specific Software)
5.2 Ampro Application Handbook
5.3 PC-104 Specifications
5.4 Parts List for the Overall Computational Block
5.5 Supporting Hardware
  5.5.1 Photographs
  5.5.2 X-Rays
  5.5.3 Specifications for the Corcom 6EC1 RFI Power-Line Filter
  5.5.4 Specifications for the Condor GSC20-5 Switching Power Supply

6 Commercial System Software

6.1 Datalight, ROM-DOS Version 6.22 – Operating System
  6.1.1 Description
  6.1.2 Software Development Kit
  6.1.3 List of Removed Extraneous Functions
  6.1.4 Narrative Regarding Use of All Retained Components
  6.1.5 List of ROM-DOS Setup Parameters
  6.1.6 Listing of CONFIG.SYS and AUTOEXEC.BAT Files
    6.1.6.1 CONFIG.SYS Listing
    6.1.6.2 AUTOEXEC.BAT Listing
  6.1.7 Memory Map Showing Configuration in All Memory ICs
  6.1.8 Interrupt Vector Locations and Values
  6.1.9 I/O Information
    6.1.9.1 IRQ Data
    6.1.9.2 DMA Data
    6.1.9.3 Ports Used
    6.1.9.4 Other I/O Parameters
  6.1.10 ROM-DOS User's Guide Manual
  6.1.11 ROM-DOS Developer's Guide Manual
  6.1.12 Commented Source-Code for ROM-DOS
  6.1.13 Third-Party ROM-DOS References
6.2 Phoenix, Award BIOS – System BIOS
  6.2.1 Narrative Description of Award ROM BIOS
  6.2.2 BIOS Function Information and Calling Protocols
  6.2.3 Ampro Information on Award ROM BIOS
  6.2.4 CMOS Setup Utility User's Guide for ALI M6117 Chipset
  6.2.5 Third-Party BIOS References

7 Computational Block Applications Software

8 PROM ICs

9 Internet References

6.1 Datalight, ROM-DOS Version 6.22 – Operating System

The ROM-DOS operating system software is an MS-DOS work-alike that is designed for use with embedded computer systems using PROMs rather than disk drives.

The commercial version of ROM-DOS comes as a Software Development Kit. This kit can be purchased from Datalight Inc. [18810 59th Avenue NE, Arlington, WA 98223 – phone 360-435-8086] for $495 plus $25 shipping.

6.1.1 Description of ROM-DOS Version 6.22

Datalight Inc. sells ROM-DOS version 6.22, which is a work-alike version of the DOS operating system that is functionally equivalent to other brands of DOS (e.g., Microsoft DOS version 6.22) and can readily run programs developed for other DOS versions. However, the ROM-DOS operating system can be loaded into read-only memory ROM and run entirely from within ROM.

Some of the vendor's claims for ROM-DOS are listed below with an indication of their value to an information barrier system follow.

- ROM-DOS is the most robust and full-featured DOS on the market today. It is important for the operating system to be robust and thus avoid software crashes behind the information barrier.
- ROM-DOS has a high level of compatibility at a relatively low cost. This allows use with analysis software developed for other DOS versions with minimal porting efforts.
- ROM-DOS has an unusually small memory footprint. Because ROM-DOS was written to be compact, it's nearly half the size of MS-DOS in ROM. This allows use of smaller PROM memories, which reduces the amount of hardware and software to be carefully inspected.
- ROM-DOS 6.22 is a work-alike of MS-DOS version 6.22, which is designed for embedded computing environments. Thus, ROM-DOS is particularly well suited for embedded applications similar to the information barrier application.
- All ROM-DOS hardware communication is done through device drivers making BIOS calls. The device drivers are provided in full source code so the interrupts and even the calls can be changed if necessary. The availability of source code is an advantage for information barrier applications.
- ROM-DOS 6 includes both a full Command Processor and a 4K, configurable, mini COMMAND processor for systems requiring only a basic command processor. The MiniCommand processor supports internal and external commands, limited batch file processing and includes full source code and instructions. The MiniCommand processor could allow reduction of extraneous functionality in an information barrier system.
- The HIMEM.SYS driver manages extended memory and the High Memory Area (HMA) in 286, 386 or 486 systems. It supports the eXtended Memory Specification (XMS) 2.0. In addition, Upper Memory Blocks (UMBs) are supported by EMM386. HIMEM serves a useful function in overwriting memory during the power up memory testing.
- The Configurable Memory Disk Driver allows any area of RAM or ROM to be used as a memory disk. This driver is designed for a variety of memory configurations, including paged memory, static battery-backed RAM, flash memory and 386/486 extended memory. The RAM disk can be used by an information barrier system to write intermediate results to volatile files between successive programs.
- ROM-DOS can take full advantage of all hardware including hard drives up to 8 gigabytes (4 logical drives at 2 GB maximum), CD-ROM drives, Flash memory and PCMCIA cards.
- ROM-DOS commented source code requires a non-disclosure agreement and is available for $10K.

Selected attributes of ROM-DOS, as obtained from the vendor's Internet site at:

http://www.datalight.com/rom-dos.htm.

WHY USE DOS?

With more than 200 million PC-compatible computers running DOS, there is a vast array of inexpensive tools, languages, and applications software, as well as an army of well-trained, capable programmers. As new, more powerful processors are introduced for desktop computers, their predecessors become affordable for alternative projects. Using DOS leads to shorter engineering cycles and lower costs.

WHY BUY ROM-DOS?

Datalight ROM-DOS 6.22, a work-alike of MS-DOS version 6.22, is designed for embedded and mobile computing environments. ROM-DOS gives you the features of MS-DOS, while adding benefits like an automated BUILD configuration utility to get you up and running quickly, together with the flexibility to meet your project's unique, non-PC needs. To this, add ROM-DOS' tools and accessories, such as remote serial disk access and optional Stacker data compression, all at a fraction of the cost of other compatible brands of DOS. You'll agree that ROM-DOS is a winner. In addition, at no additional charge, ROM-DOS is currently shipping with the Datalight Software Development Tool Kit that includes Borland tools 5.2.

WHAT IF MY SYSTEM HAS SPECIAL REQUIREMENTS?

ROM-DOS is designed to be more flexible than other operating systems. Whether your hardware is fully PC compatible or not, with ROM-DOS your embedded or mobile system will be. The only requirements for ROM-DOS are RAM, ROM and an 80x86 CPU. ROM-DOS can take full advantage of all hardware including hard drives up to 8 gigabytes (4 logical drives at 2 GB maximum), CD-ROM drives, Flash memory and PCMCIA cards.

Do you want ROM-DOS to run from ROM, but on speed critical systems, run from RAM? Tell the BUILD utility and ROM-DOS does it.

If your BIOS is not 100% PC compatible, it will not be a problem. All ROM-DOS hardware communication is done through device drivers making BIOS calls. The device drivers are provided in full source code so the interrupts and even the calls can be changed.

Are you booting on a diskless system, but need 2 Megabyte ROM disk, or four separate ROM disk drives? ROM-DOS has a built in ROM disk driver, which can be configured to handle these, and many other situations.

What if you want to have drive A: as a PCMCIA drive, drive B: as a ROM disk, drive C: as a RAM disk, D: as a floppy and drive E: as a hard drive? What if you want to reverse that order? ROM-DOS can configure the drives in any order.

WHAT IF I NEED TO SAVE SPACE?

If you are tight on space, you can leave out drivers that you won't be using. You can choose the 4K "mini" command processor or no command processor, if it isn't needed. Because ROM-DOS was written to be compact, it's nearly half the size of MS-DOS in ROM.

IS IT REALLY COMPATIBLE?

No matter how flexible, small, fast or feature-rich ROM-DOS becomes, it will remain compatible with the DOS standard. ROM-DOS was written independently by Datalight using internal testing and published material. Compatibility is the key goal for ROM-DOS. For that reason we offer a free bootable demo disk so that you can see for yourself that ROM-DOS will run your applications. We also offer this guarantee: if you should find a program that runs under MS-DOS but not under ROM-DOS, let us know and we will make sure that it runs under the next release of ROM-DOS. And if you wish, Datalight will refund 100% of the purchase price of the Developer's Kit upon it's return.

A COMPLETE ROM DEVELOPMENT SOLUTION

ROM-DOS 6.22 is much more than an inexpensive replacement for MS-DOS. It comes with a wide variety of development tools at no additional cost, including Borland Tools 5.2. The addition of these tools will speed you through your development cycle.

BUILD is an automated utility that configures ROM-DOS for ROM or disk. Many Datalight customers have commented on how quick and easy it is to build a ROM-based system with BUILD. As it walks you through a menu, you make the choices and BUILD does the rest. ROM-DOS can be running in ROM on your system in under 15 minutes.

Stacker Data Compression is available to ROM-DOS 6 Licensed OEMs from Datalight at special, low cost, embedded system rates. Award winning Stacker doubles the storage capacity of system hard disk, RAM ROM disks, Flash memory and floppy disks.

Remote Disk is a utility program that allows you to access a disk drive on a remote system via a serial cable and serial port. It can be used for exchanging files between hand-held and non-desktop systems and for data collection from diskless systems.

CONFIG.SYS options include boot-time menus to select system configuration commands tailored to individual user tasks. Also, developers may choose among CONFIG.SYS compatibility levels (DOS 3.31, 5.0 and 6.0) at the time of configuring, for saving space in memory restricted systems.

ROM-DOS 6 includes both a full Command Processor and a 4K, configurable, mini COMMAND processor for systems requiring only a basic command processor. MiniCommand supports internal and external commands, limited batch file processing and includes full source code and instructions.

The Command Processors included with ROM-DOS are optional. Unlike MS-DOS, you can configure your system to run an EXE or COM program directly and avoid the "overhead" of the command processor and security to embedded systems.

Advanced Power Management is supported by ROM-DOS for hand-held devices and other systems where prolonged battery life is desired. It requires BIOS Power Management support.

RXEs are ROMable EXE programs that can be run in place from a ROM disk, a useful alternative for embedded and mobile systems with minimal RAM.

The International Support you need is provided with ROM-DOS. Twenty-one countries are directly supported, as well as double byte characters for Japanese. Included are COUNTRY, DISPLAY.SYS and KEYB.COM. Please call to check on support for your language needs.

The HIMEM.SYS driver manages extended memory and the High Memory Area (HMA) in 286, 386 or systems. It supports the eXtended Memory Specification (XMS) 2.0. Plus Upper Memory Blocks (UMBs) are supported by EMM386.

The Configurable Memory Disk Driver allows any area of RAM or ROM to be used as a memory disk. This driver is designed for a variety of memory configurations, including paged memory, static battery-backed RAM, flash memory and 386/486 extended memory.

The ROM-DOS User's Manual is available to original equipment manufacturers (OEMs) on disk so that you can include all or part of it, or change it to match the format desired.

6.1.2 Software Development kit

The ROM-DOS 6.22 Software Developer's Kit includes:

- ROM-DOS operating system disks on bootable CD
- ROM-DOS Developer's Guide with complete, step-by-step instructions.
- ROM-DOS User's Guide. Also available as a .DOC file for inclusion in end-product documentation for licensed OEM's.
- A large set of DOS Utilities.
- Automated BUILD Configuration Utility for ROM or disk.
- HIMEM.SYS and EMM386 drivers included.
- Remote Disk serial link for accessing a hard drive on another system.
- RXE kit for running executable files directly from ROM or flash.
- Full Source Code for all device drivers.
- A Configurable Memory Disk Driver.
- International Support including KEYB and DISPLAY.
- Advanced Power Management to support BIOS APM.
- Variable CONFIG.SYS compatibility levels for developer space savings.
- Datalight OEM Agreement for distributing ROM-DOS.
- Includes, at no additional charge, Datalight Software Development Tool Kit (SDTK) with Borland Tools 5.2
- 90-Day 100% Money Back Guarantee

The directories of the two CD-ROM included in the Software Developer's Kit follow.

- ROM-DOS 6.22
- Datalight Software Development Toolkit

When these CD-ROMs are installed, files are placed in the \DL directory and four sub-directories. The files contained in these directories are also listed in a Norton Find File format.

- /ROMDOS – ROM-DOS operating system and utilities
- /RXE – kit for running executable files directly from ROM or flash
- /STACKER – Disk compression utility
- DEVTOOLS – Borland Tools 5.2

The first CD-ROM, which is titled ROM-DOS 6.22, includes the following files.

COMMAND	COM	34493	11-11-99	6:22a
DATA	TAG	118	11-17-99	2:44p
DATA1	CAB	2309245	11-17-99	2:44p
DATA1	HDR	20234	11-17-99	2:44p
LANG	DAT	4679	9-18-98	3:12p
LAYOUT	BIN	630	11-17-99	2:44p
OS	DAT	450	7-27-98	6:41p
RDOSDEV	PDF	388112	11-03-99	3:36p
RDOSUSER	PDF	567006	11-03-99	3:36p
READ	ME	33098	11-11-99	1:16p
ROM-DOS	SYS	54448	11-11-99	6:22a
SETUP	BMP	95598	7-13-99	7:52p
SETUP	EXE	60928	10-02-98	6:04p
SETUP	INI	115	11-17-99	2:44p
SETUP	INS	58794	11-17-99	2:43p
SETUP	LID	49	11-17-99	2:44p
SYS	COM	21482	11-11-99	6:22a
_INST16	EX_	289447	10-02-98	7:15p
_INST32I	EX_	297989	10-02-98	7:15p
_ISDEL	EXE	8192	10-02-98	6:05p
_SETUP	DLL	11264	9-29-98	5:44p
_SYS1	CAB	370397	11-17-99	2:43p
_SYS1	HDR	4569	11-17-99	2:43p
_USER1	CAB	22938	11-17-99	2:43p
_USER1	HDR	5008	11-17-99	2:43p

25 Files 4659283 Char.

The second CD-ROM, which is titled Datalight Software Development Toolkit, includes the following files.

DATA	TAG	135	2-02-99	5:00p
DATA1	CAB	9915731	2-02-99	5:01p
DATA1	HDR	53959	2-02-99	5:01p
LANG	DAT	4679	9-18-98	3:12p
LAYOUT	BIN	609	2-02-99	5:01p
OS	DAT	450	7-27-98	6:41p
SETUP	BMP	62818	2-02-99	4:27p
SETUP	EXE	71680	10-02-98	7:04p
SETUP	INI	123	2-02-99	5:00p
SETUP	INS	58696	2-02-99	5:00p
SETUP	LID	49	2-02-99	5:00p
_INST32I	EX_	297989	10-02-98	7:15p
_ISDEL	EXE	27648	1-12-99	11:44a
_SETUP	DLL	34816	9-29-98	5:34p
_SYS1	CAB	180837	2-02-99	5:00p
_SYS1	HDR	3905	2-02-99	5:00p
_USER1	CAB	2483	2-02-99	5:00p
_USER1	HDR	4293	2-02-99	5:00p

18 Files 10720900 Char.

The ROM-DOS Software Development Kit includes two CD-roms. When installed on Disk E:, The first CD-rom titled ROM-DOS 6.22 produced directories \DL\ROMDOS, \DL\RXE, and \DL\STACKER. The second CD-rom titled Datalight Software Development Toolkit produced the \DL\DEVTOOLS directory.

The following directories show the included software and documentation files.

FF-File Find, Advanced Edition 4.50, (C) Copr 1987-88, Peter Norton

E:\

DL	<DIR>	1:40 pm	Sat Mar 10 00

E:\DL

.	<DIR>	1:40 pm	Sat Mar 10 00
..	<DIR>	1:40 pm	Sat Mar 10 00
un_rd.isu	50,884 bytes	1:42 pm	Sat Mar 10 00
ROMDOS	<DIR>	1:41 pm	Sat Mar 10 00
STACKER	<DIR>	1:41 pm	Sat Mar 10 00
RXE	<DIR>	1:41 pm	Sat Mar 10 00
DEVTOOLS	<DIR>	1:50 pm	Sat Mar 10 00

E:\DL\ROMDOS

.	<DIR>	1:41 pm	Sat Mar 10 00
..	<DIR>	1:41 pm	Sat Mar 10 00
command.hlp	11,122 bytes	6:22 am	Thu Nov 11 99
command.com	34,493 bytes	6:22 am	Thu Nov 11 99
config.sys	220 bytes	6:22 am	Thu Nov 11 99
autoexec.bat	125 bytes	6:22 am	Thu Nov 11 99
rom-dos.sys	54,448 bytes	6:22 am	Thu Nov 11 99
longnam6.lib	25,088 bytes	6:22 am	Thu Nov 11 99
sys.com	21,482 bytes	6:22 am	Thu Nov 11 99
build.exe	48,350 bytes	6:22 am	Thu Nov 11 99
compress.exe	14,180 bytes	6:22 am	Thu Nov 11 99
loc.exe	73,486 bytes	6:22 am	Thu Nov 11 99
dosinit6.lib	11,264 bytes	6:22 am	Thu Nov 11 99
doscfg6.lib	12,800 bytes	6:22 am	Thu Nov 11 99
fdhd6_1.lib	15,360 bytes	6:22 am	Thu Nov 11 99
fdhd6_2.lib	16,896 bytes	6:22 am	Thu Nov 11 99
fdhd6_3.lib	16,384 bytes	6:22 am	Thu Nov 11 99
romdos6.lib	146,432 bytes	6:22 am	Thu Nov 11 99
read.me	33,098 bytes	1:16 pm	Thu Nov 11 99
rdosdev.pdf	388,112 bytes	3:36 pm	Wed Nov 3 99
rdosuser.pdf	567,006 bytes	3:36 pm	Wed Nov 3 99
devhdr.inc	3,474 bytes	6:22 am	Thu Nov 11 99
dos6asm.obj	7,120 bytes	6:22 am	Thu Nov 11 99
dos6cfg0.obj	544 bytes	6:22 am	Thu Nov 11 99
dos6cfg3.obj	3,011 bytes	6:22 am	Thu Nov 11 99
dos6cfg5.obj	3,859 bytes	6:22 am	Thu Nov 11 99
dos6cfg7.obj	9,296 bytes	6:22 am	Thu Nov 11 99
dosignon.c	1,778 bytes	6:22 am	Thu Nov 11 99
dosver.inc	869 bytes	6:22 am	Thu Nov 11 99
dosver.h	980 bytes	6:22 am	Thu Nov 11 99
macros.inc	2,746 bytes	6:22 am	Thu Nov 11 99

```
        sysgen.asm      28,068 bytes   6:22 am  Thu Nov 11 99
        DEVSRC            <DIR>        1:41 pm  Sat Mar 10 00
        MEMDISK           <DIR>        1:41 pm  Sat Mar 10 00
        MINICMD           <DIR>        1:41 pm  Sat Mar 10 00
        UTILS             <DIR>        1:41 pm  Sat Mar 10 00
        SUPRBOOT          <DIR>        1:41 pm  Sat Mar 10 00

E:\DL\ROMDOS\DEVSRC
        .                 <DIR>        1:41 pm  Sat Mar 10 00
        ..                <DIR>        1:41 pm  Sat Mar 10 00
        devfdhd.inc      7,428 bytes   6:22 am  Thu Nov 11 99
        devclk.asm      10,511 bytes   6:22 am  Thu Nov 11 99
        devcom.asm       8,165 bytes   6:22 am  Thu Nov 11 99
        devcon.asm       6,209 bytes   6:22 am  Thu Nov 11 99
        devfdhd.asm     86,703 bytes   6:22 am  Thu Nov 11 99
        devfdhdi.asm    83,886 bytes   6:22 am  Thu Nov 11 99
        devnul.asm       2,932 bytes   6:22 am  Thu Nov 11 99
        devprn.asm       4,974 bytes   6:22 am  Thu Nov 11 99
        devrom.asm       7,495 bytes   6:22 am  Thu Nov 11 99
        makefile         2,419 bytes   6:22 am  Thu Nov 11 99
        dostruct.inc    33,136 bytes   6:22 am  Thu Nov 11 99

E:\DL\ROMDOS\MEMDISK
        .                 <DIR>        1:41 pm  Sat Mar 10 00
        ..                <DIR>        1:41 pm  Sat Mar 10 00
        makefile        11,875 bytes   6:22 am  Thu Nov 11 99
        memdisk.asm     50,500 bytes   6:22 am  Thu Nov 11 99
        ldrbext.asm      5,411 bytes   6:22 am  Thu Nov 11 99
        vdisk.c         18,338 bytes   6:22 am  Thu Nov 11 99
        raminit.c        4,579 bytes   6:22 am  Thu Nov 11 99
        pagedldr.c       8,655 bytes   6:22 am  Thu Nov 11 99
        pagedldr.loc       120 bytes   6:22 am  Thu Nov 11 99
        memfixed.c       8,805 bytes   6:22 am  Thu Nov 11 99
        int15pag.c       4,043 bytes   6:22 am  Thu Nov 11 99
        dyndrvr.c        4,441 bytes   6:22 am  Thu Nov 11 99
        memrom.c         6,924 bytes   6:22 am  Thu Nov 11 99
        memext.c         8,112 bytes   6:22 am  Thu Nov 11 99
        sc400pag.c       4,282 bytes   6:22 am  Thu Nov 11 99
        mempaged.c       8,009 bytes   6:22 am  Thu Nov 11 99
        memdisk.inc      3,470 bytes   6:22 am  Thu Nov 11 99
        compiler.h       1,437 bytes   6:22 am  Thu Nov 11 99
        memdisk.h        7,957 bytes   6:22 am  Thu Nov 11 99
        vdisk.h          2,105 bytes   6:22 am  Thu Nov 11 99
        dyndrvr.h        1,181 bytes   6:22 am  Thu Nov 11 99
        memutil.lib     23,040 bytes   6:22 am  Thu Nov 11 99

E:\DL\ROMDOS\MINICMD
        .                 <DIR>        1:41 pm  Sat Mar 10 00
        ..                <DIR>        1:41 pm  Sat Mar 10 00
        makefile         1,523 bytes   6:22 am  Thu Nov 11 99
        cmd.asm          9,997 bytes   6:22 am  Thu Nov 11 99
        cmdcmds.c        8,893 bytes   6:22 am  Thu Nov 11 99
        cmdmain.c       14,636 bytes   6:22 am  Thu Nov 11 99
        cmdprot.h        2,711 bytes   6:22 am  Thu Nov 11 99
```

```
        minicmd.h        4,203 bytes   6:22 am  Thu Nov 11 99
        minicmd.com      4,674 bytes   6:22 am  Thu Nov 11 99

E:\DL\ROMDOS\UTILS
        .                  <DIR>       1:41 pm  Sat Mar 10 00
        ..                 <DIR>       1:41 pm  Sat Mar 10 00
        ansi.sys         3,361 bytes   6:22 am  Thu Nov 11 99
        attrib.com      10,994 bytes   6:22 am  Thu Nov 11 99
        deltree.exe     15,022 bytes   6:22 am  Thu Nov 11 99
        exe2bin.com      9,858 bytes   6:22 am  Thu Nov 11 99
        fdisk.com       36,248 bytes   6:22 am  Thu Nov 11 99
        hexcat.com      13,116 bytes   6:22 am  Thu Nov 11 99
        mode.com         9,936 bytes   6:22 am  Thu Nov 11 99
        more.com         9,416 bytes   6:22 am  Thu Nov 11 99
        print.com       12,934 bytes   6:22 am  Thu Nov 11 99
        find.com        11,218 bytes   6:22 am  Thu Nov 11 99
        label.com        9,088 bytes   6:22 am  Thu Nov 11 99
        sort.com        11,234 bytes   6:22 am  Thu Nov 11 99
        tree.com        11,522 bytes   6:22 am  Thu Nov 11 99
        sys.com         21,482 bytes   6:22 am  Thu Nov 11 99
        choice.com       2,637 bytes   6:22 am  Thu Nov 11 99
        xcopy.com       26,286 bytes   6:22 am  Thu Nov 11 99
        longdir.exe      7,976 bytes   6:22 am  Thu Nov 11 99
        mem.exe          7,761 bytes   6:22 am  Thu Nov 11 99
        backup.exe      22,298 bytes   6:22 am  Thu Nov 11 99
        restore.exe     21,454 bytes   6:22 am  Thu Nov 11 99
        move.exe        17,222 bytes   6:22 am  Thu Nov 11 99
        stackdev.sys     1,504 bytes   6:22 am  Thu Nov 11 99
        rz.bat              77 bytes   6:22 am  Thu Nov 11 99
        proto.exe       10,464 bytes   6:22 am  Thu Nov 11 99
        umblink.exe      5,197 bytes   6:22 am  Thu Nov 11 99
        rsz.exe         32,564 bytes   6:22 am  Thu Nov 11 99
        sz.bat             104 bytes   6:22 am  Thu Nov 11 99
        himem.sys        5,832 bytes   6:22 am  Thu Nov 11 99
        power.exe        6,350 bytes   6:22 am  Thu Nov 11 99
        format.com      37,762 bytes   6:22 am  Wed Nov 17 99
        vdisk.sys        7,958 bytes   6:22 am  Thu Nov 11 99
        emm386.exe      20,845 bytes   6:22 am  Thu Nov 11 99
        version.sys        256 bytes   6:22 am  Thu Nov 11 99
        nedman.txt      24,080 bytes   6:22 am  Thu Nov 11 99
        remdisk.exe     12,523 bytes   6:22 am  Thu Nov 11 99
        remserv.exe      9,875 bytes   6:22 am  Thu Nov 11 99
        remquit.com        106 bytes   6:22 am  Thu Nov 11 99
        transfer.exe    15,108 bytes   6:22 am  Thu Nov 11 99
        ned.exe         88,866 bytes   6:22 am  Thu Nov 11 99
        nedremot.cfg       791 bytes   6:22 am  Thu Nov 11 99
        nedremot.com       176 bytes   6:22 am  Thu Nov 11 99
        ata.sys          9,966 bytes   6:22 am  Thu Nov 11 99
        ata.txt            859 bytes   6:22 am  Thu Nov 11 99
        longname.txt     1,762 bytes   6:22 am  Thu Nov 11 99
        country.sys     10,597 bytes   6:22 am  Thu Nov 11 99
        chkdsk.com      18,992 bytes   6:22 am  Thu Nov 11 99
        mscdex.exe      25,307 bytes   6:22 am  Thu Nov 11 99
        comm.exe        46,824 bytes   6:22 am  Thu Nov 11 99
```

```
        disk2img.exe    10,044 bytes   6:22 am  Thu Nov 11 99
        dump.exe        10,774 bytes   6:22 am  Thu Nov 11 99
        romdisk.exe     31,380 bytes   6:22 am  Thu Nov 11 99
        subst.exe        8,910 bytes   6:22 am  Thu Nov 11 99
        keyb.com         6,417 bytes   6:22 am  Thu Nov 11 99
        display.sys      5,899 bytes   6:22 am  Thu Nov 11 99
        keyboard.sys    14,753 bytes   6:22 am  Thu Nov 11 99
        keybrd2.sys      2,056 bytes   6:22 am  Thu Nov 11 99
        wx2swap.sys        533 bytes   6:22 am  Thu Nov 11 99
        ega.cpi         34,080 bytes   6:22 am  Thu Nov 11 99
        ega3.cpi        14,624 bytes   6:22 am  Thu Nov 11 99
        share.exe        4,367 bytes   6:22 am  Thu Nov 11 99

E:\DL\ROMDOS\SUPRBOOT
        .                  <DIR>       1:41 pm  Sat Mar 10 00
        ..                 <DIR>       1:41 pm  Sat Mar 10 00
        makefile         6,522 bytes   6:22 am  Thu Nov 11 99
        first.lib       22,528 bytes   6:22 am  Thu Nov 11 99
        firstb.lib      22,528 bytes   6:22 am  Thu Nov 11 99
        middle.lib      23,040 bytes   6:22 am  Thu Nov 11 99
        middleb.lib     23,040 bytes   6:22 am  Thu Nov 11 99
        last.lib        23,040 bytes   6:22 am  Thu Nov 11 99
        lastb.lib       23,040 bytes   6:22 am  Thu Nov 11 99

E:\DL\STACKER
        .                  <DIR>       1:41 pm  Sat Mar 10 00
        ..                 <DIR>       1:41 pm  Sat Mar 10 00
        screate.sys     15,510 bytes   6:22 am  Thu Jul 22 99
        create.com       1,042 bytes   6:22 am  Thu Jul 22 99
        screxec.exe     89,380 bytes   6:22 am  Thu Jul 22 99
        screxec2.exe    64,896 bytes   6:22 am  Thu Jul 22 99
        stacker.com     43,727 bytes   6:22 am  Thu Jul 22 99
        stacker.exe     30,251 bytes   6:22 am  Thu Jul 22 99

E:\DL\RXE
        .                  <DIR>       1:41 pm  Sat Mar 10 00
        ..                 <DIR>       1:41 pm  Sat Mar 10 00
        readme.txt       8,294 bytes   6:22 am  Thu Jul 22 99
        absdisk.asm     10,525 bytes   6:22 am  Thu Jul 22 99
        rxe_cvt.exe     24,900 bytes   6:22 am  Thu Jul 22 99
        rxecheck.exe    21,430 bytes   6:22 am  Thu Jul 22 99
        romdisk.exe     31,352 bytes   6:22 am  Thu Jul 22 99
        absdisk.sys      1,127 bytes   6:22 am  Thu Jul 22 99
        rxe_init.asm     9,782 bytes   6:22 am  Thu Jul 22 99
        reset.exe        7,312 bytes   6:22 am  Thu Jul 22 99
        B-START            <DIR>       1:41 pm  Sat Mar 10 00
        rxe_init.exe     1,109 bytes   6:22 am  Thu Jul 22 99
        EXAMPLES           <DIR>       1:42 pm  Sat Mar 10 00
        LIBSRC             <DIR>       1:42 pm  Sat Mar 10 00
        MS-START           <DIR>       1:42 pm  Sat Mar 10 00

E:\DL\RXE\B-START
        .                  <DIR>       1:41 pm  Sat Mar 10 00
        ..                 <DIR>       1:41 pm  Sat Mar 10 00
```

```
        BC2                 <DIR>          1:41 pm  Sat Mar 10 00
        BC3                 <DIR>          1:42 pm  Sat Mar 10 00
        BC4                 <DIR>          1:42 pm  Sat Mar 10 00
        BC45                <DIR>          1:42 pm  Sat Mar 10 00
        BC5                 <DIR>          1:42 pm  Sat Mar 10 00

E:\DL\RXE\B-START\BC2
        .                   <DIR>          1:41 pm  Sat Mar 10 00
        ..                  <DIR>          1:41 pm  Sat Mar 10 00
        c0s.obj         2,298 bytes    6:22 am  Thu Jul 22 99
        c0c.obj         2,167 bytes    6:22 am  Thu Jul 22 99
        c0m.obj         2,303 bytes    6:22 am  Thu Jul 22 99
        c0l.obj         2,172 bytes    6:22 am  Thu Jul 22 99
        c0fs.obj        2,298 bytes    6:22 am  Thu Jul 22 99
        c0fc.obj        2,192 bytes    6:22 am  Thu Jul 22 99
        c0fm.obj        2,303 bytes    6:22 am  Thu Jul 22 99
        c0fl.obj        2,197 bytes    6:22 am  Thu Jul 22 99

E:\DL\RXE\B-START\BC3
        .                   <DIR>          1:42 pm  Sat Mar 10 00
        ..                  <DIR>          1:42 pm  Sat Mar 10 00
        c0s.obj         2,451 bytes    6:22 am  Thu Jul 22 99
        c0c.obj         2,334 bytes    6:22 am  Thu Jul 22 99
        c0m.obj         2,454 bytes    6:22 am  Thu Jul 22 99
        c0l.obj         2,337 bytes    6:22 am  Thu Jul 22 99
        c0fs.obj        2,451 bytes    6:22 am  Thu Jul 22 99
        c0fc.obj        2,359 bytes    6:22 am  Thu Jul 22 99
        c0fm.obj        2,454 bytes    6:22 am  Thu Jul 22 99
        c0fl.obj        2,362 bytes    6:22 am  Thu Jul 22 99

E:\DL\RXE\B-START\BC4
        .                   <DIR>          1:42 pm  Sat Mar 10 00
        ..                  <DIR>          1:42 pm  Sat Mar 10 00
        c0s.obj         2,492 bytes    6:22 am  Thu Jul 22 99
        c0c.obj         2,372 bytes    6:22 am  Thu Jul 22 99
        c0m.obj         2,500 bytes    6:22 am  Thu Jul 22 99
        c0l.obj         2,379 bytes    6:22 am  Thu Jul 22 99
        c0fs.obj        2,492 bytes    6:22 am  Thu Jul 22 99
        c0fc.obj        2,397 bytes    6:22 am  Thu Jul 22 99
        c0fm.obj        2,500 bytes    6:22 am  Thu Jul 22 99
        c0fl.obj        2,404 bytes    6:22 am  Thu Jul 22 99

E:\DL\RXE\B-START\BC45
        .                   <DIR>          1:42 pm  Sat Mar 10 00
        ..                  <DIR>          1:42 pm  Sat Mar 10 00
        c0s.obj         2,509 bytes    6:22 am  Thu Jul 22 99
        c0c.obj         2,389 bytes    6:22 am  Thu Jul 22 99
        c0m.obj         2,517 bytes    6:22 am  Thu Jul 22 99
        c0l.obj         2,396 bytes    6:22 am  Thu Jul 22 99
        c0fs.obj        2,509 bytes    6:22 am  Thu Jul 22 99
        c0fc.obj        2,414 bytes    6:22 am  Thu Jul 22 99
        c0fm.obj        2,517 bytes    6:22 am  Thu Jul 22 99
        c0fl.obj        2,421 bytes    6:22 am  Thu Jul 22 99
```

```
E:\DL\RXE\B-START\BC5
        .                   <DIR>       1:42 pm  Sat Mar 10 00
        ..                  <DIR>       1:42 pm  Sat Mar 10 00
        c0s.obj         2,509 bytes     6:22 am  Thu Jul 22 99
        c0c.obj         2,389 bytes     6:22 am  Thu Jul 22 99
        c0m.obj         2,517 bytes     6:22 am  Thu Jul 22 99
        c0l.obj         2,396 bytes     6:22 am  Thu Jul 22 99
        c0fs.obj        2,509 bytes     6:22 am  Thu Jul 22 99
        c0fc.obj        2,414 bytes     6:22 am  Thu Jul 22 99
        c0fm.obj        2,517 bytes     6:22 am  Thu Jul 22 99
        c0fl.obj        2,421 bytes     6:22 am  Thu Jul 22 99

E:\DL\RXE\EXAMPLES
        .                   <DIR>       1:42 pm  Sat Mar 10 00
        ..                  <DIR>       1:42 pm  Sat Mar 10 00
        cpp.cpp         5,140 bytes     6:22 am  Thu Jul 22 99
        basic.c         1,501 bytes     6:22 am  Thu Jul 22 99
        floatpt.c         862 bytes     6:22 am  Thu Jul 22 99
        do_msc8.bat     2,343 bytes     6:22 am  Thu Jul 22 99
        compiler.h        623 bytes     6:22 am  Thu Jul 22 99
        do_msc6.bat     1,917 bytes     6:22 am  Thu Jul 22 99
        check.bat         118 bytes     6:22 am  Thu Jul 22 99
        convert.bat        51 bytes     6:22 am  Thu Jul 22 99
        showall.bat     1,294 bytes     6:22 am  Thu Jul 22 99
        makefile        2,536 bytes     6:22 am  Thu Jul 22 99

E:\DL\RXE\LIBSRC
        .                   <DIR>       1:42 pm  Sat Mar 10 00
        ..                  <DIR>       1:42 pm  Sat Mar 10 00
        setargv.c       1,855 bytes     6:22 am  Thu Jul 22 99
        setargx.asm       300 bytes     6:22 am  Thu Jul 22 99
        borland.bat       757 bytes     6:22 am  Thu Jul 22 99
        makeall.bat     1,289 bytes     6:22 am  Thu Jul 22 99
        malloc.c        8,170 bytes     6:22 am  Thu Jul 22 99
        microsof.bat      755 bytes     6:22 am  Thu Jul 22 99
        compiler.h        623 bytes     6:22 am  Thu Jul 22 99
        ms.bat            314 bytes     6:22 am  Thu Jul 22 99
        respond.bat       144 bytes     6:22 am  Thu Jul 22 99
        makefile        1,104 bytes     6:22 am  Thu Jul 22 99
        dlspawn.asm     8,024 bytes     6:22 am  Thu Jul 22 99
        tstspawn.c      2,526 bytes     6:22 am  Thu Jul 22 99
        datalght.inc   42,582 bytes     6:22 am  Thu Jul 22 99
        tasm.inc        1,746 bytes     6:22 am  Thu Jul 22 99
        masm.inc       13,156 bytes     6:22 am  Thu Jul 22 99
        dlmacros.inc   33,273 bytes     6:22 am  Thu Jul 22 99

E:\DL\RXE\MS-START
        .                   <DIR>       1:42 pm  Sat Mar 10 00
        ..                  <DIR>       1:42 pm  Sat Mar 10 00
        MS-C600             <DIR>       1:42 pm  Sat Mar 10 00
        MSVC                <DIR>       1:42 pm  Sat Mar 10 00

E:\DL\RXE\MS-START\MS-C600
        .                   <DIR>       1:42 pm  Sat Mar 10 00
```

```
        ..                  <DIR>       1:42 pm  Sat Mar 10 00
        C                   <DIR>       1:42 pm  Sat Mar 10 00
        L                   <DIR>       1:42 pm  Sat Mar 10 00
        M                   <DIR>       1:42 pm  Sat Mar 10 00
        S                   <DIR>       1:42 pm  Sat Mar 10 00

E:\DL\RXE\MS-START\MS-C600\C
        .                   <DIR>       1:42 pm  Sat Mar 10 00
        ..                  <DIR>       1:42 pm  Sat Mar 10 00
        DOS                 <DIR>       1:42 pm  Sat Mar 10 00

E:\DL\RXE\MS-START\MS-C600\C\DOS
        .                   <DIR>       1:42 pm  Sat Mar 10 00
        ..                  <DIR>       1:42 pm  Sat Mar 10 00
        crt0.obj        1,185 bytes     6:22 am  Thu Jul 22 99

E:\DL\RXE\MS-START\MS-C600\L
        .                   <DIR>       1:42 pm  Sat Mar 10 00
        ..                  <DIR>       1:42 pm  Sat Mar 10 00
        DOS                 <DIR>       1:42 pm  Sat Mar 10 00

E:\DL\RXE\MS-START\MS-C600\L\DOS
        .                   <DIR>       1:42 pm  Sat Mar 10 00
        ..                  <DIR>       1:42 pm  Sat Mar 10 00
        crt0.obj        1,199 bytes     6:22 am  Thu Jul 22 99

E:\DL\RXE\MS-START\MS-C600\M
        .                   <DIR>       1:42 pm  Sat Mar 10 00
        ..                  <DIR>       1:42 pm  Sat Mar 10 00
        DOS                 <DIR>       1:42 pm  Sat Mar 10 00

E:\DL\RXE\MS-START\MS-C600\M\DOS
        .                   <DIR>       1:42 pm  Sat Mar 10 00
        ..                  <DIR>       1:42 pm  Sat Mar 10 00
        crt0.obj        1,177 bytes     6:22 am  Thu Jul 22 99

E:\DL\RXE\MS-START\MS-C600\S
        .                   <DIR>       1:42 pm  Sat Mar 10 00
        ..                  <DIR>       1:42 pm  Sat Mar 10 00
        DOS                 <DIR>       1:42 pm  Sat Mar 10 00

E:\DL\RXE\MS-START\MS-C600\S\DOS
        .                   <DIR>       1:42 pm  Sat Mar 10 00
        ..                  <DIR>       1:42 pm  Sat Mar 10 00
        crt0.obj        1,163 bytes     6:22 am  Thu Jul 22 99

E:\DL\RXE\MS-START\MSVC
        .                   <DIR>       1:42 pm  Sat Mar 10 00
        ..                  <DIR>       1:42 pm  Sat Mar 10 00
        C                   <DIR>       1:42 pm  Sat Mar 10 00
        L                   <DIR>       1:42 pm  Sat Mar 10 00
        M                   <DIR>       1:42 pm  Sat Mar 10 00
        S                   <DIR>       1:42 pm  Sat Mar 10 00
```

```
E:\DL\RXE\MS-START\MSVC\C
        .                   <DIR>        1:42 pm  Sat Mar 10 00
        ..                  <DIR>        1:42 pm  Sat Mar 10 00
        DOS                 <DIR>        1:42 pm  Sat Mar 10 00

E:\DL\RXE\MS-START\MSVC\C\DOS
        .                   <DIR>        1:42 pm  Sat Mar 10 00
        ..                  <DIR>        1:42 pm  Sat Mar 10 00
        crt0.obj          1,230 bytes    6:22 am  Thu Jul 22 99

E:\DL\RXE\MS-START\MSVC\L
        .                   <DIR>        1:42 pm  Sat Mar 10 00
        ..                  <DIR>        1:42 pm  Sat Mar 10 00
        DOS                 <DIR>        1:42 pm  Sat Mar 10 00

E:\DL\RXE\MS-START\MSVC\L\DOS
        .                   <DIR>        1:42 pm  Sat Mar 10 00
        ..                  <DIR>        1:42 pm  Sat Mar 10 00
        crt0.obj          1,244 bytes    6:22 am  Thu Jul 22 99

E:\DL\RXE\MS-START\MSVC\M
        .                   <DIR>        1:42 pm  Sat Mar 10 00
        ..                  <DIR>        1:42 pm  Sat Mar 10 00
        DOS                 <DIR>        1:42 pm  Sat Mar 10 00

E:\DL\RXE\MS-START\MSVC\M\DOS
        .                   <DIR>        1:42 pm  Sat Mar 10 00
        ..                  <DIR>        1:42 pm  Sat Mar 10 00
        crt0.obj          1,222 bytes    6:22 am  Thu Jul 22 99

E:\DL\RXE\MS-START\MSVC\S
        .                   <DIR>        1:42 pm  Sat Mar 10 00
        ..                  <DIR>        1:42 pm  Sat Mar 10 00
        DOS                 <DIR>        1:42 pm  Sat Mar 10 00

E:\DL\RXE\MS-START\MSVC\S\DOS
        .                   <DIR>        1:42 pm  Sat Mar 10 00
        ..                  <DIR>        1:42 pm  Sat Mar 10 00
        crt0.obj          1,208 bytes    6:22 am  Thu Jul 22 99

E:\DL\DEVTOOLS
        .                   <DIR>        1:50 pm  Sat Mar 10 00
        ..                  <DIR>        1:50 pm  Sat Mar 10 00
        uninst.isu      154,115 bytes    1:54 pm  Sat Mar 10 00
        LIB                 <DIR>        1:50 pm  Sat Mar 10 00
        HELP                <DIR>        1:51 pm  Sat Mar 10 00
        INCLUDE             <DIR>        1:51 pm  Sat Mar 10 00
        BIN                 <DIR>        1:53 pm  Sat Mar 10 00

E:\DL\DEVTOOLS\LIB
        .                   <DIR>        1:50 pm  Sat Mar 10 00
        ..                  <DIR>        1:50 pm  Sat Mar 10 00
        ws2_32.lib        7,680 bytes    5:02 am  Tue Mar 25 97
        bwcc.lib          2,048 bytes    5:02 am  Tue Mar 25 97
```

```
bwcc32.lib        2,048 bytes   5:02 am  Tue Mar 25 97
c0c.obj           2,416 bytes   5:02 am  Tue Mar 25 97
c0d32.obj         1,983 bytes   5:02 am  Tue Mar 25 97
c0d32dyn.obj      1,989 bytes   5:02 am  Tue Mar 25 97
c0d32x.obj        1,892 bytes   5:02 am  Tue Mar 25 97
c0dc.obj          1,749 bytes   5:02 am  Tue Mar 25 97
c0dl.obj          1,942 bytes   5:02 am  Tue Mar 25 97
c0dm.obj          1,776 bytes   5:02 am  Tue Mar 25 97
c0ds.obj          1,775 bytes   5:02 am  Tue Mar 25 97
c0fc.obj          2,441 bytes   5:02 am  Tue Mar 25 97
c0fh.obj          2,314 bytes   5:02 am  Tue Mar 25 97
c0x32w.obj        1,741 bytes   5:02 am  Tue Mar 25 97
cc.lib          450,560 bytes   5:02 am  Tue Mar 25 97
ch.lib          454,656 bytes   5:02 am  Tue Mar 25 97
cl.lib          460,288 bytes   5:02 am  Tue Mar 25 97
cm.lib          443,392 bytes   5:02 am  Tue Mar 25 97
crtldll.lib     237,568 bytes   5:02 am  Tue Mar 25 97
cs.lib          434,176 bytes   5:02 am  Tue Mar 25 97
ct.lib          290,816 bytes   5:02 am  Tue Mar 25 97
cw32.lib        713,728 bytes   5:02 am  Tue Mar 25 97
cw32i.lib       179,200 bytes   5:02 am  Tue Mar 25 97
cw32mt.lib      748,032 bytes   5:02 am  Tue Mar 25 97
cw32mti.lib     183,808 bytes   5:02 am  Tue Mar 25 97
cwc.lib         441,856 bytes   5:02 am  Tue Mar 25 97
cwl.lib         472,064 bytes   5:02 am  Tue Mar 25 97
cwm.lib         434,688 bytes   5:02 am  Tue Mar 25 97
cws.lib         420,864 bytes   5:02 am  Tue Mar 25 97
mswsock.lib       2,048 bytes   5:02 am  Tue Mar 25 97
noeh32.lib        2,560 bytes   5:02 am  Tue Mar 25 97
noehc.lib         1,536 bytes   5:02 am  Tue Mar 25 97
noehh.lib         1,536 bytes   5:02 am  Tue Mar 25 97
noehl.lib         1,536 bytes   5:02 am  Tue Mar 25 97
noehm.lib         1,536 bytes   5:02 am  Tue Mar 25 97
noehs.lib         1,536 bytes   5:02 am  Tue Mar 25 97
noehwc.lib        1,536 bytes   5:02 am  Tue Mar 25 97
noehwl.lib        1,536 bytes   5:02 am  Tue Mar 25 97
noehwm.lib        1,536 bytes   5:02 am  Tue Mar 25 97
noehws.lib        1,536 bytes   5:02 am  Tue Mar 25 97
obsolete.lib      1,024 bytes   5:02 am  Tue Mar 25 97
ole2w16.lib      30,720 bytes   5:02 am  Tue Mar 25 97
ole2w32.lib      40,448 bytes   5:02 am  Tue Mar 25 97
overlay.lib      16,384 bytes   5:02 am  Tue Mar 25 97
rpcextra.lib      2,048 bytes   5:02 am  Tue Mar 25 97
th32.lib          1,024 bytes   5:02 am  Tue Mar 25 97
w32sut16.lib      1,024 bytes   5:02 am  Tue Mar 25 97
w32sut32.lib      1,024 bytes   5:02 am  Tue Mar 25 97
c0fl.obj          2,448 bytes   5:02 am  Tue Mar 25 97
c0fm.obj          2,542 bytes   5:02 am  Tue Mar 25 97
c0fs.obj          2,534 bytes   5:02 am  Tue Mar 25 97
c0ft.obj          2,221 bytes   5:02 am  Tue Mar 25 97
c0h.obj           2,314 bytes   5:02 am  Tue Mar 25 97
c0l.obj           2,423 bytes   5:02 am  Tue Mar 25 97
c0m.obj           2,542 bytes   5:02 am  Tue Mar 25 97
c0s.obj           2,534 bytes   5:02 am  Tue Mar 25 97
```

```
        c0t.obj          2,221 bytes   5:02 am  Tue Mar 25 97
        c0w32.obj        1,739 bytes   5:02 am  Tue Mar 25 97
        c0w32w.obj       1,745 bytes   5:02 am  Tue Mar 25 97
        c0wc.obj         2,037 bytes   5:02 am  Tue Mar 25 97
        c0wl.obj         2,184 bytes   5:02 am  Tue Mar 25 97
        c0wm.obj         2,068 bytes   5:02 am  Tue Mar 25 97
        c0ws.obj         2,063 bytes   5:02 am  Tue Mar 25 97
        c0x32.obj        1,716 bytes   5:02 am  Tue Mar 25 97
        default.def        249 bytes   5:02 am  Tue Mar 25 97
        emu.lib         16,896 bytes   5:02 am  Tue Mar 25 97
        fp87.lib         4,608 bytes   5:02 am  Tue Mar 25 97
        import.lib     114,176 bytes   5:02 am  Tue Mar 25 97
        import32.lib   468,480 bytes   5:02 am  Tue Mar 25 97
        inet.lib        17,408 bytes   5:02 am  Tue Mar 25 97
        mathc.lib       35,328 bytes   5:02 am  Tue Mar 25 97
        mathh.lib       33,792 bytes   5:02 am  Tue Mar 25 97
        mathl.lib       35,840 bytes   5:02 am  Tue Mar 25 97
        mathm.lib       34,816 bytes   5:02 am  Tue Mar 25 97
        maths.lib       34,816 bytes   5:02 am  Tue Mar 25 97
        mathwc.lib      36,864 bytes   5:02 am  Tue Mar 25 97
        mathwl.lib      39,424 bytes   5:02 am  Tue Mar 25 97
        mathwm.lib      36,352 bytes   5:02 am  Tue Mar 25 97
        mathws.lib      36,352 bytes   5:02 am  Tue Mar 25 97
        msextra.lib     27,136 bytes   5:02 am  Tue Mar 25 97
        32BIT             <DIR>        1:51 pm  Sat Mar 10 00
        COMPAT            <DIR>        1:51 pm  Sat Mar 10 00
        STARTUP           <DIR>        1:51 pm  Sat Mar 10 00
        16BIT             <DIR>        1:51 pm  Sat Mar 10 00

E:\DL\DEVTOOLS\LIB\32BIT
        .                 <DIR>        1:51 pm  Sat Mar 10 00
        ..                <DIR>        1:51 pm  Sat Mar 10 00
        fileinfo.obj        183 bytes   5:02 am  Tue Mar 25 97
        files.c           1,383 bytes   5:02 am  Tue Mar 25 97
        files2.c          2,263 bytes   5:02 am  Tue Mar 25 97
        gp.obj              819 bytes   5:02 am  Tue Mar 25 97
        matherr.c         5,402 bytes   5:02 am  Tue Mar 25 97
        matherrl.c        5,013 bytes   5:02 am  Tue Mar 25 97
        wildargs.obj        234 bytes   5:02 am  Tue Mar 25 97

E:\DL\DEVTOOLS\LIB\COMPAT
        .                 <DIR>        1:51 pm  Sat Mar 10 00
        ..                <DIR>        1:51 pm  Sat Mar 10 00
        edits.obj         2,686 bytes   5:02 am  Tue Mar 25 97
        framelin.obj      1,305 bytes   5:02 am  Tue Mar 25 97
        owlmathd.obj     11,849 bytes   5:02 am  Tue Mar 25 97
        owlmathl.obj     11,849 bytes   5:02 am  Tue Mar 25 97
        setjmp.obj          268 bytes   5:02 am  Tue Mar 25 97
        swapst.obj          733 bytes   5:02 am  Tue Mar 25 97
        swndobjd.obj    125,892 bytes   5:02 am  Tue Mar 25 97
        swndobjl.obj    123,264 bytes   5:02 am  Tue Mar 25 97
        sysint.obj        3,520 bytes   5:02 am  Tue Mar 25 97
        tgrmv.obj           862 bytes   5:02 am  Tue Mar 25 97
        ttprvlns.obj      1,074 bytes   5:02 am  Tue Mar 25 97
```

```
        tvcursor.obj      1,054 bytes   5:02 am  Tue Mar 25 97
        tvexposd.obj      1,419 bytes   5:02 am  Tue Mar 25 97
        tvwrite.obj       1,641 bytes   5:02 am  Tue Mar 25 97
        versiond.obj        448 bytes   5:02 am  Tue Mar 25 97
        versionl.obj        294 bytes   5:02 am  Tue Mar 25 97

E:\DL\DEVTOOLS\LIB\STARTUP
        .                   <DIR>       1:51 pm  Sat Mar 10 00
        ..                  <DIR>       1:51 pm  Sat Mar 10 00
        buildc0.bat       2,379 bytes   5:02 am  Tue Mar 25 97
        c0.asm           26,351 bytes   5:02 am  Tue Mar 25 97
        c0d.asm          16,055 bytes   5:02 am  Tue Mar 25 97
        c0w.asm          13,183 bytes   5:02 am  Tue Mar 25 97
        rules.asi        18,852 bytes   5:02 am  Tue Mar 25 97

E:\DL\DEVTOOLS\LIB\16BIT
        .                   <DIR>       1:51 pm  Sat Mar 10 00
        ..                  <DIR>       1:51 pm  Sat Mar 10 00
        files.c           1,464 bytes   5:02 am  Tue Mar 25 97
        files2.c          1,100 bytes   5:02 am  Tue Mar 25 97
        matherr.c         5,758 bytes   5:02 am  Tue Mar 25 97
        matherrl.c        5,371 bytes   5:02 am  Tue Mar 25 97
        wildargs.obj        242 bytes   5:02 am  Tue Mar 25 97

E:\DL\DEVTOOLS\HELP
        .                   <DIR>       1:51 pm  Sat Mar 10 00
        ..                  <DIR>       1:51 pm  Sat Mar 10 00
        bctools.hlp     146,733 bytes   5:02 am  Tue Mar 25 97
        bcdos.hlp       255,735 bytes   5:02 am  Tue Mar 25 97
        bc5main.hlp      30,110 bytes   5:02 am  Tue Mar 25 97
        bc5main.gid       8,628 bytes   2:23 pm  Tue Feb  2 99
        bcdos.gid        33,234 bytes   2:22 pm  Tue Feb  2 99
        bctools.gid      16,826 bytes   2:22 pm  Tue Feb  2 99

E:\DL\DEVTOOLS\INCLUDE
        .                   <DIR>       1:51 pm  Sat Mar 10 00
        ..                  <DIR>       1:51 pm  Sat Mar 10 00
        xcmcmsxt.h          136 bytes   5:02 am  Tue Mar 25 97
        _defs.h           4,689 bytes   5:02 am  Tue Mar 25 97
        _nfile.h            378 bytes   5:02 am  Tue Mar 25 97
        _null.h             572 bytes   5:02 am  Tue Mar 25 97
        accctrl.h           122 bytes   5:02 am  Tue Mar 25 97
        aclapi.h            120 bytes   5:02 am  Tue Mar 25 97
        algorith.h      109,292 bytes   5:02 am  Tue Mar 25 97
        alloc.h           5,340 bytes   5:02 am  Tue Mar 25 97
        alphaops.h          124 bytes   5:02 am  Tue Mar 25 97
        assert.h          1,786 bytes   5:02 am  Tue Mar 25 97
        atalkwsh.h          124 bytes   5:02 am  Tue Mar 25 97
        avicap.h         30,009 bytes   5:02 am  Tue Mar 25 97
        basetyps.h          124 bytes   5:02 am  Tue Mar 25 97
        bcd.h            11,305 bytes   5:02 am  Tue Mar 25 97
        windowsx.h           94 bytes   5:02 am  Tue Mar 25 97
        winerror.h      203,089 bytes   5:02 am  Tue Mar 25 97
        wing.h            2,330 bytes   5:02 am  Tue Mar 25 97
```

```
wingdi.h          120 bytes   5:02 am  Tue Mar 25 97
winioctl.h        124 bytes   5:02 am  Tue Mar 25 97
winmem32.h        125 bytes   5:02 am  Tue Mar 25 97
winnetwk.h        124 bytes   5:02 am  Tue Mar 25 97
winnls.h          120 bytes   5:02 am  Tue Mar 25 97
winnls32.h      3,198 bytes   5:02 am  Tue Mar 25 97
winnt.h           118 bytes   5:02 am  Tue Mar 25 97
winperf.h         122 bytes   5:02 am  Tue Mar 25 97
winreg.h          120 bytes   5:02 am  Tue Mar 25 97
winresrc.h     53,541 bytes   5:02 am  Tue Mar 25 97
winsock.h      33,776 bytes   5:02 am  Tue Mar 25 97
winsock2.h     94,140 bytes   5:02 am  Tue Mar 25 97
winspool.h        124 bytes   5:02 am  Tue Mar 25 97
svrapi.h       47,122 bytes   5:02 am  Tue Mar 25 97
systypes.h        817 bytes   5:02 am  Tue Mar 25 97
tapi.h            128 bytes   5:02 am  Tue Mar 25 97
tchar.h        28,197 bytes   5:02 am  Tue Mar 25 97
time.h          3,987 bytes   5:02 am  Tue Mar 25 97
tlhelp32.h        136 bytes   5:02 am  Tue Mar 25 97
tnef.h            128 bytes   5:02 am  Tue Mar 25 97
toolhelp.h        125 bytes   5:02 am  Tue Mar 25 97
toolhelp.inc    8,916 bytes   5:02 am  Tue Mar 25 97
tree.h         40,367 bytes   5:02 am  Tue Mar 25 97
tspi.h            128 bytes   5:02 am  Tue Mar 25 97
typeinfo.h      1,996 bytes   5:02 am  Tue Mar 25 97
unknwn.h          132 bytes   5:02 am  Tue Mar 25 97
utility.h       4,240 bytes   5:02 am  Tue Mar 25 97
utime.h         1,860 bytes   5:02 am  Tue Mar 25 97
values.h        1,765 bytes   5:02 am  Tue Mar 25 97
search.h        1,614 bytes   5:02 am  Tue Mar 25 97
sehmap.h          557 bytes   5:02 am  Tue Mar 25 97
set.h          14,592 bytes   5:02 am  Tue Mar 25 97
setjmp.h        1,883 bytes   5:02 am  Tue Mar 25 97
setupapi.h    102,381 bytes   5:02 am  Tue Mar 25 97
share.h           735 bytes   5:02 am  Tue Mar 25 97
shellapi.h         94 bytes   5:02 am  Tue Mar 25 97
shlguid.h       4,786 bytes   5:02 am  Tue Mar 25 97
shlobj.h          132 bytes   5:02 am  Tue Mar 25 97
signal.h        2,027 bytes   5:02 am  Tue Mar 25 97
smpab.hr          130 bytes   5:02 am  Tue Mar 25 97
smpms.h           130 bytes   5:02 am  Tue Mar 25 97
smpxp.h           130 bytes   5:02 am  Tue Mar 25 97
snmp.h            116 bytes   5:02 am  Tue Mar 25 97
sporder.h         122 bytes   5:02 am  Tue Mar 25 97
stack.h         4,666 bytes   5:02 am  Tue Mar 25 97
mcx.h             126 bytes   5:02 am  Tue Mar 25 97
mem.h           5,395 bytes   5:02 am  Tue Mar 25 97
memory.h       14,057 bytes   5:02 am  Tue Mar 25 97
mgmtapi.h         122 bytes   5:02 am  Tue Mar 25 97
midles.h          120 bytes   5:02 am  Tue Mar 25 97
mmreg.h           118 bytes   5:02 am  Tue Mar 25 97
mmsystem.h    155,068 bytes   5:02 am  Tue Mar 25 97
mmsystem.inc   60,237 bytes   5:02 am  Tue Mar 25 97
moniker.h          92 bytes   5:02 am  Tue Mar 25 97
```

```
msacm.h          58,547 bytes   5:02 am  Tue Mar 25 97
msacmdlg.h          124 bytes   5:02 am  Tue Mar 25 97
msfs.h              128 bytes   5:02 am  Tue Mar 25 97
mspab.h             130 bytes   5:02 am  Tue Mar 25 97
mspst.h             130 bytes   5:02 am  Tue Mar 25 97
msviddrv.h          124 bytes   5:02 am  Tue Mar 25 97
msvideo.h        12,008 bytes   5:02 am  Tue Mar 25 97
varargs.h           938 bytes   5:02 am  Tue Mar 25 97
variant.h            92 bytes   5:02 am  Tue Mar 25 97
vcr.h               126 bytes   5:02 am  Tue Mar 25 97
vdmdbg.h            120 bytes   5:02 am  Tue Mar 25 97
vector.h         33,059 bytes   5:02 am  Tue Mar 25 97
ver.h                85 bytes   5:02 am  Tue Mar 25 97
vfw.h           140,054 bytes   5:02 am  Tue Mar 25 97
w32sut.h          2,140 bytes   5:02 am  Tue Mar 25 97
wdbgexts.h          136 bytes   5:02 am  Tue Mar 25 97
wfext.h              89 bytes   5:02 am  Tue Mar 25 97
winbase.h           122 bytes   5:02 am  Tue Mar 25 97
wincon.h            120 bytes   5:02 am  Tue Mar 25 97
wincrypt.h       13,402 bytes   5:02 am  Tue Mar 25 97
windef.h          7,736 bytes   5:02 am  Tue Mar 25 97
windows.h           157 bytes   5:02 am  Tue Mar 25 97
windows.inc      66,650 bytes   5:02 am  Tue Mar 25 97
stdarg.h            917 bytes   5:02 am  Tue Mar 25 97
stdcomp.h        34,095 bytes   5:02 am  Tue Mar 25 97
stddef.h          1,500 bytes   5:02 am  Tue Mar 25 97
stddefs.h         6,604 bytes   5:02 am  Tue Mar 25 97
stdexcep.h        4,767 bytes   5:02 am  Tue Mar 25 97
stdgen.h          4,114 bytes   5:02 am  Tue Mar 25 97
stdio.h          20,553 bytes   5:02 am  Tue Mar 25 97
stdiostr.h        2,272 bytes   5:02 am  Tue Mar 25 97
stdlib.h         24,876 bytes   5:02 am  Tue Mar 25 97
stdmutex.h        6,693 bytes   5:02 am  Tue Mar 25 97
stdwind.h         7,253 bytes   5:02 am  Tue Mar 25 97
storage.h            92 bytes   5:02 am  Tue Mar 25 97
stress.h            121 bytes   5:02 am  Tue Mar 25 97
string.h         80,737 bytes   5:02 am  Tue Mar 25 97
strstrea.h        4,657 bytes   5:02 am  Tue Mar 25 97
svcguid.h           122 bytes   5:02 am  Tue Mar 25 97
reconcil.h          136 bytes   5:02 am  Tue Mar 25 97
ref.h             2,395 bytes   5:02 am  Tue Mar 25 97
regexp.h          2,566 bytes   5:02 am  Tue Mar 25 97
regstr.h            132 bytes   5:02 am  Tue Mar 25 97
richedit.h       29,965 bytes   5:02 am  Tue Mar 25 97
richole.h           134 bytes   5:02 am  Tue Mar 25 97
rpc.h               172 bytes   5:02 am  Tue Mar 25 97
rpcdce.h            120 bytes   5:02 am  Tue Mar 25 97
rpcdce2.h           122 bytes   5:02 am  Tue Mar 25 97
rpcdcep.h           122 bytes   5:02 am  Tue Mar 25 97
rpcndr.h            185 bytes   5:02 am  Tue Mar 25 97
rpcnsi.h            120 bytes   5:02 am  Tue Mar 25 97
rpcnsip.h           122 bytes   5:02 am  Tue Mar 25 97
rpcnterr.h          124 bytes   5:02 am  Tue Mar 25 97
rpcproxy.h          124 bytes   5:02 am  Tue Mar 25 97
```

```
scode.h              88 bytes   5:02 am  Tue Mar 25 97
mswsock.h         4,228 bytes   5:02 am  Tue Mar 25 97
nb30.h              116 bytes   5:02 am  Tue Mar 25 97
nddeapi.h           122 bytes   5:02 am  Tue Mar 25 97
nddesec.h           122 bytes   5:02 am  Tue Mar 25 97
new.h             2,059 bytes   5:02 am  Tue Mar 25 97
nspapi.h            120 bytes   5:02 am  Tue Mar 25 97
ntsdexts.h          124 bytes   5:02 am  Tue Mar 25 97
numeric.h         7,608 bytes   5:02 am  Tue Mar 25 97
oaidl.h         197,688 bytes   5:02 am  Tue Mar 25 97
objbase.h        30,607 bytes   5:02 am  Tue Mar 25 97
objidl.h            200 bytes   5:02 am  Tue Mar 25 97
ocidl.h         244,801 bytes   5:02 am  Tue Mar 25 97
ole.h                84 bytes   5:02 am  Tue Mar 25 97
ole1cls.h           123 bytes   5:02 am  Tue Mar 25 97
ole2.h              143 bytes   5:02 am  Tue Mar 25 97
ole2dbg.h           123 bytes   5:02 am  Tue Mar 25 97
winsvc.h            120 bytes   5:02 am  Tue Mar 25 97
wintrust.h       11,279 bytes   5:02 am  Tue Mar 25 97
winuser.h           122 bytes   5:02 am  Tue Mar 25 97
winver.h            120 bytes   5:02 am  Tue Mar 25 97
winwlx.h            132 bytes   5:02 am  Tue Mar 25 97
wownt16.h         4,644 bytes   5:02 am  Tue Mar 25 97
wownt32.h         9,891 bytes   5:02 am  Tue Mar 25 97
ws2atm.h            120 bytes   5:02 am  Tue Mar 25 97
ws2spi.h         18,641 bytes   5:02 am  Tue Mar 25 97
ws2tcpip.h          124 bytes   5:02 am  Tue Mar 25 97
wshisotp.h        3,503 bytes   5:02 am  Tue Mar 25 97
wsipx.h           2,171 bytes   5:02 am  Tue Mar 25 97
wsnetbs.h         2,296 bytes   5:02 am  Tue Mar 25 97
wsnwlink.h        9,543 bytes   5:02 am  Tue Mar 25 97
wsvns.h           1,289 bytes   5:02 am  Tue Mar 25 97
wsvv.h              116 bytes   5:02 am  Tue Mar 25 97
wtypes.h         30,031 bytes   5:02 am  Tue Mar 25 97
xcmc.h              128 bytes   5:02 am  Tue Mar 25 97
xcmcext.h           134 bytes   5:02 am  Tue Mar 25 97
xcmcmsx2.h          136 bytes   5:02 am  Tue Mar 25 97
bios.h            5,383 bytes   5:02 am  Tue Mar 25 97
bitset.h         18,074 bytes   5:02 am  Tue Mar 25 97
bivbx.h           8,250 bytes   5:02 am  Tue Mar 25 97
bwcc.h            4,894 bytes   5:02 am  Tue Mar 25 97
cderr.h              88 bytes   5:02 am  Tue Mar 25 97
cguid.h             118 bytes   5:02 am  Tue Mar 25 97
checks.h          8,808 bytes   5:02 am  Tue Mar 25 97
cobjps.h            121 bytes   5:02 am  Tue Mar 25 97
coguid.h            121 bytes   5:02 am  Tue Mar 25 97
colordlg.h           95 bytes   5:02 am  Tue Mar 25 97
commctrl.h       81,049 bytes   5:02 am  Tue Mar 25 97
commdlg.h            92 bytes   5:02 am  Tue Mar 25 97
complex.h        50,132 bytes   5:02 am  Tue Mar 25 97
compnent.h        2,606 bytes   5:02 am  Tue Mar 25 97
compobj.h            92 bytes   5:02 am  Tue Mar 25 97
conio.h           8,848 bytes   5:02 am  Tue Mar 25 97
dvobj.h              88 bytes   5:02 am  Tue Mar 25 97
```

```
errno.h          4,915 bytes   5:02 am  Tue Mar 25 97
error.h            130 bytes   5:02 am  Tue Mar 25 97
except.h         2,725 bytes   5:02 am  Tue Mar 25 97
exchext.h          134 bytes   5:02 am  Tue Mar 25 97
exchform.h       1,835 bytes   5:02 am  Tue Mar 25 97
excpt.h         12,585 bytes   5:02 am  Tue Mar 25 97
fcntl.h          2,649 bytes   5:02 am  Tue Mar 25 97
float.h          5,907 bytes   5:02 am  Tue Mar 25 97
fstream.h        6,612 bytes   5:02 am  Tue Mar 25 97
ftsiface.h         136 bytes   5:02 am  Tue Mar 25 97
function.h       9,790 bytes   5:02 am  Tue Mar 25 97
generic.h        1,884 bytes   5:02 am  Tue Mar 25 97
graphics.h      16,893 bytes   5:02 am  Tue Mar 25 97
httpext.h          122 bytes   5:02 am  Tue Mar 25 97
httpfilt.h         124 bytes   5:02 am  Tue Mar 25 97
ole2ver.h           92 bytes   5:02 am  Tue Mar 25 97
oleauto.h       33,256 bytes   5:02 am  Tue Mar 25 97
olectl.h            90 bytes   5:02 am  Tue Mar 25 97
olectlid.h       7,072 bytes   5:02 am  Tue Mar 25 97
oledlg.h           132 bytes   5:02 am  Tue Mar 25 97
oleguid.h          123 bytes   5:02 am  Tue Mar 25 97
oleidl.h           132 bytes   5:02 am  Tue Mar 25 97
olenls.h           121 bytes   5:02 am  Tue Mar 25 97
pbt.h              126 bytes   5:02 am  Tue Mar 25 97
pcrt32.h           132 bytes   5:02 am  Tue Mar 25 97
pdh.h              114 bytes   5:02 am  Tue Mar 25 97
pdhmsg.h           120 bytes   5:02 am  Tue Mar 25 97
pdkver.h           132 bytes   5:02 am  Tue Mar 25 97
penwin.h       106,710 bytes   5:02 am  Tue Mar 25 97
penwoem.h          123 bytes   5:02 am  Tue Mar 25 97
plan32.h           132 bytes   5:02 am  Tue Mar 25 97
idf.h              126 bytes   5:02 am  Tue Mar 25 97
imagehlp.h      23,680 bytes   5:02 am  Tue Mar 25 97
ime.h            8,027 bytes   5:02 am  Tue Mar 25 97
imessage.h         136 bytes   5:02 am  Tue Mar 25 97
imm.h              126 bytes   5:02 am  Tue Mar 25 97
initguid.h       1,481 bytes   5:02 am  Tue Mar 25 97
initoid.h          134 bytes   5:02 am  Tue Mar 25 97
intshcut.h         136 bytes   5:02 am  Tue Mar 25 97
io.h             7,260 bytes   5:02 am  Tue Mar 25 97
iomanip.h        5,124 bytes   5:02 am  Tue Mar 25 97
iostream.h      35,952 bytes   5:02 am  Tue Mar 25 97
isguids.h          134 bytes   5:02 am  Tue Mar 25 97
iterator.h      22,157 bytes   5:02 am  Tue Mar 25 97
limits.h        36,900 bytes   5:02 am  Tue Mar 25 97
list.h          24,458 bytes   5:02 am  Tue Mar 25 97
lm.h               112 bytes   5:02 am  Tue Mar 25 97
dde.h               84 bytes   5:02 am  Tue Mar 25 97
ddeml.h             88 bytes   5:02 am  Tue Mar 25 97
ddraw.h        102,393 bytes   5:02 am  Tue Mar 25 97
deque.h         30,683 bytes   5:02 am  Tue Mar 25 97
digitalv.h      37,738 bytes   5:02 am  Tue Mar 25 97
dir.h            5,397 bytes   5:02 am  Tue Mar 25 97
direct.h         1,255 bytes   5:02 am  Tue Mar 25 97
```

```
dirent.h          3,790 bytes   5:02 am  Tue Mar 25 97
dispatch.h           94 bytes   5:02 am  Tue Mar 25 97
dispdib.h        11,214 bytes   5:02 am  Tue Mar 25 97
dlcapi.h            120 bytes   5:02 am  Tue Mar 25 97
dlgs.h               86 bytes   5:02 am  Tue Mar 25 97
dos.h            26,973 bytes   5:02 am  Tue Mar 25 97
dplay.h          10,993 bytes   5:02 am  Tue Mar 25 97
drivinit.h           94 bytes   5:02 am  Tue Mar 25 97
dsound.h         13,434 bytes   5:02 am  Tue Mar 25 97
constrea.h        6,218 bytes   5:02 am  Tue Mar 25 97
cpl.h                84 bytes   5:02 am  Tue Mar 25 97
cplext.h            132 bytes   5:02 am  Tue Mar 25 97
cstring.h        35,660 bytes   5:02 am  Tue Mar 25 97
ctl3d.h           2,712 bytes   5:02 am  Tue Mar 25 97
ctype.h           7,687 bytes   5:02 am  Tue Mar 25 97
custcntl.h           94 bytes   5:02 am  Tue Mar 25 97
d3d.h            18,648 bytes   5:02 am  Tue Mar 25 97
d3dcaps.h        13,605 bytes   5:02 am  Tue Mar 25 97
d3drm.h           5,566 bytes   5:02 am  Tue Mar 25 97
d3drmdef.h       14,250 bytes   5:02 am  Tue Mar 25 97
d3drmobj.h       32,528 bytes   5:02 am  Tue Mar 25 97
d3drmwin.h        1,336 bytes   5:02 am  Tue Mar 25 97
d3dtypes.h       32,113 bytes   5:02 am  Tue Mar 25 97
dbdaoid.h           122 bytes   5:02 am  Tue Mar 25 97
dbt.h               126 bytes   5:02 am  Tue Mar 25 97
lmaccess.h          124 bytes   5:02 am  Tue Mar 25 97
lmalert.h           122 bytes   5:02 am  Tue Mar 25 97
lmapibuf.h          124 bytes   5:02 am  Tue Mar 25 97
lmat.h              116 bytes   5:02 am  Tue Mar 25 97
lmaudit.h           122 bytes   5:02 am  Tue Mar 25 97
lmbrowsr.h          124 bytes   5:02 am  Tue Mar 25 97
lmchdev.h           122 bytes   5:02 am  Tue Mar 25 97
lmconfig.h          124 bytes   5:02 am  Tue Mar 25 97
lmcons.h            120 bytes   5:02 am  Tue Mar 25 97
lmerr.h             118 bytes   5:02 am  Tue Mar 25 97
lmerrlog.h          124 bytes   5:02 am  Tue Mar 25 97
lmmsg.h             118 bytes   5:02 am  Tue Mar 25 97
lmremutl.h          124 bytes   5:02 am  Tue Mar 25 97
lmrepl.h            120 bytes   5:02 am  Tue Mar 25 97
lmserver.h          124 bytes   5:02 am  Tue Mar 25 97
lmshare.h           122 bytes   5:02 am  Tue Mar 25 97
poppack.h         1,138 bytes   5:02 am  Tue Mar 25 97
print.h             119 bytes   5:02 am  Tue Mar 25 97
process.h         8,482 bytes   5:02 am  Tue Mar 25 97
prsht.h          13,029 bytes   5:02 am  Tue Mar 25 97
pshpack1.h        1,088 bytes   5:02 am  Tue Mar 25 97
pshpack2.h        1,088 bytes   5:02 am  Tue Mar 25 97
pshpack4.h        1,214 bytes   5:02 am  Tue Mar 25 97
pshpack8.h        1,212 bytes   5:02 am  Tue Mar 25 97
queue.h           6,296 bytes   5:02 am  Tue Mar 25 97
random.h          3,563 bytes   5:02 am  Tue Mar 25 97
ras.h               114 bytes   5:02 am  Tue Mar 25 97
rasdlg.h            120 bytes   5:02 am  Tue Mar 25 97
raserror.h          124 bytes   5:02 am  Tue Mar 25 97
```

```
rassapi.h          122 bytes   5:02 am  Tue Mar 25 97
rasshost.h         124 bytes   5:02 am  Tue Mar 25 97
recguids.h         136 bytes   5:02 am  Tue Mar 25 97
mapiform.h         136 bytes   5:02 am  Tue Mar 25 97
mapiguid.h         136 bytes   5:02 am  Tue Mar 25 97
mapihook.h         136 bytes   5:02 am  Tue Mar 25 97
mapinls.h        6,971 bytes   5:02 am  Tue Mar 25 97
mapioid.h          134 bytes   5:02 am  Tue Mar 25 97
mapispi.h          134 bytes   5:02 am  Tue Mar 25 97
mapitags.h         136 bytes   5:02 am  Tue Mar 25 97
mapiutil.h      30,879 bytes   5:02 am  Tue Mar 25 97
mapival.h          134 bytes   5:02 am  Tue Mar 25 97
mapiwin.h       15,971 bytes   5:02 am  Tue Mar 25 97
mapiwz.h           132 bytes   5:02 am  Tue Mar 25 97
mapix.h            130 bytes   5:02 am  Tue Mar 25 97
math.h          12,193 bytes   5:02 am  Tue Mar 25 97
mbctype.h        2,960 bytes   5:02 am  Tue Mar 25 97
mbstring.h       7,015 bytes   5:02 am  Tue Mar 25 97
mciavi.h           120 bytes   5:02 am  Tue Mar 25 97
lmsname.h          122 bytes   5:02 am  Tue Mar 25 97
lmstats.h          122 bytes   5:02 am  Tue Mar 25 97
lmsvc.h            118 bytes   5:02 am  Tue Mar 25 97
lmuse.h            118 bytes   5:02 am  Tue Mar 25 97
lmuseflg.h         124 bytes   5:02 am  Tue Mar 25 97
lmwksta.h          122 bytes   5:02 am  Tue Mar 25 97
loadperf.h       1,446 bytes   5:02 am  Tue Mar 25 97
locale.h         3,438 bytes   5:02 am  Tue Mar 25 97
lsapi.h            118 bytes   5:02 am  Tue Mar 25 97
lzexpand.h          94 bytes   5:02 am  Tue Mar 25 97
malloc.h         3,835 bytes   5:02 am  Tue Mar 25 97
map.h           16,755 bytes   5:02 am  Tue Mar 25 97
mapi.h             128 bytes   5:02 am  Tue Mar 25 97
mapicode.h      10,606 bytes   5:02 am  Tue Mar 25 97
mapidbg.h       24,915 bytes   5:02 am  Tue Mar 25 97
mapidefs.h     104,039 bytes   5:02 am  Tue Mar 25 97
WIN16             <DIR>        1:52 pm  Sat Mar 10 00
WIN32             <DIR>        1:52 pm  Sat Mar 10 00
SYS               <DIR>        1:53 pm  Sat Mar 10 00

E:\DL\DEVTOOLS\INCLUDE\WIN16
.                 <DIR>        1:52 pm  Sat Mar 10 00
..                <DIR>        1:52 pm  Sat Mar 10 00
cderr.h          1,584 bytes   5:02 am  Tue Mar 25 97
cobjps.h         2,420 bytes   5:02 am  Tue Mar 25 97
coguid.h         3,111 bytes   5:02 am  Tue Mar 25 97
colordlg.h       1,103 bytes   5:02 am  Tue Mar 25 97
commdlg.h       12,600 bytes   5:02 am  Tue Mar 25 97
compobj.h       34,262 bytes   5:02 am  Tue Mar 25 97
cpl.h            5,948 bytes   5:02 am  Tue Mar 25 97
custcntl.h      10,383 bytes   5:02 am  Tue Mar 25 97
dde.h            5,041 bytes   5:02 am  Tue Mar 25 97
ddeml.h         15,942 bytes   5:02 am  Tue Mar 25 97
dispatch.h      46,285 bytes   5:02 am  Tue Mar 25 97
dlgs.h           5,646 bytes   5:02 am  Tue Mar 25 97
```

```
        drivinit.h          57 bytes   5:02 am  Tue Mar 25 97
        dvobj.h         16,277 bytes   5:02 am  Tue Mar 25 97
        initguid.h       1,382 bytes   5:02 am  Tue Mar 25 97
        lzexpand.h       3,631 bytes   5:02 am  Tue Mar 25 97
        toolhelp.h      14,609 bytes   5:02 am  Tue Mar 25 97
        variant.h       10,475 bytes   5:02 am  Tue Mar 25 97
        ver.h            9,360 bytes   5:02 am  Tue Mar 25 97
        wfext.h          2,914 bytes   5:02 am  Tue Mar 25 97
        windows.h      159,388 bytes   5:02 am  Tue Mar 25 97
        windowsx.h      68,391 bytes   5:02 am  Tue Mar 25 97
        winmem32.h       1,383 bytes   5:02 am  Tue Mar 25 97
        moniker.h        9,381 bytes   5:02 am  Tue Mar 25 97
        ole.h           25,532 bytes   5:02 am  Tue Mar 25 97
        ole1cls.h        6,553 bytes   5:02 am  Tue Mar 25 97
        ole2.h          44,686 bytes   5:02 am  Tue Mar 25 97
        ole2dbg.h          516 bytes   5:02 am  Tue Mar 25 97
        ole2ver.h          256 bytes   5:02 am  Tue Mar 25 97
        olectl.h        43,472 bytes   5:02 am  Tue Mar 25 97
        oleguid.h        3,855 bytes   5:02 am  Tue Mar 25 97
        olenls.h        22,377 bytes   5:02 am  Tue Mar 25 97
        penwin.h        26,491 bytes   5:02 am  Tue Mar 25 97
        penwoem.h        2,680 bytes   5:02 am  Tue Mar 25 97
        print.h         11,203 bytes   5:02 am  Tue Mar 25 97
        scode.h         10,763 bytes   5:02 am  Tue Mar 25 97
        shellapi.h       2,870 bytes   5:02 am  Tue Mar 25 97
        storage.h       13,289 bytes   5:02 am  Tue Mar 25 97
        stress.h         1,909 bytes   5:02 am  Tue Mar 25 97

E:\DL\DEVTOOLS\INCLUDE\WIN32
        .                  <DIR>       1:52 pm  Sat Mar 10 00
        ..                 <DIR>       1:52 pm  Sat Mar 10 00
        accctrl.h        8,779 bytes   5:02 am  Tue Mar 25 97
        aclapi.h        14,236 bytes   5:02 am  Tue Mar 25 97
        alphaops.h      43,125 bytes   5:02 am  Tue Mar 25 97
        atalkwsh.h       5,702 bytes   5:02 am  Tue Mar 25 97
        basetyps.h       9,173 bytes   5:02 am  Tue Mar 25 97
        cderr.h          2,236 bytes   5:02 am  Tue Mar 25 97
        cguid.h          3,536 bytes   5:02 am  Tue Mar 25 97
        colordlg.h       1,471 bytes   5:02 am  Tue Mar 25 97
        commdlg.h       28,283 bytes   5:02 am  Tue Mar 25 97
        compobj.h          552 bytes   5:02 am  Tue Mar 25 97
        cpl.h            8,399 bytes   5:02 am  Tue Mar 25 97
        cplext.h         2,199 bytes   5:02 am  Tue Mar 25 97
        custcntl.h       8,578 bytes   5:02 am  Tue Mar 25 97
        dbdaoid.h        5,110 bytes   5:02 am  Tue Mar 25 97
        dbt.h           10,861 bytes   5:02 am  Tue Mar 25 97
        dde.h            5,254 bytes   5:02 am  Tue Mar 25 97
        mapitags.h      68,479 bytes   5:02 am  Tue Mar 25 97
        mapival.h       90,211 bytes   5:02 am  Tue Mar 25 97
        mapiwz.h         2,018 bytes   5:02 am  Tue Mar 25 97
        mapix.h         28,872 bytes   5:02 am  Tue Mar 25 97
        mciavi.h         3,028 bytes   5:02 am  Tue Mar 25 97
        mcx.h            3,954 bytes   5:02 am  Tue Mar 25 97
        mgmtapi.h        5,354 bytes   5:02 am  Tue Mar 25 97
```

```
midles.h          6,243 bytes   5:02 am  Tue Mar 25 97
mmreg.h          69,352 bytes   5:02 am  Tue Mar 25 97
moniker.h           552 bytes   5:02 am  Tue Mar 25 97
msacmdlg.h        1,068 bytes   5:02 am  Tue Mar 25 97
msfs.h           17,168 bytes   5:02 am  Tue Mar 25 97
mspab.h           2,182 bytes   5:02 am  Tue Mar 25 97
mspst.h           5,016 bytes   5:02 am  Tue Mar 25 97
msviddrv.h        4,720 bytes   5:02 am  Tue Mar 25 97
nb30.h           12,935 bytes   5:02 am  Tue Mar 25 97
ddeml.h          17,907 bytes   5:02 am  Tue Mar 25 97
dispatch.h          554 bytes   5:02 am  Tue Mar 25 97
dlcapi.h         34,039 bytes   5:02 am  Tue Mar 25 97
dlgs.h            5,731 bytes   5:02 am  Tue Mar 25 97
drivinit.h           32 bytes   5:02 am  Tue Mar 25 97
dvobj.h             502 bytes   5:02 am  Tue Mar 25 97
error.h          14,376 bytes   5:02 am  Tue Mar 25 97
exchext.h        30,638 bytes   5:02 am  Tue Mar 25 97
ftsiface.h        6,828 bytes   5:02 am  Tue Mar 25 97
httpext.h         7,880 bytes   5:02 am  Tue Mar 25 97
httpfilt.h       12,922 bytes   5:02 am  Tue Mar 25 97
idf.h            12,754 bytes   5:02 am  Tue Mar 25 97
imessage.h        8,439 bytes   5:02 am  Tue Mar 25 97
imm.h            20,673 bytes   5:02 am  Tue Mar 25 97
initoid.h         1,506 bytes   5:02 am  Tue Mar 25 97
intshcut.h       15,151 bytes   5:02 am  Tue Mar 25 97
pdkver.h            120 bytes   5:02 am  Tue Mar 25 97
plan32.h            915 bytes   5:02 am  Tue Mar 25 97
ras.h            32,425 bytes   5:02 am  Tue Mar 25 97
rasdlg.h          6,628 bytes   5:02 am  Tue Mar 25 97
raserror.h       18,260 bytes   5:02 am  Tue Mar 25 97
rassapi.h        10,676 bytes   5:02 am  Tue Mar 25 97
rasshost.h        4,725 bytes   5:02 am  Tue Mar 25 97
recguids.h          606 bytes   5:02 am  Tue Mar 25 97
reconcil.h        5,638 bytes   5:02 am  Tue Mar 25 97
regstr.h         60,367 bytes   5:02 am  Tue Mar 25 97
richole.h         6,713 bytes   5:02 am  Tue Mar 25 97
rpc.h             2,282 bytes   5:02 am  Tue Mar 25 97
rpcdce.h         42,096 bytes   5:02 am  Tue Mar 25 97
rpcdce2.h         4,480 bytes   5:02 am  Tue Mar 25 97
rpcdcep.h         9,717 bytes   5:02 am  Tue Mar 25 97
rpcndr.h         76,998 bytes   5:02 am  Tue Mar 25 97
nddeapi.h        14,759 bytes   5:02 am  Tue Mar 25 97
nddesec.h         3,150 bytes   5:02 am  Tue Mar 25 97
nspapi.h         19,062 bytes   5:02 am  Tue Mar 25 97
ntsdexts.h        1,568 bytes   5:02 am  Tue Mar 25 97
objerror.h          340 bytes   5:02 am  Tue Mar 25 97
objidl.h        331,556 bytes   5:02 am  Tue Mar 25 97
ole.h            25,806 bytes   5:02 am  Tue Mar 25 97
ole2.h           12,955 bytes   5:02 am  Tue Mar 25 97
ole2ver.h           664 bytes   5:02 am  Tue Mar 25 97
olectl.h         20,592 bytes   5:02 am  Tue Mar 25 97
oledlg.h         74,840 bytes   5:02 am  Tue Mar 25 97
oleidl.h        159,861 bytes   5:02 am  Tue Mar 25 97
pbt.h             1,448 bytes   5:02 am  Tue Mar 25 97
```

```
pcrt32.h          1,534 bytes   5:02 am  Tue Mar 25 97
pdh.h            16,379 bytes   5:02 am  Tue Mar 25 97
pdhmsg.h          8,370 bytes   5:02 am  Tue Mar 25 97
tlhelp32.h        6,323 bytes   5:02 am  Tue Mar 25 97
tnef.h           14,339 bytes   5:02 am  Tue Mar 25 97
tspi.h           41,881 bytes   5:02 am  Tue Mar 25 97
unknwn.h          9,523 bytes   5:02 am  Tue Mar 25 97
variant.h           513 bytes   5:02 am  Tue Mar 25 97
vcr.h            20,278 bytes   5:02 am  Tue Mar 25 97
vdmdbg.h         13,620 bytes   5:02 am  Tue Mar 25 97
ver.h              193 bytes    5:02 am  Tue Mar 25 97
wdbgexts.h       13,515 bytes   5:02 am  Tue Mar 25 97
wfext.h           6,570 bytes   5:02 am  Tue Mar 25 97
winbase.h       155,355 bytes   5:02 am  Tue Mar 25 97
wincon.h         14,820 bytes   5:02 am  Tue Mar 25 97
windows.h         6,901 bytes   5:02 am  Tue Mar 25 97
windowsx.h       73,354 bytes   5:02 am  Tue Mar 25 97
wingdi.h        147,988 bytes   5:02 am  Tue Mar 25 97
winioctl.h       30,577 bytes   5:02 am  Tue Mar 25 97
rpcnsi.h         14,693 bytes   5:02 am  Tue Mar 25 97
rpcnsip.h         1,215 bytes   5:02 am  Tue Mar 25 97
rpcnterr.h        1,569 bytes   5:02 am  Tue Mar 25 97
rpcproxy.h       17,162 bytes   5:02 am  Tue Mar 25 97
scode.h             292 bytes   5:02 am  Tue Mar 25 97
scrnsave.h        8,778 bytes   5:02 am  Tue Mar 25 97
shellapi.h       16,941 bytes   5:02 am  Tue Mar 25 97
shlobj.h         95,124 bytes   5:02 am  Tue Mar 25 97
smpab.h           1,272 bytes   5:02 am  Tue Mar 25 97
smpms.h           1,931 bytes   5:02 am  Tue Mar 25 97
smpxp.h           3,817 bytes   5:02 am  Tue Mar 25 97
snmp.h           17,065 bytes   5:02 am  Tue Mar 25 97
sporder.h         2,008 bytes   5:02 am  Tue Mar 25 97
storage.h           513 bytes   5:02 am  Tue Mar 25 97
svcguid.h        16,218 bytes   5:02 am  Tue Mar 25 97
tapi.h          150,324 bytes   5:02 am  Tue Mar 25 97
lmserver.h       47,846 bytes   5:02 am  Tue Mar 25 97
lmshare.h        10,539 bytes   5:02 am  Tue Mar 25 97
lmsname.h         3,104 bytes   5:02 am  Tue Mar 25 97
lmstats.h         5,318 bytes   5:02 am  Tue Mar 25 97
lmsvc.h          13,439 bytes   5:02 am  Tue Mar 25 97
lmuse.h           3,746 bytes   5:02 am  Tue Mar 25 97
lmuseflg.h          763 bytes   5:02 am  Tue Mar 25 97
lmwksta.h        18,383 bytes   5:02 am  Tue Mar 25 97
lsapi.h          11,839 bytes   5:02 am  Tue Mar 25 97
lzexpand.h        1,718 bytes   5:02 am  Tue Mar 25 97
mapi.h           12,097 bytes   5:02 am  Tue Mar 25 97
mapiform.h       28,479 bytes   5:02 am  Tue Mar 25 97
mapiguid.h       11,689 bytes   5:02 am  Tue Mar 25 97
mapihook.h        3,047 bytes   5:02 am  Tue Mar 25 97
mapioid.h         2,837 bytes   5:02 am  Tue Mar 25 97
mapispi.h        47,798 bytes   5:02 am  Tue Mar 25 97
xcmcmsxt.h        1,855 bytes   5:02 am  Tue Mar 25 97
isguids.h           786 bytes   5:02 am  Tue Mar 25 97
lm.h              2,025 bytes   5:02 am  Tue Mar 25 97
```

```
        lmaccess.h      40,890 bytes   5:02 am  Tue Mar 25 97
        lmalert.h        3,779 bytes   5:02 am  Tue Mar 25 97
        lmapibuf.h       1,362 bytes   5:02 am  Tue Mar 25 97
        lmat.h           3,644 bytes   5:02 am  Tue Mar 25 97
        lmaudit.h       10,978 bytes   5:02 am  Tue Mar 25 97
        lmbrowsr.h       6,035 bytes   5:02 am  Tue Mar 25 97
        lmchdev.h        5,507 bytes   5:02 am  Tue Mar 25 97
        lmconfig.h       1,764 bytes   5:02 am  Tue Mar 25 97
        lmcons.h         8,343 bytes   5:02 am  Tue Mar 25 97
        lmerr.h         36,493 bytes   5:02 am  Tue Mar 25 97
        lmerrlog.h      38,545 bytes   5:02 am  Tue Mar 25 97
        lmmsg.h          2,191 bytes   5:02 am  Tue Mar 25 97
        lmremutl.h       2,987 bytes   5:02 am  Tue Mar 25 97
        lmrepl.h         6,792 bytes   5:02 am  Tue Mar 25 97
        winnetwk.h      22,778 bytes   5:02 am  Tue Mar 25 97
        winnls.h        39,690 bytes   5:02 am  Tue Mar 25 97
        winnt.h        181,386 bytes   5:02 am  Tue Mar 25 97
        winperf.h       28,345 bytes   5:02 am  Tue Mar 25 97
        winreg.h        14,363 bytes   5:02 am  Tue Mar 25 97
        winspool.h      52,713 bytes   5:02 am  Tue Mar 25 97
        winsvc.h        19,110 bytes   5:02 am  Tue Mar 25 97
        winuser.h      192,731 bytes   5:02 am  Tue Mar 25 97
        winver.h         9,543 bytes   5:02 am  Tue Mar 25 97
        winwlx.h        22,022 bytes   5:02 am  Tue Mar 25 97
        ws2atm.h        16,711 bytes   5:02 am  Tue Mar 25 97
        ws2tcpip.h       2,677 bytes   5:02 am  Tue Mar 25 97
        wsvv.h           1,695 bytes   5:02 am  Tue Mar 25 97
        xcmc.h          14,242 bytes   5:02 am  Tue Mar 25 97
        xcmcext.h        2,950 bytes   5:02 am  Tue Mar 25 97
        xcmcmsx2.h       1,263 bytes   5:02 am  Tue Mar 25 97

E:\DL\DEVTOOLS\INCLUDE\SYS
        .                  <DIR>       1:53 pm  Sat Mar 10 00
        ..                 <DIR>       1:53 pm  Sat Mar 10 00
        locking.h          618 bytes   5:02 am  Tue Mar 25 97
        stat.h           4,068 bytes   5:02 am  Tue Mar 25 97
        timeb.h            849 bytes   5:02 am  Tue Mar 25 97
        types.h            587 bytes   5:02 am  Tue Mar 25 97

E:\DL\DEVTOOLS\BIN
        .                  <DIR>       1:53 pm  Sat Mar 10 00
        ..                 <DIR>       1:53 pm  Sat Mar 10 00
        bcc32.cfg           48 bytes   6:05 pm  Thu Jan 28 99
        bcc.exe        851,968 bytes   5:02 am  Tue Mar 25 97
        bcroot.inc          15 bytes   7:16 pm  Wed Jan  6 99
        bcc32.exe      700,416 bytes   5:02 am  Tue Mar 25 97
        bopnhlp.cfg      5,187 bytes   6:04 pm  Thu Jan 28 99
        builtins.mak       649 bytes   5:02 am  Tue Mar 25 97
        brc.exe         47,648 bytes   5:02 am  Tue Mar 25 97
        brc32.exe       47,648 bytes   5:02 am  Tue Mar 25 97
        brcc.exe        45,600 bytes   5:02 am  Tue Mar 25 97
        brcc32.exe      76,320 bytes   5:02 am  Tue Mar 25 97
        cw3220.dll     229,376 bytes   5:02 am  Tue Mar 25 97
        cpp.exe        159,744 bytes   5:02 am  Tue Mar 25 97
```

```
cpp32.exe       159,744 bytes   5:02 am  Tue Mar 25 97
cw3220mt.dll    249,856 bytes   5:02 am  Tue Mar 25 97
cw3230.dll      303,104 bytes   5:02 am  Tue Mar 25 97
tlink32.cfg          22 bytes   6:05 pm  Thu Jan 28 99
tlink.exe       120,874 bytes   5:02 am  Tue Mar 25 97
turboc.cfg           48 bytes   6:05 pm  Thu Jan 28 99
tlink32.exe     233,472 bytes   5:02 am  Tue Mar 25 97
touch.com         5,528 bytes   4:50 am  Thu Nov 17 94
trigraph.exe     16,460 bytes   5:02 am  Tue Mar 25 97
bc520rtl.dll    229,888 bytes   5:02 am  Tue Mar 25 97
winstub.exe          578 bytes   5:02 am  Tue Mar 25 97
ned.cfg              580 bytes   6:51 pm  Thu Jan 28 99
proto.exe        10,448 bytes   4:43 pm  Mon Feb  1 99
dumpdos.exe      16,218 bytes   6:45 pm  Thu Jan 28 99
hex2bin.exe      35,174 bytes   6:00 pm  Thu Jan 28 99
loc.exe          98,420 bytes   6:07 pm  Thu Jan 28 99
rw32core.dll    654,880 bytes   5:02 am  Tue Mar 25 97
ned.exe          88,930 bytes   6:19 pm  Thu Jan 28 99
promerge.exe     41,074 bytes   6:00 pm  Thu Jan 28 99
recurse.exe      10,386 bytes   6:00 pm  Thu Jan 28 99
rtm.exe         120,853 bytes   5:02 am  Tue Mar 25 97
rw32ui.dll    1,380,384 bytes   5:02 am  Tue Mar 25 97
rwaddon.dll     167,456 bytes   5:02 am  Tue Mar 25 97
rwctls.dll      754,720 bytes   5:02 am  Tue Mar 25 97
rwdesign.dll    458,784 bytes   5:02 am  Tue Mar 25 97
rwimged.dll     138,784 bytes   5:02 am  Tue Mar 25 97
rwinspct.dll    308,256 bytes   5:02 am  Tue Mar 25 97
rwres.dll       187,936 bytes   5:02 am  Tue Mar 25 97
tlink.cfg            22 bytes   6:05 pm  Thu Jan 28 99
tasm.exe        134,978 bytes   4:00 am  Fri May 19 95
tdmem.exe        16,752 bytes   5:02 am  Tue Mar 25 97
tdstrip.exe      16,920 bytes   5:02 am  Tue Mar 25 97
tdstrp32.exe     45,056 bytes   5:02 am  Tue Mar 25 97
tdump.exe       184,320 bytes   5:02 am  Tue Mar 25 97
tlib.exe         61,440 bytes   5:02 am  Tue Mar 25 97
cw3230mt.dll    319,488 bytes   5:02 am  Tue Mar 25 97
dpmi16bi.ovl     50,576 bytes   5:02 am  Tue Mar 25 97
dpmi32vm.ovl     58,376 bytes   5:02 am  Tue Mar 25 97
locale.bll       95,246 bytes   5:02 am  Tue Mar 25 97
grep.com         98;304 bytes   5:02 am  Tue Mar 25 97
hc31.exe        174,439 bytes   2:00 pm  Fri Sep 16 94
impdef.exe       73,728 bytes   5:02 am  Tue Mar 25 97
implib.exe       90,112 bytes   5:02 am  Tue Mar 25 97
openhelp.cfg      1,071 bytes   6:05 pm  Thu Jan 28 99
make.exe         86,016 bytes   5:00 am  Wed Feb 21 96
maker.exe        72,088 bytes   5:02 am  Tue Mar 25 97
makeswap.exe     35,154 bytes   5:02 am  Tue Mar 25 97
objxref.exe      86,016 bytes   5:02 am  Tue Mar 25 97
rlink32.dll      53,248 bytes   5:02 am  Tue Mar 25 97
openhelp.exe    311,296 bytes   5:02 am  Tue Mar 25 97
rlink.exe        68,368 bytes   5:02 am  Tue Mar 25 97
showerr.exe      34,914 bytes   9:55 am  Wed Feb  5 97
split.exe        37,672 bytes   6:00 pm  Thu Jan 28 99
dump.exe          8,014 bytes  10:08 am  Thu Aug 18 88
```

```
      bin2c.exe         7,716 bytes  12:09 pm  Tue Dec  6 88
      32rtm.exe       152,108 bytes   5:02 am  Tue Mar 25 97

1,084 files found
```

6.1.3 List of Removed Extraneous Functions

The following is the list of ROM-DOS external commands that have been removed as extraneous. Datalight says that ROM-DOS follows the standard DOS conventions.

External DOS 6.2 Commands Removed

Command	Description
ANSI.SYS	A console device driver that allows support of ANSI codes on the local screen.
ATA.SYS	A PCMCIA ATA disk device driver.
ATTRIB	Displays or modifies the attributes associated with a file.
BACKUP	Backs up a single directory tree to a floppy drive, hard disk, or network drive.
CALL	Batch file command. Invokes execution of a secondary batch file.
CHKDSK	Checks the integrity of data on a disk. Displays information.
CHOICE	Allows a user to make a processing choice during the execution of a batch file.
COMMAND	Starts a second DOS command processor.
DEFRAG	Reorganizes fragmented disk files to optimize disk space and system performance.
DELTREE	Deletes one or more directory trees or individual files.
DISKCOPY	Copies the contents of one floppy disk to another of the same type.
DISPLAY	Displays international letters and symbols.
ECHO	Batch file command. Turns on or off display of batch execution on the monitor.
EGA/EGA3.CPI: DISPLAY.SYS.	Font data files for use with the International video display driver,
EMM386	Enables expanded memory support for capable systems.
FDISK	Initializes and partitions a hard disk for DOS.
FIND	Works as a filter to display only lines that contain a specified string.
FOR	Batch file command. Performs one DOS command on a set of files.
FORMAT	Initializes a disk so that ROM-DOS can access files on that disk.
GOTO	Batch file command. Moves control to a specified line in the batch file.
HELP	Lists all available ROM-DOS commands along with brief descriptions.
IF	Batch file command. Performs a command based on a specified condition.
INCLUDE	Allows instructions in one configuration block to be included with instructions in another configuration block.
KEYB	Allows altering of the keyboard layout for a different language or nationality.
KEYBOARD/ KEYBRD2.SYS	Keyboard code page data files for use with the International keyboard driver, KEYB.COM.

External DOS 6.2 Commands Removed

Command	**Description**
LABEL	Creates, changes, or deletes a disk volume label.
LONGDIR.EXE	Provides a directory listing of files in a single directory including files with long file names.
MCDEX	Enables the use of CD-ROM drives.
MEM	Displays the used and free memory in your system.
MENUCOLOR	Allows setting of text and background colors for the startup menu.
MENUDEFAULT	Sets the default menu-item choice and time-out value for making a selection.
MENUITEM	Specifies an item to be placed on the startup menu display during system boot.
MODE	Modifies the operation of the printer, serial port, and active video display.
MORE	Displays a text file one screen at a time.
MOVE	Moves files and renames files and directories.
NEWFILE	Allows continuation of CONFIG.SYS processing from a new file.
NUMLOCK	Sets the NUMLOCK keyboard key to on or off when your computer starts.
PAUSE	Batch file command. Causes execution to halt until a key is pressed.
POWER	Conserves power on the system that employs an APM BIOS.
PRINT	Prints a list of files, up to ten files.
REMQUIT.EXE	Terminates the REMSERV program via the serial connection.
RESTORE	Restores files previously saved with BACKUP to the hard disk.
SHARE	Installs the capabilities for file sharing and file locking on your hard disk.
SHIFT	Batch file command. Shifts replaceable parameters one position to the left.
SORT	Sorts a text file and displays the output to the standard device.
STACKDEV.SYS	Increases the number of stacks available for IRQ handlers and Int13h.
SUBMENU	Defines a menu item that represents a secondary menu.
SUBST	Allows one drive to appear as another drive.
SYS	Transfers the hidden system files to a specified drive.
TREE	Displays the path of each directory on a specified drive.
UMBLINK.SYS	A non-protected mode program that can allow the creation of Upper Memory Blocks using existing RAM areas.
VDISK	Allows the use of memory as a simulated disk driver.
VERSION.SYS	Modifies the version number ROM-DOS reports.
XCOPY	Copies multiple files and optionally subdirectories.

6.1.4 Narrative Regarding Use of All Retained Components

Only a very minimal subset of ROM-DOS was loaded and used. The disk image in the PROM contains only the following files.

```
 Volume in drive A has no label
 Volume Serial Number is 2D69-14D7
 Directory of A:\

IBMBIO   COM        54,542  07-22-99   6:22a -- Hidden
IBMDOS   COM            74  07-22-99   6:22a -- Hidden
COMMAND  COM        34,099  07-22-99   6:22a
AUTOEXEC BAT            73  11-04-99   9:07a
HIMEM    SYS         5,832  07-22-99   6:22a
DATA_ATT EXE        87,608  01-31-00   1:05p
CONFIG   SYS            85  02-08-00  10:16a
```

The BIOS functions and the DOS services functions, which are invoked by software interrupts are retained.

There are several DOS commands that can be invoked from the command line or a batch file. Only the DOS commands internal to COMMAND.COM are retained. These are listed in the following table. Two MS-DOS internal commands (e.g., CHCP and DRIVPARM) are not in the ROM-DOS list of commands and these were tested and found not to be internal ROM-DOS commands.

HIMEM.SYS to handle extended memory is also retained as an external DOS command.

Internal DOS 6.2 Commands Retained

Command	**Description**
CHDIR (also CD)	Changes the current directory (also CD).
CLS	Clears all information from the monitor's screen.
COPY	Copies files from one storage location to another.
CTTY	Changes the default terminal interacting with ROM-DOS.
DATE	Displays the date from the system's internal calendar. Allows revision.
DEL	Deletes specified files.
DIR DIRectory.	Lists contents of a specified directory.
ERASE	Erases specified files (same as DEL).
EXIT	Used to exit nested running of ROM-DOS within another program.
LOADHIGH	Loads a program into the upper memory area, if available.
MKDIR	Creates a new subdirectory.
PATH	Displays current command search path(s). A new path line can be specified.
PROMPT	Resets the appearance of the system prompt line.
RMDIR (also RM)	Deletes a specified subdirectory.
REN	Renames files.
SET	Sets environment variables and command processor strings.
TIME	Displays current time from the system's internal clock. Also allows revision.
TYPE	Displays the contents of a text file on the monitor.
VER	Displays current version of ROM-DOS on the monitor.
VERIFY	Displays the current VERIFY state or sets the VERIFY state to on or off.
VOL	Displays the volume label on a disk.

Internal DOS 6.2 Commands for use in CONFIG.SYS Retained

Command	**Description**
BREAK	Turns on or off the ability to stop program execution at a non-I/O point.
BUFFERS	Sets the number of internal data buffers.
COUNTRY	Designates the country code for displays.
DEVICE	Installs a device driver into ROM-DOS.
DEVICEHIGH	Loads a device into the upper memory area, if available.
DOS	Installs ROM-DOS into High Memory Area (HMA).
FCBS	Specifies the number of File Control Blocks (FCBS) open at one time.
FILES	Sets the maximum number of files that can be open at one time on the system.
INSTALL	Loads Terminate and Stay Resident (TSR) programs during CONFIG.SYS processing.
LASTDRIVE	Sets the maximum number of drives.
REM	A batch file command for identifying non-executing lines.
SHELL	Allows selections of a command interpreter other than COMMAND.COM.
STACKS	Allows for the use of dynamic data stacks to handle interrupts.
SWITCHES	Allows special CONFIG.SYS file options.

External DOS 6.2 Commands Retained

Command	**Description**
HIMEM	Manages extended memory and the high memory area in a 286, 386, PS/2 system.

6.1.5 List of ROM-DOS Setup Parameters

The ROM-DOS setup parameters are found in the CONFIG.SYS and AUTOEXEC.BAT files listed in the subsequent section.

If a CONFIG.SYS file is not found, ROM-DOS uses the following as default values for the following commands (page 13 of ROM-DOS User's Guide):

```
BREAK = OFF
BUFFERS = 15
COUNTRY = 001
FCBS = 4
FILES = 8
NUMLOCK = ON
SHELL = COMMAND.COM /P /E:128
STACKS = 0,0
```

6.1.6 Listing of CONFIG.SYS and AUTOEXEC.BAT Files

Copies of the CONFIG.SYS and AUTOEXEC.BAT files used in the computational block are listed as follows.

6.1.6.1 CONFIG.SYS Listing

```
DEVICE=HIMEM.SYS

FILES=60
BUFFERS=20
DOS=HIGH,UMB
STACKS=9,256
```

Documented Version of CONFIG.SYS

```
;       Memory manager for extended memory and the High Memory Area (HMA)
;       See page 57 of the ROM-DOS user's guide
;       There is no evidence in the manual that HIMEM performs a memory check
DEVICE=HIMEM.SYS

;       Maximum number of files that can be open at one time
;       This value can be between 10 and 255
;       The default value is FILES=8 (see page 13 of user's guide)
;       See page 53-54 of the ROM-DOS user's guide
FILES=60

;       Maximum number of internal buffers used to hold file data
;       This value can be between 2 and 40.
;       The default value is BUFFERS=15 (see page 13 of user's guide)
;       See page 31-32 of the ROM-DOS user's guide
BUFFERS=20

;       DOS=HIGH → Loads ROM-DOS into the High Memory Area (HMA)
;       DOS=UMB → Allows DOS to load resident programs into upper memory blocks
;                       (UMBs) with DEVICEHIGH= and LOADHIGH=.  [MS-DOS]
;       DOS=UMB is not documented for ROM-DOS and is apparently the default condition
;       See page 47 of the ROM-DOS user's guide
DOS=HIGH,UMB

;       Enables the dynamic use of data stacks
;       Argument 1 = The number of stacks must be between 8 and 64
;       Argument 2 = Size of each stack in bytes must be between 32 and 512
;       The default value is STACKS = 0,0 (see page 13 of user's guide)
;       See page 83 of the ROM-DOS user's guide
STACKS=9,256
```

6.1.6.2 AUTOEXEC.BAT Listing

```
@ECHO OFF
SET DIRCMD=/O:N
CLS
PATH=A:\
Data_Att.exe
```

Documented Version of AUTOEXEC.BAT

```
;       @ Prevents this line from being echoed to the screen.  See page 27 of user's guide
;       Prevents future lines from being echoed to the screen.  See page 47 of user's guide
@ECHO OFF

;       Sets the environmental variable DIRCMD.  See page 79 of ROM-DOS user's guide
;       This DIR command option causes the files displayed by the directory command
;               to be sorted alphabetically by name.  See page 45 of ROM-DOS user's guide
SET DIRCMD=/O:N

;       Clear the monitor screen.  See page 36 of ROM-DOS user's guide
CLS

;       Sets the path to the disk image in the PROM.  See page 72 of ROM-DOS user's guide
PATH=A:\

;       Run the application program
Data_Att.exe
```

6.1.7 Memory Map Showing Configuration in All Memory ICs

The memory configuration is important to understand for system authentication. The following information is provided for clarity.

- Memory IC information
- MEM /B – ROM-DOS memory map of the system, which displays each BIOS extension and its size
- MEM /C – ROM-DOS memory map of the system, which classifies the memory usage
- MEM /R – ROM-DOS memory map of the system, which does a raw dump of the memory control blocks (MCB)

The ROM-DOS MEM command is explained on page 64 of the ROM-DOS user's guide. The output redirection command ">" is the same as for MS-DOS and is explained on page 9 of the ROM-DOS user's guide.

Table Showing Available Memory within 3SXi and Potential Memory Map Locations

ADDRESS RANGE Hex	SIZE bytes	FUNCTION
DRAM	1Mx16	2-Mbyte main memory = Toshiba, TC5118160CFT-60
200000 – 3FFFFF	2048k	Extended memory - 2 Mbyte of DRAM – in 2nd DRAM ←
100000 – 1FFFFF	1024k	Extended memory - 1 Mbyte of DRAM
0F0000 – 0FFFFF	64k	Ampro ROM-BIOS – possible shadow copy in RAM
0E0000 – 0EFFFF	64k	Possible memory window into extended memory
0D0000 – 0DFFFF	64k	Flash or Socket memory window into 1 Mbyte of memory
0C0000 – 0CFFFF	64k	Video BIOS – possible shadow copy in RAM for speed
0A0000 – 0BFFFF	128k	Video Screen RAM window into 512k bytes of video DRAM
000000 – 09FFFF	640k	Possible 640-kbytes of DRAM available for programs
		See MEM memory maps for details
FLASH – A ←	1Mx8	1-Mbyte BIOS & OEM flash memory = Intel, 28F008SA
010000 – 0FFFFF	960k	Unused
000000 – 00FFFF	64k	Ampro BIOS
FLASH – B	1Mx8	128-kbyte BIOS & OEM flash memory = Intel, 28F010
010000 – 01FFFF	64k	Unused
000000 – 00FFFF	64k	Ampro BIOS
PROM – A ←	1Mx8	Byte-Wide Socket =1-Mbyte memory = Atmel, AT27C080
000000 – 0FFFFF	1024k	User files
PROM – B	512kx8	Byte-Wide Socket = 512-kbyte memory = 27F040
000000 – 07FFFF	512k	User files
Setup EEPROM	256	2-kbit EEPROM = Motorola, 93LC56X
080 – 0FF	128	**Documentation is unclear**
040 – 07F	64	512 bits for OEM use
000 - 040	64	Setup info
Setup NVSRAM	128	Real-time clock w/ 114x8 NVSRAM = Benchmarq BQ3285S
00E – 07F	114	Setup storage resisters
000 – 00D	14	Clock & control status resisters
		Requires battery backup
Setup internal RAM	128	Internal real-time clock in M6117 w/ 114x8 CMOS RAM
00E – 07F	114	Setup storage resisters
000 – 00D	14	Clock & control status resisters
		Requires battery backup & documentation unclear

← indicates option installed in the computational block

The following memory maps were made with the 1-Mbyte PROM installed on a 3SXi board with two 2-Mbyte DRAM ICs installed. The system found and booted ROM-DOS from the PROM.

ROM-DOS MEM/B output of 3SXi running ROM-DOS version 6.22 with 4-Mbyte DRAM

```
ROM-DOS MEM v6.22 (Revision 3.00.1)
Copyright (c) 1989-1999 Datalight, Inc.

Bios Extension at C000:0000  32K  (32,768 Bytes)
Bios Extension at D000:0000   2K  (2,048 Bytes)
```

ROM-DOS MEM/C output of 3SXi running ROM-DOS version 6.22 with 4-Mbyte DRAM

```
ROM-DOS MEM v6.22 (Revision 3.00.1)
Copyright (c) 1989-1999 Datalight, Inc.

Module     Address          Size
---------------------------------------
HIMEM      0394:0      3K   (3,824 bytes)
COMMAND    05FC:0      0K     (256 bytes)
COMMAND    060D:0      3K   (3,744 bytes)
COMMAND    06F8:0      0K     (240 bytes)
MEM_RD     0708:0      0K      (96 bytes)
MEM_RD     070F:0      8K   (8,816 bytes)

Memory Type         Total   =   Used   +   Free
-----------------  -------   -------   -------
Conventional          626K       28K       598K
Upper                   0K        0K         0K
Adapter RAM/ROM       398K      398K         0K
Extended (XMS)      3,072K       64K     3,008K
-----------------  -------   -------   -------
Total memory        4,096K      490K     3,606K

Total under 1 MB      626K       28K       598K

Largest executable program size       598K (612,096 bytes)
Largest free upper memory block         0K       (0 bytes)
ROM-DOS is resident in the high memory area.
```

ROM-DOS MEM/R output of 3SXi running ROM-DOS version 6.22 with 4-Mbyte DRAM

```
ROM-DOS MEM v6.22 (Revision 3.00.1)
Copyright (c) 1989-1999 Datalight, Inc.

Address     Sig  Owner    Size      Name
   0303:0000    M    0008     02F8      SD
   05FC:0000    M    060E     0010      COMMAND
   060D:0000    M    060E     00EA      COMMAND
   06F8:0000    M    060E     000F      COMMAND
   0708:0000    M    0710     0006      MEM_RD
   070F:0000    M    0710     0227      MEM_RD
   0937:0000    Z    0000     9348      Free
```

The following memory maps were made without the 1-Mbyte PROM installed on a 3SXi board with only one 2-Mbyte DRAM IC installed. The system found and booted ROM-DOS from the floppy disk.

ROM-DOS MEM/B output of 3SXi running ROM-DOS version 6.22 with 2-Mbyte DRAM

```
ROM-DOS MEM v6.22 (Revision 3.00.1)
Copyright (c) 1989-1999 Datalight, Inc.

Bios Extension at C000:0000  32K  (32,768 Bytes)
```

ROM-DOS MEM/C output of 3SXi running ROM-DOS version 6.22 with 2-Mbyte DRAM

```
ROM-DOS MEM v6.22 (Revision 3.00.1)
Copyright (c) 1989-1999 Datalight, Inc.

Module    Address         Size
----------------------------------------
HIMEM     0396:0     3K   (3,824 bytes)
COMMAND   05FE:0     0K     (256 bytes)
COMMAND   060F:0     3K   (3,744 bytes)
COMMAND   06FA:0     0K     (240 bytes)
MEM_RD    070A:0     0K      (96 bytes)
MEM_RD    0711:0     8K   (8,816 bytes)

Memory Type         Total  =   Used  +   Free
-----------------  -------   -------   -------
Conventional          639K       28K      611K
Upper                   0K        0K        0K
Adapter RAM/ROM       385K      385K        0K
Extended (XMS)      1,024K       64K      960K
-----------------  -------   -------   -------
Total memory        2,048K      477K    1,571K

Total under 1 MB      639K       28K      611K

Largest executable program size       611K (625,376 bytes)
Largest free upper memory block         0K       (0 bytes)
ROM-DOS is resident in the high memory area.
```

ROM-DOS MEM/R output of 3SXi running ROM-DOS version 6.22 with 2-Mbyte DRAM

```
ROM-DOS MEM v6.22 (Revision 3.00.1)
Copyright (c) 1989-1999 Datalight, Inc.

Address     Sig  Owner   Size     Name
   0303:0000    M   0008    02FA     SD
   05FE:0000    M   0610    0010     COMMAND
   060F:0000    M   0610    00EA     COMMAND
   06FA:0000    M   0610    000F     COMMAND
   070A:0000    M   0712    0006     MEM_RD
   0711:0000    M   0712    0227     MEM_RD
   0939:0000    Z   0000    9686     Free
```

6.1.8 Interrupt Vector Locations and Values

The 256 interrupt vectors are located in the first 1024 bytes [000-3FF] of memory and point to interrupt-service routines (ISR) using a 4-byte far address. The interrupt vectors contain the SEGMENT:OFFSET address that execution jumps to following the corresponding interrupt. These vectors are stored as FAR-pointers, where the 16-bit OFFSET value is stored first in memory followed by the 16-bit SEGMENT value. These 16-bit words are stored with the least significant byte (LSB) stored first in memory followed by the most significant byte (MSB)[1].

These interrupt vectors control program flow whenever a hardware (IRQ) or software (INT) interrupt occurs by providing the address for the corresponding ISR, to which execution jumps. The last instruction the ISR is the interrupt-return instruction, IRET, which pops off the stack a far pointer to the location following the last instruction executed before the interrupt and jumps there. Thus, normal program execution resumes with the next instruction to be processed in the absence of the interrupt. Since interrupts unpredictably alter program flow from that determined by line-by-line examination, special attention is warranted. This section provides an illustrative list of interrupt-vector values from the computational block as an example only.

The following output is from a stand-alone compiled-version of a Quick Basic program, GETVECT.BAS. Since programs often temporarily capture and redirect some interrupts, one has to clearly understand these values correspond to the values when GETVECT.EXE rather than DATA_ATT.EXE is running. The table contains:

- The vector number in both decimal and hex
- The address of the location of the first byte of the 4-byte vector in the standard SEGMENT:OFFSET format
- The jump address is listed as found in the interrupt vector using the standard SEGMENT:OFFSET format
- The jump address is also converted into an offset relative to segment 0
- A brief description of the cause associated with that interrupt vector[2] [3].

Note that the IRQ0-7 interrupts overlap some processor interrupts due to historical choices.

The raw memory dump command, MEM /R, shows that system programs reside below 05FC:0000 (05FC0h). Many interrupts are associated with segment 0089 and point into this low-memory area. COMMAND.COM resides between 05FC:0000 (5FC0h) and 0708:000 (07080h). The user area (MEM_RD or GETVECT) resides above 0708:000 (07080h) in the example memory map. This list contains several interrupt vectors captured and redirected into the user program (GETVECT.EXE), which point into the user memory area. A few interrupt vectors point into the video BIOS located between C0000 and CFFFF hex. Some interrupt vectors point into the BIOS image located between F0000 and FFFFF hex.

A listing of GETVECT.BAS is also provided.

---

[1] Michael Tisher and Bruno Jennrich, PC Intern, Abacus, page 18, ISBN 1-55755-304, 1996
[2] Michael Tisher and Bruno Jennrich, PC Intern, Abacus, page 21-23, ISBN 1-55755-304, 1996
[3] Frank van Gilluwe, The Undocumented PC, Addison-Wesley, page 282-284, ISBN 0-201-47950, 1997

Table. Interrupt Vector Details [see PC Intern 6th Ed page 21-23]

Vector#	Location	Jump Addr	Abs Addr	Cause of Interrupt
0 00	0:000	232C:0C50	0:023F10	CPU: Divide error
1 01	0:004	0070:0465	0:000B65	CPU: Single Step
2 02	0:008	3B13:2462	0:03D592	CPU: Non-maskable interrupt
3 03	0:00C	0070:0465	0:000B65	CPU: Break Point
4 04	0:010	232C:0C54	0:023F14	CPU: Numeric Overflow
5 05	0:014	F000:FF54	0:0FFF54	BIOS: Print Screen
6 06	0:018	F000:B2D0	0:0FB2D0	CPU: Invalid Opcode
7 07	0:01C	F000:B3FE	0:0FB3FE	CPU: Coprocessor not avail
8 08	0:020	0593:0000	0:005930	IRQ0: Timer Tick / CPU: Double exception
9 09	0:024	232C:7151	0:02A411	IRQ1: Keyboard / CPU: Segment overflow
10 0A	0:028	F000:B3FE	0:0FB3FE	IRQ2: Cascade / CPU: Invalid TSS
11 0B	0:02C	F000:B3FE	0:0FB3FE	IRQ3: COM2 / CPU: Segment absent
12 0C	0:030	F000:B3FE	0:0FB3FE	IRQ4: COM1 / CPU: Stack Exception
13 0D	0:034	F000:B3FE	0:0FB3FE	IRQ5: 2nd LPT / CPU: Protection Fault
14 0E	0:038	DE50:009A	0:0DE59A	IRQ6: Floppy / CPU: Page Fault
15 0F	0:03C	0070:0465	0:000B65	IRQ7: LPT1
16 10	0:040	4731:084D	0:047B5D	BIOS: Video functions
17 11	0:044	F000:F84D	0:0FF84D	BIOS: Determine configuration
18 12	0:048	F000:F841	0:0FF841	BIOS: Determine RAM size
19 13	0:04C	FDB0:24A2	0:0FFFA2	BIOS: Disk drive functions
20 14	0:050	F000:E739	0:0FE739	BIOS: Access to serial port
21 15	0:054	0291:053A	0:002E4A	BIOS: Cassettes / System services
22 16	0:058	0070:042D	0:000B2D	BIOS: Keyboard inquiry
23 17	0:05C	D85E:0A28	0:0D9008	BIOS: Access to printer port
24 18	0:060	F000:6760	0:0F6760	Call ROM BASIC / Boot Failure
25 19	0:064	047C:002F	0:0047EF	BIOS: Boot system
26 1A	0:068	F000:FE6E	0:0FFE6E	BIOS: Prompt date/time
27 1B	0:06C	D85E:0604	0:0D8BE4	BIOS: Keyboard Ctrl-Break key pressed
28 1C	0:070	0593:001D	0:00594D	BIOS: Called after Int 08 (timer tick)
29 1D	0:074	F000:F0A4	0:0FF0A4	BIOS: Addr of video parameter table
30 1E	0:078	0000:0522	0:000522	BIOS: Addr of diskette parameter table
31 1F	0:07C	C000:21ED	0:0C21ED	BIOS: Addr of character bit pattern
32 20	0:080	00C9:0FA8	0:001C38	DOS: Program terminate
33 21	0:084	04AD:042F	0:004EFF	DOS: Function dispatcher
34 22	0:088	FE06:1F42	0:0FFFA2	DOS: Addr of DOS quit program routine
35 23	0:08C	3B13:244A	0:03D57A	DOS: Addr of DOS Ctrl-Break routine
36 24	0:090	232C:7B0E	0:02ADCE	DOS: Addr of DOS Error routine
37 25	0:094	00C9:0FBC	0:001C4C	DOS: Absolute disk read
38 26	0:098	00C9:0FC6	0:001C56	DOS: Absolute disk write
39 27	0:09C	00C9:0FD0	0:001C60	DOS: Terminate and Stay Resident
40 28	0:0A0	00C9:106C	0:001CFC	DOS: DOS is unoccupied or idle
41 29	0:0A4	0070:0466	0:000B66	DOS: Display character
42 2A	0:0A8	D85E:05B4	0:0D8B94	DOS: NETBIOS
43 2B	0:0AC	00C9:106C	0:001CFC	DOS: Reserved
44 2C	0:0B0	00C9:106C	0:001CFC	DOS: Reserved
45 2D	0:0B4	00C9:106C	0:001CFC	DOS: Reserved

Table. Interrupt Vector Details [see PC Intern 6th Ed page 21-23]

Vector#	Location	Jump Addr	Abs Addr	Cause of Interrupt
46 2E	0:0B8	092D:0162	0:009432	DOS: Command execute
47 2F	0:0BC	0B31:0FF0	0:00C300	DOS: Multiplexer
48 30	0:0C0	C90F:E4EA	0:0D75DA	DOS: Reserved
49 31	0:0C4	F000:B300	0:0FB300	DOS: Reserved
50 32	0:0C8	00C9:106C	0:001CFC	DOS: Reserved
51 33	0:0CC	058D:0001	0:0058D1	User: Mouse driver functions
52 34	0:0D0	3B13:0717	0:03B847	DOS: Reserved
53 35	0:0D4	3B13:0717	0:03B847	DOS: Reserved
54 36	0:0D8	3B13:0717	0:03B847	DOS: Reserved
55 37	0:0DC	3B13:0717	0:03B847	DOS: Reserved
56 38	0:0E0	3B13:0717	0:03B847	DOS: Reserved
57 39	0:0E4	3B13:0717	0:03B847	DOS: Reserved
58 3A	0:0E8	3B13:0717	0:03B847	DOS: Reserved
59 3B	0:0EC	3B13:0717	0:03B847	DOS: Reserved
60 3C	0:0F0	3B13:06CD	0:03B7FD	DOS: Reserved
61 3D	0:0F4	3B13:06FA	0:03B82A	DOS: Reserved
62 3E	0:0F8	00C9:106C	0:001CFC	DOS: Reserved
63 3F	0:0FC	232C:1FE9	0:0252A9	DOS: Reserved
64 40	0:100	F000:EC59	0:0FEC59	BIOS: Revectored diskette services
65 41	0:104	F000:E761	0:0FE761	BIOS: Fixed disk parameter table #1
66 42	0:108	F000:F065	0:0FF065	Reserved
67 43	0:10C	C000:2600	0:0C2600	Reserved
68 44	0:110	F000:B3FE	0:0FB3FE	Reserved
69 45	0:114	F000:B3FE	0:0FB3FE	Reserved
70 46	0:118	F000:E401	0:0FE401	BIOS: Fixed disk parameter table #2
71 47	0:11C	F000:B3FE	0:0FB3FE	Free: Available to programs
72 48	0:120	F000:B3FE	0:0FB3FE	Free: Available to programs
73 49	0:124	F000:B3FE	0:0FB3FE	Free: Available to programs
74 4A	0:128	F000:B3FE	0:0FB3FE	BIOS: Alarm occurred
75 4B	0:12C	FD96:2642	0:0FFFA2	Free: Available to programs
76 4C	0:130	F000:B3FE	0:0FB3FE	Free: Available to programs
77 4D	0:134	F000:B3FE	0:0FB3FE	Free: Available to programs
78 4E	0:138	F000:B3FE	0:0FB3FE	Free: Available to programs
79 4F	0:13C	0070:04FC	0:000BFC	Free: Available to programs
80 50	0:140	F000:B3FE	0:0FB3FE	Free: Available to programs
81 51	0:144	F000:B3FE	0:0FB3FE	Free: Available to programs
82 52	0:148	F000:B3FE	0:0FB3FE	Free: Available to programs
83 53	0:14C	F000:B3FE	0:0FB3FE	Free: Available to programs
84 54	0:150	F000:B3FE	0:0FB3FE	Free: Available to programs
85 55	0:154	F000:B3FE	0:0FB3FE	Free: Available to programs
86 56	0:158	F000:B3FE	0:0FB3FE	Free: Available to programs
87 57	0:15C	F000:B3FE	0:0FB3FE	Free: Available to programs
88 58	0:160	F000:B3FE	0:0FB3FE	Free: Available to programs
89 59	0:164	F000:B3FE	0:0FB3FE	Free: Available to programs
90 5A	0:168	F000:B3FE	0:0FB3FE	Free: Available to programs

Table. Interrupt Vector Details [see PC Intern 6th Ed page 21-23]

Vector#	Location	Jump Addr	Abs Addr	Cause of Interrupt
91 5B	0:16C	F000:B3FE	0:0FB3FE	Free: Available to programs
92 5C	0:170	05BC:0001	0:005BC1	NETBIOS functions
93 5D	0:174	F000:B3FE	0:0FB3FE	Free: Available to programs
94 5E	0:178	F000:B3FE	0:0FB3FE	Free: Available to programs
95 5F	0:17C	F000:B3FE	0:0FB3FE	Free: Available to programs
96 60	0:180	0000:0000	0:000000	Free: Available to programs
97 61	0:184	0000:0000	0:000000	Free: Available to programs
98 62	0:188	0000:0000	0:000000	Free: Available to programs
99 63	0:18C	0000:0000	0:000000	Free: Available to programs
100 64	0:190	0000:0000	0:000000	Free: Available to programs
101 65	0:194	0000:0000	0:000000	Free: Available to programs
102 66	0:198	0000:0000	0:000000	Free: Available to programs
103 67	0:19C	058E:0040	0:005920	User: EMS memory manager functions
104 68	0:1A0	F000:B3FE	0:0FB3FE	Free: Available to programs
105 69	0:1A4	F000:B3FE	0:0FB3FE	Free: Available to programs
106 6A	0:1A8	F000:B3FE	0:0FB3FE	Free: Available to programs
107 6B	0:1AC	F000:B3FE	0:0FB3FE	Free: Available to programs
108 6C	0:1B0	F000:B3FE	0:0FB3FE	Free: Available to programs
109 6D	0:1B4	C000:13B9	0:0C13B9	Free: Available to programs
110 6E	0:1B8	F000:B3FE	0:0FB3FE	Free: Available to programs
111 6F	0:1BC	F000:B3FE	0:0FB3FE	Free: Available to programs
112 70	0:1C0	DE50:0035	0:0DE535	IRQ8: Real-time clock
113 71	0:1C4	F000:B2A1	0:0FB2A1	IRQ9: Video
114 72	0:1C8	F000:B3FE	0:0FB3FE	IRQ10: COM4
115 73	0:1CC	F000:B3FE	0:0FB3FE	IRQ11: COM3
116 74	0:1D0	DE50:00E2	0:0DE5E2	IRQ12: Resvd: Mouse port
117 75	0:1D4	F000:B292	0:0FB292	IRQ13: Resvd: Math Coprocessor Error
118 76	0:1D8	DE50:00FA	0:0DE5FA	IRQ14: IDE Disk Controller
119 77	0:1DC	DE50:0112	0:0DE612	IRQ15:
120 78	0:1E0	0000:0000	0:000000	Reserved
121 79	0:1E4	0000:0000	0:000000	Reserved
122 7A	0:1E8	0000:0000	0:000000	Reserved
123 7B	0:1EC	0000:0000	0:000000	Reserved
124 7C	0:1F0	0000:0000	0:000000	Reserved
125 7D	0:1F4	0000:0000	0:000000	Reserved
126 7E	0:1F8	0000:0000	0:000000	Reserved
127 7F	0:1FC	0000:0000	0:000000	Reserved
128 80	0:200	0000:0000	0:000000	Basic interpreter
129 81	0:204	0000:0000	0:000000	Basic interpreter
130 82	0:208	0000:0000	0:000000	Basic interpreter
131 83	0:20C	0000:0000	0:000000	Basic interpreter
132 84	0:210	0000:0000	0:000000	Basic interpreter
133 85	0:214	0000:0000	0:000000	Basic interpreter
134 86	0:218	0000:0000	0:000000	Basic interpreter
135 87	0:21C	0000:0000	0:000000	Basic interpreter

Table. Interrupt Vector Details [see PC Intern 6th Ed page 21-23]

Vector#	Location	Jump Addr	Abs Addr	Cause of Interrupt
136 88	0:220	0000:0000	0:000000	Basic interpreter
137 89	0:224	0000:0000	0:000000	Basic interpreter
138 8A	0:228	0000:0000	0:000000	Basic interpreter
139 8B	0:22C	0000:0000	0:000000	Basic interpreter
140 8C	0:230	0000:0000	0:000000	Basic interpreter
141 8D	0:234	0000:0000	0:000000	Basic interpreter
142 8E	0:238	0000:0000	0:000000	Basic interpreter
143 8F	0:23C	0000:0000	0:000000	Basic interpreter
144 90	0:240	0000:0000	0:000000	Basic interpreter
145 91	0:244	0000:0000	0:000000	Basic interpreter
146 92	0:248	0000:0000	0:000000	Basic interpreter
147 93	0:24C	0000:0000	0:000000	Basic interpreter
148 94	0:250	0000:0000	0:000000	Basic interpreter
149 95	0:254	0000:0000	0:000000	Basic interpreter
150 96	0:258	0000:0000	0:000000	Basic interpreter
151 97	0:25C	0000:0000	0:000000	Basic interpreter
152 98	0:260	0000:0000	0:000000	Basic interpreter
153 99	0:264	0000:0000	0:000000	Basic interpreter
154 9A	0:268	0000:0000	0:000000	Basic interpreter
155 9B	0:26C	0000:0000	0:000000	Basic interpreter
156 9C	0:270	0000:0000	0:000000	Basic interpreter
157 9D	0:274	0000:0000	0:000000	Basic interpreter
158 9E	0:278	0000:0000	0:000000	Basic interpreter
159 9F	0:27C	0000:0000	0:000000	Basic interpreter
160 A0	0:280	0000:0000	0:000000	Basic interpreter
161 A1	0:284	0000:0000	0:000000	Basic interpreter
162 A2	0:288	0000:0000	0:000000	Basic interpreter
163 A3	0:28C	0000:0000	0:000000	Basic interpreter
164 A4	0:290	0000:0000	0:000000	Basic interpreter
165 A5	0:294	0000:0000	0:000000	Basic interpreter
166 A6	0:298	0000:0000	0:000000	Basic interpreter
167 A7	0:29C	0000:0000	0:000000	Basic interpreter
168 A8	0:2A0	0000:0000	0:000000	Basic interpreter
169 A9	0:2A4	0000:0000	0:000000	Basic interpreter
170 AA	0:2A8	0000:0000	0:000000	Basic interpreter
171 AB	0:2AC	0000:0000	0:000000	Basic interpreter
172 AC	0:2B0	0000:0000	0:000000	Basic interpreter
173 AD	0:2B4	0000:0000	0:000000	Basic interpreter
174 AE	0:2B8	0000:0000	0:000000	Basic interpreter
175 AF	0:2BC	0000:0000	0:000000	Basic interpreter
176 B0	0:2C0	0000:0000	0:000000	Basic interpreter
177 B1	0:2C4	0000:0000	0:000000	Basic interpreter
178 B2	0:2C8	0000:0000	0:000000	Basic interpreter
179 B3	0:2CC	0000:0000	0:000000	Basic interpreter
180 B4	0:2D0	0000:0000	0:000000	Basic interpreter

Table. Interrupt Vector Details [see PC Intern 6th Ed page 21-23]

Vector#	Location	Jump Addr	Abs Addr	Cause of Interrupt
181 B5	0:2D4	0000:0000	0:000000	Basic interpreter
182 B6	0:2D8	0000:0000	0:000000	Basic interpreter
183 B7	0:2DC	0000:0000	0:000000	Basic interpreter
184 B8	0:2E0	0000:0000	0:000000	Basic interpreter
185 B9	0:2E4	0000:0000	0:000000	Basic interpreter
186 BA	0:2E8	0000:0000	0:000000	Basic interpreter
187 BB	0:2EC	0000:0000	0:000000	Basic interpreter
188 BC	0:2F0	0000:0000	0:000000	Basic interpreter
189 BD	0:2F4	0000:0000	0:000000	Basic interpreter
190 BE	0:2F8	0000:0000	0:000000	Basic interpreter
191 BF	0:2FC	0000:0000	0:000000	Basic interpreter
192 C0	0:300	0000:0000	0:000000	BIOS: User hard disk 0 data
193 C1	0:304	0000:0000	0:000000	BIOS: User hard disk 0 data
194 C2	0:308	0000:0000	0:000000	BIOS: User hard disk 0 data
195 C3	0:30C	0000:0000	0:000000	BIOS: User hard disk 0 data
196 C4	0:310	0000:0000	0:000000	BIOS: User hard disk 1 data
197 C5	0:314	0000:0000	0:000000	BIOS: User hard disk 1 data
198 C6	0:318	0000:0000	0:000000	BIOS: User hard disk 1 data
199 C7	0:31C	0000:0000	0:000000	BIOS: User hard disk 1 data
200 C8	0:320	0000:0000	0:000000	Basic interpreter
201 C9	0:324	0000:0000	0:000000	Basic interpreter
202 CA	0:328	0000:0000	0:000000	Basic interpreter
203 CB	0:32C	0000:0000	0:000000	Basic interpreter
204 CC	0:330	0000:0000	0:000000	Basic interpreter
205 CD	0:334	0000:0000	0:000000	Basic interpreter
206 CE	0:338	0000:0000	0:000000	Basic interpreter
207 CF	0:33C	0000:0000	0:000000	Basic interpreter
208 D0	0:340	0000:0000	0:000000	Basic interpreter
209 D1	0:344	0000:0000	0:000000	Basic interpreter
210 D2	0:348	0000:0000	0:000000	Basic interpreter
211 D3	0:34C	0000:0000	0:000000	Basic interpreter
212 D4	0:350	0000:0000	0:000000	Basic interpreter
213 D5	0:354	0000:0000	0:000000	Basic interpreter
214 D6	0:358	0000:0000	0:000000	Basic interpreter
215 D7	0:35C	0000:0000	0:000000	Basic interpreter
216 D8	0:360	0000:0000	0:000000	Basic interpreter
217 D9	0:364	0000:0000	0:000000	Basic interpreter
218 DA	0:368	0000:0000	0:000000	Basic interpreter
219 DB	0:36C	0000:0000	0:000000	Basic interpreter
220 DC	0:370	0000:0000	0:000000	Basic interpreter

Table. Interrupt Vector Details [see PC Intern 6th Ed page 21-23]

Vector#	Location	Jump Addr	Abs Addr	Cause of Interrupt
221 DD	0:374	0000:0000	0:000000	Basic interpreter
222 DE	0:378	0000:0000	0:000000	Basic interpreter
223 DF	0:37C	0000:0000	0:000000	Basic interpreter
224 E0	0:380	0000:0000	0:000000	Basic interpreter
225 E1	0:384	0000:0000	0:000000	Basic interpreter
226 E2	0:388	0000:0000	0:000000	Basic interpreter
227 E3	0:38C	0000:0000	0:000000	Basic interpreter
228 E4	0:390	0000:0000	0:000000	Basic interpreter
229 E5	0:394	0000:0000	0:000000	Basic interpreter
230 E6	0:398	0000:0000	0:000000	Basic interpreter
231 E7	0:39C	0000:0000	0:000000	Basic interpreter
232 E8	0:3A0	0000:0000	0:000000	Basic interpreter
233 E9	0:3A4	0000:0000	0:000000	Basic interpreter
234 EA	0:3A8	0000:0000	0:000000	Basic interpreter
235 EB	0:3AC	0000:0000	0:000000	Basic interpreter
236 EC	0:3B0	0000:0000	0:000000	Basic interpreter
237 ED	0:3B4	0000:0000	0:000000	Basic interpreter
238 EE	0:3B8	0000:0000	0:000000	Basic interpreter
239 EF	0:3BC	DE50:0028	0:0DE528	Basic interpreter
240 F0	0:3C0	0593:0000	0:005930	Basic interpreter
241 F1	0:3C4	0000:0000	0:000000	Reserved
242 F2	0:3C8	0000:0000	0:000000	Reserved
243 F3	0:3CC	0000:0000	0:000000	Reserved
244 F4	0:3D0	0000:0000	0:000000	Reserved
245 F5	0:3D4	0000:0000	0:000000	Reserved
246 F6	0:3D8	0000:0000	0:000000	Reserved
247 F7	0:3DC	0000:0000	0:000000	Reserved
248 F8	0:3E0	0000:0000	0:000000	Reserved
249 F9	0:3E4	0000:0000	0:000000	Reserved
250 FA	0:3E8	0000:0000	0:000000	Reserved
251 FB	0:3EC	0000:0000	0:000000	Reserved
252 FC	0:3F0	0000:0000	0:000000	Reserved
253 FD	0:3F4	0000:0000	0:000000	Reserved
254 FE	0:3F8	64AF:0000	0:064AF0	Reserved
255 FF	0:3FC	0202:F000	0:011020	Reserved

```
' Quick BASIC program to get the 256 interrupt vectors into a file

OPEN "Ivector.txt" for OUTPUT as #1
OPEN "Vector.lst" for INPUT as #2          ' List of I-vector causes
Cflg = 0                                   ' 0 Cause of vector in C$
Zero$ = "000000"                 ' Zeros to adjust HEX output strings
Hd$ = "Vector#  Location  Jump Addr  Abs Addr  Cause of Interrupt"
print #1, Hd$
Segment& = 0
DEF SEG = Segment&               ' Set SEGMENT = 0 for I-vector table

FOR I = 0 TO 255                 ' Step through all interrupt vectors
      VN$ = Hex$(I)              ' VN$ = Vector number in hex
      M = len(VN$)               ' Adjust to 2 hex digits
      if M < 2 then
            Q$ = left$(Zero$, 2-M)+ VN$
            VN$ = Q$
      endif

      J% = I * 4                 ' Offset of 1st byte of I-vector
                                 ' VL$ = Vector Location in hex
      X$ = Hex$(J%)
      M = len(X$)                ' Adjust to 3 hex digits
      if M < 3 then
            VL$ = "0:" +left$(Zero$, 3-M)+ X$
      else
            VL$ = "0:" +X$
      endif

      FOR K% = 0 TO 3            ' Read out all 4 bytes of the I-vector
            L% = J% + K%         ' Offset of I-vector byte
            I&(K%) = PEEK(L%)
      NEXT K%

            ' Note integers are stored LSB first in memory then MSB
            '           See Qbasic page 545  PEEK command
            ' Note Far-Pointers are stored OFFSET first in memory and
            '           SEGMENT second in memory
            '         See PC Intern page 18
      Qo& = I&(0) + 256 * I&(1)      ' OFFSET of interrupt pointer
      Qs& = I&(2) + 256 * I&(3)      ' SEGMENT of interrupt pointer

      Qs$ = HEX$(Qs&)                ' Fix SEGMENT string
      M = LEN(Qs$)                   ' Adjust to 4 hex digits
      IF M < 4 THEN
            Q$ = LEFT$(Zero$, 4 - M) + Qs$
            Qs$ = Q$
      END IF

      Qo$ = HEX$(Qo&)                ' Fix OFFSET string
      M = LEN(Qo$)                   ' Adjust to 4 hex digits
      IF M < 4 THEN
            Q$ = LEFT$(Zero$, 4 - M) + Qo$
            Qo$ = Q$
      END IF

      JA$ = Qs$ + ":" + Qo$          ' Build Jump address
```

```
            ' AA$ = Convert to absolute address - 16 bytes per segment
        X$ = hex$(16*Qs& +Qo&)          ' Absolute address
        M = len(X$)                     ' Adjust to 6 hex digits
        if M < 6 then
              AA$ = "0:" +left$(Zero$, 6-M)+ X$
        else
              AA$ = "0:" +X$
        endif

              ' VC$ = Cause of the Interrupt from the vector list file
        if Cflg = 0 then
              line input #2, CV$
              CV$ = ltrim$(CV$)
              Cflg = 1                  ' Cause awaits use
        endif
        CVN$ = left$(CV$,2)             ' Vector number
        if CVN$ = VN$ then
              VC$ = mid$(CV$,6,40)
              Cflg = 0                  ' Cause used
        endif

        PRINT      using "###";I; tab(5);
        PRINT      VN$; TAB(10); VL$; TAB(20); JA$; TAB(31); AA$; tab(41);VC$

        PRINT #1, using "###";I; tab(5);
        PRINT #1, VN$; TAB(10); VL$; TAB(20); JA$; TAB(31); AA$; tab(41);VC$
NEXT I

stop
end
```

6.1.9 I/O Information

The flow of information into and out of the CPU is critical to data integrity. The data transfer is often accomplished using subroutines and/or interrupt driven routines, for which source code is often difficult to obtain or understand. Tabled values of the critical I/O data can be very useful in understanding both the state of the computer and the transfer software.

6.1.9.1 IRQ Data

When the hardware initiates an interrupt request, one of the interrupt-request, IRQ, lines are pulled low on the PC104 bus. The following table indicates which devices are assigned to which IRQ lines of the computational block. IRQs are a means of externally altering program flow and thus memory content.

Table Showing the Utilization of Interrupt Request (IRQ) Levels

IRQ	DMA	PORT hex	I vector Location	CARD	FUNCTION
0		040 – 043	0:020	3SXi	ROM BIOS clock tick function from programmable timer 3
1		060 – 064	0:024		Keyboard interrupt
2			0:028		Cascade for IRQ8-15
3		108-10F [2F8 - 2FF]	0:02C	DIO	➔ DIO COM2 = PU300/PU600 [3SXi Serial Port 2 IRQ disabled]
4		100-107 [3F8 – 3FF]	0:030	DIO	➔ DIO COM1 = NMC [3SXi Serial Port 1 IRQ disabled]
5		278 – 27F	0:034		Reserved for: 2nd LPT
6	2	3F0 – 3F7	0:038	3SXi	Floppy controller 8-bit transfers
7		378 – 37F	0:03C	3SXi	LPT1 = Parallel Port (DMA 1 or 3 possible)
8			0:1C0		Reserved for: battery-backed clock alarm
9			0:1C4	SVG-2	Video Controller if Jumper W1 is On
10		118-11F	0:1C8	DIO	➔ DIO COM4 = Symmetry
11		110-117	0:1CC	DIO	➔ DIO COM3 = PU900
12			0:1D0		
13			0:1D4		M6117: Math Coprocessor interface
14		1F0 - 1F7	0:1D8	3SXi	IDE hard disk controller
15			0:1DC		

6.1.9.2 DMA Data

Some I/O devices can directly access the CPU's main memory without the aid of the CPU. This occurs via a direct memory access (DMA) channel. DMAs are a means of externally altering memory content.

Table Showing the Utilization of Direct Memory Access (DMA) Channels

IRQ	DMA	PORT hex	CARD	FUNCTION
	0			Available for 8-bit transfers
	1			Available for 8-bit transfers
6	2	3F0 – 3F7	3SXi	Floppy controller 8-bit transfers
	3			Available for 8-bit transfers
	4			Cascade for DMA channel 0-3
	5			Available for 16-bit transfers
	6			Available for 16-bit transfers
	7			Available for 16-bit transfers

6.1.9.3 Ports Used

Table Showing the Utilization of the I/O Ports [see PC Intern 6th Ed, page 19]

IRQ	DMA	PORT hex	CARD	FUNCTION
		000 – 01F	CPU	DMA controller (8237A)
		020 – 03F	CPU	Interrupt controller (8259A)
		022 ←	3SXi	M6117: index – M6117configuration registers
		023 ←	3SXi	M6117: data – M6117configuration registers
0		040 – 043	3SXi	ROM BIOS clock tick function from programmable timer 3
1		060 – 06F		Keyboard interrupt M6117: Port 61h = speaker signals
		070 – 07F		Real-time clock
		080 – 09F		DMA page register
		0A0 – 0BF		Interrupt controller #2 (8259A)
		0C0 – 0DF		DMA controller #2 (8237A)
		0E0 – 0EF		
		0F0 – 0FF		Reserved for Math coprocessor
4		100 – 07	DIO	→ DIO COM1 = NMC
3		108 – 10F	DIO	→ DIO COM2 = PU300/PU600
11		110 – 117	DIO	→ DIO COM3 = PU900
10		118 – 11F	DIO	→ DIO COM4 = Symmetry
		170 – 177	3SXi	M6117: IDE channel #2 (HDCS0J)
14		1F0 – 1F7	3SXi	Reserved for IDE hard disk controller M6117: IDE channel #1 (HDCS0J)
		200 – 207		Reserved for game port (joysticks)
5		278 – 27F		Reserved for 2nd LPT
[3]		2F8 – 2FF	3SXi	3SXi Serial Port 2 = COM2 [IRQ disabled]
		300	DIO	DIO#0 → Input bits from Input switches
		301	DIO	DIO#1 unused
		302	DIO	DIO#2 unused
		303	DIO	DIO#3 → Output bit for latching into data barrier
		304	DIO	DIO#4 → Output bits to Display lights
		305	DIO	DIO#5 → Output bits to Display lights
		360 – 36F		Reserved for network card
		376	3SXi	M6117: IDE channel #2 (HDCS1J)
7		378 – 37F	3SXi	LPT1 = Parallel Port (DMA 1 or 3 possible)
		3B0 – 3BF		Reserved for monochrome display adapter
		3B4 – 3DA	SVG-2	Standard VGA port addresses
		3D0 – 3DF		Reserved for Color/Graphics adapter
6	2	3F0 – 3F7	3SXi	Floppy controller 8-bit transfers 3F6 = M6117: IDE channel #1 (HDCS1J)
[4]		3F8 – 3FF	3SXi	3SXi Serial Port 1 = COM1 [IRQ disabled]
8				Reserved for battery-backed clock alarm
		46E8	SVG-2	Standard VGA port addresses

The above table may not be a complete listing of port assignments. Possible additions might be found in other documentation. For example, information regarding ports 022 & 023 is found in the M6117C data sheet and is specific to the hardware.

The M6117C has 49 internal configuration registers, which are accessed via port 22 (index) and port 23 (data). These control the operation of the M6117C and are addressed in the data sheet provided in section 2.3.3.

6.1.9.4 Other I/O Parameters

The M6117C has 49 internal configuration registers, which are accessed via port 22 (index) and port 23 (data). These internal configuration registers are very important and determine memory banks, memory shadowing, BIOS access, interrupts, and many other I/O related items. The internal configuration register information is spelled out in the M6117C data sheet pages 24-52. This data sheet is provided in section 2.3.3. There is a register summary table on pages 39-40 of the data sheet. The register bit definitions are detailed on pages 41-52 of the data sheet. Memory addressing is detailed in an MA table on pages 74-76 of the data sheet.

The timing of signals to and from I/O devices is another example of additional I/O parameters necessary for operating the computational block. The following is a white paper on the timing of I/O pulses is provided by Ampro on the Internet at

http://www.ampro.com/university/wpapers/ISAtiming.pdf.

ISA Bus Timing Diagrams

P/N 5001321 Revision A

COMPUTERS, INCORPORATED

4757 Hellyer Avenue, San Jose, CA 95138

Phone: 408 360-0200, FAX: 408 360-0222, Web: www.ampro.com

TRADEMARKS

The Ampro logo is a registered trademark, and Ampro, CoreModule, MiniModule, and CoreModule are trademarks of Ampro Computers, Inc. Pentium is a registered trademark of Intel, Incorporated. All other marks are the property of their respective companies.

NOTICE

No part of this document may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language or computer language, in any form or by any means, electronic, mechanical, magnetic, optical, chemical, manual, or otherwise, without prior written permission from Ampro Computers, Incorporated.

DISCLAIMER

Ampro Computers, Incorporated makes no representations or warranties with respect to the contents of this manual or of the associated Ampro products, and specifically disclaims any implied warranties of merchantability or fitness for any particular purpose. Ampro shall under no circumstances be liable for incidental or consequential damages or related expenses resulting from the use of this product, even if it has been notified of the possibility of such damages. Ampro reserves the right to revise this publication from time to time without obligation to notify any person of such revisions. If errors are found, please contact Ampro at the address listed on the title page of this document.

REVISION HISTORY

Revision	Reason for Change	Date
A	Initial Release	6/98

ISA Bus Timing Diagrams

Ampro's ISA bus timing diagrams are derived from diagrams in the IEEE P996 draft specification which were, in turn, derived from the timing of the original IBM AT computer.

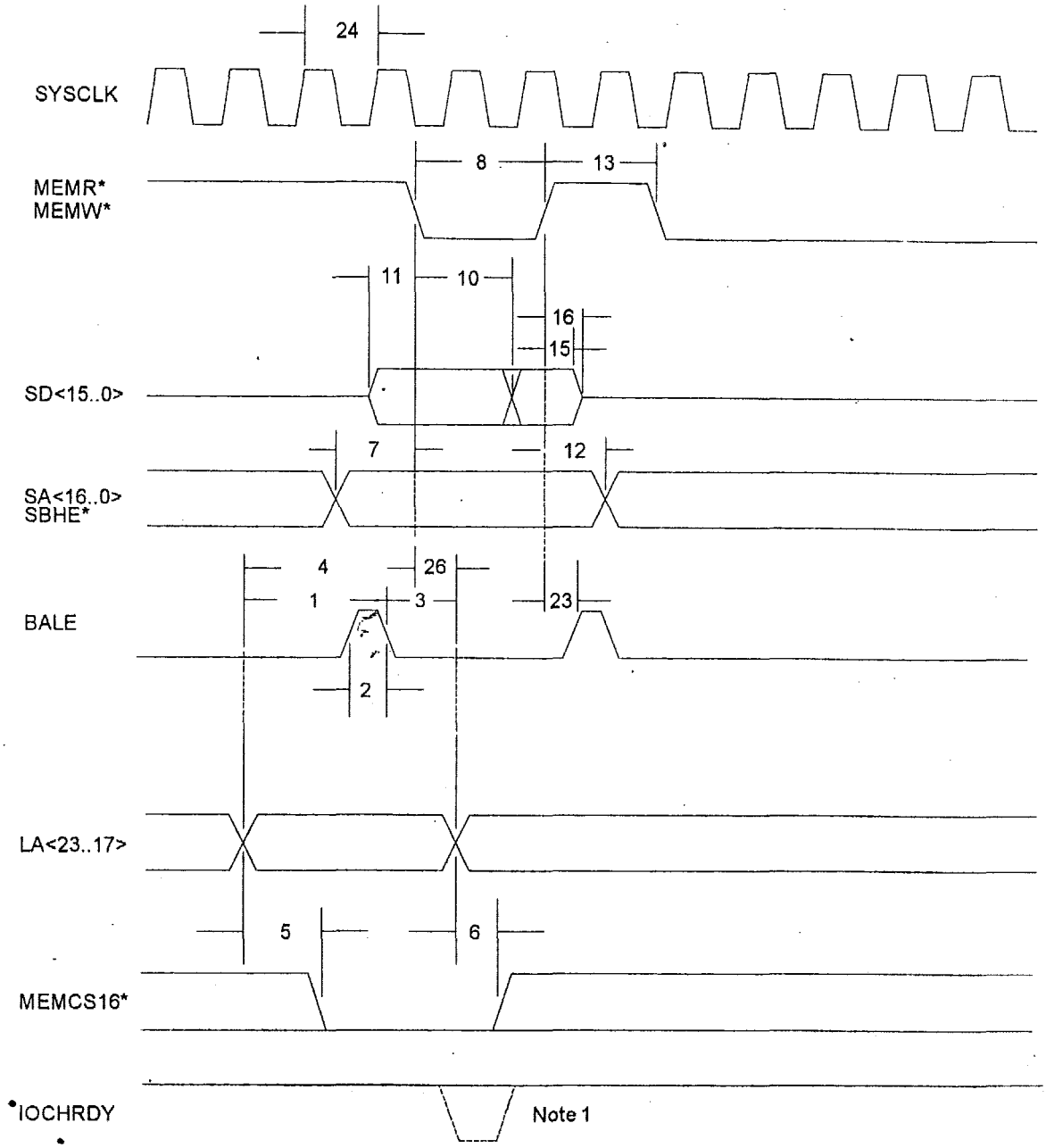
Please note that the IEEE P996 draft specification was never completed by the IEEE and is not an IEEE approved spec. Also, the "latest" IEEE draft is known to contain errors. In the absence of an approved IEEE specification, manufacturers of PC chip sets attempt to meet a "consensus" ISA bus standard. This has resulted in minor variations in signal interpretation and timing among the various PC chipset vendors. For this reason, Ampro recommends that designers of interfaces to the ISA bus use the minimum number of bus signals needed to perform a required function (e.g. chip selection or signal synchronization). For example, at least one popular chipset does not drive AEN high during REFRESH. In certain instances, Ampro has added logic to improve bus timing and/or signal relationships on CPU and peripheral boards.

Ampro's ISA bus timing diagrams include several corrections relative to the IEEE P996 draft specification. However, since these diagrams are derived from an uncompleted and unapproved IEEE specification, they may contain other errors.

For comprehensive technical details on the ISA architecture and bus, Ampro recommends the following book: *ISA & EISA Theory and Operation*, by Edward Solari; published by Annabooks (www.annabooks.com). This book contains a detailed technical exposition of the ISA and EISA buses and is written by the principal author of the IEEE P996 draft specification.

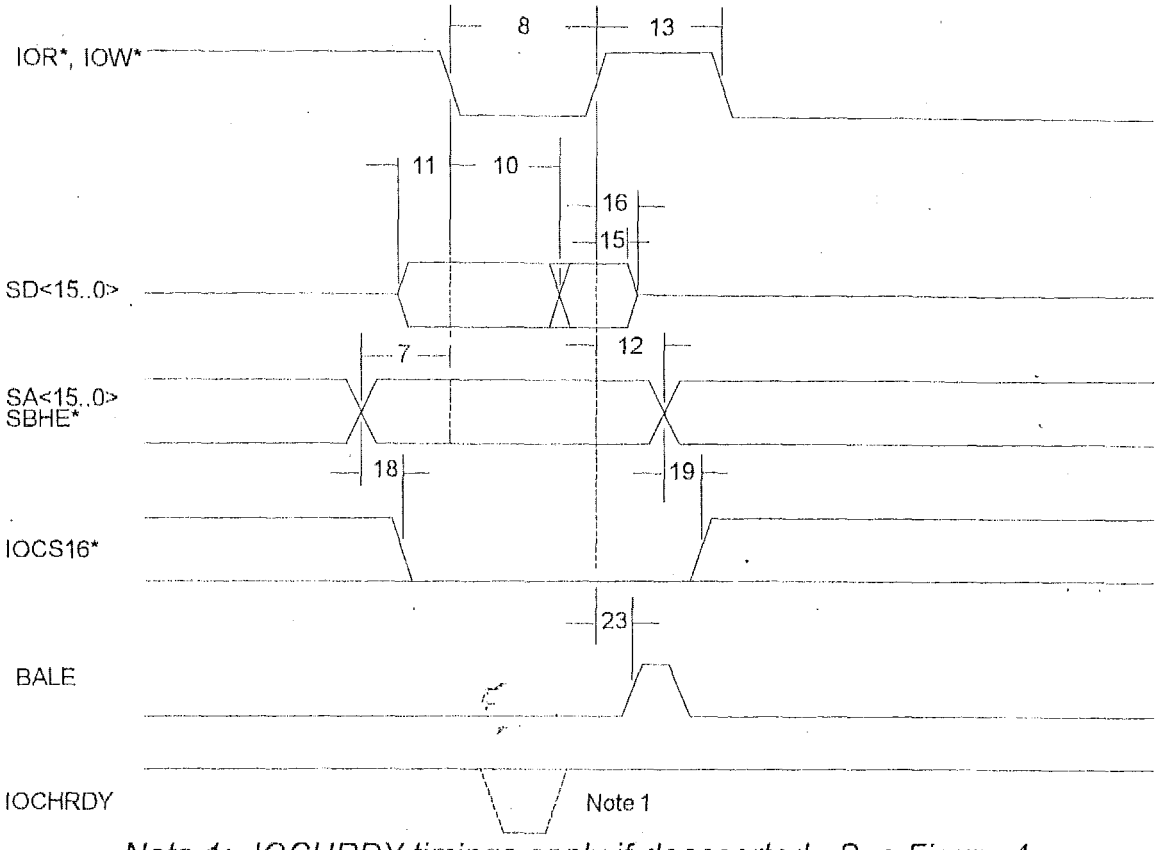
REF	TYPE	SIZE	DESCRIPTION	DRIVER		RECEIVER	
				MIN	MAX	MIN	MAX
1	M,IO	8/16	LA setup to BALE deasserted	111		100	
2	M,IO	8/16	BALE pulse width	61		50	
3	M,IO	8/16	LA hold from BALE deasserted	26		15	
4a	M	16	LA setup to MEMx* asserted	120		109	
4b	M	8	LA setup to MEMx* asserted	183		172	
5	M	8/16	MEMCS16* valid from LA		66		102
6	M	8/16	MEMCS16* hold from LA	0		0	
7a	M	16	SA, SBHE* setup to MEMx*	39		28	
7b	IO	16	SA, SBHE* setup to IOx*	102		91	
7c	M,IO	8	SA, SBHE* setup to IOx* or MEMx*	102		91	
8a	M	16	Command width	240		219	
8b	IO	16	Command width	165		154	
8c	M	16	Command width with ENDXFR* asserted	103		92	
8d	M,IO	8	Command width	541		530	
10a	M	16	Read data access		173		195
10b	IO	16	Read data access		110		132
10c	M,IO	16	Read data access with ENDXFR* asserted		48		70
10d	M,IO	8	Read data access		482		504
11a	M	16	Write data setup	-34		-45	
11b	IO	16	Write data setup	33		22	
11c	M,IO	8	Write data setup (even)	7		-4	
11d	M,IO	8	Write data setup (odd)	-45		-56	
12	M,IO	8/16	SA, SBHE* hold	53		42	
13a	M	16	Command deasserted	108		97	
13b	M	8	Command deasserted	170		159	
13c	IO	8/16	Command deasserted	170		159	
15a	M,IO	8/16	Read data hold	0		0	
15b	M,IO	8/16	Write data hold	25		25	
16	M,IO	8/16	Read command to SD disabled		30		30
17	M	16	ENDXFR* asserted from command		10		32
18	IO	8/16	IOCS16* asserted from SA		74		122
19	IO	8/16	IOCS16* hold from SA	0		0	
20a	M,IO	8/16	IOCHRDY valid from command asserted		70		159
20b	M,IO	8	IOCHRDY valid from command asserted		373		462
21	M,IO	8/16	IOCHRDY deasserted pulse width	125	15600	125	15611
22	M,IO	8/16	Command hold from IOCHRDY	125			
23	M,IO	8/16	BALE asserted from command deasserted	46		35	
24	M,IO	8/16	Clock period (Tclk)	120	167	120	167
25a	M,IO	8/16	Data setup to IOCHRDY deasserted (8-bit even)		85		74
25b	M,IO	8	Data setup to IOCHRDY deasserted (8-bit odd)		75		64
26a	M	16	LA hold to MEMx* active	41		30	
26b	M	8	LA hold to MEMx* active	-21		-32	
28	M	16	ENDXFR* setup to SYSCLK falling edge	22			
29	M	16	ENDXFR* hold from SYSCLK falling edge	22			
36	M	16	LA setup to ENDXFR* asserted		180		158
37	M	16	SA setup to ENDXFR* asserted		83		61

Table 1. Memory and I/O Timing



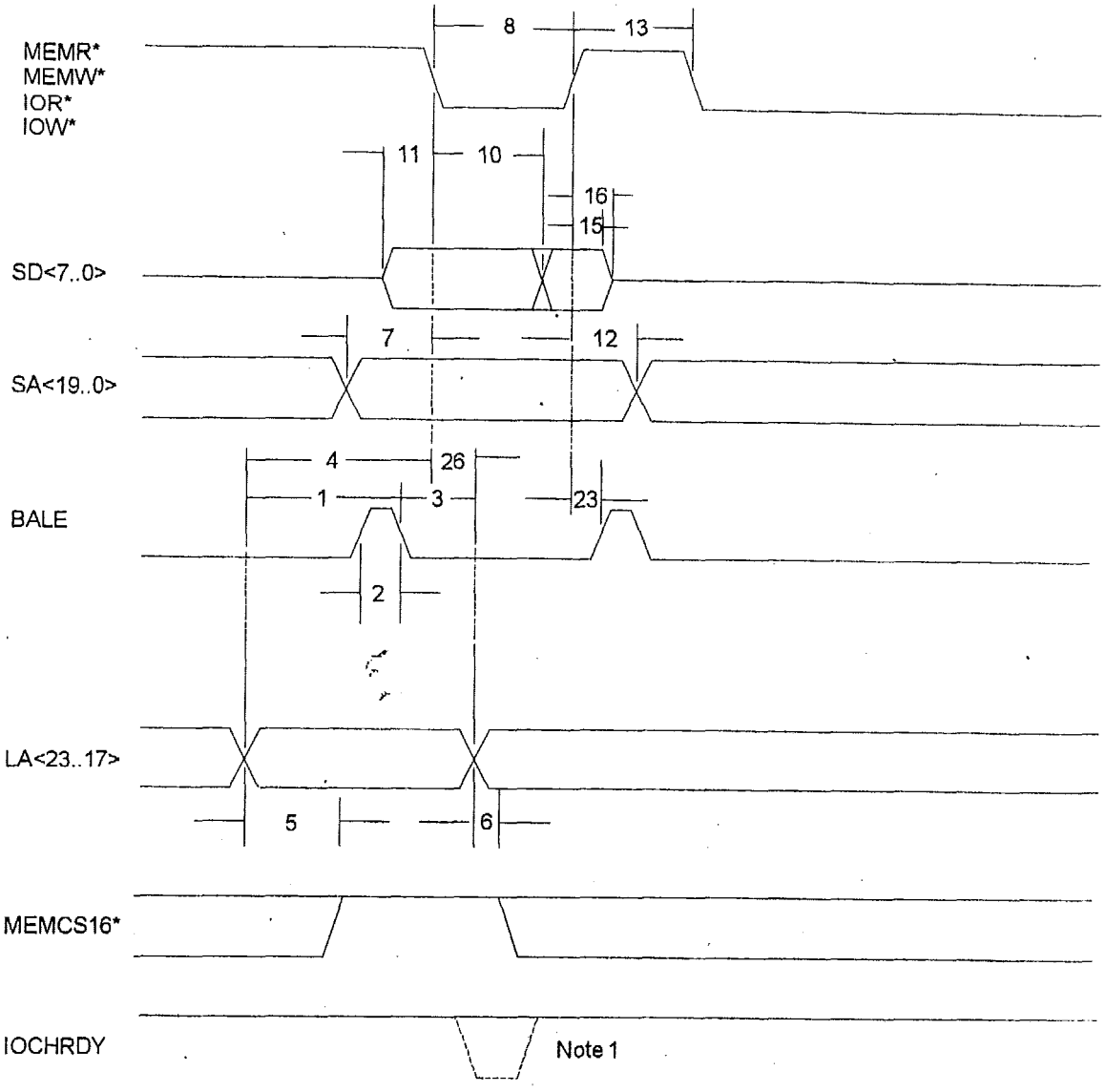
Note 1: IOCHRDY timings apply if deasserted. See Figure 4.

Figure 1. 16-bit Memory Timing



Note 1: IOCHRDY timings apply if deasserted. See Figure 4.

Figure 2. 16-bit I/O Timing



Note 1: IOCHRDY timings apply if deasserted. See Figure 4.

Figure 3. 8-bit Memory and I/O Timing

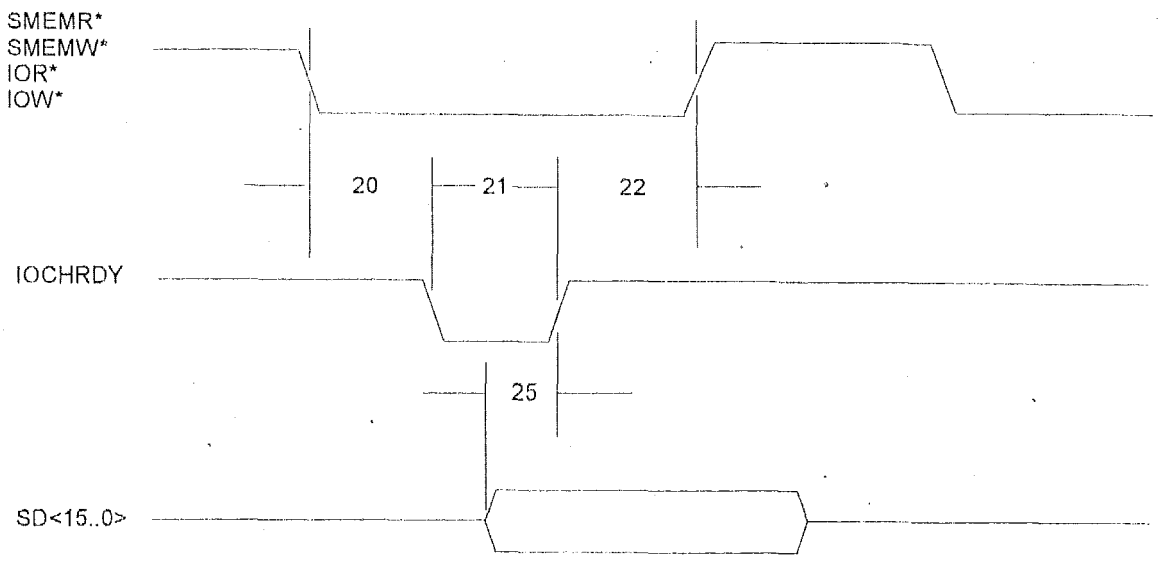
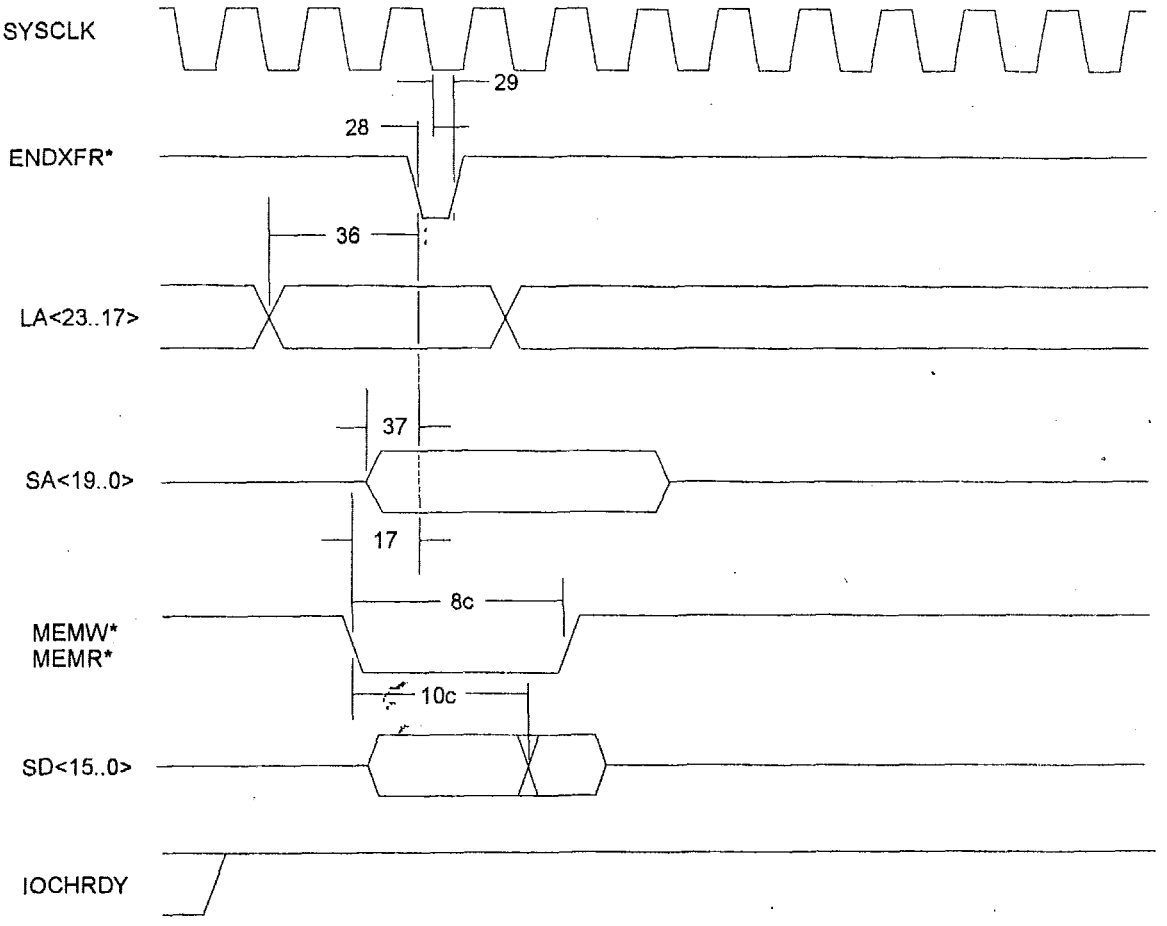


Figure 4: IOCHRDY Timing

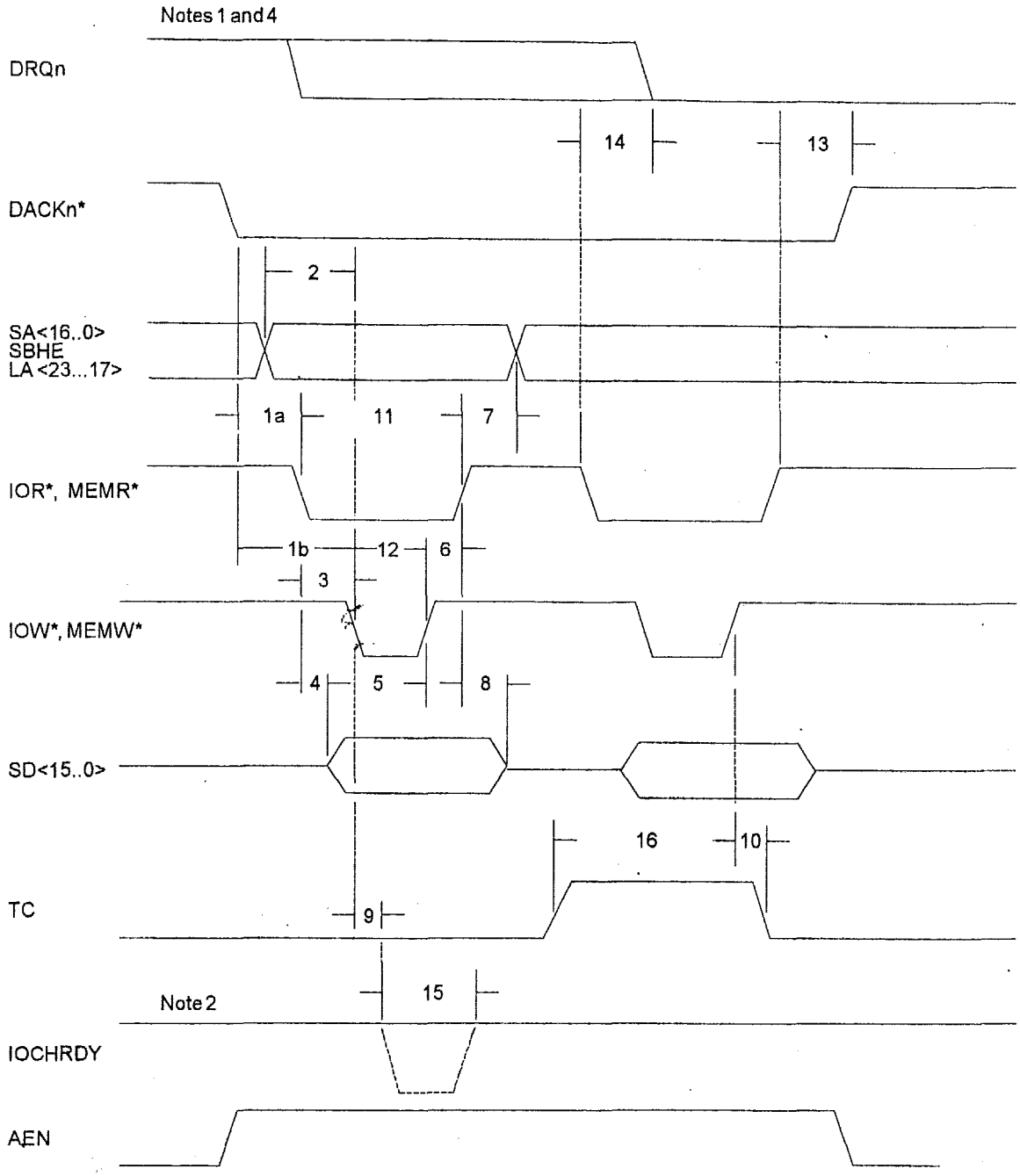


Note 1: Assertion of ENDXFR* within the maximum time from command is only required for a 16-bit cycle with zero wait states. Otherwise, ENDXFR* may be asserted at any time during the cycle while command is asserted.

Figure 5. ENDXFR Timing*

REF	DESCRIPTION	DRIVER		RECEIVER	
		MIN	MAX	MIN	MAX
1a	DACKn*, AEN setup to IOR*	76		65	
1b	DACKn*, AEN setup to IORW*	321		310	
2	Address setup to MEMW*, IOW*	102		91	
3a	IOR* setup to MEMW*	246		234	
3b	MEMR* setup to IOW*	0		0	
4a	Data access from IOR* 8/16bit		220		242
4b	Data access from MEMR* 16bit		173		195
4c	Data access from MEMR* 8bit		332		337
5	Data setup to IOW* unasserted	164		142	
6	Read command hold from write command	50		39	
7	SBHE*, address hold	53		42	
8	Data hold from read command	11		0	
9a	IOCHRDY deasserted from 16bit memory command		81 384		103 406
9b	IOCHRDY deasserted from 8bit memory command				
10	TC hold from command unasserted	60		49	
11a	IOR* pulse width	797		786	
11b	MEMR* pulse width	547		536	
12	IOW*, MEMW* width	500		489	
13a	DACKn* hold from IOW*	114		103	
13b	DACKn* hold from IOW*	173		162	
13c	AEN hold from command	41		30	
14	DREQ inactive from IOx*		119		141
15	IOCHRDY low width	Tclk	15600	Tclk	15611
16	TC setup to command unasserted	511		500	

Table 2. DMA Timing



Note 1: DRQn may be deasserted any time after DACKn* during a block mode DMA transfer.

Note 2: IOCHRDY may be deasserted to insert additional wait states. Additional bus wait states are added in units of two bus clocks.

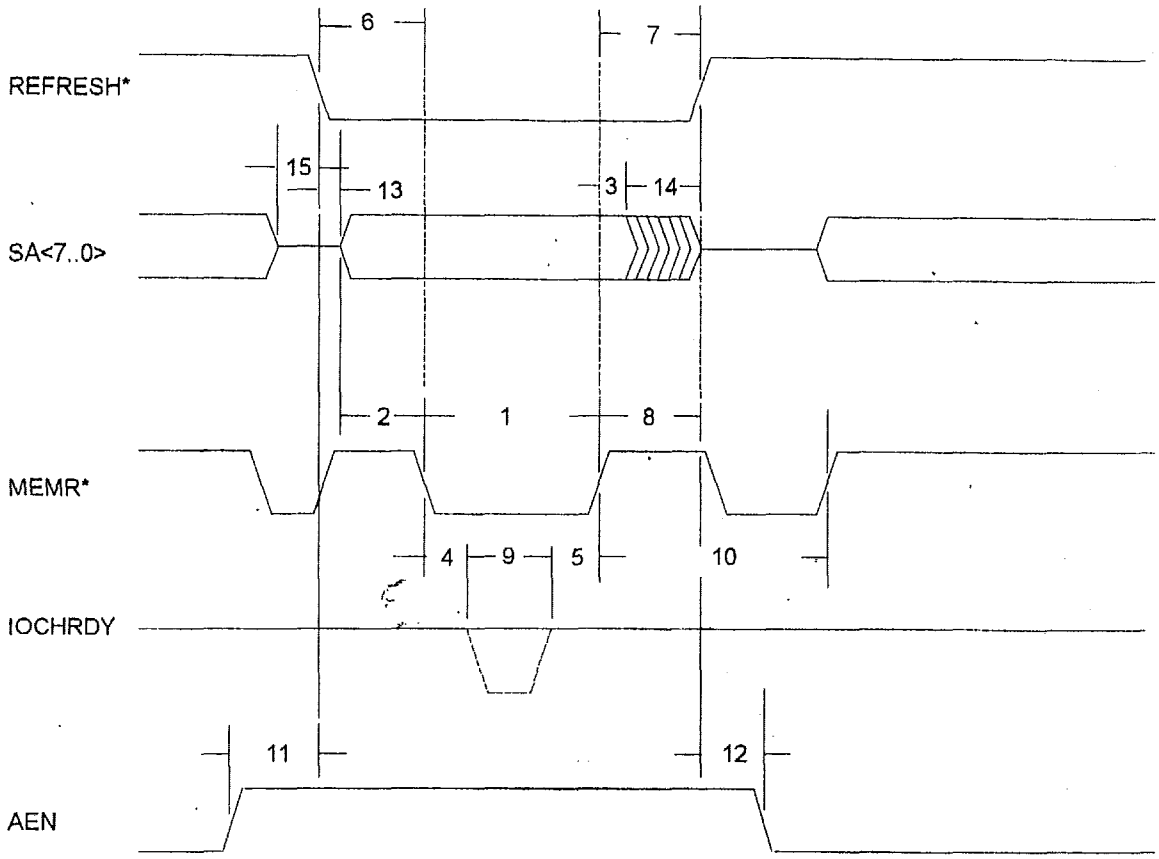
Note 3: The DMA controller activates TC during the last cycle of a DMA request.

Note 4: DMA transfers may be broken up into multiple back-to-back cycles where the DMA controller removes DACKn* and optionally releases the bus to allow higher priority cycles to occur. In this case, DACKn* will be temporarily deasserted even though DRQn is still asserted.

Figure 6. DMA Timing

REF	DESCRIPTION	DRIVER		RECEIVER	
		MIN	MAX	MIN	MAX
1	MEMR* pulse width	214		203	
2	SA<0...7> setup to MEMR*	81		70	
3	SA<0...7> hold from MEMR*	36		25	
4	IOCHRDY deasserted from MEMR*		81		159
5	MEMR* deasserted from IOCHRDY	125		125	
6	REFRESH* setup to MEMR*	125		114	
7	REFRESH* hold from MEMR* (Note 1)	31	250	20	239
8	SA<11...0> tri-state from MEMR* high		Tclk		
9	IOCHRDY width	Tclk		Tclk	
10	AM ownership delay (Note 2)	2*Tclk		2*Tclk	
11	AEN asserted to REFRESH* active	11		0	
12	AEN hold to REFRESH* inactive	11		0	
13	REFRESH* asserted to SA<0...7> valid	11		0	
14	REFRESH* hold from SA<0...7> valid	11		0	
15	Address and Control disabled to REFRESH* asserted	0		0	

Table 3. Refresh Timing



Note 1: The temporary master may exceed the maximum REFRESH* hold time in order to conduct another refresh operation.

Note 2: The temporary master, if the current master, must tri-state the address and command signals prior to driving REFRESH* high (1).

Figure 7. REFRESH Timing

6.1.10 ROM-DOS User's Guide Manual

The ROM-DOS User's Guide is a booklet included in the purchased ROM-DOS Software Development Kit. It describes ROM-DOS commands that a typical user would encounter. The following is a copy for the ROM-DOS User's Guide that was printed from the Datalight Internet site at:

http://www.datalight.com/eval/rd-ug.pdf

Datalight ROM-DOS™ 6.22

User's Guide

Printed: July 1999

Datalight **ROM-DOS**™ User's Guide

Part Number: 0200-0110

Contents

Chapter 1, Introduction

ROM-DOS is a disk operating system that can be loaded in Read Only Memory (ROM) and can run entirely from within ROM and also from a hard or floppy disk, such as in a desktop system. ROM-DOS is functionally equivalent to other brands of DOS and can run programs that are executable under a standard DOS (which executes from RAM). With ROM-DOS, the executable program resides on a disk or is placed in ROM along with ROM-DOS.

Conventions Used in this Manual

This manual uses several notation conventions to denote specific actions and types of information.

- Key combinations, where two or more keys must be pressed simultaneously, are shown with a plus sign. For example, Shift+F1.

- Command line entries and displayed messages are shown in `this font`. For example,

  ```
  DEL MYLETTER.DOC
  ```

- Command line entries indicated by italicized lowercase letters represent information that you supply. For example,

  ```
  DEL <filename>
  ```

  indicates a need to enter the name of the file to be deleted.

- Command line entries enclosed within square brackets represent optional information. For example,

  ```
  PAUSE [message]
  ```

  indicates the *message* portion may be omitted from the PAUSE command.

- Command line entries that include mutually-exclusive options separate those options with a vertical bar (|). In the following example, anything more than the BREAK command must be either the ON or OFF.

  ```
  BREAK [ON|OFF]
  ```

- You must press the Enter key for ROM-DOS to accept your command line data. The command line entries shown in this manual do not show the Enter key.

Terminology Used in this Manual

Computer files are stored either in the computer's internal memory (RAM and ROM) or on magnetic media, typically disks. Regardless of where the files reside, all information is stored in bytes. A byte can store a single character of data. The following terms describe the size of memory, files, and disk space:

- **KB** (kilobyte)—One kilobyte equals 1024 bytes, although the number is often rounded to one thousand bytes.
- **MB** (megabyte)—One megabyte equals 1,048,576 bytes but is usually thought of as one million bytes.
- **GB** (gigabyte)—One gigabyte equals 1,073,741,824 bytes but usually is thought of as one billion bytes.

Random Access Memory (RAM)

RAM can be written to, read from, erased, and rewritten. RAM is your computer's electronic workspace during operation. RAM is also called volatile memory. Its storage ability is temporary, only holding information while the power is on. When the power is turned off or interrupted, everything stored in RAM is lost.

Within limits, you can change the amount of RAM in a system. Typically, embedded systems have 512KB, 640KB, or 1MB of RAM. Greater amounts of RAM are also possible. In desktop systems, 16MB to 128MB are common.

Read Only Memory (ROM)

ROM is more permanent than RAM. Data is programmed into a ROM device before the device is installed in the computer. Information stored in ROM remains intact whether the system power is on or off.

Disks and Disk Drives

Computer disks are classified into two basic groups; rotating media such as floppy disks and hard disks, and memory disks such as those formatted in RAM, ROM and flash memory.

Floppy Disks

A floppy disk is a disk-shaped piece of magnetic material much like audio recording tape. The information is stored in concentric tracks that are subdivided into sectors. Typical storage space on the 3.5-inch disk is 1.44MB.

Hard Drives

Hard drives work much like floppy disks but are fixed in the computer chassis and have a much higher storage capacity. ROM-DOS 6.22 is capable of utilizing hard drives of up to 8GB. Larger drives need to be partitioned into two or more drives.

RAM Disks

Portions of RAM can be made to behave like disk drives, complete with tree-structured directories. When a disk drive is created in RAM, it can be read from and written to in the same manner as the physical disk media. However, when system power is interrupted, all information on the drive is lost, unless the drive is formatted on static RAM.

ROM Disks

Portions of ROM can be made to behave like disk drives, complete with tree-structured directories. ROM drives differ from RAM drives in that they are written to only once. Thereafter, they can only be read from, much like a write-protected floppy disk. Also, ROM disks are non-volatile, in that the information is not lost when power is lost.

Flash Memory Disks

Both PCMCIA cards (PC cards) and on-board (resident) flash arrays can serve as disk drives when used with flash file system software such as Datalight's FlashFX for on-board flash memory. Unlike ROM disks, flash memory disks support both the reading and writing of data.

Chapter 2, Files, Directories, and Command Line Entries

Naming Files

A file is a defined set of related information that your computer stores electronically. A file may be stored on a floppy disk (also called a floppy), on a hard drive, on a CD, or may reside in computer memory (RAM or ROM). To maintain control of interaction between various computer files, each must have its own name that both you and the computer can recognize.

Filenames

Files used in the ROM-DOS environment have two-part names separated by a period. The first part is the filename; the second part is the filename extension. For example, the command interpreter file provided with ROM-DOS is named COMMAND.COM, where COMMAND is the filename and .COM is the filename extension.

Filenames range from one to eight characters in length and consist of any combination of letters, numbers, and the following symbols:

underscore	_	ampersand	&
caret	^	hyphen	-
dollar sign	$	braces	{ }
tilde	~	parenthesis	( )
exclamation point	!	at sign	@
number sign	#	apostrophe	'
percent sign	%		

Filename Extensions

The second part of the filename is the filename extension. The filename extension has one, two, or three characters and may use the same symbols as the filename. A filename extension is not required, although filename extensions can be helpful in identifying the type of file. Commonly used filename extensions include .DOC for documents, .DAT for data, and .TXT for text files.

You may use any filename extension you choose. However, certain filename extensions have a special meaning to ROM-DOS and should only be used when appropriate. These include:

.COM—used for executable files

.EXE—used for executable files

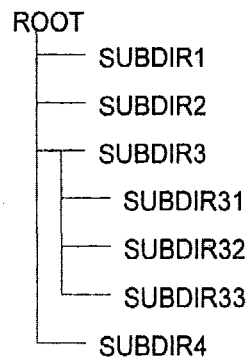
.BAT—used for batch files

Some application programs, such as word processors, may use or require particular filename extensions for output or input files. It is best to follow the application instructions regarding proper filename extensions for that particular program. For example, a file named LETTER1 may include a filename extension such as LETTER1.DOC or LETTER1.TXT.

It is possible to have several files with the same filename but different extensions. ROM-DOS searches for and accesses the filename extensions in the following order: .COM, .EXE, .BAT, and then all others. For example, you could have an executable file named MYPROG.EXE and a batch file call MYPROG.BAT in the current directory. When you enter MYPROG on the command line, the file MYPROG.EXE is executed. If you want to execute the batch file MYPROG.BAT, you must specify the .BAT extension when entering MYPROG on the command line.

Tree-Structured Directory System

ROM-DOS uses a tree-structured directory system. In this system, each branch of the directory, called a subdirectory, is either attached to the main or root directory or is attached to another subdirectory. The following diagram illustrates the directory system and shows two levels of subdirectories under the root directory.



Naming Subdirectories

You can create any subdirectory structure you choose, giving each subdirectory the name of your choice. The naming of subdirectories is similar to the naming of files; there is an eight-character limit and the same character-choice limitations as for filenames (letters, numbers, and symbols). A subdirectory name can also have an extension. For more information on creating and deleting subdirectories, refer to the MKDIR and RMDIR descriptions later in this manual.

Moving around the Directory Tree

When the computer is first turned on, ROM-DOS boots into the root directory. From the root, you can change to any other directory by means of the CD (CHDIR) command. At any given time, ROM-DOS considers you to be in a specific directory, referred to as the current directory.

You can make the computer automatically move to a different directory upon system startup by adding the CD command to your AUTOEXEC.BAT file. Refer to the description of the CHDIR command on page 33 for more information on changing the current directory.

Drive Specifications

Since ROM-DOS can store and retrieve information from more than one disk drive, disk drives are given unique names such as A:, B:, C:, and so on. By convention, floppy disk drives are identified as drive A: and drive B:. On systems having only one physical floppy drive, ROM-DOS can treat the one drive as either A: or B:.

The hard drive, if your system has one, is identified as drive C:. Hard drives can be partitioned (divided) into smaller sections with the FDISK utility. Disks exceeding 8GB must be partitioned into two or more areas, with a maximum size of 8GB per partition. A separate drive letter identifies each hard-drive partition. The first partition is drive C:, the next drive D:, and so on. The highest available drive identifier is the letter Z.

To refer to the C: drive, enter on the command line

```
C:
```

Note: The drive name may be entered in either uppercase or lowercase.

Using Wildcard Characters

To simplify a task performed on a large group of similarly named files, use wildcard characters. Wildcard characters allow you to reference groups of files without entering the complete filename for each file in the group. A wildcard character can substitute all or part of a filename or extension. The two wildcard characters are the asterisk (*) and the question mark (?). The asterisk represents an entire name or a group of characters found within a name. The question mark represents a single character. The following table lists some examples of wildcard character usage.

Example	Description
DIR C:\TEST*.EXE	Lists all files in the TEST directory having the extension .EXE.
DIR D*.*	Lists all files in the current directory that start with the letter D.
COPY C:*.BAT B:*.BAK	Copies all files with a .BAT extension from the C: drive root directory onto the B: drive. The files on the B: drive will have an extension of .BAK. This example backs up a group of files with a single command.
DIR B:\????.*	Lists all the files on the B: drive that have exactly four characters in the filename but have any extension. The question mark substitutes for a single character.
REN TEST?.BAT TEST?.OLD	Renames all files having TEST for the first four characters in the filename, followed by any single character and the .BAT extension. The files retain the same TEST? filename but gain the file extension OLD.

Example	Description
	The ? can also be used to match a single specific character in a filename.

System Prompt

After the execution of each command, ROM-DOS displays the system prompt indicating that it is ready for the next instruction. Unless you define the system prompt otherwise with the PROMPT command, the prompt includes only the current disk drive designation followed by a right angle bracket. For example,

```
A:>
```

One common choice for prompt line configuration is to include the current path in addition to the drive designation. For example,

```
A:\MY_FILES>
```

For more information on configuring the system prompt, refer to the PROMPT command description on page 76.

Command Line

Your keystrokes appear to the right of the system prompt on the command line. You can use the following keys to edit the contents of the command line:

Key	Editing Function
F1	Displays one character at a time from the command line buffer. The right-pointing direction key works in the same way.
F3	Displays entire contents of command buffer.
Ins	Allows insertion of one or more characters in the command line.
Del	Allows deletion of a character from the command line buffer.
Esc	Cancels the current command line and returns you to a new, empty line.
Backspace	Deletes to the left. Allows backing up on the command line.

The last command entered on the command line is stored in a command line buffer. You can recall and edit the contents of this buffer as a way of reentering the command to repeat it or make changes to it. For example, if you intended to enter CHKDSK but accidentally entered CHKDSI, a message appears indicating that CHKDSI is a nonexistent command or filename. Rather than reentering the entire string of characters, press F3 to recall the string and use the Backspace key to back up and make the correction.

You may also make corrections to the beginning of long command line entries. For example, suppose you enter the command

```
CPY DATA1.DAT A:DATA1BAK.DAT
```

where the command is misspelled (COPY is missing the O). To correct the command line, press F1 to display the first character. Then press Ins followed by O.

You can now enter the rest of the command by pressing F3 once.

Redirecting Input and Output

Certain conventions dictate where each ROM-DOS command receives input data and where it sends output data. However, by using the right angle bracket (>) and the left angle bracket (<), you can redirect the input and output.

Input Redirection

The syntax for changing standard input source from keyboard input to file input is

< *filename*

When this is added at the end of the command line, ROM-DOS receives its instructions from the file named *filename*.

Caution: If input redirection is used and the input file is incomplete, the system will hang and refuse to accept information from the keyboard (except for Ctrl+Alt+Del to reboot).

Output Redirection

The syntax for redirecting output to a file is

> *filename*

When this is added to the end of the command line, standard output is temporarily directed to the file named *filename*. If the named file already exists, its contents are replaced with the ROM-DOS function's output. Otherwise, a new file is created to hold the output. Output can also be redirected to a device such as PRN (the printer).

To append output to the receiving file, rather than replace its contents, use >>.

>> *filename*

This adds the ROM-DOS function's output to the existing contents of the named file. If the named file does not exist, ROM-DOS creates the file.

For example, if you want to save a list of the current directory contents to a file, you can redirect the DIR command's output to a file by entering:

```
DIR > MYDIR.TXT
```

where MYDIR.TXT is the name of the file you want to contain the directory listing.

Chapter 3, Using Batch Files

A batch file is a standard text file containing a list of commands that can be submitted to ROM-DOS for automatic sequential execution. Using batch files helps you avoid unnecessary retyping of command sequences that are commonly repeated, complex, or difficult to remember.

The ROM-DOS command processor provides full batch file processing, compatible with standard DOS version 6.22. Batch files can include internal DOS commands, external DOS commands, batch file commands, names of other executable files or programs, or even the names of other batch files.

Batch Filenames

When naming batch files, use the .BAT extension on the filename. This extension tells ROM-DOS to execute the batch file when its name is entered on the command line. The name of the batch file cannot be the name of other internal commands. For example, COPY.BAT is an invalid batch filename.

To execute the batch file, enter the filename on the command line. You need not include the .BAT extension unless a file with the same filename and a .EXE or .COM extension is present in the same directory. Batch file execution begins when you press the Enter key.

Creating a Batch File

You can create a batch file by using any word processor or text editor that saves output as unformatted (ASCII) text. Or you can create them by typing directly from the keyboard into a file. This is done with the command

```
COPY CON filename.BAT
```

This tells ROM-DOS to copy the output from the console (keyboard) to the specified file. Once you have entered the above command, you may enter the contents of your batch file.

At the completion of each line, press Enter. As you enter each line, you can make corrections using Backspace and retyping. If you enter an incorrect line or wish to discontinue without saving your work, press Ctrl+C.

When you have finished entering all the lines in your batch file, press Ctrl+Z and then Enter to complete the file and return to the command line prompt.

Batch File Command Line Parameters

A batch file may use parameters placed on the command line. Insert these parameters as arguments for commands or instructions within the batch file. For example, the following batch file, named ARCHIVE, accepts a command line parameter:

```
PRINT %1
COPY %1 \ARCHIVE\*.*
DEL %1
```

Execute the ARCHIVE batch file by entering

```
ARCHIVE THISFILE.DAT
```

The %1 parameter takes on the name THISFILE.DAT, and the batch file executes the following to make a copy of THISFILE.DAT in the ARCHIVE subdirectory:

```
PRINT THISFILE.DAT
COPY THISFILE.DAT \ARCHIVE\*.*
DEL THISFILE.DAT
```

Note: %1 represents the first parameter in a batch file command. If a command has multiple parameters, they are represented by %2, %3, and so on.

Batch File Commands

In addition to the standard ROM-DOS commands, there are other commands specifically for batch files. Refer to CALL, CHOICE, ECHO, FOR, GOTO, IF, PAUSE, REM, and SHIFT in 'Chapter 5, Command Descriptions.'

For example, you may often run a program named MY_INFO1 followed by a program named MY_INFO2, both of which display a screen of information. After running each of these programs, you always clear the screen before proceeding. Your normal keystroke sequence is:

```
MY_INFO1
CLS
MY_INFO2
CLS
```

You could create a batch file named INFO containing the following commands:

```
MY_INFO1
PAUSE
CLS
MY_INFO2
PAUSE
CLS
```

After executing MY_INFO1 (by entering INFO on the command line), the system pauses. When you press a key, the batch file clears the screen and executes MY_INFO2, then pauses again. Press a key to return to the command line.

Note: You can bypass some or all of the commands in your AUTOEXEC.BAT files during system boot. Refer to 'Bypassing CONFIG.SYS and AUTOEXEC.BAT Commands' on page 16.

Chapter 4, Configuring ROM-DOS

Basic Configuration

Certain standard settings for your system's operation can be stored in a file named CONFIG.SYS. You may create or edit your own CONFIG.SYS file using a word processor or the COPY CON command. (See 'Creating a Batch File' on page 11.)

ROM-DOS offers three levels of CONFIG.SYS processing: DOS 3.31 compatible, DOS 5.0 compatible, and DOS 6.22 compatible. The level of processing available is determined when ROM-DOS is configured.

- DOS 3.31 compatible commands include BREAK, BUFFERS, COUNTRY, DEVICE, FCBS, FILES, LASTDRIVE, NEWFILE, REM, and SHELL.
- DOS 5.0 level processing includes all of the commands available with DOS 3.31 plus DOS, INSTALL, and STACKS.
- DOS 6.22 commands include those from both the DOS 3.31 and 5.0 levels, plus INCLUDE, MENUCOLOR, MENUDEFAULT, MENUITEM, NUMLOCK, SET, SUBMENU, and SWITCHES.

You must place the CONFIG.SYS file in the root directory of the drive that is used for system startup or boot. If a CONFIG.SYS file is not found, the following default values are used for the following commands:

```
BREAK = OFF
BUFFERS = 15
COUNTRY = 001
FCBS = 4
FILES = 8
NUMLOCK = ON
SHELL = COMMAND.COM /P /E:128
STACKS = 0,0
```

Example

A typical CONFIG.SYS file might look like this.

```
BREAK = ON
FILES = 15
BUFFERS = 15
DEVICE = C:\ROMDOS\UTILS\HIMEM.SYS
```

Using Multiple-User Configurations

Your CONFIG.SYS file can be used to define multiple system configurations. This is handy when several people share a computer and require different working environments. It is also useful for booting your own computer using different device drivers, paths, or settings, depending on the intended computer tasks.

To define multiple configurations within the CONFIG.SYS file, you first need to define a startup menu. Each menu item represents a different system configuration option. Then, for each item on the menu, define a configuration block. Each configuration block contains the specific commands to be implemented as the system completes booting.

The menu-item definition and all configuration blocks are marked with a block header. A block header is a descriptive label enclosed in square brackets ([ ]). The start of the menu items must be marked with the block header [MENU]. Each configuration block may have a unique label of your choice. This label can be up to 70 characters long and can contain most printable characters, including spaces, backslashes (\), forward slashes (/), commas (,), semicolons (;), and equal signs (=). Square brackets ([]) cannot be used in block names.

The menu block (or submenu block) may contain only the following commands (A full description for each command can be found in the Command Descriptions section in chapter 6):

- Menuitem
- Menudefault
- Menucolor
- Submenu
- Numlock

Note: Although NumLock may be used outside of a menu/submenu block, it is typically used to enable the keypad for menu-choice selections in the menu block.

A sample menu block might look as follows:

```
[MENU]
menuitem=Research, Research and Development
menuitem=WP, Word Processing
menuitem=Games, Games
menucolor=8,5
menudefault=WP, 10
```

When the system boots, the following menu displays

```
ROM-DOS 6.22 Startup Menu
1. Research and Development
2. Word Processing
3. Games
Enter a choice: 1
```

Each menu item has its own configuration block. Items that are common to all menu choices can be placed in a Configuration Block labeled [COMMON]. All instructions in the common block are carried out along with the specific instructions for any menu item. The [COMMON] block can also be placed at the end of your CONFIG.SYS file so that applications can append commands into this area as the application installs. You may have as many common blocks as you want. The instructions found in the common block(s) are processed in the order they are listed in the CONFIG.SYS file.

When the CONFIG.SYS file is processed by ROM-DOS, it first displays the startup menu that was defined in the [MENU] configuration block, and then waits for your response. The choice

made from the menu determines the configuration block whose commands are to be executed. After the menu selection, processing starts with any instructions in CONFIG.SYS prior to the menu block. Then, instructions in the selected configuration block (including instructions added in via an INCLUDE statement) and all common blocks are processed in the order they are listed in CONFIG.SYS. ROM-DOS ignores the instructions in any nonselected configuration blocks or submenus.

To continue the above example, the configuration blocks might appear as follows:

```
[COMMON]
device=c:\romdos\himem.sys
dos=high
break=on
[RESEARCH]
files=20
buffers=50
device=vdisk.sys 128 /e
[WP]
files=10
buffers=10
lastdrive=m
device=c:\network\loadnet.sys
[GAMES]
include=wp
device=mouse.sys
[COMMON]
```

If choice number 3 is made, selecting [GAMES], the instructions in the [COMMON] configuration block are processed first, followed by the instructions in the [GAMES] configuration block. The [GAMES] section makes use of the INCLUDE command. All of the instructions provided for the WP menu choice also apply to [GAMES]. If any instructions are in the final [COMMON] configuration block, they are processed last.

Extending Menu Items to AUTOEXEC.BAT

The defined name of the menu item you have chosen becomes the value of the environment variable CONFIG. For example, if you choose number 3, GAMES, from the preceding menu, the variable CONFIG is set to GAMES. The CONFIG environment variable can then be used in your AUTOEXEC.BAT file to further customize the startup sequence. This environment variable is referenced by %CONFIG% in your AUTOEXEC.BAT file.

An example of an AUTOEXEC.BAT file that continues the customization process from the preceding MENU may look like this.

```
prompt $p$g
set temp=c:\mystuff\temp
c:\virus\scanit.com
rem Go to section that matches menu
rem choice made in CONFIG.SYS
goto %config%
:RESEARCH
path c:\bin;c:\ROMDOS;c:\ROMDOS\utils;c:\BORLANDC
cd \ROMDOS
rem Skip other sections and move to end
goto end
:WP
```

```
path c:\bin;c:\ROMDOS;c:\wp
wp
rem Skip next section and move to end
goto end
:GAMES
path c:\bin;c:\ROMDOS;c:\gamedir
cd \gamedir
gamelist.bat
goto end
:end
```

Bypassing CONFIG.SYS and AUTOEXEC.BAT Commands

ROM-DOS offers the capability to bypass some or all of the commands in your AUTOEXEC.BAT and CONFIG.SYS files during the boot process. This feature may be useful in tracking system problems that may be related to one or more commands in either of these two files.

To bypass the instructions in both your AUTOEXEC.BAT and CONFIG.SYS files, follow these steps:

1. Turn on, or restart your computer if it is already on, and wait for the following message.
   ```
   Starting ROM-DOS...
   ```
2. As the above message is being displayed, press the F5 key or hold down the SHIFT key to display the following message.
   ```
   ROM-DOS is bypassing your CONFIG.SYS and AUTOEXEC.BAT files.
   ```

Your system then continues the boot process using the basic default configurations. You may notice differences in the way your system behaves. For instance, installable device drivers and memory device drivers are not loaded, and system prompts and path have default values. If the command interpreter COMMAND.COM is not in the root directory, ROM-DOS may not be able to locate it.

Stepping Through CONFIG.SYS and AUTOEXEC.BAT Commands

If you suspect that one or more commands in either the CONFIG.SYS or AUTOEXEC.BAT files are causing problems in your system, you can choose to process or bypass each command as follows:

1. Turn on, or restart your computer if it is already on, and wait for the following message.
   ```
   Starting ROM-DOS...
   ```
2. As the above message is being displayed, press the F8 key to display the first command in the CONFIG.SYS file.
3. For each command, ROM-DOS displays a [Y,N]? prompt. To process the instruction, press Y. To bypass the instruction, press N. ROM-DOS then moves to the next command in the CONFIG.SYS file. To bypass the confirmation prompt for the remaining

instructions in the CONFIG.SYS file and skip the AUTOEXEC.BAT file, you can press the F5 key at any [Y,N]? prompt.

Note: You can prompt for a single CONFIG.SYS command by using the question mark (?) command prior to the equals sign (=) in the command line. For a complete description of each CONFIG.SYS command, refer to the ROM-DOS 6.22 Command Descriptions starting on page 23.

When ROM-DOS finishes all of the commands in the CONFIG.SYS file, it prompts with:

```
Process AUTOEXEC.BAT [Y, N]?
```

Press Y to selectively process the commands in AUTOEXEC.BAT, otherwise press N to bypass all commands in the AUTOEXEC.BAT file.

Environment Variables

A block of system memory is reserved for the definition of certain strings, called environment variables, to be used in the command processor environment. These include the settings you may establish with the PATH and PROMPT commands as well as COMSPEC, which is automatically defined by ROM-DOS at system startup. Environment variables may be defined using the SET command that is explained in the Command Descriptions section in chapter 6.

Configuring ROM-DOS for International Use

You can configure ROM-DOS to conform to local conventions for date, time, and currency formats by giving the COUNTRY= command. You can also use the COUNTRY command to select an international character set (known as a code page) that determines the sort order. Any code page can also be shown on EGA and VGA displays by loading DISPLAY.SYS. You can remap a keyboard to provide support for various languages and layouts by running KEYB.COM.

ROM-DOS uses a country code to identify the country conventions that are to be used. In most cases, the country code is the same as the international long distance telephone dialing code.

A code page is a set of 256 symbols, including letters, digits, punctuation, and graphic characters. The first 128 symbols in a code page are the standard ASCII characters and are identical in all code pages. The last 128 symbols vary depending on the code page. These symbols include the graphic, line-drawing characters, plus many international letters, currency symbols, and other assorted symbols. A number, such as 437 identifies each code page.

A computer display has one hardware code page built into it. Typically, this is code page 437, which is the standard US code page. CGA and monochrome monitors can only display the hardware code page in text mode. EGA and VGA monitors display the hardware code page unless you load special software (like DISPLAY.SYS). Each country supports a default code page and an alternate code page. The following table lists the valid combinations.

Country	Code	Code Page	Alternative Code Page
Australia	061	437	850
Belgium	032	850	437
Brazil	055	850	437
Canadian-French	002	863	850
Czech Republic	042	852	850
Denmark	045	850	437
Finland	358	850	437
France	033	850	437
Germany	049	850	437
Hungary	036	852	850
Italy	039	850	437
Japan	081	932	---
Latin America	003	850	437
Netherlands	031	850	437
Norway	047	850	865
Poland	048	852	850
Portugal	351	850	860
Russia	007	866	437
Spain	034	850	437
Sweden	046	437	850
Switzerland	041	850	437
United Kingdom	044	437	850
United States	001	437	850
Yugoslavia	038	852	850

Changing Country Conventions

The command to instruct ROM-DOS to use German conventions, for example, is

```
COUNTRY=049
```

The COUNTRY= command requires COUNTRY.SYS to be present in the root directory of the boot drive. Setting a country code affects

- Date and time formats
- The symbol used to denote currency

If you only specify a country code, ROM-DOS uses the default code page for that country. You can choose the alternate code page by including it in the COUNTRY= command. This command

tells ROM-DOS to use German conventions for things such as date and time but using code page 437 instead of 850, the default code page.

```
COUNTRY=049,437
```

Setting a system code page affects

- The sort order for alphabetizing
- The rules for converting international letters to uppercase.

The individual application programs determine whether they make use of these conventions. For example, DOS uses the date format for displaying directories and for showing and getting the current date and time. Some programs may choose to ignore the country information and continue to display dates in a specific format.

Displaying Different Code Pages

To display a code page other than the hardware code page, you must load DISPLAY.SYS in CONFIG.SYS. The following command sets the display to show code page 850, assuming both the DISPLAY.SYS driver and the EGA.CPI font file are located in the C:\DOS directory:

```
DEVICE=C:\DOS\DISPLAY.SYS 850 C:\DOS\EGA.CPI
```

The available font files are named EGA.CPI and EGA3.CPI. They are both used for EGA and VGA systems.

If you have an EGA or VGA system, the character fonts are immediately switched to the requested code page. Some characters may look different after you load DISPLAY.SYS because ROM-DOS uses its own font for all 256 characters. For example, your hardware font might use a square-like zero character, but ROM-DOS might use a round zero character. The differences are minor.

Printing Different Code Pages

At this time, ROM-DOS does not support printing code pages other than those stored in the printer hardware.

Changing the Keyboard Layout

To alter the keyboard layout, issue the KEYB command from within DOS. You can do this by running KEYB in AUTOEXEC.BAT or directly from a DOS prompt. Use this following command to switch to a German keyboard layout, for example:

```
KEYB GR
```

Each country has two valid code pages. If you do not specify a code page, the default code page is used. The following table lists the valid combinations.

Country	Country Identifier	Code Page	Alternate Code Page
Australia	au	437	850
Belgium	be	850	437
Brazil	br	850	437
Canadian-French	cf	863	850
Czech Republic	cz	852	850
Denmark	dk	850	865
Finland	su	850	437
France	fr	850	437
Germany	gr	850	437
Hungary	hu	852	850
Italy	it	850	437
Japan	---	932	---
Latin America	la	850	437
Netherlands	nl	850	437
Norway	no	850	865
Poland	pl	852	850
Portugal	po	850	860
Russia	ru	866	437
Spain	sp	850	437
Sweden	sv	437	850
Switzerland	sf	850	437
United Kingdom	uk	437	850
United States	us	437	850
Yugoslavia	yu	852	850

After you have loaded KEYB, your keyboard layout reflects the country you chose. You can switch back to the US keyboard layout at any time by pressing Ctrl+Alt+F1 (Alt+Left-Shift from Russian and Czech Republic keyboards). You can return to the modified keyboard layout by pressing Ctrl+Alt+F2 (Alt+Right-Shift from Russian and Czech Republic keyboards). You can also switch to a completely different layout by running KEYB again and specifying another country identifier.

Appendix F includes diagrams of the different keyboard layouts. The diagrams show native-language keyboards that tend to have different layouts from US keyboards. KEYB does its best to map the available hardware keys to the desired layout. Some symbols may not be available when using a US keyboard and a non-US layout. In the diagrams, symbols appearing in the lower right

corner of a key are activated by pressing the AltGr key along with the desired key. On keyboards without a right AltGr, pressing Ctrl+Alt represents the AltGr key.

Note: The AltGr key is not found on a standard US keyboard.

Some keys are prefix keys that don't generate any symbol by themselves but modify the following keystroke. For example, on most European keyboards, the apostrophe key (') causes the next letter to be accented. To produce an apostrophe alone, press the apostrophe key followed by the space bar. Other keys that may behave as prefixes, depending on the current keyboard layout, are the backward apostrophe (`), tilde (~), and caret (^).

Some keys represent symbols that are not available in all code pages. For example, the German keyboard can produce a capital A with a caret above it. In the default German code page (850), that symbol is represented by the code 182. However, in the alternate German code page (437) there is no such symbol. If you are using the German layout and code page 437, and you try to produce a capital A with a caret above it, you get a caret character followed by an uppercase A (^A).

Note that the keyboard code page could be set *not* to match the display code page. This can lead to confusion, as the keyboard may produce characters that appear on screen as other symbols. Continuing the above example, if you are using the German layout with keyboard code page 850, but your display code page is 437, and you produce an uppercase A with a caret above it, the screen displays a box drawing character.

Configuring Your System: an Example

To completely configure your system, you need to include commands in your CONFIG.SYS and AUTOEXEC.BAT files. The following sample files set up a computer to use German conventions; to use code page 850 for sorting, uppercase conversions, and the display; and to switch the keyboard layout to German. COUNTRY.SYS is assumed to be in the root directory of the boot drive, and DISPLAY.SYS, EGA.CPI, KEYB.COM, and KEYBOARD.SYS are assumed to be in the C:\DOS directory.

CONFIG.SYS

```
BUFFERS=20
FILES=20
COUNTRY=049
DEVICE=C:\DOS\DISPLAY.SYS 850 C:\DOS\EGA.CPI
```

AUTOEXEC.BAT

```
@ECHO OFF
PROMPT $P$G
PATH C:\DOS;C:\UTILITY
KEYB GR
```

Chapter 5, Command Descriptions

Command Summary

Following are brief descriptions of all ROM-DOS commands, including batch file commands.

Command	Description
?	CONFIG.SYS command. It directs ROM-DOS to pause for confirmation before processing a command.
@	Used to suppress the display of a single batch-file command line.
;	Identifies nonexecuting lines. The same as the REM command.
ANSI.SYS	A console device driver that allows you to support Ansi codes on the local screen.
ATA.SYS	A PCMCIA ATA disk device driver.
ATTRIB	Displays or modifies the attributes associated with a file.
BACKUP	Backs up a single directory tree to a floppy drive, hard disk, or network drive.
BREAK	Turns on or off the ability to stop program execution at a non-I/O point.
BUFFERS	Sets the number of internal data buffers.
CALL	Batch file command. Invokes execution of a secondary batch file.
CHDIR (also CD)	Changes the current directory (also CD).
CHKDSK	Checks the integrity of data on a disk. Displays information.
CHOICE	Allows a user to make a processing choice during the execution of a batch file.
CLS	Clears all information from the monitor's screen.
COMMAND	Starts a second DOS command processor.
COPY	Copies files from one storage location to another.
COUNTRY	Designates the country code for displays.
CTTY	Changes the default terminal interacting with ROM-DOS.
DATE	Displays the date from the system's internal calendar. Allows revision.
DEFRAG	Reorganizes fragmented disk files to optimize disk space and system performance.
DEL	Deletes specified files.
DELTREE	Deletes one or more directory trees or individual files.
DEVICE	Installs a device driver into ROM-DOS.
DEVICEHIGH	Loads a device into the upper memory area, if available.
DIR	DIRectory. Lists contents of a specified directory.
DISKCOPY	Copies the contents of one floppy disk to another of the same type.

Command	Description
DISPLAY	Displays international letters and symbols.
DOS	Installs ROM-DOS into High Memory Area (HMA).
ECHO	Batch file command. Turns on or off display of batch execution on the monitor.
EGA/EGA3.CPI	Font data files for use with the International video display driver, DISPLAY.SYS.
EMM386	Enables expanded memory support for capable systems.
ERASE	Erases specified files (same as DEL).
EXIT	Used to exit nested running of ROM-DOS within another program.
FCBS	Specifies the number of File Control Blocks (FCBS) open at one time.
FDISK	Initializes and partitions a hard disk for DOS.
FILES	Sets the maximum number of files that can be open at one time on the system.
FIND	Works as a filter to display only lines that contain a specified string.
FOR	Batch file command. Performs one DOS command on a set of files.
FORMAT	Initializes a disk so that ROM-DOS can access files on that disk.
GOTO	Batch file command. Moves control to a specified line in the batch file.
HELP	Lists all available ROM-DOS commands along with brief descriptions.
HIMEM	Manages extended memory and the high memory area in a 286, 386, PS/2 system.
IF	Batch file command. Performs a command based on a specified condition.
INCLUDE	Allows instructions in one configuration block to be included with instructions in another configuration block.
INSTALL	Loads Terminate and Stay Resident (TSR) programs during CONFIG.SYS processing.
KEYB	Allows altering of the keyboard layout for a different language or nationality.
KEYBOARD/ KEYBRD2.SYS	Keyboard code page data files for use with the International keyboard driver, KEYB.COM.
LABEL	Creates, changes, or deletes a disk volume label.
LASTDRIVE	Sets the maximum number of drives.
LOADHIGH	Loads a program into the upper memory area, if available.
LONGDIR.EXE	Provides a directory listing of files in a single directory including files with long file names.
MCDEX	Enables the use of CD-ROM drives.
MEM	Displays the used and free memory in your system.
MENUCOLOR	Allows setting of text and background colors for the startup menu.
MENUDEFAULT	Sets the default menu-item choice and time-out value for making a selection.
MENUITEM	Specifies an item to be placed on the startup menu display during system boot.

Command	Description
MKDIR	Creates a new subdirectory.
MODE	Modifies the operation of the printer, serial port, and active video display.
MORE	Displays a text file one screen at a time.
MOVE	Moves files and renames files and directories.
NEWFILE	Allows continuation of CONFIG.SYS processing from a new file.
NUMLOCK	Sets the NUMLOCK keyboard key to on or off when your computer starts.
PATH	Displays current command search path(s). A new path line can be specified.
PAUSE	Batch file command. Causes execution to halt until a key is pressed.
POWER	Conserves power on the system that employs an APM BIOS.
PRINT	Prints a list of files, up to ten files.
PROMPT	Resets the appearance of the system prompt line.
REM	A batch file command for identifying non-executing lines.
REMQUIT.EXE	Terminates the REMSERV program via the serial connection.
REN	Renames files.
RESTORE	Restores files previously saved with BACKUP to the hard disk.
RMDIR (also RM)	Deletes a specified subdirectory.
SET	Sets environment variables and command processor strings.
SHARE	Installs the capabilities for file sharing and file locking on your hard disk.
SHELL	Allows selections of a command interpreter other than COMMAND.COM.
SHIFT	Batch file command. Shifts replaceable parameters one position to the left.
SORT	Sorts a text file and displays the output to the standard device.
STACKDEV.SYS	Increases the number of stacks available for IRQ handlers and Int13h.
STACKS	Allows for the use of dynamic data stacks to handle interrupts.
SUBMENU	Defines a menu item that represents a secondary menu.
SUBST	Allows one drive to appear as another drive.
SWITCHES	Allows special CONFIG.SYS file options.
SYS	Transfers the hidden system files to a specified drive.
TIME	Displays current time from the system's internal clock. Also allows revision.
TREE	Displays the path of each directory on a specified drive.
TYPE	Displays the contents of a text file on the monitor.
UMBLINK.SYS	A non-protected mode program that can allow the creation of Upper Memory Blocks using existing RAM areas.
VDISK	Allows the use of memory as a simulated disk driver.
VER	Displays current version of ROM-DOS on the monitor.

Command	Description
VERIFY	Displays the current VERIFY state or sets the VERIFY state to on or off.
VERSION.SYS	Modifies the version number ROM-DOS reports.
VOL	Displays the volume label on a disk.
XCOPY	Copies multiple files and optionally subdirectories.

Command Descriptions

The following pages provide a complete description of each ROM-DOS command, including batch file commands. Each entry includes a description of the command's purpose, command entry syntax, remarks, and examples as appropriate. Each command also has a label to designate whether it is an internal or external command.

- *Internal* commands are part of the command processor program COMMAND.COM. These functions are only available while COMMAND.COM is running.
- *External* commands are stand-alone utility programs. A special notation is also made for those internal commands that are unique to CONFIG.SYS processing. These commands, labeled Internal/CONFIG.SYS, can only be used inside a CONFIG.SYS file.

For on-line help information and syntax descriptions, use the /? option with any command. For example:

```
DIR /?
```

Note: The file COMMAND.HLP must be available in the root directory on the boot drive to access help information for internal commands.

?

Internal Command

The question mark (?) command directs ROM-DOS to pause and ask for confirmation before processing the command. Place it on a command line in the CONFIG.SYS file following the actual command.

Syntax

[command]? = *command_arguments*

Remarks

The *command* can be any of the following standard CONFIG.SYS commands.

BREAK=	BUFFERS=
DEVICE=	FCBS=
DOS=	INSTALL=

FILES= LASTDRIVE=
STACKS= SWITCHES=

command_arguments can be any of the available options defined for the command. Refer to the individual command description for complete instructions.

The question mark (?) should be placed just before the equal sign (=) in the command line.

Example

DEVICE?=VDISK.SYS 64 /E

Causes ROM-DOS to pause and ask for confirmation before installing the VDISK. If Yes (Y) is answered, the installation will continue. If No (N) is answered, the device will not be loaded and processing moves on to the next CONFIG.SYS command line.

@

Internal Command

The @ sign command prevents a single command in a batch file from being echoed to the screen as the batch file is being run. Place the @ sign in front of the command whose display is to be suppressed.

Format

@ [batch file command]

Remarks

The *batch file command* argument can be any executable line in your batch file.

Examples

```
@COPY FILE1.BAT FILE1.SAV
```

Executes the COPY instruction, but the instructions are not echoed to the screen as the batch file runs.

```
@ECHO OFF
```

The ECHO OFF command differs from the @ sign in that it causes all subsequent commands *not* to be displayed on the screen. To prevent the ECHO OFF command from displaying itself, place the @ sign in front of the command.

;

Internal Command

The semicolon (;) command has two purposes: to allow comments in a batch or CONFIG.SYS file, and to temporarily disable a command without physically deleting the command from the file. Refer also to the REM command.

Syntax

; *[any text here]*

Remarks

Use the (;) command to functionally remove a command from the CONFIG.SYS file without actually deleting it from the CONFIG.SYS file.

Examples

```
;C:\BIN\VDISK.SYS 64 /E
```

Prevents the VDISK command from executing until the (;) command is removed.

ANSI.SYS

Installable Device Driver

ANSI.SYS is a console device driver that allows you to support ANSI codes on the local display.

Syntax

Device=ANSI.SYS [options]

Remarks

ANSI.SYS supports standard ANSI escape sequences.

ANSI.SYS writes directly the screen when using text video mode.

Options

The /K option forces use of the extended keyboard BIOS calls which sense F11 and F12.

The /X option lets you redefine the extended keys independently.

The /S option disables the keyboard redefinition feature.

The /T*nn* option indicates that the video mode *nn* is a text mode. By default, modes 0, 1, 2, 3 and 7 are text modes.

Examples

```
DEVICE=ANSI.SYS
```

This example loads ANSI.SYS with default settings.

```
DEVICE=ANSI.SYS /T54 /S
```

Load ANSI.SYS with mode 54h as a video text mode and disable keyboard redefinition.

ATA.SYS

Installable Device Driver

ATA.SYS is a PCMCIA ATA disk device driver.

Syntax

DEVICE = ATA.SYS [/A *xxxx*] [/I*xxxx*]

Remarks

ATA.SYS requires an Intel 82365 or compatible controller. The PCMCIA controller card itself is always mapped to addresses 3E0h and 3E1h. These are fixed addresses and can not be changed. This driver supports only 5-volt ATA cards. Up to two drives can be supported with the driver. Use the Datalight FORMAT command to format an ATA disk, if it is not already formatted.

ATA Cards tested:

- Integral 1841PA 17MB ATA card 660KB/s write, 690KB/s read
- Toshiba TH6SS160201AA 20MB ATA card 900KB/s write, 1200KB/s read
- SanDisk SDCFB 4MB CompactFlash card 300KB/s write, 1000KB/s read
- SanDisk SDP3B 2MB ATA card 275KB/s writen 1000KB/s read

Controllers tested:

- Vadem VG-465
- Intel 82365SL rev A

Options

The /A*xxxx* option sets the memory window that the ATA card gets mapped to for use. This is a 4KB window. The default is C000. Replace *xxxx* with the correct memory segment window for your installation.

The /Ixxxx options sets the I/O address where the ATA card is mapped to by the controller. The default I/O address is 240h. The memory usage for this is 16 bytes starting at the given address.

Examples

```
DEVICE = ATA.SYS
```

Load ATA.SYS with the default memory segment and I/O address settings.

```
DEVICE=ATA.SYS /AD000 /I290
```

Load ATA.SYS using memory segment address D000h and I/O address 290h

ATTRIB

External Command

The ATTRIB command either displays or modifies the attribute of a file.

Syntax

ATTRIB [+ | -][*option*][[*drive*:][*path*][*filename*]

Remarks

The file attributes define the characteristics of a file. They determine if a file may be deleted or modified, or if it is archived. Use the ATTRIB command to manage these file attributes.

Wildcard characters may be used in the ATTRIB *filename*.

The ATTRIB command modifies file attributes if modify commands are given to ATTRIB. The modify commands are

Option	Description
+/-	Add(+) or remove(-) attribute inserted before each option.

Option	Description
A	Archive attribute
C	Clear all attributes
H	Hidden file attribute
R	Read Only attribute
S	System file attribute

If ATTRIB finds no modify commands, then it displays the files in the specified directory along with the filenames and their current attributes.

Examples

```
ATTRIB +r myfile.dat
```

Adds the Read Only attribute to the file `myfile.dat`.

```
ATTRIB -a -r *.dat
```

Removes the Read Only attribute and the Archive attribute of all files with the .DAT extension.

```
ATTRIB *.dat
```

Displays the attributes of all files with the .DAT extension.

BACKUP

External Command

The BACKUP command backs up a single directory tree to a floppy drive, hard disk, or network drive.

Syntax

BACKUP <*srcpath*> <*dstdrive*> [/A] [/H] [/L[:<*logname*>]] [/M] [/S] [/Y] [/?]

Remarks

The complement program, RESTORE, restores a backup set to hard disk. Backup creates one or more backup volumes in the form of DL970507.001, where the year, month, and day compose the name, and the volume number is the file extension.

The <*srcpath*> (source path) can be any legal DOS path, with an optional file mask, such as D:\SOURCE*.C.

The <*dstdrive*> (destination drive) can be any legal DOS drive.

Options

The A option appends the data to an existing backup file.

The /H option backs up hidden files as well as normal files.

The /L option writes a .LOG file in ASCII text form. If no log filename is specified, BACKUP creates a .LOG file in the current directory.

The /M option backs up only files that do **not** have the archive bit set. BACKUP automatically clears the archive bit for each file it backs up. Use Datalight ATTRIB to view/set the archive bit manually.

The /S option backs up the entire tree, not just the one directory.

The /Y option allows BACKUP to operate in batch files with no user input (affirmative is given for all prompts). However, if the backup set requires multiple floppies, BACKUP prompts for all floppies after the first one.

Note: BACKUP operates much faster while using any disk cache program.

Example

The following command backs up all files in the C:\DEV\ROMDOS directory, including subdirectories, to the B: drive.

```
BACKUP C:\DEV\ROMDOS B: /S
```

BREAK

Internal Command

The BREAK command expands the list of operations that can be stopped by pressing Ctrl+C or Ctrl+Break. Alternatively, returns to the default setting of a limited number of break-able operations.

Syntax

BREAK [ON|OFF]

Remarks

In the normal default condition, the BREAK switch is off. In the off mode, the stop commands, Ctrl+C and Ctrl+Break, affect activities that read from or write to the keyboard, the screen, or the printer. ROM-DOS does not look for these stop commands during any other activities.

With the BREAK switch set to ON, ROM-DOS looks for Ctrl+C and Ctrl+Break during activities such as disk reads and writes.

Examples

```
BREAK ON
```

Expands the BREAK list.

```
BREAK OFF
```

Returns to limited BREAK list.

```
BREAK
```

Displays the current BREAK setting.

BUFFERS

CONFIG.SYS Command

ROM-DOS has internal buffers to temporarily hold data read from the disk. Increasing the number of internal buffers speeds system performance.

Syntax

BUFFERS = *number*

Remarks

Each buffer used by ROM-DOS requires 512 bytes of RAM. The BUFFERS command increases or decreases the amount of RAM used by the operating system.

The minimum *number* of buffers is two, and the maximum number is 40. When the number is less than two, the number of BUFFERS is set to two. When the number is larger than 40, then BUFFERS is set to 40. The default number of buffers is calculated using a scale. The ratio is 15 to 640KB. Consequently, a system having 640KB of conventional memory will have 15 buffers. If the calculated number is less than two, then two buffers is used.

Example

```
BUFFERS = 10
```

Causes ROM-DOS to have ten buffers. These ten buffers use 5120 bytes of RAM.

CALL

Batch File, Internal Command

The CALL command invokes execution of a secondary batch file without exiting the primary batch file. When the secondary batch file is done executing, control is returned to the primary batch file.

Syntax

CALL *batchfile* [*batchfile arguments*]

Remarks

Parameters for the secondary batch file may also be included, if appropriate.

Examples

```
CALL BATCH2
```

Executes the batch file BATCH2.BAT.

```
CALL MYBATCH FILEX FILEZ
```

Executes the batch file MYBATCH.BAT. The arguments passed to MYBATCH.BAT are

```
%1 = FILEX
%2 = FILEZ
```

CHDIR (CHange DIRectory)

Internal Command

The CHDIR command changes the current directory.

Syntax

CHDIR [*drive*:][*path*]*subdir*

CD [*drive*:][*path*]*subdir*

Remarks

Subdir is the name of the new current subdirectory. You may use CD in place of CHDIR.

The new directory that is to become the current directory must already exist. Refer to MKDIR for information on the creation of subdirectories.

A series of two periods (..) may be used to indicate a move back to the next-higher or parent directory.

Specifying only the backslash (\) for the *subdir* argument moves you to the root directory of the current drive.

Examples

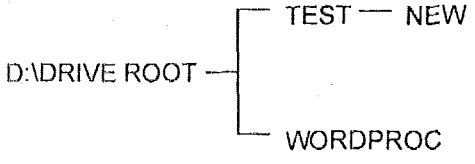
```
CHDIR \TOOLS
```

Moves you into the subdirectory named TOOLS, whose parent directory is the root of the current drive.

```
CD A:
```

Displays the current directory on drive A:. Any valid drive letter can be substituted to get the current directory on that drive.

The following examples use this directory tree structure:




```
CD D:\TEST\NEW
```

Moves you into the subdirectory named NEW, located on the D: drive, under the parent directory TEST.

```
CHDIR..
```

Moves you back to the parent directory of the current subdirectory. If you were in the directory D:\TEST\NEW (from the previous example), this CHDIR command moves you from NEW back into the TEST directory.

```
CD ..\WORDPROC
```

Moves you back to the parent directory and then into a subdirectory named WORDPROC. If you start in the TEST directory, you will move back to the ROOT directory and then into the WORDPROC subdirectory.

```
CD \
```

Moves you back to the root directory from any starting point in the directory tree.

CHKDSK (CHecK DiSK)

External Command

The CHKDSK command checks the disk directories and File Allocation Table (FAT) and displays a disk and memory report.

Syntax

CHKDSK [*drive:*][*path*][*filename*][/C] [/F] [/V]

Remarks

CHKDSK examines a disk and determines whether it has any errors in the File Allocation Table (FAT) and optionally fixes errors.

Options

The /C option allows CHKDSK to correct errors without user confirmation. This option can be used along with the /F option for corrections to be made without user confirmation.

The /F option causes CHKDSK to fix FAT or directory errors on the disk if any are found. If /F is not specified, then CHKDSK acts as if fixing the disk, but no corrections are written to the disk.

If errors are detected, you are prompted with a message similar to the following:

```
15 lost allocation units found in 5 chains.
Convert lost chains to files?
```

If you answer Y for Yes, each lost chain is written to a file in the root directory of the current drive. Each file will have the name FILE*nnnn*.CHK. *nnnn* will be a sequential number. The first chain will be in FILE000.CHK. These files can be verified to see if they contain valuable information, and then deleted if desired. If you answer N for No to the above prompt, CHKDSK still makes the corrections, however, the lost chains are not saved to the disk.

The /V options causes CHKDSK to display each path and file as it is processed.

If a file specification is specified, then CHKDSK displays all files matching the specification that have noncontiguous data areas on the disk. Files that are stored in noncontiguous areas, especially .EXE files, have slower disk access times. If CHKDSK reports a large number of files with this problem, you should use a utility program that optimizes the files and frees space on your disk.

After checking the disk, CHKDSK displays any error messages followed by a report on the state of the disk that was checked. An example of the report is shown below.

```
Volume ROM-DOS created June 1,1999 1:00a
Volume Serial Number is 190E-4AA2
  362496       bytes total disk space
       0       bytes in 1 hidden files
    6144       bytes in 2 user files
  356352       bytes available on disk
  655360       bytes total memory
  595360       bytes free
```

CHKDSK does not wait for a disk to be inserted before the checking is initiated, nor does it repair any errors.

Examples

```
CHKDSK a:
```

Checks the integrity of drive A:. The report is printed to the console.

```
CHKDSK d: >drive_d.rpt
```

Checks the integrity of hard drive disk D. The report is saved in a file named DRIVE_D.RPT.

CHOICE

External Command

The CHOICE command allows a user to select between different options during the processing of a batch file.

Syntax

[*drive:*][*path*]CHOICE [/C[:]*keys*] [/N] [/S] [/T[:]*c,nn*] [*Prompt Text*]

Remarks

/C[:]*keys* specifies the allowed keys for the user prompt. The colon is optional for the command syntax. The default selection for the keys is YN. More than two key choices may be entered.

/N prevents display of the user prompt. The selected keys are still valid, but they are not displayed in the prompt message.

/S selects case-sensitivity regarding the prompt. With the /S switch, you must enter the response in the exact case used with the /C option.

/T[:]*c,nn* sets a time delay. CHOICE pauses for the specified number of seconds waiting for a response. If none is given, the default key choice is used. The argument *c* is the default-key choice character. The *nn* argument specifies the number of seconds to pause. Valid number selections are from 0 to 99. A 0 setting produces no pause.

Prompt Text for the prompt is optional. You can display different output by using the text field, or not, and by using or removing the /N switch along with the text.

Examples

When the following CHOICE command is used

```
CHOICE /c:ync
```

you will see:

```
[Y,N,C]?
```

If the text argument is added

```
CHOICE /c:ync Please select Yes, No, or Continue
```

the display will be:

```
Please select Yes, No, or Continue [Y,N,C]?
```

If the key choice prompt is left off

```
CHOICE /n Continue reading file
```

only the following appears (note key choice of YN is default)

```
Continue reading file?
```

A complete batch file example follows. Each prompt selection returns an errorlevel that can be trapped. The errorlevel corresponds to the order of the key choice. For example, with /c:tco, t returns 1, e returns 2, and o returns 3.

```
@echo off
cls
echo.
echo T Run TIME-IN Program
echo E Run Employee Update
echo O Run TIME-OUT Program
```

```
echo.
choice /teo /t:t,5 Please select option
if errorlevel 3 goto timeout
if errorlevel 2 goto update
if errorlevel 1 goto timein
:timeout
tmout.exe
goto end
:update
updat.exe
goto end
:timein
tmin.exe
goto end
:end
```

CLS (CLear Screen)

Internal Command

The CLS command clears the monitor to display a blank screen.

Syntax

CLS

Remarks

CLS clears the screen, and then redisplays the DOS prompt and cursor in the upper left-hand corner. There are no additional options for CLS.

COMMAND

External Command

The COMMAND command starts a new command processor.

Syntax

COMMAND [/e:*number*] [/k *filename*] [/p] [/c *string*][/msg]

Remarks

Use this command to start a new instance of the ROM-DOS command processor, the program that contains the internal ROM-DOS commands.

Starting a new instance of the command processor also produces a new environment. The size of the environment is 128 bytes by default but can be changed using the /e switch. The environment size may be larger if this is a secondary copy of COMMAND.

Command and its arguments can also be used in a SHELL= statement in your CONFIG.SYS file. Refer to the description of SHELL for more details.

Options

The /e:number switch sets the environment size. *Number* represents the size of the environment in bytes. *Number* must be in the range from 160 to 32,768. All other values are ignored and replaced with the default value of 256. ROM-DOS rounds up the value entered to the nearest multiple of 16.

The /k filename option tells the command processor to run the specified filename and then display the ROM-DOS command prompt. It is not recommended that this option be used in a CONFIG.SYS SHELL= statement.

The /p switch causes COMMAND not to exit but to remain permanent. Use the /p switch only when COMMAND is used in a CONFIG.SYS SHELL statement.

The /c string switch causes COMMAND to execute the command in string and then terminate. The string command can be any internal or external command.

The /msg option causes all error messages to be stored in memory. This option is recommended only for floppy disk-based systems. ROM-DOS keeps many of its error messages in the nonresident portion of COMMAND.COM rather than using valuable memory to store them. If an error message is needed, and you have loaded ROM-DOS from a floppy disk, the message will only be available if the boot disk is still in the drive. By using the /msg option, the message is available in memory at all times. The /p option must be used along with the /msg option.

Examples

```
COMMAND /C DIR C:
```

Causes a new copy of COMMAND to be executed. It performs a DIR command on the C: drive and then exits back to the previous Command Processor.

```
SHELL=C:\COMMAND.COM /P /E:256
```

Loads a permanent copy of COMMAND with an environment size of 256 bytes.

COPY

Internal Command

The COPY command copies a file or set of files to a specified destination: another disk, another subdirectory on the current disk, or on a completely different drive. COPY may also be used to alter the *filename* within the current directory. In addition, this command can be used to direct communication between files and devices (for example, file contents to a printer or keyboard input to a file).

Syntax

This command has several possible formats. The essential structure of each command is

COPY source target option

The *source* is the copy from *filename* or *device*, and the *target* is the copy to *filename* or *device*. Following are various configurations of the COPY command format.

COPY [*drive:*] [*path*] *filename*[*/option*] [*drive:*] [*path*] *filename*[*/option*]

Where the first *filename* indicates the source file(s) to be copied, and the second *filename* indicates the target area on which to copy.

COPY [*drive:*] [*path*] *filename*[*/option*] + [*drive:*] [*path*] *filename*[*/option*]

[*drive:*] [*path*] *filename* [*/option*]

As shown, several source *filenames* may be listed to be copied into the target *filename* that is listed last. The source files are concatenated into the target file.

COPY [*drive:*] [*path*] *filename* [*/option*] *device*

The target *device* is a console or printer (PRN).

COPY *device* [*drive:*] [*path*] *filename* [*/option*]

The *device* is the source such as a keyboard or console, the output of which is directed to the target *filename.*

Options

The /A and /B options represent ASCII and binary, respectively, and act as switches that allow each of these file types to be copied. When /A or /B is used, it applies to the preceding *filename* . The option remains in effect for any *filenames* following in the command line until superseded by another /A or /B.

/A and /B options are only needed when combining ASCII and binary files.

/A treats the file as an ASCII file (text file). When used with the source file, everything is copied up to, but not including, the first CTRL+Z end-of-file marker. When /A is used on the target file, a Ctrl+Z is added as the last character in the file.

/B treats the file as a binary file. When /B is used with the source file, the entire file is copied regardless of any Ctrl+Z characters. When /B is used with the target filename, no Ctrl+Z end-of-file marker is added.

/H copies hidden files.

/V is not implemented in ROM-DOS for code size reasons. It is included to maintain command-line compatibility.

/Y copies the current file(s) over the existing file(s) of the same name(s) without confirmation. This option overrides the setting made by the COPYCMD environment variable.

/-Y confirms the copy of one file over the existing file of the same name. This option overrides the setting made by the COPYCMD environment variable.

Set the COPYCMD environment variable with the SET command. This allows you to set confirmation on or off for the COPY command. If you always want to be prompted for confirmation when a file will copy over an existing file, set COPYCMD= /-Y. To automatically overwrite without confirmation during a copy instruction, set COPYCMD= /Y. For proper usage, refer to the SET command.

Remarks

When no filename is specified for the target, the new copy is given the same name as the source filename.

When no *drive* or *path* is specified for the source, the current drive and directory are assumed. When no *drive* or *path* is specified for the target, the current drive and directory are assumed.

If a drive name only is specified without a *path*, the current directory for that drive is assumed.

Both source and target *filenames* may include wildcard characters (*) and (?) to specify a set of several files.

Examples

```
COPY LETTER.TXT A:
```

Copies the file LETTER.TXT (in your current drive and path) to the current directory on the disk in drive A:.

```
COPY *.DOC A:
```

Copies all files in the current directory with an extension of .DOC to the default directory of drive A:.

```
COPY DATAORIG.DOC DATABACK.DOC
```

Creates a backup copy, DATABACK.DOC, from the file DATAORIG.DOC. The new file is located in the current directory.

```
COPY JAN.DAT + FEB.DAT + MAR.DAT QTR1.DAT
```

Copies the files JAN.DAT, FEB.DAT, and MAR.DAT in sequence into the single file, QTR1.DAT.

```
COPY CON NEWFILE.TXT
```

Sets up your console (keyboard) to input directly to NEWFILE.TXT. Ctrl+Z followed by Enter closes the file and returns to normal command line operation.

COUNTRY

CONFIG.SYS Command

ROM-DOS supports multiple country formats for time, date, and currency, and other basic country-specific information. A country is identified by a three-digit, international telephone country code.

Syntax

COUNTRY = *countrynumber* [*codepage*]

Remarks

The file COUNTRY.SYS must be present in the same directory as your CONFIG.SYS file.

If you do not specify a code page, ROM-DOS uses the default code page for the chosen country. If a code page is specified, it must be either the default or alternate code page for the chosen country.

This command affects the ROM-DOS DATE and TIME commands. Applications that use DOS functions to determine the date, time or currency format, or request that DOS provide character sort order or uppercase information, are affected as well.

Refer to the table provided on page 18 for a list of the currently supported countries.

Examples

```
COUNTRY= 049
COUNTRY = 049, 437
```

The next time you start ROM-DOS with either of these COUNTRY commands, the DATE and TIME will be displayed as follows:

```
DATE
   Current date is Wed 20.06.1998
   Enter new date (dd.mm.yyyy):
TIME
   Current time is 16:39:54,45
   Enter new time:
```

The first COUNTRY command above uses codes page 850, by default, for sorting and case conversion. The second COUNTRY command example uses the specified code page 437 instead.

CTTY (Change TeleTYpe)

Internal Command

The CTTY command directs input and output to a different device other than your computer's standard keyboard and monitor.

Syntax

CTTY *device*

Remarks

Use CTTY for any situation requiring interaction with an alternate console.

The CTTY command only affects communication with ROM-DOS and with programs that work through ROM-DOS for input and output. For example, BASIC uses standard keyboard input regardless of previous CTTY command usage.

Examples

```
CTTY COM2
```

Sets the device on COM2 as the input/output device.

```
CTTY CON
```

Returns control to the standard keyboard.

DATE

Internal Command

The DATE command displays the current date (day, month, year) as known to ROM-DOS and also allows you to change it.

Syntax

DATE [*mm-dd-yyyy*]

Remarks

The date set by this command is used, among other things, for date stamping your file revision dates. This information is displayed when you execute a directory listing of your files.

You may want to include the DATE command in your AUTOEXEC.BAT file, so you can set the date during boot. If your computer has an internal, battery-operated clock, you won't need to do so.

The format of the date command is also dependent on the country specified in CONFIG.SYS. The date is displayed according to local standards for the specified country.

Refer also to the TIME command.

Examples

When entering this command without specifying *mm-dd-yyyy*, the current date as known to ROM-DOS is displayed, and you are prompted to enter a new date.

```
Current date is Sat 6-10-1989
Enter new date (mm-dd-yyyy):
```

If you do not want to change the date, press Enter. Otherwise, key in the current date and press Enter.

Alternatively, you may skip the display and prompting by entering the current date on the command line. To enter June 10, 1999 (assuming US country support), enter the DATE command as follows:

```
DATE 6-10-1999
```

Valid entries for months, days, and years are

mm = 1-12 *dd* = 1-31 *yyyy* = 1980-2099

ROM-DOS calculates the day of the week; do not include it in your entry.

DEFRAG

External Command

The DEFRAG command reorganizes fragmented files on hard and floppy disks to optimize disk space and system performance.

Syntax

DEFRAG [drive:] [[/D | /F | /U] [/S[:]order] [/B]]

Remarks

To defragment files on your disk, quit all other programs prior to running DEFRAG.

Options

Option	Description
[drive:]	Drive letter of the disk to be optimized.
/B	Reboot the system when defragmentation/optimization is completed.
/D	Optimize directories only.
/U	Defragment files and optimize directories (default option).
/S	Sort files by specified order. Use a hyphen (-) suffix to reverse the order. N = by name (alphabetic) E = by extension (alphabetic) D = by date & time (earliest first) S = by size (smallest first) - = reverse previous sort order.
/C	Prevents prompting for confirmation. DEFRAG does not prompt for confirmation when run from a batch file.

Example

The first example defragments drive C, places the files in reverse alphabetical order, and reboots the system when finished. The second example defragments files on the current drive, optimizes directories and sorts in the order of increasing file size.

```
DEFRAG c: /s:n-e- /b
```

or

```
DEFRAG /u /ss
```

DEL (DELete)

Internal Command

The DEL command deletes a specified file or set of files.

Syntax

DEL [*drive:*] [*path*] *filename* [/P]

Remarks

The DEL command and the ERASE command are functionally identical.

When no drive is specified, the default is assumed. When no path is specified, the default path is assumed.

Global filename characters ? and * (wildcards) can be used in the *filename*. This should be done with caution as it is possible to delete multiple files unintentionally.

When the *filename* *.* is used to delete all files in the specified subdirectory, a verification message is displayed:

```
Are you sure (Y/N) ?
```

Enter Y to delete all files in the specified subdirectory.

DEL deletes files within a subdirectory, not the subdirectory itself. For subdirectory removal, refer to the RMDIR command.

Caution: No ROM-DOS command can undelete a file. Although utilities exist that can attempt an undelete, certain factors can cause the deleted file to be partially or totally lost. The DEL command should be treated as a permanent deletion.

Options

The /P option causes DEL to pause and prompt you before each file is deleted. This option is most useful when deleting files with wildcards. A sample prompt is shown below:

```
<Filename> Delete (Y/N) ?
```

Examples

```
DEL A:*.DOC /P
```

Deletes all files on the A: drive with a .DOC extension. Before each file is deleted, you are prompted to determine if that file should be deleted.

```
DEL MYLETTER.DOC
```

Deletes the file MYLETTER.DOC from the current default subdirectory.

```
DEL *.DOC
```

Deletes all files in the current subdirectory with a .DOC file extension.

DELTREE

External Command

The DELTREE command deletes one or more directory trees or individual files.

Syntax

DELTREE [/Y] [drive:]path [[drive:]path[...]]

Remarks

/Y prevents DELTREE from prompting before deleting.

[drive:]path indicates the name(s) of the file(s) or directory tree(s) to delete. Wildcards are allowed.

Examples

Datalight DELTREE deletes one or more directory trees. For example, to delete all files and directories in the tree C:\TEMP, enter

```
DELTREE C:\TEMP.
```

You can also use DELTREE to delete individual files, one at a time, using wildcards. For example, to delete selected files in the current directory, enter

```
DELTREE *.*
```

DELTREE then prompts you for each file it finds, allowing you to choose whether to delete them.

Caution: Take care when using wildcards with DELTREE. DELTREE deletes all specified files and subdirectories regardless of their attributes. Multiple files and/or subdirectories may be specified.

DEVICE

CONFIG.SYS Command

The DEVICE command installs a device driver.

Syntax

DEVICE = [*drive*] [*path*] *driver name* [*arguments*]

Remarks

A device driver allows ROM-DOS to access hardware that is not common in all PCs.

The full drive path and filename of the device must be specified. The arguments are different depending on the device driver.

Example

```
DEVICE=C:\BIN\VDISK.SYS 120 /e
```

Installs the ROM-DOS RAM disk driver, VDISK.SYS, via the DEVICE command and dedicates 120KB of extended memory to the RAM disk.

DEVICEHIGH

CONFIG.SYS Command

The DEVICEHIGH command loads an installable device driver into the upper memory area, if available.

Syntax

DEVICEHIGH = [*drive*] [*path*] *driver name* [*arguments*]

Remarks

A device driver allows ROM-DOS to access hardware that is not common in all PCs. A device can be loaded into the upper memory areas if they are available and there is enough free upper memory to accommodate the driver's needs. To make high memory available, the EMM386.EXE and HIMEM.SYS utilities must be loaded. If these utilities are not loaded or there is not enough upper memory available, the device is loaded into conventional memory.

The full drive path and filename of the device must be specified. The arguments differ depending on the device driver.

Example

```
DEVICEHIGH=C:\BIN\MYDEVICE.SYS /20 /M
```

Installs a driver MYDEVICE with its command line arguments as specified. The device is loaded into upper memory, if available.

DIR (DIRectory)

Internal Command

The DIR command displays a list of the files that are in a specific directory.

Syntax

DIR [*drive:*] [*path*] [*filename*] [*/option*]

Remarks

Use the DIR command to list all the files in a directory or to show the directory entries of specific files. The standard directory display format includes columns for filenames, filename extensions, file sizes, and the dates and times the files were created.

Options

The /A option causes the DIR command to display only the files that match the specified *filename* and have the given attribute. The following list shows the legal attribute descriptions.

Attribute	Description
A	Archive-ready for archiving
D	Directories
H	Hidden files
R	Read only files
S	System files
X	Show attributes
-	The minus sign can be used to negate listed attributes. For example, to select all files that do not have the archive bit set, use /A-A option.

The /B or bare option causes the filenames to be displayed without volume label, date, time, or size information.

The /L option causes the filenames to be displayed in lowercase.

The /P option selects page mode, which makes ROM-DOS pause the display each time the screen is full. Press any key to go onto the next page of entries.

The /O option causes the filenames to be displayed in sorted order. The sort order can contain one or more of the following attributes.

Attribute	Description
D	By date and time, newest first
E	Alphabetic order by extension
G	Directories grouped before files
N	Alphabetic order by name
S	Size smallest first
-	The minus sign can precede the sort option to reverse the sort order. For example, to sort all files in the directory in reverse alphabetic order, use /O-N option.

The /S option causes the display to include files in subdirectories also.

The /W option displays the list in a wide format without date, time, or size.

The DIRCMD environment variable can be used to set the default preferences for the DIR command. The SET command assigns the values to an environment variable. Refer to the SET command for proper usage. For example, to always have the /P option set for DIR, use the statement SET DIRCMD=/P. You can override the default settings in DIRCMD by using the minus sign (-) preceding the option. To cancel the paging for a single use of the DIR command, enter DIR /-P.

The DIRSIZE environment variable is useful for nonstandard screen sizes. As with the DIRCMD variable, the SET command assigns the values. The syntax is

SET DIRSIZE *rows*[,*columns*]

The values for *rows* and *columns* only have an affect when you use the /P or /W options with the DIR command. The /P (paging) option uses *rows* to display the correct number of lines before

pausing. The /W (wide display) uses the *columns* argument to display the correct number of columns across the width of the screen.

Examples

```
DIR
```

Lists the directory entries of all files in the current drive and directory.

```
DIR B:\MEMOS
```

Lists all files in the subdirectory MEMOS on drive B:.

```
DIR A:*.RPT/P
```

Lists the directory entries for all files in drive A: with the extension RPT, displayed one screen at a time.

```
DIR /ON
```

Lists all files sorted by filename order.

```
DIR /AH
```

Lists all hidden files.

```
DIR *.DOC /B
```

Lists all files with a .DOC extension without file sizes or volume labels.

DISKCOPY

External Command

The DISKCOPY command copies the entire contents of one floppy disk to another.

Syntax

DISKCOPY *<drive1:> <drive2:> [/option]*

Remarks

The first drive specifies the source disk, and the second drive specifies the target disk.

The disks that may be copied are 360KB, 720KB, 1.2MB, and 1.44MB disks. Both the source disk and the target disk must be of the same type. If the target disk is not formatted, or is formatted with another format, then it is reformatted before copying.

If any problem occurs during the copy process, DISKCOPY indicates the side, track, and sector where the problem occurred.

The source or the target disk may not be a RAM or virtual disk.

DISKCOPY copies only floppy disks, not hard disks.

Options

The /V option verifies the copy after completion.

Examples

```
DISKCOPY A: B:
```

Duplicates the contents of the disk in drive A: onto drive B:. Drive B: must support the same type of disk as drive A:.

```
DISKCOPY A: A:
```

Duplicates the contents of the disk in drive A:. You are prompted for swapping the source and target disk as needed to perform the entire copy.

DISPLAY

Installable Device Driver

DISPLAY is a device driver that allows you to view international letters and symbols (code pages) on EGA and VGA displays.

Syntax

device =[*drive:*][*path*]DISPLAY.SYS *codepage*[*fontfilename*]

Remarks

DISPLAY immediately reconfigures your video adapter to display characters from the selected code page instead of those characters built into the hardware.

The *codepage* argument specifies the code page you wish to display. ROM-DOS supports code pages 437, 850, 852, 860, 863, 865, and 866.

The *fontfilename* argument, when included, tells ROM-DOS where to find the EGA.CPI or EGA3.CPI font file. EGA.CPI is the default file choice. If EGA.CPI is in the same directory as CONFIG.SYS, this can be omitted.

Currently, font information for all countries other than Russia and Czechoslovakia can be found in EGA.CPI. Russia's and Czechoslovakia's information is contained in EGA3.CPI.

For more information on DISPLAY, see the section on 'Configuring ROM-DOS for International Use' on page 17.

Examples

```
device=DISPLAY.SYS 850
device=C:\DOS\DISPLAY.SYS 850 C:\DOS\EGA.CPI
```

These examples configure the video adapter to display code page 850. The second example is used if the DISPLAY.SYS and EGA.CPI files are in the C:\DOS directory instead of the same directory as CONFIG.SYS.

DOS

CONFIG.SYS Command

ROM-DOS can be loaded into an upper portion of memory referred to as the High Memory Area (HMA), freeing more of conventional (lower 640KB) DOS memory for use by applications.

Syntax

DOS=HIGH

Remarks

The DOS=HIGH command frees up more of the standard DOS memory for use by applications.

This command only works on 286 and higher CPUs with extended memory and Datalight's HIMEM.SYS High Memory Manager, or equivalent, installed. It will not work on standard XT class PCs. Setting DOS=HIGH is ignored when ROM-DOS is in ROM.

Refer also to the HIMEM device driver description on page 58.

Example

```
DEVICE=HIMEM.SYS
DOS=HIGH
```

Loads the high memory area device driver and then loads ROM-DOS into the HMA for increased conventional memory. The high-memory-area device driver must be loaded first, before DOS=HIGH.

ECHO

Batch File, Internal Command

The ECHO command controls whether ROM-DOS commands and other messages are displayed during batch file execution. ECHO also allows you to create your own messages for display.

Syntax

ECHO [ON|OFF]

ECHO *message*

Remarks

The ON option is the default ECHO setting. It causes commands in a batch file to be displayed as ROM-DOS executes them. Typing ECHO OFF turns off such display, after which the ON option switches it back on again.

The ECHO command alone, entered without the ON or OFF option, displays the current ECHO setting.

The *message* option is a string of characters, such as a warning or a reminder, that you want ROM-DOS to display. Although your message displays whether ECHO is on or off, the message display is useful only when ECHO is off.

To create a message, enter ECHO followed by your message. If your message is more than one line long, the ECHO command must begin each line of the message.

The @ symbol can be used to suppress the echoing of a single command when ECHO is on. Place the @ symbol first on the command line. Refer also to the description of the @ command for additional information.

Examples

```
ECHO This batch file moves files
ECHO to another directory.
```

A batch file message with more than one line.

```
ECHO OFF
```

Sets the ECHO to off.

EGA.CPI/EGA3.CPI

Support Files

EGA.CPI and EGA3.CPI are font data files for use with the International video display driver, DISPLAY.SYS. These files contain alternate font sets to display in place of the hardware code page built into the EGA or VGA display monitor.

Syntax

Refer to the instructions for DISPLAY.SYS for usage.

Remarks

Currently, font information for all countries other than Russia and the Czechoslovakia can be found in the file EGA.CPI. Russia's and Czechoslovakia's information is contained in EGA3.CPI.

EMM386

Installable Device Driver

The EMM386 device driver enables expanded memory support for systems capable of supporting expanded memory such as the 386 and higher CPUs. HIMEM.SYS, or another extended memory specifications (XMS) manager, must be installed prior to EMM386.

Syntax (in CONFIG.SYS)

device=[*drive:*] [*path*] HIMEM.SYS

device=[*drive:*] [*path*] EMM386.EXE {I=xxxx-yyyy} {X=xxxx-yyyy} [FRAME=seg[,memK]]

[ROM=xxxx-yyyy] [D=xxx]

xxxx and yyyy define a range of memory. I= includes UMBs (Upper Memory Blocks) in that range. X= excludes UMBs in that range. ROM= creates ROM shadowing in that range.

Remarks

The HIMEM.SYS driver must be loaded for the EMM386 utility to function properly. Datalight's EMM386 supports Expanded Memory Services (EMS), Upper Memory Blocks (UMB), Virtual DMA Services (VDS), Virtual Control Program Interface (VCPI), and ROM shadowing.

MS-DOS checks for the presence of an XMS UMB provider after it loads any device. If such a device is found, it allocates all available UMBs, builds MCB chains within them, and records the fact that an upper memory chain is available in its SDA. However, ROM-DOS does not function in this manner. The Datalight EMM386 driver builds its own MCB chains within its allocated UMBs and sets up the previously-mentioned data structures with the help of the DOS kernel. Consequently, XMS UMBs that are made available by programs such as MS-DS\OS' EMM386.EXE are not recognized by ROM-DOS.

The *I=xxxx-yyyy* option tells EMM386 to include UMBs in the range specified by *xxxx-yyyy*. The *X= xxxx-yyyy* option tells EMM386 to exclude or **not** to make UMBs in a specified range. EMM386, by default, attempts to make UMBs in the range from C800-F800, excluding areas already occupied by ROM or RAM. Up to eight ranges can be specified. The range can be

specified as an exact range, for example, D000 to DFFF or as the starting and ending marks for the range D000 to E000.

The *FRAME=* option defines the starting segment for four 16K pages that can be mapped in and out at will. The optional *memK* argument specifies how much memory to reserve for EMS. The default behavior is to allow all XMS memory to be used as EMS but also to share it, if not in use, with other processes such as Windows or Datalight's WinLight. The EMS memory is simulated using extended memory. The segment must be on an even 16K boundary. For example,

```
DEVICE = EMM386.EXE FRAME=E000,512
```

The above CONFIG.SYS statement allocates 512KB of extended memory to be used as EMS memory, with four Windows into that memory at E000:0. By default, EMM386 uses memory as requested for EMS up to the maximum in the system.

The *ROM=* option specifies a region of ROM to be supported by shadow RAM. This can speed up system performance if the BIOS resides in ROM and does not do its own ROM shadowing. As with all memory ranges, the range must begin and end on 4KB boundaries.

The *D=* option specifies the amount of RAM in kilobytes (specified as a base-10 number) that is to be reserved for a VDS buffer. The default value is zero. Values for the D= option must be between 16KB and 256KB and are rounded up to the nearest 4KB.

The *MAX=memk* option specifies the maximum amount of EMS memory in KB.

The *LOW* argument prevents the relocation of EMM386 into the first UMB with sufficient size to hold it. EMM386 remains in conventional memory, using approximately 20KB, but reserving UMB areas for use by other drivers.

The *PS2* argument forces EMM386 to use the PS2-style (port92h) A20 line control.

The *RAM* argument is included for compatibility and has no effect on the function or setup for EMM386.

Datalight EMM386 contains auto detection of BIOS extensions. EMM386 automatically searches for UMBs in the range of C800-F800, similar to MS EMM386. It supports VCPI, VDS, and all Int 67H functions, shared XMS/EMS, and works with both Datalight WinLight and MS Windows. Datalight's EMM386.EXE is significantly smaller than other extended memory managers.

Datalight's EMM386 does not support reallocation of EMS pages. Only four map-able pages are supported. EMS handle 0 is not supported. LIM 3.2 contexts obtained through Int 67h function 47h share eight storage slots with EMS handle names.

Examples

```
DEVICE = EMM386.EXE I=C800-F000
```

Maps RAM from extended memory into the address C800:0 to just under F000:0 and defines an Upper Memory Block region there. The range could also have been specified as C800-EFFF. When no FRAME= option is supplied, neither EMS nor VCPI services are provided by EMM386.

```
DEVICE = EMM386.EXE I=C800-D7FF I=F000-F7FF FRAME=E000, 1024
```

Maps in RAM from extended memory into C800-D7FF and into F000-F7FF and define Upper Memory Block Regions there. Also, EMS support is allowed with four 16KB pages starting at E000:0 with 1024KB (1MG) worth of 16KB pages.

```
DEVICE = EMM386.EXE ROM=F000-FFFF
```

Specifies an address range for EMM386 to use for shadow RAM. The ROM represented by the address range is copied into RAM. The RAM area is remapped into the ROM address space and write protected. In this example, the BIOS that normally occupies the 64KB block at F000:0h is copied into RAM and run from there. This may speed up your system if it does not already make use of shadow RAM.

```
DEVICE = EMM386.EXE D=20
```

Sets aside 20KB of memory for Direct Memory Access (DMA). Another client, such as a network TSR, can then utilize this memory buffer for disk I/O.

ERASE

Internal Command

The ERASE command deletes a specified file or set of files.

Syntax

ERASE [*drive:*] [*path*] *filename* [/P]

Remarks

The DEL command and the ERASE command are functionally identical.

When no drive is specified, the default drive is assumed. When no path is specified, the default path is assumed.

Global filename characters ? and * can be used in the *filename*. This should be done with caution as it is possible to delete multiple files unintentionally.

When the *filename* *.* is used to delete all files in the specified subdirectory, a verification message is displayed.

```
Are you sure (Y/N) ?
```

Enter Y to erase (delete) all the files in the specified subdirectory.

Caution: ROM-DOS has no command to unerase a file. Although utilities exist that can attempt an unerase, certain factors can cause the erased file to be partially or totally lost. The ERASE command should be treated as a permanent erase.

ERASE deletes files within a subdirectory, not the subdirectory itself. For subdirectory removal, refer to the RMDIR command.

Options

The /P option causes ERASE to pause and prompt before each file is deleted. This option is most useful when deleting files with wildcards. A sample prompt is shown below.

```
Filename, Erase (Y/N) ?
```

Examples

```
ERASE MYLETTER.DOC
```

Erases the file MYLETTER.DOC from the current default subdirectory.

```
ERASE *.DOC
```

Erases all files in the current subdirectory with a .DOC file extension.

```
ERASE A:*.DOC /P
```

Erases all files on the A: drive with a .DOC extension. Before each file is erased, you are prompted to determine whether that file should be erased.

EXIT

Internal Command

The EXIT command exits a secondary nested ROM-DOS operation and returns control of the system to the primary program.

Syntax

```
EXIT
```

Remarks

The EXIT command has no affect if a secondary COMMAND.COM command processor has not been loaded since the primary COMMAND.COM is always loaded in a permanent mode. A secondary COMMAND.COM is affected if it is loaded without the /P permanent option.

FCBS

CONFIG.SYS Command

The FCBS command allows you to specify the number of File Control Blocks (FCBs) open at one time.

Syntax

FCBS = number [,*minimum number*]

Remarks

Number specifies the maximum number of FCBs open at any given time. The default for this value is 4. The value for *number* must be in the range from 1 to 255. The *minimum number* specifies the minimum number of FCBs to be open at all times. The *minimum number* argument has the same default and range value as the *number* argument.

Example

```
FCBS = 8, 4
```

Sets the maximum number of FCBs to 8 and leaves at least 4 open at all times.

FDISK

External Command

The FDISK command initializes a hard disk for ROM-DOS to use. FDISK allows you to specify the number of logical drives that are available using a single or multiple physical hard disks. FDISK can be run from its menu or by command line arguments. FDISK supports drives that require LBA support.

Syntax

FDISK [*drive*:] [*options*]

Options

The *drive* option specifies the BIOS drive number to use with other command line arguments. For hard drives, this number is 80h, 81h, and so on, depending on the number of hard drives and partitions in the system.

The /B option creates a FAT16 partition and forces the update of the master boot record code.

The /C option causes FDISK to proceed without confirmation messages.

The /I# argument creates a Super-boot partition with an identification (ID) number of #. 98h is the default value for #.

The /N option prevents writing of the master boot record code for the Super-boot partitions.

The /R option refreshes the master boot record code only

The /S# option sets the partition size in MB. The default setting creates a partition as large as possible.

The /V argument displays the current partition information then exits.

Remarks

The FDISK command, when entered with no command line arguments, prompts with the command menu:

```
V) View partition(s)
R) Raw display of partition sectors
C) Create DOS partition(s)
P) Create Super-boot partition
D) Delete a partition
A) Delete all partitions
T) Toggle boot status
M) Write Master Boot Code
Q) Quit without saving
S) Save changes (and reboot)
```

Enter the letter that corresponds to your choice. If you make a mistake, press Esc to return to the previous menu.

Initialization of a hard disk is usually performed when it is first setup for use. Do this by using choice C (or possibly N). Choice C displays a description of available disk space and a recommendation for using that available space. Follow these recommendations to initialize a hard disk with as many 2GB partitions as possible.

If choice C indicates that the hard disk has partitions already, then they may be deleted if needed. The only reason for deleting partitions is to build smaller-sized partitions or fewer larger-sized partitions.

The M choice writes the code for the master boot record. By default, FDISK only writes the disk partition information. If FDISK finds that the Boot Code is invalid, it automatically writes the boot record. If the existing Boot Code appears valid, it is not rewritten unless you select the M option.

After completing all desired modifications and/or additions, select S to save changes. Once FDISK has modified the disk, the following message appears.

```
Press any key and ROM-DOS will reboot
```

After rebooting, each newly initialized drive must be formatted prior to use. Format only those drives that are new or have been resized. Formatting destroys any existing data.

If no changes were made with FDISK, or you wish to abandon the changes made, select Q to quit.

Examples

```
FDISK 81 /R /C
```

Refreshes the master boot record code on drive 81h without confirmation.

```
FDISK 80 /I98 /S5 /C
```

Creates a 5MB Super-boot partition with an ID of 98h and without confirmation.

```
FDISK 80 /S
```

Creates a partition that is as large as possible.

FILES

CONFIG.SYS Command

The FILES command specifies the maximum number of files that may be open at one time.

Syntax

FILES = *number*

Remarks

The number of files includes the standard files, *stdin*, *stdout*, *stderr*, *stdprn*, and *stdaux*. The minimum value is 8, and the maximum is 255. All other values are ignored.

Example

```
FILES = 10
```

Specifies the maximum number of open files to ten.

FIND

External Command

The FIND command displays lines, within a disk file, that contain a specified string of characters.

Syntax

FIND [*/option*] *string* [*filename*]

Options

The /C option displays only the count of lines found with the specified *string*.

The /N option displays the line number of the line found containing the *string*.

The /V option displays the lines that do not contain the string.

The *string* argument specifies the string of characters to search for.

The *filename* argument specifies the file or group of files to search in. The complete drive and path can be specified. Wildcard characters can be used in the *filename*.

Examples

```
FIND printf junk.c
```

Displays each line in the file JUNK.C that contains the *string* printf.

```
dir | FIND DIR
```

Displays each line in a directory listing that contains a DIR. The command first executes a DOS DIR command with the output piped into the FIND command. The FIND command then displays each line that contains the DIR *string*.

```
FIND /C ROM-DOS MANUAL.TXT
.....MANUAL.TXT: 105
```

Displays a count of the lines in the file MANUAL.TXT that contain the string ROM-DOS.

FOR

Batch File Command

The FOR command allows repeated execution of a ROM-DOS command applied to a set of files.

Syntax

FOR %%*variable* IN (*set*) DO *command* %%*variable*

Remarks

During execution, this command attaches the *variable* as an identifier to each file in the *set* of files described; it then applies the *command* to each of these identified files. The *set* may be an exact list of complete filenames or a global file specification using wildcard characters.

The FOR subcommand can be used directly on the command line and within a batch file. To use on the command line, substitute a single percent (%) symbol for the double percent signs (%%).

Examples

```
FOR %%N IN (Q1.TXT Q2.TXT) DO PRINT %%N
```

Prints only the files Q1.TXT and Q2.TXT.

```
FOR %%N IN (*.TXT) DO PRINT %%N
```

Prints all files in the current default directory with a .TXT extension.

FORMAT

External Command

The FORMAT command initializes a disk so ROM-DOS can access files on that disk. A disk must be formatted before ROM-DOS can use it.

Syntax

FORMAT [*drive:*] [*/options*]

Remarks

FORMAT initializes the disk and directory of the specified drive. The size of the formatted disk is the largest possible size that the specified drive supports, unless a different size is specified via a command line option.

Options

The /4 switch causes the floppy disk to be formatted as a 360KB disk even if the drive is a 1.44MB, 2.88MB, or 1.2MB drive.

The /7 switch causes the floppy disk to be formatted as a 720KB disk even if the drive is a 1.44MB or 2.88MB drive.

The /B option causes FORMAT to used BIOS Int 13h calls. By default, FORMAT checks the DOS version, and if it is DOS 5.0 or higher, it uses the floppy device driver to do the format. Using the /B option forces FORMAT to bypass the floppy or hard disk controller and use BIOS calls. /B makes FORMAT device independent.

The /C switch causes FORMAT to format one disk without operator input. The disk is assumed to be in the specified drive, and FORMAT exits immediately when the format is complete. This switch is useful in batch files or programs that require a formatted disk without user input.

The /F:*size* option specifies the size of the floppy disk to be formatted. Available size values are 360, 720, 1.2, 1.44, and 2.88, and are entered as /F:*size*. For example, /F:1.2.

The /H switch causes the system files not to be hidden or write-protected. This can be used along with the /S option.

The /I option forces FORMAT to use IOCTL calls and never use BIOS calls. Normally, FORMAT first tries to access the device driver IOCTL calls to format the disk. If this fails, BIOS calls are used (unless the /B option is specified). BIOS calls are always used for DOS 3.3 and earlier.

The /Q option causes FORMAT to do a quick format. A quick format reinitializes the disk, deleting each file and subdirectory from the disk. A quick format can only be performed on a previously fully formatted disk.

The /S switch causes FORMAT to copy the ROM-DOS system files, ROM-DOS.SYS and COMMAND.COM, onto the disk. The file ROM-DOS.SYS is renamed and stored on the disk as files IBMBIO.COM and IBMDOS.COM, which are stored as hidden files, unless the /H option is used.

The /V:LABEL switch causes FORMAT to place a volume label on the disk. If the volume label is not provided on the command line, you are prompted for the volume label once the format is complete.

The /[switch causes FORMAT to run without display of the sign-on message.

If FORMAT encounters an error, the exit code returned to DOS indicates the type of error. The error codes are listed in the following table.

Error Level	Type of Error
0	No error encountered
1	Invalid drive
2	Unsupported drive format
3	Attempted hard drive format (unsupported)
4	Write-protect error

GOTO

Batch File, Internal Command

The GOTO subcommand transfers control to another line of the batch file.

Syntax

GOTO *label*

Remarks

The *label* is another line in the batch file consisting of a string up to eight characters long. The label may be an environment variable.

If the specified *label* is not found, then the batch file terminates with the error message

```
Label not found.
```

Example

```
GOTO MESSAGE
```

This command moves the control of execution within the batch file to a line that says

```
:MESSAGE
```

Note: A batch file label must be preceded by a colon (:).

HELP

Internal Command

The HELP command provides on-line help of each ROM-DOS command.

Syntax

HELP <command>

Remarks

- The COMMAND.HLP file must be available (i.e., in the path) to use this command. If it is not available, an error message occurs, indicating that COMMAND.HLP cannot be found.
- HELP serves as a memory aid. For complete information about ROM-DOS commands, always consult this manual.

  HELP for each command can also be displayed by entering a /? following the command name.

  All available batch file commands are also listed by HELP.

Examples

```
HELP DIR
DIR /?
```

Both commands list the help of the DIR command.

HIMEM

Installable Device Driver

The HIMEM.SYS device driver manages extended memory and the High Memory Area (HMA) in a 286, 386 or greater, or PS/2 systems. HIMEM prevents programs from simultaneously using the same area of memory for two different purposes. HIMEM supports the Extended Memory Specification (XMS) 2.0. HIMEM is installed as a device driver in CONFIG.SYS.

Syntax

DEVICE = [*d:*] [*path*] HIMEM.SYS [/machine: *n*] [/A20[+]] [/PS2] [/CONTROLA20:OFF]

Remarks

The HIMEM driver can be used to allow ROM-DOS to run in High Memory.

HIMEM supports a default of 32 handles.

HIMEM should not be used with older versions of Datalight's VDISK. Current versions of VDISK use XMS memory if it is available.

HIMEM recognizes PS/2-style A20 line control and determines whether to use the PS/2 A20 control or the AT A20 control method automatically by calling Int 15h, function C0h (get system configuration). This automatic detection can be overridden with the /Machine: *n*, /A20, A20+, or /PS2 command line switches in the event that the auto detection on a given system fails.

/Machine:1 and /A20 both designate the PC AT A20 control method. These switches instruct HIMEM *not* to wait for the A20 line to settle.

/Machine:2 and /PS2 both designate the PS/2 control method.

/A20+ is similar to /A20 but instructs HIMEM to wait for the A20 line to settle.

/Machine:3 designates support for the Phoenix Cascade BIOS A20 control methods.

Alternately, /CONTROLA20:OFF instructs HIMEM to *not* detect the control method for the A20 line and assumes the A20 line is always on.

The /BIOS switch forces the use of BIOS Int15h, Function 87h, for data transfers to and from XMS memory.

The /QUIET switch forces HIMEM to remove the sign-on message when loading.

Error Conditions

No Extended Memory—An extended memory error condition can occur if the BIOS (via Int 15H, function 88H) notifies HIMEM that there is no extended memory. In this situation, HIMEM displays an appropriate error message and does not install.

Failure to Control the A20 Line—When HIMEM installs, it attempts to control the A20 line; which controls access to the HMA. HIMEM first attempts control via the AT method (using the 8259 keyboard control). If that fails, HIMEM then attempts control via the PS/2 method (using I/O port 60H). If both methods fail, HIMEM assumes it can't control the A20 line and displays the message

```
A20 Control (OFF)
```

If either of these errors occur, try using the /A20, /A20+, or /PS2 in the HIMEM command line.

Note also that some older programs assume that the machine is a 1MB 8086 and so require that the A20 line to be disabled (OFF) while they run. Current programs typically do not require that the A20 line be disabled.

Examples

```
Device = HIMEM.SYS
```

Installs the XMS device driver. Once this driver is installed, accessing the HMA and Extended Memory (XMS) memory areas are legal. The Extended Memory area can contain up to 2GB of memory. Typical systems have 4, 8, or 16MB of XMS memory installed.

```
Device = HIMEM.SYS /machine:1
```

Forces the use of the AT-style A20 line control.

The HIMEM driver fails to load when either the machine does not have memory above the 1MB boundary or the BIOS does not provide support for it. It also fails to load when another XMS manager has been previously installed.

IF

Batch File Command

The IF subcommand allows conditional execution of commands.

Syntax

IF [NOT] *condition command*

Remarks

The *condition* may be any one of the following:

ERRORLEVEL *number*

string1 == *string2*

EXIST [*drive:*] [*path*]*filename*

If the *condition* is true, then the *command* is executed. Otherwise the *command* is bypassed, and the next command in the batch file is executed. The [NOT] option tests the opposite of any condition.

The ERRORLEVEL *number* is true if the last program to execute had an exit code equal or greater than *number*. Using the [NOT] option with this condition tests if the exit code is less than the *number* argument.

The condition *string1* == *string2* is only true when *string1* and *string2* are identical. The strings must match exactly; uppercase/lowercase mismatches are not allowed. Applying the [NOT] option creates a condition that is true only when the strings are not identical.

The EXIST *condition* is true if the specified *filename* is found. Wildcard characters are allowed in the *filename*. The [NOT] EXIST condition is true when the *filename* cannot be found.

Examples

```
IF ERRORLEVEL 15 GOTO EXIT
```

Will GOTO the :EXIT label if the ERRORLEVEL was equal to or greater than 15.

```
IF %1 == CONFIG.SYS PRINT %1
```

Prints the file stored as the %1 parameter only if its exact name is CONFIG.SYS.

```
IF NOT EXIST OLD COPY CONFIG.SYS OLD
```

Copies CONFIG.SYS to OLD if a file named OLD does not exist.

INCLUDE

CONFIG.SYS Command

The INCLUDE command includes the contents of one configuration block into another. The instructions from the originating instruction block, as well as the included block, are carried out. This command can only be used within a CONFIG.SYS configuration block.

INCLUDE = *blockname*

Remarks

This command is useful for sets of instructions common to several system configurations. The commands are defined once in a single configuration block and then inserted into other configuration blocks via the INSERT command. For additional details, refer to the section ' Using Multiple-User Configurations' on page 13.

Example

```
:
:
[MISC]
device=mouse.sys
device=c:\network\loadnet.sys

[WORDPROC]
files=20
buffers=10
set path=c:\bin;c:\wp;c:\dict
INCLUDE=MISC
.
.
.
```

When you choose WORDPROC from a CONFIG.SYS menu, the instructions in the configuration block labeled [WORDPROC] are carried out. The instructions in the INCLUDED block labeled [MISC] are also implemented as part of the [WORDPROC] block of instructions.

INSTALL

CONFIG.SYS Command

The INSTALL command loads Terminate and Stay Resident (TSR) programs during CONFIG.SYS processing.

Syntax

INSTALL = [*d:*\][*path*] *TSR_Program TSR_Arguments*

Remarks

The TSR program is loaded the same as if loaded from AUTOEXEC.BAT, except that an environment is not created. The lack of an environment may cause some programs to execute incorrectly. These programs must be loaded from the AUTOEXEC.BAT file.

Example

```
INSTALL = C:\BIN\MOUSE.COM
```

Loads a mouse driver from CONFIG.SYS using INSTALL. Command line arguments can be included.

KEYB

External Command

The KEYB command allows you to alter the keyboard layout for a different language or nationality.

Syntax

KEYB

KEYB *countryid*

KEYB *countryid*, [*codepage*] [,*keyboard filename*]

Remarks

KEYB is a terminate and stay resident program (TSR). Running KEYB with no arguments shows the current settings of a resident copy of KEYB, if there is one.

The *countryid* argument is a two-letter code that specifies which country, region, or language is to become current.

When no *codepage* is included, KEYB uses the default code page for the *countryid*. You can specify either the default or the alternate code page for any *countryid*.

The *keyboard filename* argument tells KEYB where to find its data file (KEYBOARD.SYS or KEYBRD2.SYS). When no *keyboard filename* is given, KEYB first looks for KEYBOARD.SYS in the current directory, then in the directory containing KEYB.COM.

Currently, keyboard data for all countries except Russia and Czechoslovakia are found in KEYBOARD.SYS. The file KEYBRD2.SYS contains the data for Russia and Czechoslovakia.

Refer to the table on page 20 for a list of the supported *countryid* codes, along with their default and alternate code pages.

If a copy of KEYB has already been run, it is reconfigured to the new specifications. While KEYB is active, you can switch back to a U.S. layout at any time by pressing Ctrl+Alt+F1 (Alt+Left-Shift for Russian and Czech Republic keyboards). You can toggle back to the alternate layout by pressing Ctrl+Alt+F2 (Alt+Right-Shift for Russian and Czech Republic keyboards).

Examples

```
KEYB GR
KEYB GR,437
KEYB GR,,C:\TOOLS\KEYBOARD.SYS
```

Each of these commands establishes a German keyboard layout. The first and third use code page 850, while the second uses code page 437. In the third case, the KEYBOARD.SYS file is located in the C:\TOOLS directory.

KEYBOARD.SYS/KEYBRD2.SYS

Support Files

KEYBOARD.SYS and KEYBRD2.SYS are keyboard code page data files for use with the International keyboard driver, KEYB.COM.

Syntax

Refer to KEYB.COMERASE for usage instructions.

Remarks

Currently , keyboard date for all countries except Russia and Czechoslovakia are found in KEYBOARD.SYS. The file KEYBRD2.SYS contains the data for Russia and Czechoslovakia.

LABEL

External Command

The LABEL command sets or deletes a disk volume label.

Format

LABEL [*drive:*] [*volume string*]

Remarks

The volume label may be up to 11 characters in length. LABEL only uses the first 11 characters of a volume label. The characters that are acceptable in a volume label are the same as those for a filename.

LABEL prompts as follows. Enter the new label and press enter to modify the existing label.

```
Volume in drive C is xxxxxxxxxxx

Volume label (11 characters, ENTER for none)?
```

If a volume label has previously been assigned to a disk, and you do not enter a new volume label, the following message is printed.

```
Delete current volume label (Y/N)?
```

If the disk did not have a volume label prior to running the LABEL command, the above message will not appear.

Example

```
LABEL a:
```

LABEL displays the volume label of drive A:, if one exists, and allows it to be modified or deleted.

LASTDRIVE

CONFIG.SYS Command

The LASTDRIVE command sets the maximum number of drives.

Syntax

LASTDRIVE = *letter*

Remarks

letter may be any character between A and Z and is the last drive letter that ROM-DOS can access. The default value for *letter* is E.

The minimum number for LASTDRIVE is the number of drives in your computer. If *letter* is less than number of drives in your computer, then the LASTDRIVE command is ignored.

LASTDRIVE is often used to cause ROM-DOS to make more space for nonstandard drives that are not in your system. These drives may be CD-ROM drives, flash disk drives, or network drives.

Example

```
LASTDRIVE = H
```

Causes ROM-DOS to allocate space for eight drives. If the computer has five drives installed, there is room for three additional nonstandard drives.

LOADHIGH

CONFIG.SYS, Internal Command

The LOADHIGH command loads an executable or TSR program into the upper memory area, if available. LOADHIGH can be run as a batch file command or from the DOS command line.

Syntax

LOADHIGH = *executable [arguments]*

-or-

LH = *executable* [*arguments*]

Remarks

An executable or TSR program can be loaded into the upper memory areas when they are available and have enough free upper memory to accommodate the program's needs. To make high memory available, the EMM386.EXE and HIMEM.SYS utilities must be loaded. If these utilities are not loaded or there is not enough upper memory available, the program loads into conventional memory.

The full drive path and filename of the device must be specified. The arguments are different depending on the device driver.

Example

```
LOADHIGH=C:\apps\checkit.exe /p
```

Installs an executable named CHECKIT with its command line arguments as specified. The program loads into upper memory, if available.

LONGDIR.EXE

External Command

The LONGDIR.EXE command provides a directory listing of files in a single directory including files with long file names.

Syntax

LONGDIR [*] [*x**][*"directory name\file name mask*"*]

Remarks

The LONGDIR command operates only if Long File Name support is enabled within the ROM-DOS kernel. This option is included at development time and cannot be added without the ROM-DOS Development Kit tools.

Subdirectories and paths are not supported with LONGDIR, except within the name mask syntax.

Options

The * option causes LONGDIR to display all files and directory names in the current directory. Both long and standard file named files are displayed.

The *x** option is used to find files or directory names within the current directory that have the character "x" in the name. "x" should be replaced with any valid character.

The "*directory name\file name mask**" option is used to find files that begin with the "file name mask". If no directory name is provided, the files in the current directory are searched for matches. A drive letter can be used as part of the directory name. If an * is used for the file name mask, all files in the named directory are listed.

Examples

```
LONGDIR *
```

Lists all files in the current directory, including those with long file names.

```
LONGDIR *v*
```

Lists all files in the current directory that have the letter "v" in the file name.

```
LONGDIR "a:\Long File Named Files\test*"
```

Lists all files in the "a:\Long File Named Files" directory that have a file name starting with "test".

The directory name need not be a long file name, it can be a standard name.

```
LONGDIR "temp\*"
```

Lists all files in the subdirectory "temp" (one level below the current directory).

MEM

External Command

The MEM command displays the used and free memory in your system.

Syntax

MEM [/BiosExtensions] [/Classify] [/Raw]

Remarks

Options	Description
/B	Displays each BIOS extension and its size.
/C	Classifies the memory usage.
/R	Does raw dump of the MCB chain.

MEM displays a list of the DOS memory contents, what free space is available, and how much memory is in conventional memory, upper memory, the HMA and extended memory. This program is useful to fine tune the system to have as much free memory as possible for applications.

Options

The /B option displays BIOS extensions in the range from C000:0 to F800:0.

The /C option shows program, TSR, and device driver sizes.

The /R option shows a low-level DOS listing of MCBs (Memory Control Blocks).

MENUCOLOR

CONFIG.SYS Command

The MENUCOLOR command allows you to set the text and background colors for the startup menu. This command can only be used in a menu block within your CONFIG.SYS file.

Syntax

MENUCOLOR = *text_color* [,*background_color*]

Remarks

The *text_color* argument selects the display color for the screen text. The color numbers 0 to 15 can be selected from the list below for the text color.

The *background_color* argument is optional. If a value is not entered, the default color 0 (Black) is used. Be sure to specify different colors for background and text, and separate the numbers with a comma. For best results, choose contrasting colors. Only the color numbers 0 to 7 can be used as the background color designation.

For systems whose BIOS does not directly support a video display, such as Datalight's miniBIOS, the standard CONFIG.SYS menu commands, which rely on BIOS screen support, are unusable.

To use these commands, the color number sequence of 0 for text_color and the default background color (black), or 0,0 for text and background colors can be selected. These numbers represent a color choice of black text with a black background, which is an unusable choice for screen viewing. Using the black/black combination in the MENUCOLOR command line signifies to ROM-DOS to display the startup menu in TTY mode without using BIOS screen/cursor positioning or color changing commands.

Color Values:

0–Black	1–Blue	2–Green
3–Cyan	4–Red	5–Magenta
6–Brown	7–White	8–Gray
9–Bright Blue	10–Bright Green	11–Bright Cyan
12–Bright Red	13–Bright Magenta	14– Bright Yellow
15–Bright White		

Examples

```
MENUCOLOR=14,1
```

Displays the menu text in bright yellow on a blue background.

```
MENUCOLOR=5
```

Displays the menu text in magenta with a default background of black.

MENUDEFAULT

CONFIG.SYS Command

The MENUDEFAULT command allows you to set the default menu-item choice and a time-out value for making a menu selection. This command can only be used within a menu configuration block in the CONFIG.SYS file. Refer also to ' Using Multiple-User Configurations ' on page 13.

Syntax

MENUDEFAULT = *blockname*[,*timeout*]

Remarks

The *blockname* argument specifies the default menu item. The value for *blockname* must match a configuration block name defined elsewhere in your CONFIG.SYS file.

The optional time-out argument represents the number of seconds ROM-DOS waits for a user input selection before initializing your system with the default configuration. The time-out period can be set to a value between 0 and 90. If you select 0, the default menu item is automatically implemented without a wait. If you do not enter a time-out value, ROM-DOS will not continue until the Enter key is pressed.

If your system BIOS does not support a video display directly, such as Datalight's miniBIOS, please refer to the MENUCOLOR command for special instructions.

Example

```
[MENU]
menuitem=Word_Proc, Word Processing
menuitem=Network, Network
menuitem=Research, Research and Development
```

```
menucolor=15,1
menudefault=Word_Proc,20
```

Makes the Word_Proc configuration block the default menu item. If you fail to make a selection within 20 seconds, the Word_Proc block is processed.

MENUITEM

CONFIG.SYS Command

The MENUITEM command allows you to specify an item on the startup menu. This command can only be used within a menu configuration block in the CONFIG.SYS file. Refer also to 'Using Multiple-User Configurations' on page 13.

Syntax

MENUITEM = *blockname* [,*menu_text*]

Remarks

The *blockname* argument is a user-defined label given to a configuration block defined elsewhere in the CONFIG.SYS file. If a user selects the menu item, all commands in the selected configuration block are processed, along with the instructions that are common to all menu choices (denoted by block header [COMMON]). The *blockname* can be up to 70 characters long and may contain most printable characters, including spaces, backslashes (\), forward slashes (/), commas, semicolons (;), equal signs (=). Square brackets ([]) cannot be used in block names.

The *menu_text* option is a descriptive statement that defines the *blockname*. The *menu_text* is displayed on the screen as a line item in the startup menu. The *menu_text* argument can be up to 70 characters long and can contain any characters. If this argument is left off, the *blockname* is used for the startup menu display.

If your system BIOS does not support a video display directly, such as Datalight's miniBIOS, please refer to the MENUCOLOR command for special instructions.

Examples

```
[MENU]
menuitem=Word_Proc, Word Processing
menuitem=Network, Network
menuitem=Research, Research and Development
menudefault=Word_Proc,20
```

Defines three menu items: Word_Proc, Network, and Research and Development. Each of these has descriptive text and a set of commands defined later in the CONFIG.SYS file. At boot time, these menu items are displayed in the startup menu as follows:

```
ROM-DOS 6.22 STARTUP MENU
1. Word Processing
2. Network
3. Research and Development
Enter a choice: 1
```

MKDIR (MaKe DIRectory)

Internal Command

The MKDIR command creates a new subdirectory.

Syntax

MKDIR [*drive:*] [*path*] *subdir*

MD [*drive:*] [*path*] *subdir*

where *subdir* is the name of the new subdirectory to be created. Note that MD may be used instead of MKDIR.

Remarks

If no drive or path is specified, the new subdirectory is created within (one level below) the current default directory.

If drive and/or path is specified, everything specified must exist or the command displays an error message.

Examples

```
MKDIR TEMPDIR1
```

Creates a new subdirectory named TEMPDIR1 within the current default directory.

```
MD C:\UTIL\TOOLS
```

Creates a new subdirectory named TOOLS within UTIL, assuming the subdirectory exists.

MODE

External Command

The MODE command modifies the operation of the printer, serial port, and active video display.

Syntax

MODE LPT#[:]=COM#[:]

MODE COM#:*baud*[,*parity*[,*databits*[,*stopbits*[,*P*]]]]

MODE <*video mode*>

MODE <*display lines*>

Remarks

The first syntax above redirects line printer output to the serial port.

The second syntax above changes the operation of the specified communications port. The options that can be modified are listed below. Invalid values for any of the options are flagged with an error message.

baud	110, 150, 300, 600, 1200, 2400, 4800, 9600
parity	N - None, O - Odd, E - Even
databits	Either 7 or 8
stopbits	Either 1 or 2 stop bits
P	Printer Port

Using the P option as the last argument causes output to be sent repeatedly to the printer port until successfully received. Without the P, output is sent only once, causing a critical error if unsuccessful.

The third syntax changes the active video mode for the display terminal. The valid choices for this version of the MODE command are as follows:

40—Indicates 40 characters per line.

80—Indicates 80 characters per line.

bw40—For a color graphics adapter with color disabled and 40 characters per line.

bw80—For a color graphics adapter with color disabled and 80 characters per line.

co40—Indicates a color monitor with color enabled and 40 characters per line.

co80—Indicates a color monitor with color enabled and 80 characters per line.

mono—For a monochrome display. Assumes 80 characters per line.

The final syntax sets the number of display lines. Valid values included L25, L43, and L50.

Note: A serial port should be initialized before an LPT device is redirected to it.

Examples

```
MODE COM1:9600,n,8,1
```

Modifies the settings for the COM1 device to a baud rate of 9600, no parity, eight data bits, and one stop bit.

```
MODE LPT2:=COM2
```

Redirects the output from LPT2 to the COM2 serial port. All following output to LPT2 actually goes to the COM2 device.

```
MODE mono
```

Indicates a monochrome display adapter.

MORE

External Command

The MORE command displays a text file one screen at a time.

Syntax

MORE [*filename*]

or

<*command*> | MORE

Remarks

The input to MORE may come from a file, or it may be piped in from another filter or a DOS command. If the *filename* is present, then the file is viewed; otherwise MORE reads from the Standard Input.

Once a screen has been viewed, a line is displayed on the bottom of the screen indicating the percent of the file that has been viewed. At this point, there are several options for the next lines of text to be viewed.

B	Display the previous full page.
<enter>	Display just one more line.
T	Display starting at the top of the file.
Spacebar	Display the next full page of text.
Q	Exit MORE

Examples

```
DIR | MORE
```

Displays a directory one screen at a time.

```
MORE READ.ME
```

Displays the file READ.ME one page at a time.

MOVE

Internal Command

The MOVE command moves files and renames files and directories.

Syntax

To move one or more files:

MOVE [/Y | /-Y] [*drive*:][*path*]*filename1*[,...] *destination*

To rename a directory:

MOVE [/Y | /-Y] [*drive*:][*path*]*dirname1 dirname2*

Remarks

[*drive*:][*path*]*filename1* specifies the location and name of the file or files you want to move.

destination specifies the new location of the file. Destination can consist of a drive letter and colon, a directory name, or a combination. If you are moving only one file, you can also include a filename if you want to rename the file when you move it.

[*drive*:][*path*]*dirname1* specifies the directory you want to rename.

dirname2 specifies the new name of the directory.

Options

/Y suppresses prompting to confirm creation of a directory or overwriting of the destination.

/-Y causes prompting to confirm creation of a directory or overwriting of the destination. The /Y option may be present in the COPYCMD environment variable. This may be overridden with /-Y on the command line.

MSCDEX

Installable Device Driver

The Datalight CD-ROM driver enables CD-ROM drives.

Syntax

MSCDEX [options]

Remarks

Options	Description
/D:<Name>	CD-ROM device name (default MSC002)
/L:<Letter>	Specifies the drive letter for CD-ROM drive
/M:<Number>	Specifies the number of sector buffers (default 4)
/X	Use extended memory for CD-ROM buffers

A low-level driver from the CD-ROM drive manufacturer interfaces to the actual CD-ROM hardware while the Datalight CD-ROM driver provides the interface between ROM-DOS and that hardware driver. This interface is called the Microsoft CD-ROM extensions (or MSCDEX).

For each hardware driver loaded, use the /D: option to specify the name of that driver. The default name is MSC002.

The /L: option specifies the drive letter to use for the CD-ROM drive. The default driver letter is the next available. To increase the number of available drive letters, use the LASTDRIVE command in CONFIG.SYS.

The /M: option specifies the number of buffers to use to speed up CD-ROM drive access.

The /X option puts those buffers in extended memory rather than conventional memory.

Examples

```
MSCDEX /D:MSC003
```

Installs the CD-ROM driver using MSCD003 as the device name.

```
MSCDEX /L:G
```

Installs the CD-ROM driver using G as drive letter.

```
MSCDEX /M:6 /X
```

Installs the CD-ROM driver using six buffers allocated in XMS.

NEWFILE

CONFIG.SYS Command

The NEWFILE command allows you to continue CONFIG.SYS file processing from a new file. The file can be located in another directory or even on a different drive.

Syntax

NEWFILE=*filename*

Remarks

The NEWFILE command is especially useful when the CONFIG.SYS file is located on an inaccessible drive or in ROM. Additional device drivers or instructions can be added easily to the new file and is processed along with the main CONFIG.SYS file upon starting the system.

When the NEWFILE= instruction is processed, control passes from the present file (the one containing the NEWFILE instruction) to the file specified in the command. Any commands placed after the NEWFILE instruction in the original file are not processed.

If the specified filename cannot be located, CONFIG.SYS processing returns to the original CONFIG.SYS file and the next instruction is processed. The next instruction can even be a second NEWFILE= statement, allowing flexibility for systems which may have a variety of drives installed or not installed at boot time. The first successful NEWFILE statement is processed, transferring control to the specified filename. The remaining instructions in the original CONFIG.SYS file, including NEWFILE= statements, are not processed.

NEWFILE commands can be nested. That is, your original CONFIG.SYS can call a second set of instructions via the NEWFILE command. The second file can, in turn, call a third file by using the NEWFILE command, and so on. Be sure that each filename in the successive steps has a unique name, otherwise, you will create an infinite loop as control is passed back to the same file repeatedly.

When NEWFILE is used, it is also created as an environment variable, retaining the value assigned by the NEWFILE= statement. This can be used in an AUTOEXEC.BAT file for further boot-up decision-making.

Examples

```
NEWFILE=C:\BIN\NEWCFG.SYS
```

Causes instructions in the file NEWCFG.SYS, located in the C:\BIN directory, to be executed as part of the CONFIG.SYS file. The contents of NEWCFG.SYS may include any of the commands listed in this section.

Also, the environment will contain the following entry:

```
NEWFILE=C:\BIN\NEWCFG.SYS
```

Verify this by running the SET command with no parameters.

NUMLOCK

CONFIG.SYS Command

The NUMLOCK command sets the NumLock key on the keyboard to ON or OFF when your computer starts.

Syntax

NUMLOCK=[on|off]

Remarks

Selecting ON designates that the NumLock key is set to on when DOS boots. Selecting OFF designates that the NumLock is off when DOS boots. In either case, you still have the ability to manually turn the NumLock key on and off after booting with the NUMLOCK command.

Example

```
NUMLOCK=on
```

Sets the NumLock key to on when the system boots.

PATH

Internal Command

The PATH command sets the search path for command files that are not in the current directory.

Syntax

PATH [*drive:*] [*path*] [; [*drive:*] [*path*]] ...

Remarks

Without a specified search path, ROM-DOS looks for an external command file (one with a .BAT, .COM, or .EXE extension) only in the current directory. The PATH command tells ROM-DOS which other directories to search after searching the current directory.

To append one or more additional directories to the PATH, use %PATH% for the current path. For example, to add C:\DOS to the current path, enter:

```
PATH %PATH%;C:\DOS
```

at the command prompt.

Enter PATH without parameters to display the current path.

To cancel previously-set command paths, enter:

```
PATH =
```

or

```
PATH =;
```

Example

If your application programs reside on a fixed disk, the PATH command enables you to start any of them from any drive or directory. To access utilities, a word processor, and a spreadsheet in subdirectories C:\UTIL, C:\WP, and C:\123, set the path command as follows:

```
PATH C:\UTIL;C:\WP;C:\123
```

PAUSE

Batch File, Internal Command

The PAUSE command suspends the execution of a batch file and resumes operation when any key is pressed.

Syntax

PAUSE

Remarks

A batch program may require that you perform some action such as changing disks or choosing to continue or terminate the operation. When the command processor encounters PAUSE, it suspends execution and displays the message

```
Strike a key when ready...
```

After you perform the appropriate action, or make a decision, striking any key other than the combinations Ctrl+C or Ctrl+Break resumes the batch job.

If you press Ctrl+C or Ctrl+Break at this point, ROM-DOS displays

```
Terminate batch job (Y/N)?
```

Responding Y ends the batch job. Strategic placement of the PAUSE command, working with this query, allows you to divide the batch file into sections so you can end it at some intermediate point.

The *message* option allows you to display a reminder on the screen during the pause. Your message precedes the "Strike a key" message. Note, however, that your message appears only if ECHO is on.

Example

```
PAUSE Place blank disk in drive A:
```

Prompts the user to insert a disk and suspends operation until a key has been hit.

POWER.EXE

Installable Device Driver

POWER.EXE conserves power on a system that has APM (Advanced Power Management) by shutting down various subsystems (screen, disk drives, etc.) that are not being used. POWER.EXE can be used in an INSTALL command in CONFIG.SYS, in AUTOEXEC.BAT or on the command line.

Syntax

From command line or AUTOEXEC.BAT, enter

[*path*]power[*options*]

From CONFIG.SYS, enter

Install=[*path*]power[*options*]

Remarks

The system BIOS must support the following APM functions for POWER.EXE to work

Function	Description
00	APM Installed
01	Connect
04	Disconnect
05	CPU idle
06	CPU busy
07	Set Power State
08	Get PM Event

For each device, you can specify the length of time that the device must be inactive before it is turned off. If you specify a length of zero for a device, POWER.EXE disables power management for that device.

The (#) argument for each command option defines the number of seconds the device can remain inactive before it is powered down.

Options

The /C# option sets the inactive time for the COM ports. The default value is two seconds.

The /D# option sets the inactive time for the disks. The default value is 30 seconds. Currently, all disk drives are treated as a single device.

The /P# option sets the inactive time for printers. The default time is two seconds.

The /S# option sets the inactive time for the screen. The default time is nine seconds.

The /H option displays a user help screen.

The /K options sets the keyboard time.

The /ADV:MIN argument provides the minimum (most responsive) power reduction.

The /ADV:REG argument provides the standard power reduction.

The /ADV:MAX argument provides the maximum (least responsive) power reduction.

The /STD argument provides the standard power reduction.

The /OFF argument turns power management off.

Example

```
POWER /S20 /C0 /P0
```

Runs POWER, turning off the screen after 20 seconds of inactivity, never turning off the COM and printer ports, and uses the default inactivity period for the disk drives (30 seconds) and keybaord.

PRINT

External Command

The PRINT command prints a single file or a list of files.

Syntax

PRINT [*/drive:*] [*filename*] [*/options*]

Remarks

PRINT allows you to enter between one and 32 files for spooling to the printer. The files are output to the device in a spooled manner (while you perform other operations).

If PRINT is entered without any parameters, it displays all the files that are in the queue.

The first time PRINT is used, the operator is prompted with this message for the device to perform the operation.

```
Name of list device [PRN]:
```

The legal devices for printing are LPT1, LPT2, LPT3, LPT4, COM1, COM2, COM3, COM4, AUX, or PRN.

Options

The /B option sets the buffer size. The default buffer size is 512 bytes. A larger buffer size causes print to operate faster. The maximum buffer size is 32KB and the minimum size is 256 bytes. This option is only allowed the first time PRINT is run.

The /C option cancels only the filenames listed after the /C command.

The /F option sets the maximum number of files to be queued up at one time. The default number of files is ten. The minimum is two and the maximum is 32. Support for more files is often useful when using wildcards in filenames. This option is only allowed the first time PRINT is run (or until the next system reboot).

The /P option causes all files listed after this option to be submitted for printing. This is the default for filenames encountered on the PRINT command line.

The /T option cancels all the files from the print queue (list).

The /H option displays the help screen.

Examples

```
PRINT FILE1.TXT FILE2.TXT FILE3.TXT
```

Puts three files into the print queue. The first file prints after the command ends.

```
PRINT /C FILE2.TXT
```

Removes file FILE2.TXT from the print queue. All other files in the queue print normally.

```
PRINT /T
```

Cancels all files in the print queue. Printing may continue for a short time because of the buffer in your printer.

PROMPT

Internal Command

The PROMPT command changes the ROM-DOS command prompt.

Syntax

PROMPT [*text*] [$*character*] [$*character*...]

Remarks

The prompt that ROM-DOS normally displays is the letter of the current drive followed by a right angle bracket (>) (the greater-than symbol). By using the PROMPT command, you can change this prompt to include any combination of a message, the current directory, the date, the time, and some other features.

Code	Corresponding Prompt
$T	Current time
$D	Current date
$P	Current drive and path
$V	ROM-DOS version number
$N	Current drive
$G	The > character

Code	Corresponding Prompt
$L	The < character
$B	The | character
$Q	The = sign
$H	A backspace (which erases the previous character)
$E	ASCII code for Escape (X'1B')
$_	Start a new line (carriage return)
$$	The $ character

Examples

To show this prompt

```
Current directory is drive:\path:
Ready for <command>
```

enter

```
PROMPT Current directory is $P;$_Ready for $Lcommand$G
```

To show the date, time, and current directory on separate lines followed by the greater-than character and a space, enter:

```
PROMPT $D$_$T$_$P$G<space>
```

where <*space*> refers to pressing the spacebar once. The resulting prompt is

```
Mon 6-26-1989
10:17:45.99
A:\> _
```

REM (REMark)

Batch File, Internal Command

The REM command has two purposes: to allow comments in a batch or CONFIG.SYS file, and to temporarily disable a command without physically deleting the command from the file. See also the (;) command.

Syntax

REM [*message*]

Remarks

The REM command provides information but has no affect on the execution of the batch file.

The comment may consist of any set of characters. You may also create a blank line by omitting the *message* portion of the line.

REM can also be used to temporarily disable a command in a batch file or CONFIG.SYS without having to delete the line from the file.

Examples

```
REM This batch file created by
REM Jane Doe
```

These lines may be added at any point in a batch file as user information only.

```
DEVICE=HIMEM.SYS
DOS=HIGH
REM DEVICE=TESTDEV.SYS /P
```

Temporarily removes the DEVICE=TESTDEV.SYS statement from these CONFIG.SYS instructions. This statement is not processed again until REM is removed.

REMQUIT.EXE

External Command

The REMQUIT command terminates the REMSERV program via the serial port. This command allows the host machine to terminate the program running on the target without accessing the target machine directly.

Syntax

REMQUIT

Remarks

This program only terminates the REMSERV program and does not work in reverse to terminate the REMDISK program.

Examples

```
REMQUIT
```

Terminates the running REMSERV program when run from the remote system attached via a serial connection.

REN (REName)

Internal Command

The REN command changes the name of a file.

Syntax

REN [*drive:*] [*path*] *filename1 filename2*

Remarks

REN renames files within a directory; it does not move a file to a different drive or directory as part of the command.

The wildcard characters * and ? may be used to rename more than one file at a time.

ROM-DOS does not allow you to give a file a name that matches the name of an existing file in the same directory.

Examples

```
REN B:NOTES.DOC REPORT.DOC
```

Renames the file NOTES.DOC in drive B: to REPORT.DOC.

```
REN *.DOC *.TXT
```

Assigns the extension .TXT to all files with the current extension .DOC.

RESTORE

External Command

The RESTORE command is the complement program to BACKUP. It restores files previously saved with BACKUP to the hard disk.

Syntax

RESTORE <*srcdrive*> [*dstpath*] [options]

Remarks

Option	Description
/D	Show directory of files in backup volume(s).
/P	Prompt only if destination file already exists and is a hidden, system, or read-only file, or is marked as changed. Do not prompt on unchanged existing files.
/S	Restore all subdirectories.
/Y	Answer Yes to all prompts.

The <*srcdrive*> is the drive to restore from and may include an optional file mask. For example, RESTORE A:*.C restores only those files ending with .C from the backup set in drive A:.

The <*dstpath*> must be a DOS destination path. If no destination path is given, RESTORE assumes the current directory. The destination may contain wildcards (which override wildcards placed in the <*srcdrive*> option.

Options

The /D option just displays a directory of what files are in the backup set, similar to DIR. This is useful if you are not sure what files or dates/times are in the backup set.

The /S option restores from subdirectories as well.

The /Y option is useful when running RESTORE from a batch file. Any entire set from another hard drive or network drive can be restored without user input.

Examples

```
RESTORE A: /D
```

Displays files in the backup.

```
RESTORE A: D:TEMP\fx*.fh /S
```

Completes restore, with subdirectories, of all files whose filenames begin with fx and have a .fh file extension.

```
RESTORE B:*.c /S
```

Restores only files with a .C file extension to the current directory.

RMDIR (ReMove DIRectory)

Internal Command

The RMDIR command removes (deletes) a specified empty subdirectory.

Syntax

RMDIR [*drive:*][*path*]*subdir*

RD [*drive:*] [*path*] *subdir*

subdir is the name of the subdirectory being deleted. Note that RD may also be used.

Remarks

If no drive or path is specified, RMDIR looks for the specified *subdir* within (one level below) the current default directory. If a drive or path is specified, everything specified must exist or ROM-DOS displays an error message.

RMDIR does not remove a subdirectory unless it is empty. An error message is displayed when you attempt to remove a subdirectory that still contains files or other subdirectories.

Example

```
RD TOOLS
```

Removes the TOOLS subdirectory from the current directory, assuming TOOLS is an empty directory.

SET

Internal Command

The SET command sets, displays, or removes environment variables.

Syntax

SET [*variable* = [*string*]]

Remarks

Use the environment variables to control the behavior of programs and batch files and also the behavior of ROM-DOS. Use this command in the AUTOEXEC.BAT and CONFIG.SYS files and on the DOS command line. The environment variables that can be defined with the set command include, but are not limited to, PATH, COMSPEC, PROMPT, and user-defined variables.

Using SET *variable* = with no argument string clears the current environment string for the named variable.

Examples

```
SET PROMPT = $p$g
```

Sets the prompt, although the prompt can also be set with the PROMPT command.

```
SET PROMPT =
```

Clears any previously set prompt settings and returns the prompt to its default state.

SHARE

External Command

The SHARE command installs the capabilities for file sharing and file locking on your hard disk.

Syntax

SHARE [/L:*nn*] [/u]

Or from CONFIG.SYS

INSTALL=[*d:*][*path*]SHARE.EXE [/*options*]

Remarks

The SHARE utility is most commonly used in a network or multitasking environment where file sharing is necessary. When SHARE is loaded, DOS utilizes the SHARE utility to validate read and write requests from application programs and users.

Options

The /L:*nn* option specifies the maximum number of files that can be locked at one time. The default number is 20.

The /U option unloads the share utility and frees the memory. SHARE does not unload if other TSRs have been loaded on top of it. The other TSRs must be unloaded first before trying to unload SHARE.

Examples

```
SHARE
```

Loads the SHARE program from the command line.

```
INSTALL=C:\UTILS\SHARE.EXE /l:30
```

Installs SHARE from the CONFIG.SYS file and changes the maximum number of locked files to 30.

```
SHARE /U
```

Unloads SHARE and frees the used memory.

SHELL

CONFIG.SYS Command

The SHELL command allows you to specify a command interpreter other than the default COMMAND.COM or to load COMMAND.COM with non-default arguments (parameters). ROM-DOS boots this new program, with arguments, instead of that specified internally.

Syntax

SHELL = *cmd_interpreter arguments*

Remarks

The SHELL command is most often used to start the initial copy of COMMAND with special parameters. One parameter provides a larger environment than the default 128 bytes.

The *cmd_interpreter* can be any executable program. The full path, including drive letter, should be specified if the program is not in the root directory of the default drive.

Arguments are optional and program-specific and vary depending on the *cmd_interpreter* being executed by the SHELL command.

Examples

```
SHELL=C:\COMMAND.COM /E:512 /P
```

Boots the standard Command Processor but sets the environment space to 512 bytes (up from the default 128). The /P parameter tells COMMAND that it is permanent (cannot terminate).

```
SHELL = C:\TEMP\MYPROG.EXE
```

Boots a program named MYPROG.EXE, located in the directory TEMP, instead of the standard Command Processor.

SHIFT

Batch File, Internal Command

The SHIFT command moves each replaceable parameter for a batch file one position to the left. Execution of the SHIFT command allows use of more replaceable parameters in a batch file beyond the standard set of %0 through %9.

Syntax

SHIFT

Remarks

This command moves the string or value stored for each replaceable parameter one position to the left. Upon execution of SHIFT, the %0 argument assumes the value of the %1 argument, the %1 argument then assumes the value of the %2 argument, and so on.

Example

The following batch file reads in a list of files (provided as arguments on the command line) and displays each one to the screen. After displaying each one, the SHIFT command copies the next file in the argument list into the %1 slot, verifying the existence of the file, and continues.

```
Command line argument:
TYPEIT autoexec.bat config.sys net.bat
TYPEIT.BAT batch file:
   :repeat
   if EXIST %1 goto doit
   goto end

   :doit
   type %1
   pause
   shift
   goto repeat

   :end
   @echo All Done
```

SORT

External Command

The SORT command sorts a text file and displays the output to the standard device.

Syntax

SORT [*/options*] [*filename*]

Remarks

SORT normally starts its comparisons at the first character in a line.

The input to SORT may come from a file, or it may be piped in from another filter or a DOS command.

Options

The /+n option causes SORT to begin its alphabetical sorting starting at the n th position in the string.

The /r option causes SORT to sort in the reverse alphabetical order.

Examples

```
SORT NAMES.LST
```

Sorts the file NAMES.LST and displays the output to the screen.

```
DIR | SORT /+14 | MORE
```

Produces a directory and then sorts the directory by file size (the file size in a directory display starts on the 14th position in each line or string). The output display is then shown one screen at a time using the MORE command.

STACKDEV.SYS

Config.sys Command

The STACKDEV.SYS command is used to increase the number of stacks available for IRQ handlers and Int13h. The standard STACKS= command for ROM-DOS increases stack space for the DOS stacks only.

Syntax

DEVICE=STACKDEV.SYS

Remarks

Under various conditions, a stack overflow error occurs while booting ROM-DOS, usually during the CONFIGY.SYS or AUTOEXEC.BAT file processing. For example, some other program may service an interrupt and not provide its own stack for that purpose. If, inside its interrupt handler, the program consumes a lot of stack space by allocating many automatic variables, calling other interrupt handlers, or by deeply nesting sub-routines, then the interrupt handler may overflow the stack of the program that it interrupted. This situation usually results in a system crash.

The STACKDEV.SYS device driver handles stack support similar to MS-DOS' internal Stacks support. STACKDEV increases the stack space available for IRQ handlers and Int13h by

switching to a local stack each time an interrupt occurs. STACKDEV does not allocate the stacks from a pool, instead, each interrupt on the list is given exactly one stack with a fixed stack size of 128 bytes.

STACKDEV differs from the internal STACKS= support with ROM-DOS in that the STACKS= command is used to allocate stack space for ROM-DOS' own internal stacks and does not add additional stack support for IRQs.

Examples

```
DEVICE = STACKDEV.SYS
```

Loads the STACKDEV.SYS device driver during CONFIGY.SYS processing. This driver should be loaded early in the file so that it can handle the IRQ stack needs of other device drivers or programs.

STACKS

CONFIG.SYS Command

The STACKS command enables the dynamic use of data stacks to handle hardware and software interrupts that use large amounts of stack space. You may use this command only in your CONFIG.SYS file. Use the STACKS command if the system crashes or encounters a stack overflow during the boot or at runtime.

STACKS uses more RAM for the DOS stacks, which you can calculate with the formula (number-of-stack*stack-size). The maximum extra DOS stack size is 32KB (64*512).

Syntax

STACKS = *n,s*

Remarks

n specifies the number of stacks. Valid values for n are 0 and integers in the range 8 through 64.

s specifies the size (in bytes) of each stack. Valid values for s are 0 and integers in the range 32 through 512.

SUBMENU

CONFIG.SYS Command

The SUBMENU command defines a menu item that represents a secondary menu when selected. This command may only be used within a menu configuration block in the CONFIG.SYS file.

Syntax

SUBMENU=*blockname* [, *menu_text*]

Remarks

The *blockname* argument defines the name of the secondary menu block of commands. The block menu must be defined elsewhere in the CONFIG.SYS file, otherwise, ROM-DOS leaves this item off of the startup menu. The label can be up to 70 characters long and can contain most printable

characters, including spaces, backslashes (\), forward slashes (/), commas (,), semicolons (;), and equal signs(=). Square brackets ([]) cannot be used in blocknames.

The optional *menu_text* argument specifies the text that ROM-DOS displays for this menu item on the startup menu. If this argument is left out, ROM-DOS displays the *blockname* as the text. The *menu_text* can be up to 70 characters long and may contain any character.

The submenu can be defined with any user-provided descriptive label. It need not have the [MENU] label.

Example

```
[MENU]
menuitem=Word_Proc, Word Processing
menuitem=Network, Network
submenu=Research, Research and Development
menucolor=15,1
menudefault=Word_Proc,20
   [WORD PROC]
   files=10
   buffers=10
   lastdrive=m
   device=c:\network\loadnet.sys
[NETWORK]
include=Word_Proc
numlock=off
   [RESEARCH]
   menuitem=proj1, Project 1
   menuitem=proj2, Project 2
   menudefault=proj1
   [PROJ1]
   files=50
   buffers=25
   numlock=on
   [PROJ2]
   files=10
   buffers=20
   device=vdisk.sys 64 /e
   numlock=off
```

In the preceding example, a submenu is defined as one of the startup menu choices. When you select Research and Development from the first menu, a secondary menu is displayed, offering the choices of Project 1 and Project 2. The actual commands for Project 1 and Project 2 are defined in the configuration blocks labeled PROJ1 and PROJ2.

SUBST

External Command

The SUBST command allows one drive to appear as another drive. This is useful for creating a consistent drive letter when the drive and/or path may change.

Syntax

SUBST [*d*: [*drive:path* | /D]]

Options

The /D switch disables the substituted drive letter (un-installs it)

Remarks

SUBST without any options displays the currently substituted drives. The path may be any legal DOS path, including network drives.

Examples

```
SUBST e: b:\subdir
```

SUBSTitute drive E: to use B:\SUBDIR

```
SUBST
```

Displays all SUBSTituted drives.

```
SUBST e: /D
```

Drive E: is no longer SUBSTituted.

SWITCHES

CONFIG.SYS Command

The SWITCHES command allows special CONFIG.SYS file options.

Syntax

SWITCHES=[/k][/n][/f]

Remarks

The /k argument makes an enhanced keyboard behave like a conventional keyboard.

The /n argument prevents the use of the F5 and F8 function keys to bypass the startup commands.

The /f argument instructs ROM-DOS to skip the delay after displaying the "Starting ROM-DOS..." message at boot time. The delay allows you time to use the F5 and F8 options to alter the processing of the CONFIG.SYS and AUTOEXEC.BAT files as described in page 16.

Example

```
switches = /n
```

Prevents you from using the F5 and F8 keys at boot time.

SYS

External Command

The SYS command copies the ROM-DOS system files ROM-DOS.SYS and COMMAND.COM from the disk in the default drive to the disk in the specified drive. The file ROM-DOS.SYS is renamed and stored on the disk as files IBMBIO.COM and IBMDOS.COM, which are stored as hidden files.

Syntax

SYS *drive:* [*/options*]

Remarks

Use the SYS command to transfer the ROM-DOS system files to a floppy disk or hard disk. The disk can be a formatted blank disk or can contain files; it is not necessary for the system files to be the first files on the disk. The only requirement is that there is enough contiguous free space on the disk for the new system files to be placed. If the disk already contains system files, installing the new system files deletes the existing files.

The command processor, COMMAND.COM, is also transferred to the disk and does not need to be copied into the same contiguous space as the system files.

You can run SYS three different ways. The first is to boot and run your system with ROM-DOS. When you run the SYS command this way, SYS copies the ROM-DOS system files and COMMAND.COM from the root directory of the default/current disk drive.

The second method is to run SYS from the root directory of a disk drive that has been previously prepared with the SYS command, but isn't booted and running. For example, you can run SYS from a bootable floppy disk to copy the files to the hard disk without actually booting from the floppy disk itself.

The third method uses the file ROM-DOS.SYS, the equivalent of the hidden system files IBMBIO.COM and IBMDOS.COM. ROM-DOS.SYS should be present in the same directory with COMMAND.COM and SYS.COM. These three files can be placed in the root directory or subdirectory on a floppy disk (that need not be booted or bootable), or in a subdirectory on the hard drive. Run the SYS command from the directory where the files reside to transfer the system files to the destination drive.

Options

The /C option prevents confirmation before transferring system files.

The /H option shows the newly transferred system files on the destination disk.

The /] option prevents display of the sign-on message.

Example

```
SYS B:
```

Copies the ROM-DOS system files to drive B:.

TIME

Internal Command

The TIME command displays the current time as shown on the system's internal clock. Allows resetting of the clock.

Syntax

TIME [*hh:mm:ss*] [pm|am]

Remarks

The time set by this command is used, among other things, for time stamping your file revision dates. This information is displayed when you execute a directory listing of your files.

You may want to include the TIME command in your AUTOEXEC.BAT file to set the date at boot time. If your computer has an internal, battery-operated clock, you won't need to do so.

The format of the time command is also dependent on the country specified in CONFIG.SYS. The time is displayed according to local standards for the specified country. Refer also to the DATE command.

If you just want to check the time maintained by ROM-DOS, enter the TIME command alone. ROM-DOS displays something like this

```
Current time is 3:00:02.48p
Enter new time:_
```

After which you press Enter to return to an empty command line.

When you want to change the time, you can include the desired time on the prompt line after the word TIME. Or you may enter the command with no option (as you do to check the time) and enter the new time before pressing Enter.

ROM-DOS displays the time according to the 24-hour clock with the *a* or *p* indicator to show AM or PM. The AM / PM indicator can be entered as *a* or *p* or as *am* or *pm*. The time may be entered in a 24-hour format or a 12-hour format with the AM or PM designators.

The allowed options for hours and minutes are

hh = 0-24 *mm* = 0-59 *indicator* = a, p, am, or pm

ROM-DOS displays time to hundredths of seconds. When entering time, however, you needn't enter seconds or hundredths; ROM-DOS assumes a value of zero if they are not specified.

You may skip the display and prompting by typing the current time after the word TIME on the command line

```
TIME 23:24
```

ROM-DOS accepts your entry as the current time.

Examples

To set the time to 11:15 p.m., enter

```
TIME 23:15 or TIME 11:15 p
```

To set the time to 9:26 a.m., enter

```
TIME 9:26 or TIME 9:26 am
```

TREE

External Command

The TREE command displays each subdirectory and, optionally, the files within them for a specified drive.

Syntax

TREE [*drive:*] [*/options*]

Remarks

The TREE command displays the full path of each subdirectory on a specified disk.

drive: specifies the drive that TREE displays the subdirectories from. This argument must be specified.

Option

The /F option causes TREE to display the files in each subdirectory.

Examples

```
TREE C:
```

Displays all subdirectories on drive C:.

```
TREE A: /F
```

Displays all subdirectories on drive A: along with the files within each subdirectory.

TYPE

Internal Command

The TYPE command displays the contents of a text file on the screen.

Syntax

TYPE [*drive:*] [*path*] *filename*

Remarks

If a file containing formatting codes or other nonalphanumeric characters is displayed with TYPE, unintelligible characters are displayed. This does not harm the system.

Example

```
TYPE A:AUTOEXEC.BAT
```

Displays the AUTOEXEC.BAT file on drive A.

UMBLINK.SYS

Config.sys Command

UMBLINK builds upper memory blocks (UMBs) which have a distinct MCB chain and may or may not be linked into the DOS MCB chain. UMBLINK can be used on some systems where EMM386 can not be used.

Syntax

DEVICE=UMBLINK.EXE [*x=mmmm-nnnn]*

Remarks

UMBLINK allows for the conversion of various RAM memory regions into DOS UMBs into which device drivers and TSRs may be loaded. UMBLINK is a simple device driver that, once installed, does nothing but filter the Int21h, Function 5803h calls to link and unlink the UMB arena from the normal DOS MCB chain. UMBLINK must appear before any DEVICEHIGH lines in your CONFIG.SYS File.

Unlike EMM386.EXE, UMBLINK.EXE is not a protected mode control program. It finds certain kinds of memory that already exists in the adapter space and links that memory into the DOS upper memory chain. It then becomes resident and, like Datalight's EMM386, becomes the UMB LINK handler for ROM-DOS. The memory types that UMBLINK can recognize are S-ICE.EXE UMBs (eliminating the need for Numega's UMB.SYS), XMS UMBs, and pre-existing RAM that is already either physically or logically addressable within the adapter space at the time the program is run.

Options

The x=mmmm-nnnn argument specified a hex segment range of existing RAM to be excluded from the UMB conversion process. Both mmmm and nnnn must be in the range C000:0h to F000:0h.

Examples

```
DEVICE=UMBLINK.SYS X=C000 - C800
```

Load UMBLINK.SYS and exclude the range from C000:0h to C800:0h from the UMB creation process.

VDISK

Installable Device Driver

VDISK is a device driver that allows you to use RAM as a disk.

Syntax

device = VDISK [*size* [*secs*[*dirs*]]] [/E]

Remarks

VDISK partitions some of your computer's memory as a disk. This disk is called a RAM disk or virtual disk and is much faster than either a floppy or hard disk. The RAM disk can use either standard DOS program memory or extended memory (above 1MB) for the disk. Any data on the VDISK is lost when the system power is turned off.

The *size* argument specifies the size of the VDISK in kilobytes. The default is 64KB. The memory selected is allocated from the DOS memory pool, decreasing the amount of memory available for programs unless the extended memory switch is used.

The *secs* argument specifies the sector size in bytes. The default is 512 bytes per sector. This value must be 128, 256, 512, or 1024. All other values are not valid, and the default of 512 bytes is used.

The *dirs* argument specifies the number of root directory entries. The default value is 64 directory entries. There may be any number of root directory entries between 2 and 1024. If an odd number is given, it is rounded up to the nearest multiple of 16 to fill the entire sector.

The /E argument causes VDISK to use extended memory (memory above the 1MB boundary) instead of DOS program memory for the disk.

The VDISK driver increases the resident size of DOS.

Note: Interrupts are turned off during the transfer of data from extended memory to conventional memory.

Examples

```
device = VDISK.SYS
```

Builds a 64KB RAM disk in DOS memory.

```
device = C:\DOS\VDISK.SYS 220 /E
```

Builds a 220KB RAM disk in extended memory. The VDISK device driver is loaded from the C: drive and the \DOS directory. VDISK assumes the default 512 byte sector size and 64 directory entries.

```
device = VDISK.SYS 45 128 18
```

Builds a 45KB RAM disk in DOS memory. There are 128 byte sectors and 18 root directory entries.

VER

Internal Command

The VER command displays the version number of the ROM-DOS in use and allows revision of this version number.

Syntax

VER [*n.nn*] [/R]

Remarks

If a new version number is specified, two digits after the decimal are required. Note that this command revises only the record of the DOS version number; it does not change the actual operating system loaded in the computer.

The version command shows both the version of the VER command itself and the version of DOS in operation.

Option

The /R option shows the full version and revision number of ROM-DOS.

Example

```
VER 5.0
```

Changes the record of the current DOS version in use to DOS 5.0. Any programs that are executed, following this command, recognize that DOS 5.0 is running.

VERIFY

Internal Command

The VERIFY command displays or modifies the VERIFY state.

Syntax

VERIFY [ON | OFF]

Remarks

The VERIFY command does not perform any data verification (same as the COPY /V option). It is included to provided batch file compatibility.

VERSION.SYS

CONFIG.SYS Command

VERSION.SYS modifies the version number that ROM-DOS reports. This device performs the same function as the Internal VER command. The difference is that VERSION.SYS allows the change to occur during CONFIG.SYS processing so that version-specific device drivers can load properly.

Syntax

DEVICE=VERSION.SYS n.nn

Remarks

Specify the new version number, *n.nn*, with two digits after the decimal. Note that this command revises only the DOS version number record; it does not change the actual operating system loaded in the computer.

The version number change can be verified after booting using the VER command.

Example

```
DEVICE=VERSION.SYS 5.0
```

Changes the record of current DOS version in use to DOS 5.0. Any programs that are executed, following this command, recognize that DOS 5.0 is running.

VOL

Internal Command

The VOL command displays the volume label on a specified disk.

Syntax

VOL *<drive:>*

Remarks

If you do not specify a drive, the current drive is assumed. VOL does not allow the setting of volume labels. Refer to the LABEL command on page 62 for instructions on setting the volume labels.

Examples

```
VOL
```

Causes ROM-DOS to display the volume label on the default drive, which is the A: drive.

```
VOL C:
```

Causes ROM-DOS to display the volume label on the C: drive.

XCOPY

External Command

The XCOPY command copies multiple files and, optionally, subdirectories from one disk to another.

Syntax

XCOPY [*source*] [*target*] [*/options*]

Remarks

Use the XCOPY command to copy multiple files and subdirectories, if they exist.

The *source* and the *target* parameter are complete drive-path and file-specification descriptions. If you do not specify a path, XCOPY assumes the default path. If a filename is not specified, then *.* is assumed.

The ATTRIB command may be used to modify the archive bit for the various XCOPY options that check the archive status of files. Refer to the ATTRIB command on page 28 description for instructions.

Options

The /A option copies only source files that have the archive bit set in them. The archive is not reset.

The /D<mm-dd-yy> option copies only those files with a date later than that specified.

The /E option creates subdirectories on the target even if they are empty.

The /M option copies only those source files that have the archive bit set. Once the source file is copied, the archive bit is reset.

The /P option prompts before each file is copied. The prompt appears as follows; enter Y to copy the file:

```
C:\COMMAND.COM (Y/N)?
```

The /S option copies files in subdirectories of the source directory.

The /V option verifies each write to the disk.

The /W option waits before starting to copy files and prompts with the following message.

```
Press any key to begin copying file(s)
```

Example

```
XCOPY \bin\*.exe a: /a
```

Copies all files in the BIN subdirectory to the A: drive that have an .EXE extension and that have the archive bit set.

Appendix A, The TRANSFER Program

TRANSFER is a file-exchange utility that allows embedded systems to upload and download files over a serial link using the XMODEM protocol. The program running on the host system may be either the COMM terminal program, another of Datalight's serial communications utility, or another instance of TRANSFER. Refer to 'Appendix B, The COMM Program' for information regarding the COMM program.

TRANSFER may be used with the Datalight BIOS to TRANSFER files by means of the console, in cases where the console is implemented by means of a serial port. In this case, TRANSFER uses BIOS Int 10h function E and Int 16h functions 0 and 1. In systems having a BIOS that does not supply a remote console, TRANSFER may also be used to transfer files over the same serial link as that used for the CTTY console. A specific serial port (COM1, COM2, and so on) can also be specified for TRANSFER to use for file transfers.

To move a file between systems, run TRANSFER on the target system. Either TRANSFER or COMM may be run on the host PC. If COMM is running on the host PC, press the PgUp key on the PC for COMM to send a file to the target system. COMM prompts for the filename and the protocol for TRANSFER; specify the Xmodem protocol. If you are using TRANSFER on the host PC, select the COM port, the baud rate, and specify either send or receive.

Run TRANSFER as follows:

```
A>TRANSFER [Options] FileName
```

The *FileName* parameter specifies the file to be uploaded or downloaded. A path and drive letter may precede the filename. Wildcards are not allowed in the *FileName* parameter.

The options for TRANSFER are listed in the following table.

Option	Description
/S	Sends a file.
/R	Receives a file.
/B#.	Sets the baud rate. The *rate* number may be 300, 1200, 2400, 4800, 9600, 19200, 38400, 57600, or 115200. The default rate is 9600.
	Specifying the baud rate causes TRANSFER to use a serial port rather than the default console.
	The default port is COM1; it can be changed using the /COM option.
/BC	Uses the BIOS console to transfer files (instead of CTTY).
/COM#	The /COM# tells TRANSFER to not Disable use of the console and allow the user to select the COM port.
	The COM number (#) may be either 1, 2, 3 or 4.
	This option (as well as /B#) causes TRANSFER to use interrupt driven serial I/O.
	An NS8250/16450/16550 or compatible UART is assumed at standard PC addresses.

Option	Description
	If the baud rate option is not used along with the /COM option, the default baud rate is 9600.
/Q	Prevents the display of the output to the screen.
/IRQ	Allows changing of the COM port IRQ value from the default setting. The default values are IRQ4 for COM1 and COM3; IRQ3 for COM2 and COM4.

When transferring files over the console, TRANSFER uses DOS calls, by default, to allow operation with CTTY. On some smaller systems it may be preferable to use a BIOS console interface to achieve higher throughput. The /BC option tells TRANSFER to use the BIOS' console interface rather than that of DOS.

TRANSFER Program Examples

The following example receives a file via the console. The data of the file is placed on drive B: in a file named *file.dat*.

```
A:>transfer /r B:file.dat
```

This example sends the file JUNK.ABC over COM4 at 1200 baud, using IRQ 11.

```
A:>transfer /s /B1200 /COM4 /IRQ11 junk.abc
```

The following two examples show the use of TRANSFER on both the host PC and the target system. The file ED.EXE is sent from the host PC to the target system. The file received on the target system is named VI.EXE.

Target system command:

```
A:>TRANSFER /r  B:VI.EXE
```

Host PC system command:

```
A:>TRANSFER /s C:\BIN\ED.EXE
```

Appendix B, The COMM Program

The COMM communications program provides the ability to communicate with a remote ROM-DOS system. COMM supports Xmodem file transfer, autodialing, Zmodem, and terminal emulation and time zones.

Command Line Options

All command line options must be separated by a space.

Option	Description
/B#	Sets the baud rate to # on startup. The available baud rates are 300, 1200, 2400, 4800, 9600, 19200, 38400, 57600, and 115200.
/COM#	Sets the communications port to COM1 or COM2; both are supported.
/8N1	Sets the serial port to 8 data bits, no parity, one stop bit.
/7E1	Sets the serial port to 7 data bits, even parity, one stop bit.
/8N2	Sets the serial port to 8 data bits, no parity, two stop bits.
/8O1	Sets the serial port to 8 data bits, odd parity, one stop bit.
/8E1	Sets the serial port to 8 data bits, even parity, one stop bit.

You can reverse the placement of the parity and data bits on the command line. For example, /8N1 is equivalent to /N81.

Environment Variables

The environment variable COMM is supported, which may set certain communications parameters. The switches are identical to the command line options, for example:

```
SET COMM= /COM2 /B2400 /7E1
```

Causes COMM to start using COM2, at 2400 baud, with 7 data bits, even parity, and 1 stop bit. If new options are specified on the command line, they override the environment variable settings. Invalid options are ignored.

An additional environment variable for time zones is also supported. The TZ variable allows you compensate for time zone differences when sending and receiving files and when the time stamp on a received file is critical. This variable only affects transfers done with the Z-modem protocol. X-modem transfers are not affected. By default, if no time zone is set, the Datalight utilities COMM and RSZ assume Greenwich Mean Time (GMT).

Setting the TZ variable is not necessary when using all programs. If you are transferring files between two Datalight utilities, the file time stamp is not affected. Some utilities, such as Windows Hyper-terminal, automatically convert the time stamp on the file to GMT and assumes that the receiving program will adjust the time to the correct local time upon receipt.

The syntax for the TZ environment variable is:

SET TZ= <*abbreviation*> +|- *value*

Abbreviation represents any three-letter abbreviation for the chosen time zone. The variable serves as a reminder to the user. For example, if setting the time zone for Pacific Standard Time, the variable could be set as PST; and for Eastern Standard Time as EST. The abbreviation is only a placeholder in the syntax for the TZ variable. There are no incorrect abbreviation choices as long as only three letters are used.

Value represents the number of hours this time zone varies from GMT. For example, the west Coast of the United States is –8 hours relative to GMT. This value may have to be adjusted to compensate for daylight savings time. There should be no spaces between the abbreviation, plus or minus sign, and the value. Some examples are:

```
SET TZ=PST-8
SET TZ=CMT-3
SET TZ=GMT+2
```

If an incorrect format for the time zone is entered, the default of GMT is used.

On a desktop PC running Windows programs, such as Hyper-terminal, determining the time zone is part of the setup of the operating system. However, if COMM is run from a DOS box, the TZ variable still will need to be set on the target machine.

COMM Commands

Enter most commands by pressing an Alt-key combination. Some commands take effect immediately (such as changing the baud rate), while others require further information (such as a filename) before continuing.

If you do not want to execute a command, or want to stop a command while it is running (such as a file transfer), press the Esc key to return to terminal mode.

Command	Description
Alt+B	Sets the baud rate. This command toggles between all the available baud rates. Continue to press Alt+B until the desired baud rate appears on the status line at the bottom of the display.
Alt+C	Clears the display.
Alt+D	Autodial allows entry of number to be dialed. Press Enter to redial the previous number.
Alt+E	Toggle echo (duplex). Press Alt+E to toggle the duplex between full (echo off) and half (echo on).
Alt+H	Hangs up if the modem is capable of hanging up with an ATH0 command.
Alt+P	Toggles through the available parameters allowing settings to be made. Esc cancels the command.
Alt+T	When enabled, pressing the Enter key generates a CR/LF instead of just a CR.

Command	Description
Alt+X	Exits the program. This command does not drop the carrier, so use this command if you need to do MS-DOS operations while online. You can run COMM again without losing the carrier and continue with telecommunications.
PgUp	Sends a file to a remote computer, giving you the option of either Xmodem or ASCII file transfer protocols. Esc cancels at any time during the transfer.
PgDn	Receives a file using the Xmodem or ASCII file transfer protocol. Press Esc to cancel at any time during the transfer.

Terminal Emulation

Currently, COMM supports a subset of the ANSI escape codes that compose the only terminal emulation available. These escape codes, A, B, C, D, H, J, and K, should meet most needs since the emulation includes such features as cursor positioning and erase to the end of the line and/or page.

File Transfer Recovery with Zmodem Protocol

The Zmodem file transfer protocol has the ability to resume a set of file transfers at the point of interruption, such as in the case of a call hang-up or disconnected cable. In the event of a failed Zmodem upload or download, run COMM as follows to resume the file transfer:

- Press the PgDn key to initiate a receive-file operation
- Select Zmodem in the file transfer option list.
- Select Y to enable the ZMODEM crash recovery option.

Auto downloads do not use crash recovery.

Appendix C, The RSZ Program

RSZ.EXE is a Zmodem file transfer utility used to transfer files over a serial port to another machine running the Zmodem file transfer protocol. RSZ.EXE can be used in place of the COMM program and can be started from within a DOS batch file to send and receive files In addition, RSZ.EXE does not require that the system have a video display as does the COMM program. RSZ.EXE uses approximately 24KB of RAM. The syntax for RSZ is:

RSZ /Pn /Bn /Hn /Fn /[R|[S file1 [file2 ...]]]

Some examples of RSZ usage include:

```
RSZ /R
RSZ /P3 /B115200 /H2 /C./F0 /S a.b lmno.p z.*
```

RSZ Program Options

All command line options must be separated by a space.

Option	Description
/P	Port number:n = 1, 2, 3, or 4 for COM1 to COM4 (default setting is 1 for COM1)
/B	Baud rate:n = 50,110,300,..115200.
	Always N81 when changed. If this option is not specified, program uses the current port parameters.
/I*nn*	IRQ Numbers – valid IRQ selections are 3 – 15. Default value for COM2 and COM4 is IRQ3; the default value for COM1 and COM3 it is IRQ4.
/H	Handshaking options.
	Both sides must use the same value: 0 = none (default), 1 = software, 2 = hardware.
/F	File management options (all files are binary):
	0 = skip, 1 = resume, 2 = make duplicate, 3 = replace (default)
/R	Receive files specified by sender using Zmodem protocol.
/S	Send the specified files using Zmodem protocol. Wildcards are allowed.
/V	Verbose switch forces the status display to ON when the console is redirected to a second COM port, and allows the Esc key to break out of a transfer.
	Esc will not work when RSZ is using the same COM port as the remote console.
/Q	Do not display/send status information while transferring files. This option is automatically selected when running CTTY or when running ROM-DOS on the Datalight BIOS to prevent errors due to the placement of status data in the transfer data stream.

File Transfer Recovery

For Zmodem file transfer recovery to work after a failed transfer, you must use RSZ with the /F1 option on the receiver command line. This option selects the resume file management option.

Time Zones

An additional environment variable for time zones is also supported. The TZ variable allows you compensate for time zone differences when sending and receiving files and when the time stamp on a received file is critical. This variable only affects transfers done with the Z-modem protocol. X-modem transfers are not affected. By default, if no time zone is set, the Datalight utilities COMM and RSZ assume Greenwich Mean Time (GMT).

Setting the TZ variable is not necessary when using all programs. If you are transferring files between two Datalight utilities, the file time stamp is not affected. Some utilities, such as Windows Hyper-terminal, automatically convert the time stamp on the file to GMT and assumes that the receiving program will adjust the time to the correct local time upon receipt.

The syntax for the TZ environment variable is:

SET TZ= <*abbreviation*> +|- *value*

Abbreviation represents any three-letter abbreviation for the chosen time zone. The variable serves as a reminder to the user. For example, if setting the time zone for Pacific Standard Time, the variable could be set as PST and for Eastern Standard Time as EST. The abbreviation is only a placeholder in the syntax for the TZ variable. There are no incorrect abbreviation choices as long as only three letters are used.

Value represents the number of hours this time zone varies from GMT. For example, the west coast of the United States and Canada is –8 hours relative to GMT. This value may have to be adjusted to compensate for daylight savings time. There should be no spaces between the abbreviation, plus or minus sign, and the value. Some examples are:

```
SET TZ=PST -8
SET TZ=CMT -3
SET TZ=GMT+2
```

If an incorrect format for the time zone is entered, the default of GMT is used.

On a desktop PC running Windows programs, such as Hyper-terminal, determining the time zone is part of the setup of the operating system. However, if COMM is run from a DOS box, the TZ variable still will need to be set on the target machine.

Appendix D, NED (Editor) Program

The NED editor is a menu-based text editor available for use with ROM-DOS. This editor is similar to other desktop editors but has special functions designed for use in editing C-source and assembly code.

Starting the Editor

To start the editor, enter

```
NED [filename] [filename]
```

NED may be initiated with or without filename arguments. Wildcard file specifications are allowed.

Up to ten files can be entered on the command line. If NED is run without arguments, it loads all files accessed during the last editing session, returning you to the exact position in the file. You can switch between the open files.

You can also enter

```
NED @errfile
```

where *errfile* is the name of your compiler error output file. NED loads all files that had errors and allows you to move between errors.

Once NED is running, you may load files into memory by using the File/Open menu command. File/Reload replaces the current file with a new file or reloads a new copy of the same file. File/Reload confirms before replacing an unsaved file.

Basic Editor Operation

NED uses the standard Windows interface for cut, copy, and paste operations. Del and Shift+Del both move the selected block to the clipboard. There is no true undo command, but Ctrl+V or Shift+Ins may be used to paste the clipboard contents to the current cursor position. Table 1 lists the all the default shortcut keys.

If a search string is all lowercase, NED treats it as a case-insensitive search. If a search string contains any uppercase letters, it is case sensitive. The replacement string is inserted exactly as entered. Repeating a Search command repeats the last Forward or Backward Search operation, not the last Replace operation.

There is one bookmark for all files. Once the bookmark is set, going to the bookmark returns you to the file and position where you set it.

The Indent and Remove-indent (referred to as Undent in the Options/Do Command) commands work on tabs. Indent inserts a tab at the beginning of the current line, or if a block is active, at

the beginning of each line in the block. Remove-indent removes the first tab from the current line or from each line in the block. If there are no tabs, Remove-indent has no effect.

Toggle case inverts the case of the current character if no block is active. If a block is active, Toggle case sets the entire block to uppercase if the first character was lower and to lowercase if the first character was uppercase.

Tabs are currently set to 3 for .C, .H, .CPP, .HPP, and .T files. They are set to 8 for all other files.

File/Print prints the current block if there is one, otherwise it prints the current file. NED prompts for a device to print to, which may be a filename. Tabs are expanded to spaces.

The Options/Do Command is intended primarily for debugging. This command allows you to execute any editor command by choosing it from a menu list.

The macro commands (Record Macro/Play Macro) allow you to define a sequence of keystrokes that can be repeated repetitively. Select Record Macro (ALT=), enter the keystrokes, then press ALT= again. The macro sequence can be played by selecting Play Macro or by pressing ALT-. Keyboard bindings are saved in NED.CFG in the same directory as NED.EXE. NED.CFG also contains the list of active files and positions.

If you record and play a recursive macro, it plays continuously.

If you press an invalid key on a menu, NED operates as if you pressed enter.

If you run out of memory, such as when you have more than 300KB of files open, NED returns to DOS.

Remote Editing

NED will operate as a full-screen editor, even through a serial port, this using ANSI Escape codes. Any communication program capable of emulating an ANSI terminal will work with NED in remote mode.

NED automatically detects if the console is redirected through a serial port, either via CTTY, or when using the Datalight BIOS with a serial console. NED does not support ANSI key codes, so the use of PC function keys and standard PC cursor keys is supported through control keys. To use the special control keys, copy the NEDREMOT.CFG to the name NED.CFG in the same directory that NED.EXE is run from on the target system. This NED configuration file was created using the Option Map a key... function, and can be modified in the same manner.

Always use the Esc key to get to the menus. Use Ctrl-K to enable/disable blocking mode when selecting text. The remote key mapping is provided in the following list.

Ctrl-A	Left arrow
Ctrl-B	Find backward

Ctrl-C	Copy to clipboard
Ctrl-D	Go to mark
Ctrl-E	Delete to end of line
Ctrl-F	Find forward
Ctrl-G	Go to line number
Ctrl-H	Delete previous character (same as Backspace)
Ctrl-I	Insert tab (same as Tab)
Ctrl-J	Page down
Ctrl-K	Toggle block mode (for cutting to clipboard)
Ctrl-L	Delete the entire line
Ctrl-M	Insert return (same as Enter)
Ctrl-N	Toggle insert/overwrite mode
Ctrl-O	Open a file
Ctrl-P	Toggle through previous 3 positions
Ctrl-Q	Home
Ctrl-R	Search/Replace
Ctrl-S	Right arrow
Ctrl-T	Top of document
Ctrl-U	Page up
Ctrl-V	Insert clipboard at cursor
Ctrl-W	Up arrow
Ctrl-X	Delete to clipboard
Ctrl-Y	End of document
Ctrl-Z	Down arrow
Ctrl-[	Menu/Cancel operation (same as Esc)
Ctrl-]	Brace match
Ctrl-\	Do a command (opens a menu with all NED commands)

Troubleshooting Remote NED

If nothing appears on the terminal screen, check the baud rate of the terminal program, check the serial cable (should normally be a null-modem cable), and check that the terminal program is set to emulate ANSI escape codes.

In some cases, it is possible for the remote auto-detect to fail. In this case, run the program NEDREMOT prior to running NED. NEDREMOT sets a word at 40:E8h to inform NED to operate remotely.

Default Hot Keys

Many of the editor commands can be accessed directly by pressing key combinations. For example, press Alt-X to exit the editor and save any open files. The following table lists the default hot keys. You can redefine the commands and keys using the Bind HotKey command available on the Options Menu.

Key	Function	Key	Function
Alt-Q	Quit without saving	F1	Help
Alt-X	Exit, saving as needed	F7	Load file into current buffer
Ctrl-A	Search again	F9	Save file
Ctrl-B	Search backward	F10	Exit asking for save as needed
Ctrl-C	Copy the block to clipboard	Left-Arrow	Left one character
Ctrl-D	Find the mark	Right-Arrow	Right one character
Ctrl-E	Erase to end-of-line	Up arrow	Up one line
Ctrl-F	Search forward	Down arrow	Down one line
Ctrl-G	Go to a line number	Home	Beginning of line
Ctrl-I	Indent the block	End	End of line
Ctrl-K	Toggle block mode	Page Up	Up one screen
Ctrl-L	Delete line to the clipboard	Page Down	Down one screen
Ctrl-M	Set the mark	Center (5)	Center the cursor onscreen
Ctrl-N	Read a file into a new buffer	Ctrl-Left-Arrow	Left one word
Ctrl-P	Move to the previous position	Ctrl-Right-Arrow	Right one word
Ctrl-Q	Quote the next character	Ctrl-Up-Arrow	Up one C function
Ctrl-R	Replace text	Ctrl-Down-Arrow	Down one C function
Ctrl-S	Switch to the next buffer	Ctrl-Home	Scroll toward beginning of file
Ctrl-T	Toggle the case of character(s)	Ctrl-End	Scroll toward end of file
Ctrl-U	Remove indent from the block	Ctrl-Page Up	Beginning of file
Ctrl-V	Insert the clipboard	Ctrl-Page Down	End of file
Ctrl-W	Delete word to the clipboard	Ins	Toggle Insert/Overwrite mode
Ctrl-X	Delete block to the clipboard	Del	Delete character
Ctrl-Z	Cancel the selected block	Backspace	Delete character backward
Alt =	Start/end recording macro	Ctrl-Ins	Copy block to clipboard

Key	Function	Key	Function
Alt -	Playback macro	Ctrl-BackSpace	Delete word backward
Alt-F7	Previous error	Shift-Ins	Insert the clipboard
Alt-F8	Next error	Shift-Del	Delete block to clipboard

Appendix E, Remote Disk Program

The remote disk program allows you to access a disk drive on a remote system via a serial cable and standard PC-style (8250UART) serial port. Remote disk allows added flexibility for diskless systems and systems tight on available space.

In a remote disk setup, one system, the one that shares its drives, is termed the server. The other system, the one that accesses and uses the remote drives, is called the client. The serial ports on both systems must be connected via a null modem cable. Remdisk/Remserv works across a standard 3-pin serial cable, similar to other Datalight serial I/O utilities (COMM and TRANSFER). The cable does not require the CTS/RTS DTS/DTR pins.

Server Program

The server system runs the program REMSERV.EXE that can make a single drive on the server system available to the client. The available drive can be changed at any time by quitting the REMSERV program and then running the program again with a new drive letter. The syntax of REMSERV.EXE is

```
REMSERV.EXE d: [/Bnnnn] [+|-] [/COMn] [/Tnnn] [/S] [/H]
```

where d: represents the letter of the drive the server makes available to the client.

Option	Description
/B*nnnn*	Selects the baud rate for transmission. Available baud rates are 300, 1200, 2400, 4800, 9600, 19200, 38400, 57600, and 115000. The default baud rate is 115000.
+/-	The plus sign (+) specifies packet-style transmission and is recommended for any baud rate over 19200. The default setting is to include + for packet transmission. Use the minus sign (-) to specify polling operation under Windows95.
COM*n*	Selects the communication port. Available ports are 1 and 2. COM1 is the default port.
/IRQ*n*	Set the IRQ for the communications port. Valid settings are 3 – 15. Default is IRQ3 for COM 2 and COM4, and IRQ4 for COM1 and COM3.
/T*nnn*	Sets the time-out in the range of 2 to 3,640 seconds.
/S	Instructs REMSERV to run without any display output.
/H	Selects hardware handshaking for flow control. To select drive B: as the available server drive at 115000 baud, packet transmission, using COM1, enter `REMSERV B:` To set drive C: as the server disk at 9600 baud, without packet-style transmission, on COM2, enter: `REMSERV C: /B9600 /COM2`

The server program can be terminated at any time by pressing the Esc key. The client can then no longer access the server's drive until the REMSERV program is run again.

Client Program

The program REMDISK runs on the client system and creates a new drive letter for the client. REMDISK uses the next available system drive letter. For example, if the last assigned drive was D:, REMDISK creates drive E:. This drive acts like any other drive, except that it requires the serial port. REMDISK.EXE can be loaded by a DEVICE= command in the CONFIG.SYS file can be entered at the DOS prompt. The syntax for REMDISK is:

```
REMDISK [/U] [/H] [/Bnnnn] [+|-][/Tnnn] [/COMn]
```

Option	Description
/U	Unloads REMDISK from memory, thereby disabling the drive letter and freeing the memory occupied by REMDISK. This option can only be used when REMDISK is installed from the DOS command line. A remote disk installed via CONFIG.SYS cannot be unloaded.
/H	Selects hardware handshaking for flow control.
/B*nnnn*	Selects the baud rate for transmission. Available baud rates are 300, 1200, 2400, 4800, 9600, 19200, 38400, 57600, and 115000. The default baud rate is 115000.
+/-	The plus sign (+) specifies packet-style transmission and is recommended for any baud rate over 19200. The default setting is to include + for packet transmission. Use the minus sign (-) to specify polling operation under Windows95.
/T*nnn*	Sets the time-out in the range of 2 to 3,640 seconds.
/IRQ*n*	Set the IRQ for the communications port. Valid settings are 3 – 15. Default is IRQ3 for COM 2 and COM4, and IRQ4 for COM1 and COM3.
COM*n*	Selects the communication port. Available ports are 1 and 2. COM1 is the default port.

To install the REMDISK program from CONFIG.SYS at 19200 baud, on COM1, using packet-style transmission, insert the following line in CONFIG.SYS and then reboot the system (remember to include the full path to find REMDISK.EXE if not located in the root directory).

```
DEVICE=REMDISK.EXE /B19200 +
```

To display a help screen for REMDISK from the DOS prompt, enter

```
REMDISK /?
```

To install REMDISK from the DOS prompt or from a batch file (such as AUTOEXEC.BAT) at 9600 baud, without packet-style transmission, on COM2, enter

```
REMDISK /B9600 /COM2
```

To unload the REMDISK installed from the batch file or the DOS prompt, enter

```
REMDISK /U
```

Using the Remote Disk

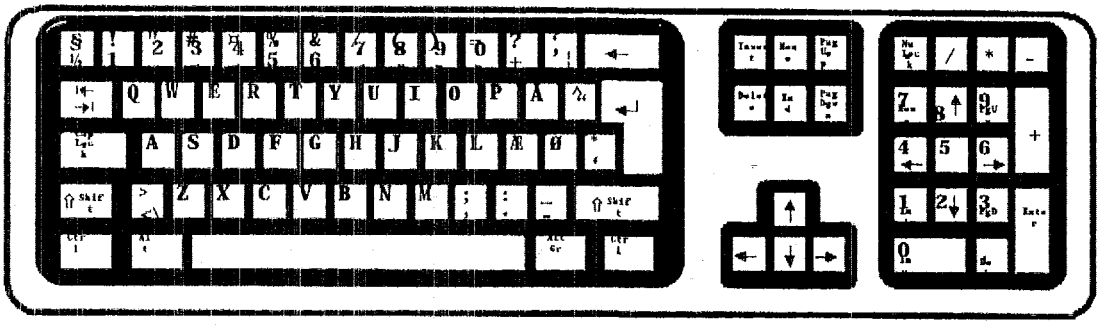
To use the remote disk, both REMDISK and REMSERV must be running on their respective systems and must use the same baud rate and packet or nonpacket-style transmission. After starting both programs, you can access the new drive on the client system. You can change the default directory to this new drive, copy files to and from the remote drive, and also run utilities such as CHKDSK on the drive. The remote drive on the server system can be used as any other drive on the client system.

Appendix F, Keyboard Layouts

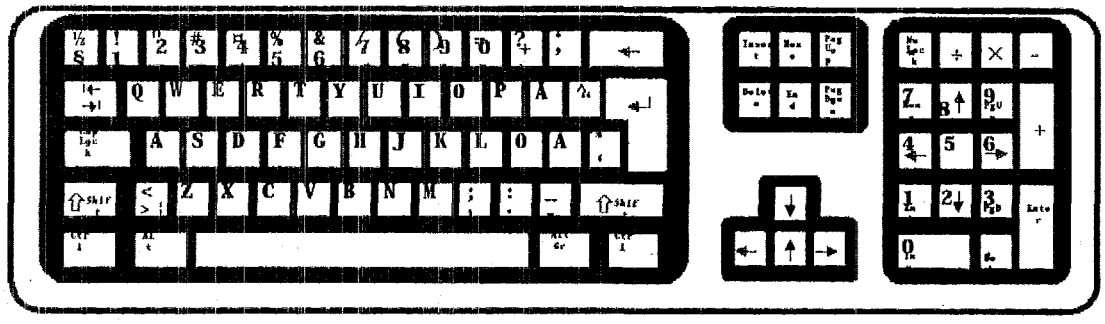
Keyboard Layouts

The following keyboard charts represent several countries supported by ROM-DOS. If you need a keyboard layout not displayed in this Appendix, please contact Datalight.

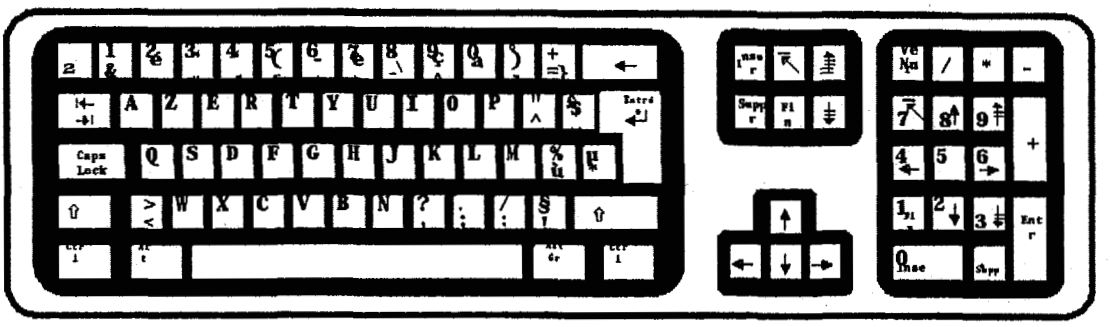
Denmark



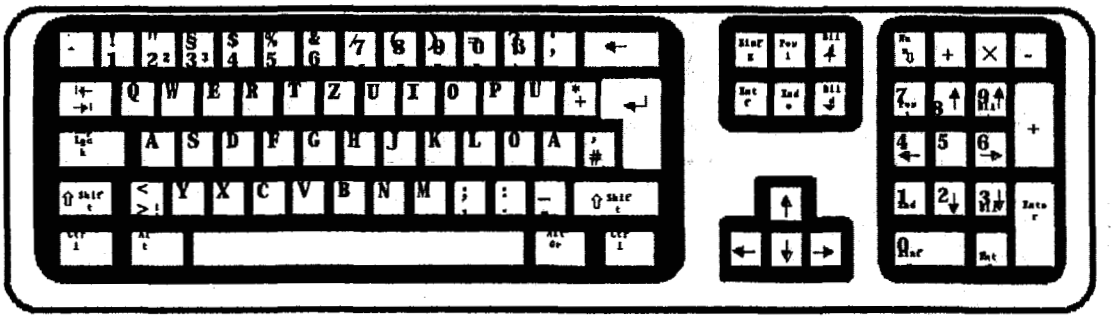
Finland



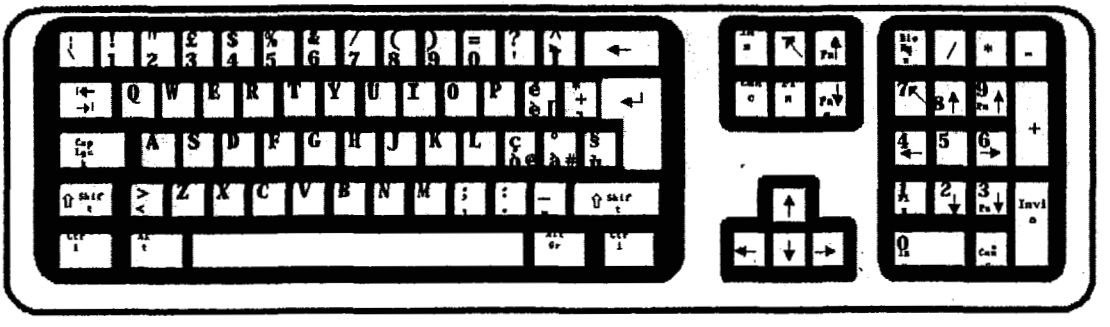
France



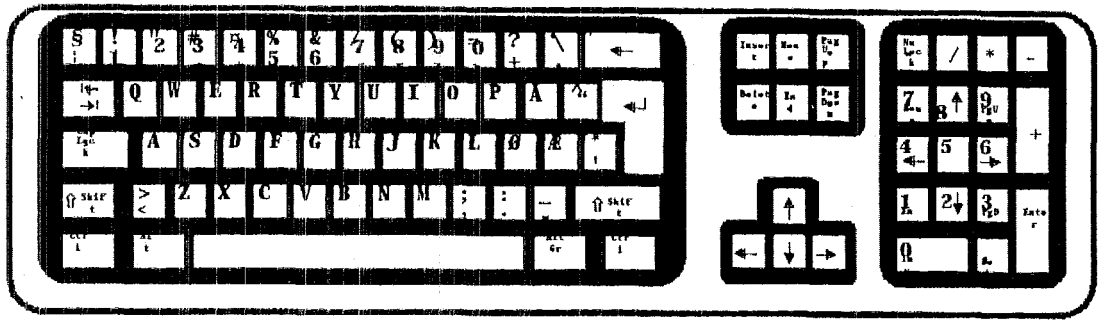
Germany



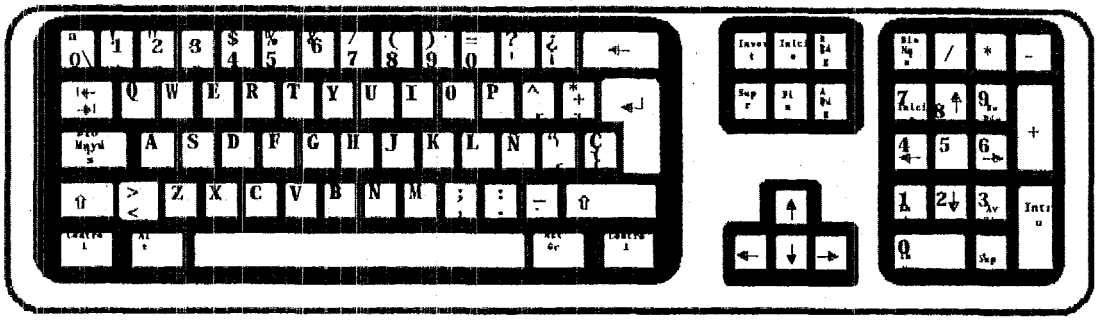
Italy



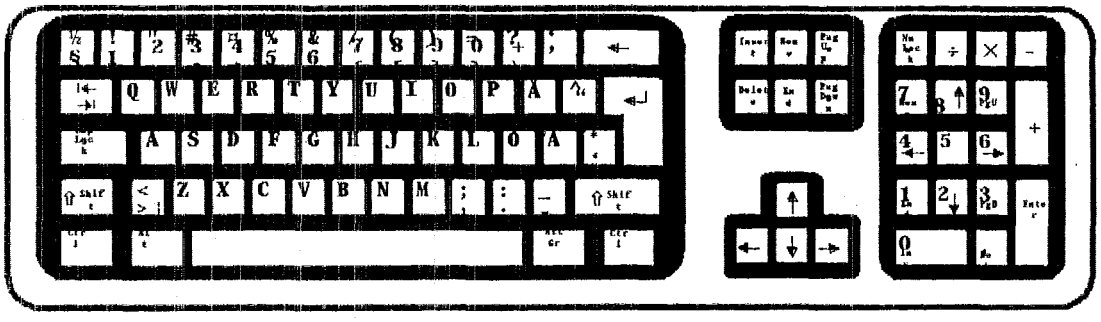
Norway



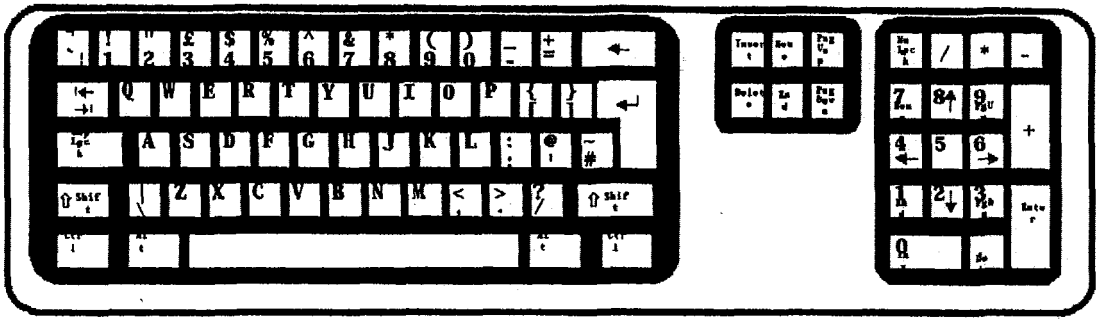
Spain



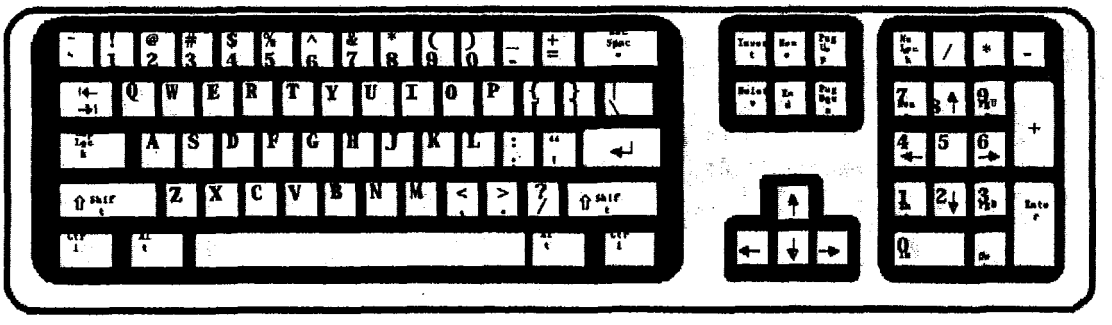
Sweden



United Kingdom



United States



Index

D

E

F

R

S

T

U

V

W

X

Z

6.1.11 ROM-DOS Developer's Guide Manual

The ROM-DOS Developer's Guide is included in the ROM-DOS Software Development Kit, which was purchased.

Copyright and export control restrictions limit options for including the manual in this documentation set to the following.

- A copy of the purchased manual can be included because an extra kit was purchased to allow this copy while honoring the copyright. Since this requires purchase of one kit per distribution copy, any information copies of this documentation will not include the courtesy copy.
- An unopened copy of the entire ROM-DOS Software Development Kit can be included as an enclosure.
- Independent purchase of the ROM-DOS Software Development Kit can be recommended.

6.1.11 ROM-DOS Developer's Guide Manual

The ROM-DOS Developer's Guide is another booklet included in the ROM-DOS Software Development Kit. It describes how to build and install the ROM-DOS operating system. This documentation explains how the ROM-DOS operating system works directly from the PROM without a disk drive in the computational block. It also explains how to create a ROM disk, which the computational block uses.

Datalight ROM-DOS™ 6.22

Developer's Guide

Printed: July 1999

Datalight **ROM-DOS®** Developer's Guide

Part Number: 3010-0200-0027

Contents

Chapter 1, About ROM-DOS

ROM-DOS 6.22, a work-alike of MS-DOS Version 6.*x*, is specifically designed for embedded and mobile computing environments. ROM-DOS allows a developer to easily create an embedded system with DOS functionality using Read Only Memory (ROM), flash, or hard disks. Support for various devices is easily added using software device drivers. ROM-DOS works equally well on a minimal system with limited hardware resources; or a complete system with a hard disk, CD-ROM disk, flash disk, PCMCIA drive, floppy disk, display and keyboard. ROM-DOS operates in conjunction with a standard PC-style Basic Input Output System (BIOS) or with the Datalight BIOS for diskless embedded systems. ROM-DOS runs on the AMD ELAN and the Intel 386EX, as well as on work-alike CPUs such as the NEC V-series, and Chips and Technology's PC chip.

For a rugged and low-cost system, you may choose to eliminate floppy disks and hard disks. ROM-DOS supports the use of memory disks, like ROM disks, in place of physical disks. The ROM-DOS SDK includes a variety of utilities for placing an application in ROM as well as full source for all device drivers. ROM-DOS, provided in library and executable form, offers the following advantages:

- Fully DOS 6.*x* compatible.
- Executes directly from within ROM.
- Uses a minimum of memory (as little as 50KB ROM, 10KB RAM).
- Configurable both at compile time and run time (via CONFIG.SYS).
- Per copy royalty is a fraction of the price of MS-DOS.
- Designed specifically for the embedded system developer.
- Does not require COMMAND.COM to boot and run the target program.
- Supports installable device drivers via CONFIG.SYS.
- Directly supports LS-120 drives without the need for drivers.
- Full source code available.
- ROM-DOS makes no assumptions about hardware, and performs all access through the BIOS.
- New Century Date Rollover - Some BIOSs do not handle date rollover from the year 1999 to 2000. ROM-DOS now automatically detects and corrects the wrong date problem. To correct this problem for older ROM-DOS or MS-DOS systems, you can request a TSR driver named RTC2000.EXE from Datalight.

ROM-DOS Target System Requirements

The target system is the hardware in which ROM-DOS will be running. At a minimum, ROM-DOS requires that the target system include (additional memory may be required for a BIOS and/or a emulated disk drive):

- any Intel x186 or compatible CPU.
- a minimum of 50KB ROM (if in ROM environment).
- a minimum of 10KB RAM.
- as few as eight BIOS calls (depending on configuration).

No special hardware or software, other than that specified above, is required by ROM-DOS.

ROM-DOS Development System Requirements

To configure and build a version of ROM-DOS for installation in your target hardware, you'll need Borland TASM and TLINK version 5.2. Compiling source code for device drivers or features such as the mini-command interpreter may also require Borland's compiler BCC. These tools are included in the Datalight Software Developer's Tool Kit (SDTK)

Requesting Technical Assistance

If you encounter a problem in configuring, building, or programming ROM-DOS, please:

- Attempt to resolve the problem by referring to this manual. You can use the table of contents and the index to locate information.
- Check the README file located for any late-breaking changes or additions to the product not covered in the manual.

You can contact Datalight:

- via telephone at **800.221.6630**
- via email at **support@datalight.com**
- via the web at **www.datalight.com**

In any communication with Datalight, be sure to include the version information from the original ROM-DOS SDK installation disks. If you have comments or suggestions about ROM-DOS or documentation, please contact us.

Chapter 2, ROM-DOS Basics

Datalight had three goals when designing ROM-DOS: compatibility, flexibility, and affordability. A compatible DOS remains the key goal for ROM-DOS. If you find a program that does not run correctly under ROM-DOS, but runs under MS-DOS 6.22, please contact our Technical Support Department.

Whether your hardware is PC-compatible or not, with ROM-DOS, your operating system will be compatible. The only requirements for ROM-DOS are RAM, ROM and an 80x186 or higher CPU (including the NEC V-series). ROM-DOS can take full advantage of all hardware in your system, including large hard drives, CD-ROM drives, flash memory and PCMCIA cards.

ROM-DOS performs all interaction with the hardware through device drivers. These drivers, provided in full source, work as you'd expect from a desktop DOS, whether your system is a small embedded computer, palmtop, or a workstation. ROM-DOS provides a DOS platform on virtually any system.

The following sections describe the various software components that make up a complete system using ROM-DOS. This includes a general description of DOS-based computer system, as well as some design ideas to aid in hardware selection.

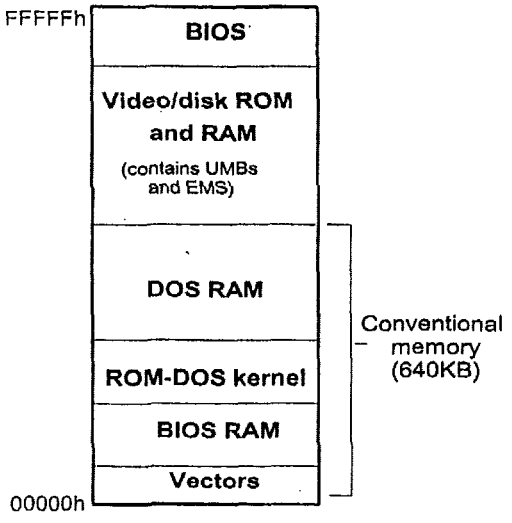
The Major Software Components

At a minimum, ROM-DOS requires a BIOS and a command interpreter to boot the system. The command interpreter can be Datalight's COMMAND.COM (which provides the familiar C:> prompt), or just an application that ROM-DOS runs directly. This program is typically run from a ROM disk on diskless systems, or from a floppy/hard disk on systems that have them.

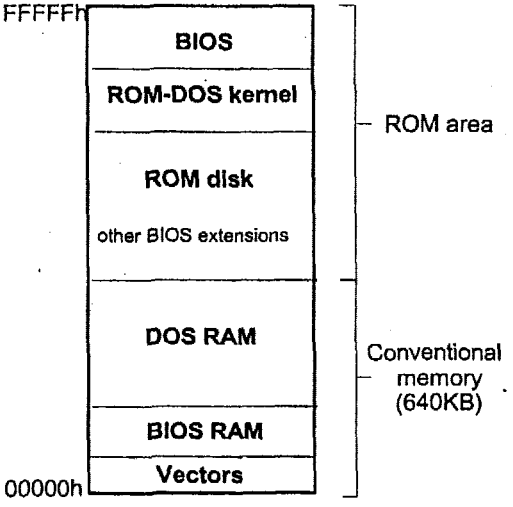
The BIOS gains control as power is applied to the computer, initializes hardware and RAM, then passes control to ROM-DOS. ROM-DOS then determines what hardware support is available through its device drivers and loads your application or a command interpreter to complete the boot process.

The BIOS always resides in ROM (or some other non-volatile memory). The ROM-DOS kernel can reside and run in ROM, or be loaded from a disk. The command interpreter, COMMAND.COM, or the application program resides on a disk, either floppy, hard, ROM, RAM, flash (Datalight's FlashFX), CD-ROM or any other disk that DOS can access.

The following illustrations show the locations of the software components in a typical embedded system and in a desktop PC.

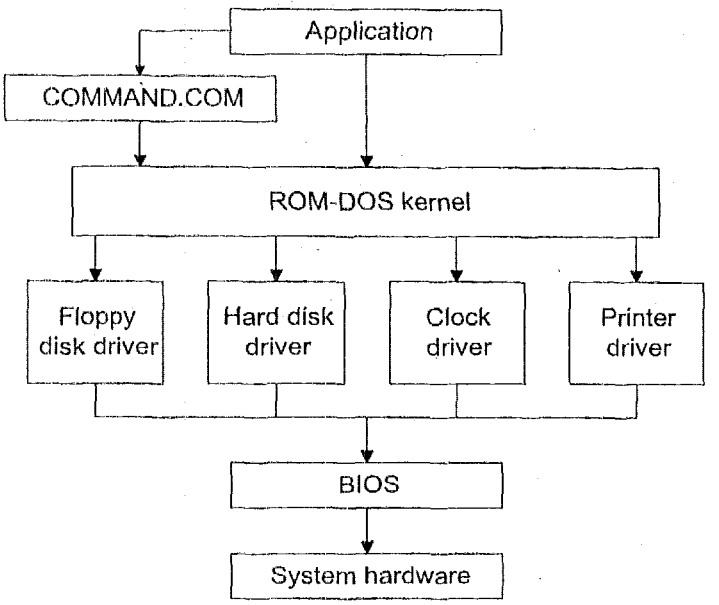


Typical PC System Memory Layout



Typical Embedded System Memory Layout

The following illustration depicts the interaction of software components. The user application communicates with the command interpreter and/or ROM-DOS. These operating system components then communicate with the appropriate device drivers. The device drivers communicate with the BIOS which then makes requests to the system hardware.



Software/Hardware Hierarchy

BIOS

BIOS is an acronym for Basic Input/Output System. All I/O in a system goes through the BIOS, unless the application ties directly to the hardware. The BIOS program is placed in ROM in every desktop computer and any system that can run DOS. It acts as the interface between DOS and the hardware, as well as performing such important issues as starting the computer from a power off state. Other functions of the BIOS include initializing any hardware required for the system to run (such as disk drives, the monitor, and so on) and loading DOS.

ROM-DOS performs all I/O operations through its device drivers. The device drivers in turn use BIOS calls. Whether ROM-DOS is writing to the disk or printer, reading from the keyboard, getting the amount of available RAM or time of day, ROM-DOS uses BIOS calls. By using BIOS calls to communicate with the hardware, ROM-DOS does not need to be aware of hardware details.

Most standard PC motherboards include a BIOS specifically configured to operate with the hardware on the motherboard. However, if you are developing proprietary hardware, you can use the Datalight BIOS which includes support for ROM-DOS.

ROM-DOS Kernel

The ROM-DOS kernel is the heart of ROM-DOS. The kernel provides file and directory management, input and output through character devices (console, serial port, and printers), along with time and date support. The kernel also provides the ability to load and execute programs, manage memory, and make country information available to applications.

The primary purpose of the kernel is to provide the DOS call services (Int 21h) to programs. The kernel also processes the CONFIG.SYS file during its initialization process and loads the initial program run by ROM-DOS. This program is by default the command interpreter, COMMAND.COM, although it may be any application program.

The ROM-DOS kernel, like the BIOS, can execute directly from ROM and does not need to copy its code into RAM. Like the command interpreter and other programs, the ROM-DOS kernel can also load from disk and run in RAM. On disk-based systems the ROM-DOS kernel files are named IBMBIO.COM and IBMDOS.COM and are hidden from view. You can list these hidden system files at the command line prompt with the following DIR command.

```
C:\>DIR /as
```

Command Interpreter

The ROM-DOS SDK includes a program known commonly as the command interpreter or shell. The program, named COMMAND.COM is, in most cases the first program loaded by ROM-DOS after boot up. COMMAND.COM provides the C:\> prompt interface, batch file processing, DIR, ERASE, and other commands. COMMAND.COM allows the user to enter commands using a keyboard and shows the results of the commands on the display.

The primary duties of the command interpreter are processing the AUTOEXEC.BAT batch file as it is starting up, executing internal commands, loading user programs, and executing batch (.bat) files. The internal commands include DATE, DIR, COPY, TIME, TYPE, and many others.

COMMAND.COM is not the only command interpreter available. The Norton Utilities™, for instance, provides a replacement command interpreter named NDOS. The command interpreter is loaded by an entry in the CONFIG.SYS file such as:

```
SHELL=NDOS.COM
```

Datalight also offers a smaller version of COMMAND.COM called mini-COMMAND. This command interpreter requires only 4KB of ROM or disk space, as opposed to the 34KB of Datalight's full command interpreter.

ROM-DOS can boot directly into any compatible program without the need for a command interpreter. Because ROM-DOS systems are sometimes dedicated to a single program, it makes sense to load that program directly, without the additional overhead of COMMAND.COM.

Using a ROM Disk

On systems that have no physical rotating or solid-state disk, a ROM disk can be used to support ROM-DOS. ROM-DOS contains a built-in ROM disk driver along with the more familiar floppy and hard disk drivers. A ROM disk utility, named ROMDISK.EXE, creates memory images that includes the files you specify. To use the ROM disk utility, specify a directory tree and ROMDISK.EXE creates an image file suitable for your PROM programmer, complete with all the files and the directory structure contained in the directory tree. This ROM disk image can be placed in the same or different ROM as ROM-DOS.

A ROM disk in a diskless system usually contains COMMAND.COM, user applications and data files. From the point of view of ROM-DOS, the ROM disk is nothing more than a fast write-protected floppy disk drive.

Another type of disk is a memory disk. Many MS-DOS developers have used RAMDRIVE.SYS or some other RAM disk equivalent, such as Datalight's VDISK, to speed development. A ROM disk is just a read-only RAM disk. Both RAM and ROM disks are memory disks which, with some special software (a memory disk device driver), appear to DOS as a conventional disk drive. The directories and files for the disk are located in memory rather than on rotating magnetic media.

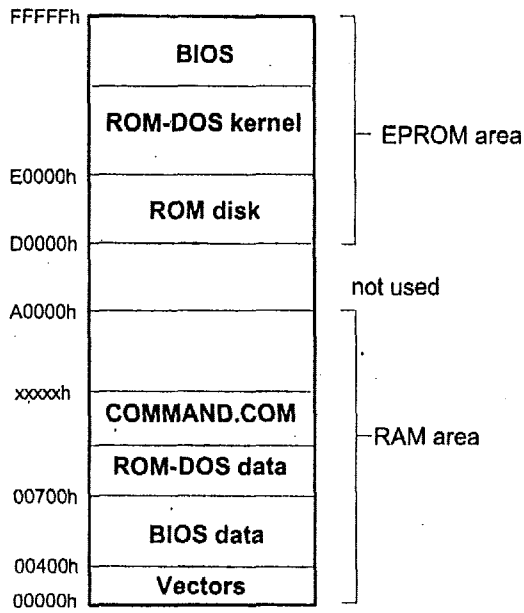
Placing ROM-DOS in a ROM

There are three separate ROM files — the ROM-DOS kernel, ROM disk, and BIOS — that may be programmed into the target system ROM before attempting a boot. When placing ROM-DOS in ROM, its kernel must be configured to execute from ROM. The ROM-DOS SDK contains a utility, BUILD.EXE, to configure the ROM-DOS kernel for a ROM or disk environment.

If the system requires a ROM disk, run the ROMDISK.EXE utility to create the ROM disk image. If the ROM disk is being programmed into ROM, the ROM-DOS ROM disk driver works as-is for ROM disks placed within the 1MB real address space. If your ROM disk is to be placed above the 1MB boundary or your hardware is designed for memory paging, a customized memory disk driver must be created to correctly access the ROM.

If the system requires either the Datalight BIOS or a custom BIOS, then this ROM must be created and placed on the system.

The following ROM memory map diagram shows a typical memory layout for embedded ROM-DOS system.



ROM-DOS In ROM Memory Map

DOS on a Disk

ROM-DOS does not require any reconfiguration for use on a standard PC platform. A new bootable disk can be made using either the Datalight SYS.COM or FORMAT.COM utilities. The SYS.COM and FORMAT.COM utilities place the hidden system files IBMBIO.COM and IBMDOS.COM on the bootable disk along with the command interpreter COMMAND.COM.

Note: To run SYS.COM or FORMAT.COM, you must boot your system from a disk that contains the hidden system files or you can build a ROM-DOS.SYS file (which is equivalent to the hidden system files) as described in 'Chapter 4, Building ROM-DOS' and use it to create a bootable disk.

Using ROM-DOS with Flash Memory

ROM-DOS supports the two basic types of flash memory, PCMCIA cards and on-board flash arrays. ROM-DOS does not directly support PCMCIA cards of any type since this is handled by the BIOS in conjunction with DOS loaded device drivers (generally available from BIOS

vendors) called card and socket services. ROM-DOS supports all popular PCMCIA card and socket services device drivers. Onboard flash arrays can be used as a programmable linear memory area or as a disk with read/write capabilities, when used in conjunction with a flash file manager such as FlashFX.

ROM-DOS's ROM-disk device driver and ROM-disk building program are suitable for creating an image to place in a linear flash memory and then reading the image as a read-only ROM disk (a ROM disk works with either ROM or flash). The disk image must be programmed into the flash memory using a custom flash loader utility (typically provided by the hardware vendor) or using a PROM programmer capable of programming flash devices.

The use of flash memory differs somewhat from ROM. However, there can be advantages. Flash memory can be less expensive and faster than standard ROM, sometimes even faster than RAM. The high speed of flash is advantageous for software such as RXE's (executable programs running in-place in ROM, a feature of ROM-DOS) and when running the ROM-DOS kernel from ROM. Some hardware is even set up so that the flash can be reprogrammed on-board while in the field, offering quick and easy updating.

In addition, ROM-DOS includes an ATA device driver named ATA.SYS that supports a variety of ATA cards. Refer to the file ROM-DOS User's Guide for more information on this driver.

Chapter 3, Installing and Running ROM-DOS

This chapter provides an overview of the process for installing ROM-DOS on a development system and running on a target platform. It also contains several examples of creating ROM-DOS to run on different systems. To create a version of ROM-DOS that runs on any system, follow these steps (described in detail in subsequent chapters of this manual).

- Select the hardware platform.
- Select the BIOS.
- BUILD (configure) ROM-DOS for that system.
- Create a ROM disk (*diskless* systems).
- Program PROMs and install them (*diskless* systems), and reboot.

Installation Procedure (development system)

The development system is used to install the ROM-DOS files from the distribution CD-ROM and then configure and build ROM-DOS to your target system specifications as described in the following chapters.

To install the ROM-DOS SDK software on your development system:

1. Place the CD-ROM in CD drive.
2. Choose Start/Run and enter

   <cd-rom drive>: **setup**

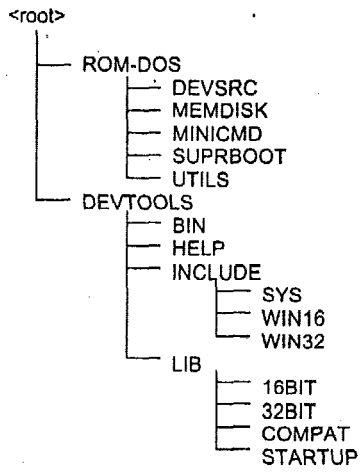
3. Follow the onscreen instructions presented by the setup program to install the required components.

We strongly recommend you install all Datalight software under a DL directory tree.

You should also install the enclosed Datalight Software Developer's Tool Kit (SDTK) in the same manner. When the installation is complete, modify your path statement to include the C:\DL\DEVTOOLS\BIN directory. This path allows the ROM-DOS BUILD utility to locate all necessary tools.

Installed Files

When installation of the ROM-DOS SDK and the SDTK software is complete, ROM-DOS and the Tool Kit software exist on your development system in the following directories/folders.



ROM-DOS Considerations

ROM-DOS runs on most x86 hardware platform, including the 186, 286, 386, 486 or Pentium, as well as the NEC V-series and the growing number of work-alike processors. Companies such as NEC, Vadem and Chips and Technologies offer all the parts of a PC in a single piece of silicon. ROM-DOS contains no hardware dependencies.

Selecting the BIOS to use depends on the availability of a BIOS for the hardware and the ROM space requirements for the BIOS. If a BIOS is already available for your hardware, use it. When selecting the BIOS, consider not only the availability, but the features of the BIOS. The BIOS should recognize the basic hardware you intend to have available in your system. It is generally the responsibility of the BIOS to support PCMCIA cards (at least SRAM cards) through the BIOS disk Int 13h. The BIOS should also support extended memory on those CPUs that can address RAM beyond 1MB.

Building (or more appropriately, customizing) ROM-DOS for any given system requires a minimum of effort. Building ROM-DOS can be as simple as answering a few prompts that the BUILD program displays and then placing ROM-DOS on disk or in ROM. A chapter is devoted to running the BUILD program later in this manual.

When building ROM-DOS, consider such issues as which drive letters you want associated with the disk drives. Will drives A: and B: be assigned to the floppy drives, as ordered on a desktop system? Or will the ROM disk be drive A:, a PCMCIA disk drive B:, followed by floppy and hard disks? ROM-DOS makes no restrictions in this area, so choose what best suits your system.

You may need to make adjustments to the file SYSGEN.ASM described under 'Ordering Floppy and Hard Disk Drives' on page 53 to order the drive letters.

If a ROM disk is needed, either for booting, or as another disk on the ROM-DOS system, use the ROMDISK.EXE program to create this disk. A ROM disk driver, which searches memory for the ROM disk, is built into ROM-DOS.

If ROM-DOS will be placed in ROM, the final step is to program the ROMs and place them in the target machine. At this point, the system can be booted using ROM-DOS.

Systems running ROM-DOS directly from a hard or floppy disk do not need to program any ROMs. Run FORMAT or SYS on the disk and reboot.

Example 1: Creating a Bootable Version of ROM-DOS on a Floppy Disk

This example creates a ROM-DOS on a floppy disk that can be used to boot your system. To begin, insert a formatted floppy disk in drive A. It doesn't matter if there are files on the disk, as long as there is enough free space for ROM-DOS and its command interpreter (about 80KB). If you do not have a formatted floppy disk, use the Datalight FORMAT.COM program to format the floppy.

Change to the ROMDOS directory on your hard disk (or whichever directory you chose to create during the INSTALL) and run the Datalight SYS program located in the UTILS subdirectory.

```
C:\ROMDOS\UTILS>SYS A:
```

The SYS utility creates a bootable disk, creates a special boot sector, and copies the ROM-DOS kernel files and the command interpreter (COMMAND.COM) onto the disk. SYS uses the single file ROM-DOS.SYS, produced by BUILD, to generate the two system files, IBMBIO.COM and IBMDOS.COM. These two files are placed as hidden files on the bootable disk. To verify the existence of these files, you can use the DIR command with the system file attribute options as follows:

```
C:\>DIR A: /AS
```

The SYS program requires that both ROM-DOS.SYS and COMMAND.COM are available in the current directory, or that Datalight's IBMBIO.COM and IBMDOS.COM are in the root directory of a currently booted floppy or hard disk. If you cannot find the file ROM-DOS.SYS, you can easily create it using the Quick-Build selection of the BUILD.EXE program. Refer to the following examples for more information on the building process. If SYS cannot find these files, it prompts you for a path for their location.

You can also use the FORMAT utility to both format a disk and add the system onto it as follows:

```
C:\>FORMAT A: /S
```

Example 2: Creating a Version of ROM-DOS in a ROM

This example uses a standard PC with a ROM card to produce a version of ROM-DOS (in ROM) that may be used on any desktop PC/AT. In this example, ROM-DOS processes CONFIG.SYS and loads COMMAND.COM from the floppy, but boots and executes from ROM. The system files IBMBIO.COM and IBMDOS.COM are not required to be on the floppy disk.

The BUILD.EXE utility is the tool used to create a ROM version of ROM-DOS and prompts with a number of questions during the custom-build session described below. The Quick-Build is usually more appropriate and much easier to run through, but in this example we are booting from ROM but loading CONFIG.SYS from floppy disk.

The following list shows the output that BUILD provides showing the prompts (explained in 'Chapter 4, Building ROM-DOS') and the appropriate responses for this example. The recommended responses are in bold letters.

```
C:\ROM-DOS\BUILD
Build Utility for ROM-DOS v6.22 (Revision 3.00)
Copyright (c) 1989-1999 Datalight, Inc.

Will ROM-DOS boot from Floppy/Hard disk?                    N
Copy ROM-DOS to RAM?                                        N
Where in ROM shall ROM-DOS code reside                      D000
Where shall ROM-DOS data reside                             70
Can a Floppy DOS superscede ROM-DOS in ROM?                 N
Do you want to include the Floppy/Hard disk driver?         Y
Always believe the BPB information?                         N
Include the Generic Memory Disk Driver?                     N
Include the built-in ROM-DISK driver in ROM-DOS?            Y
Do you want to change the default ROM-DISK search segment?  N
Read CONFIG.SYS from a specific drive letter?               Y
Read CONFIG.SYS from which drive letter:                    A
What level of CONFIG.SYS processing (None, 3, 5, 6)?       6
Do you want ROM-DOS boot diagnostics?                       Y
Include the Boot Menu?                                      Y
Use Real Time Clock Exclusively?                            N
Create Binary or Intel HEX file(s) as output (B/H):         B
Split the output into Odd byte and Even byte files?         N

TASM.EXE /DBCHECK=1 /DBEXT=1 /DBOOTDRV=0 /DBOOTMENU=1 /Mx SYSGEN.ASM;
Turbo Assembler Version 4.1 Copyright (c) 1988, 1993 Borland International
Assembling file:     SYSGEN.ASM
Error messages:      None
Warning messages:    None
Passes:              1
Remaining memory     412k
TLINK.EXE @ROM-DOS.LNK
Turbo Line Version 7.1.32.2 Copyright (c) 1987, 1996 Borland International
Warning:  No stack
COMPRESS ROM-DOS
LOC @ROM-DOS.LOC
DATALIGHT 80x86 Locator v2.0
(c) Copyright 1988-1992, Datalight Inc.
  53k ROM at D000:0000 is 99% full (File 'ROM-DOS.img').
Program entry point at D000:0025
The file ROM-DOS.IMG has been created.
ROM-DOS has now been Built to boot from ROM.  This
version of ROM-DOS requires 54k (54928 bytes) of ROM and
```

```
can reside at any address between C000:0 and F000:0.
The ROM-DOS data will reside at 70:0.
ROM-DOS is 100% relocatable so the PROM can be moved
to any address without burning another image.
However, ROM-DOS in ROM boots via BIOS extension so
it MUST reside between the address of C000:0 and F000:0
on a 2K boundary for the BIOS to recognize it.  See
the manual about BIOS extensions.
```

The preceding example assumes you have a ROM board to plug into your desktop PC/AT, an EPROM programmer and a ROM large enough to hold ROM-DOS (54KB). We recommend a 27C512 EPROM (64KB x 8).

Program the ROM-DOS.IMG file into an EPROM, plug it into the ROM board and set the address to D000:0. Plug the board into your desktop PC/AT, place a floppy in drive A: with Datalight's COMMAND.COM on it and apply power. ROM-DOS proceeds to check for a CONFIG.SYS file and COMMAND.COM on drive A: (the floppy). At the DOS prompt, type:

```
A>VER /r
Datalight ROM-DOS Version 6.22
Copyright (c) 1989-1999 Datalight, Inc.
Kernel Revision 3.00
Kernel Resides in RAM
Command Revision 3.00
```

The VER command (with the revision option) displays the ROM-DOS version and where it is running (ROM, RAM or high memory area).

If you want to boot from the DOS on your hard disk, and bypass ROM-DOS in ROM, it is not necessary to remove the ROM card. Hold down the Alt-key while the system boots and ROM-DOS displays a menu of boot options. (Select Yes to the boot menu option during BUILD to activate this feature.) Choose the menu option to boot DOS from hard disk.

Example 3: Creating a Diskless System with ROM-DOS

This example places the Datalight BIOS, ROM-DOS and a ROM disk into ROM on an AT motherboard. The console is supported via the COM1 serial port by the BIOS. The example assumes the AT motherboard has 128KB of ROM space. The Datalight BIOS, the ROM-DOS kernel, and a ROM disk are placed in the ROM. The ROM disk contains the files COMMAND.COM, TRANSFER.EXE, VDISK.SYS and CONFIG.SYS. This example creates binary images for input by the PROM programmer. The image files can be Intel hex files or split files, depending on the needs of your programmer.

The BIOS requires about 28KB of ROM, ROM-DOS requires about 52KB ROM and the ROM disk another 58KB for a total of 138KB ROM space. These sizes may change as new features are added to the BIOS, ROM-DOS or the command interpreter.

These three files are common to most diskless systems.

BIOS (DLBIOS.IMG)

ROM-DOS kernel (ROM-DOS.IMG)

ROM disk (ROMDISK.IMG)

Note: The file DLBIOS.IMG is created using the Datalight BIOS SDK.

The file ROM-DOS.IMG is created using the BUILD program as in the previous example. This example uses Quick-Build instead of the Custom-Build to simplify the operation shown below.

```
Build Utility for ROM-DOS v6.22 (Revision 3.00)
Copyright (c) 1989-1999 Datalight, Inc.
Do you wish to Quick-Build or Custom Build ROM-DOS: Q
Will ROM-DOS boot from Floppy/Hard disk? N
Create Binary or Intel HEX file(s) as output (B/H): B
Split the output into Odd byte and Even byte files?: N
The file ROM-DOS.IMG has been created.
ROM-DOS has now been Built to boot from ROM.  This
version of ROM-DOS requires 54k (54480 bytes) of ROM and
can reside at any address between C000:0 and F000:0.
The ROM-DOS data will reside at 70:0.
ROM-DOS is 100% relocatable so the PROM can be moved
to any address without burning another image.
However, ROM-DOS in ROM boots via BIOS extension so
it MUST reside between the address of C000:0 and F000:0
on a 2K boundary for the BIOS to recognize it.  See
the manual about BIOS extensions.
```

BUILD has now created the file ROM-DOS.IMG. Place this file along with DLBIOS.IMG in your PROM programmer directory. Refer to 'Chapter 4, Building ROM-DOS' on page 20 if you have any difficulty during this stage.

Lastly, create the ROM disk. You can do this by placing the previously mentioned files into a directory tree and running the ROMDISK.EXE utility. Use the following DOS commands:

```
C:\>MKDIR TEMPDIR
C:\>COPY COMMAND.COM TEMPDIR
C:\>COPY TRANSFER.EXE TEMPDIR
C:\>COPY VDISK.SYS TEMPDIR
C:\>COPY CON TEMPDIR\CONFIG.SYS
DEVICE=VDISK.SYS 64 <Ctrl+Z>
```

Now run the ROMDISK.EXE program as shown to create the ROM disk with the files COMMAND.COM, TRANSFER.EXE, VDISK.SYS, and CONFIG.SYS.

```
C:\>ROMDISK TEMPDIR
\COMMAND.COM
\TRANSFER.EXE
\VDISK.SYS
\CONFIG.SYS
ROM disk   Volume 'ROM-DISK    '
Built from C:\ROMDOS\TEMPDIR\*.*
Placed in  ROM-DISK.IMG
    58624 bytes total ROM disk size
      128 bytes in boot sector
      768 bytes in 4 FAT sectors
      256 bytes in root directory
    57472 bytes in 5 user files
        0 bytes available on disk
      128 bytes in each of 458 sectors
```

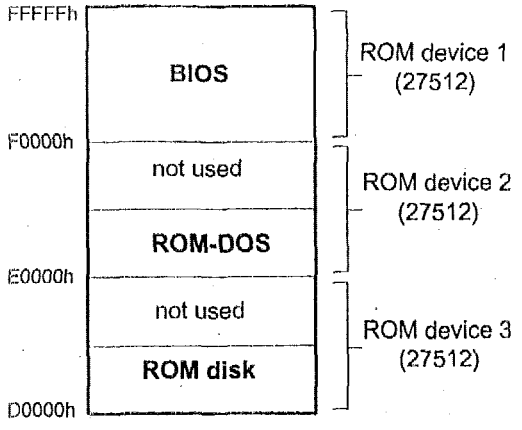
The ROMDISK program creates the file ROM-DISK.IMG. Place this file into the PROM programmer directory along with DLBIOS.IMG, ROM-DOS.IMG and ROM-DISK.IMG. If you have difficulty, see 'Chapter 5, Creating a ROM Disk.'

There are three ROMs to program; one contains the BIOS, the second contains ROM-DOS, while the other contains the ROM disk. All ROMs should be 27512 (64KB) devices.

Run your PROM programmer software and set the appropriate ROM size. Load the file DLBIOS.IMG into the upper 28KB of the ROM. The PROM is placed at F000:0, the upper 64KB of memory.

After the BIOS PROM, load the file ROM-DOS.IMG into the lowest portion of the programmer buffer and program the ROM. This ROM is placed at E000:0.

After the ROM-DOS and BIOS PROM have been programmed, place a 64KB PROM in the programmer and program in the ROM-DISK.IMG file. Place the ROM disk at D000:0. Your memory MAP should look like the following:



Diskless ROM-DOS Memory Map

The ROM-DISK PROM is placed in physical address D000:0 through D000:FFFF. The ROM-DOS PROM is located at E000:0 through E000:FFFF, and the BIOS at F000:0 through F000:FFFF, the high end of memory.

If you have configured the BIOS to run with a remote console for I/O, you can connect a null-modem cable from COM1 on your PC to the serial port on the AT-motherboard (a null-modem normally connects two PCs together). Run the program COMM.EXE (a utility installed with the ROM-DOS SDK) as shown.

```
C:\>COMM /B9600
```

The COMM program defaults to 9600 baud, 8 data bits, no parity, 1 stop bit, on COM1.

Turn on the power to the AT motherboard and the system boots using ROM-DOS. You can type commands to ROM-DOS and have it respond back via the serial port. The serial port is effectively your console.

ROM-DOS is now up and running on a diskless system. This system has a ROM disk as drive A and a 64KB RAM disk (with the help of VDISK.SYS) as drive B. The TRANSFER program placed on the ROM disk allows you to copy programs over the console serial port into the RAM disk.

Chapter 4, Building ROM-DOS

Building ROM-DOS is accomplished with the BUILD utility. BUILD allows you to specify such parameters as the boot drive for ROM-DOS, where ROM-DOS will run (in ROM or RAM); and, if in ROM, what sort of file(s) your PROM programmer requires.

BUILD operates interactively and prompts for information and option selections. BUILD requires an assembler and a linker — we recommend the Borland 5.2 tools provided with the Datalight STDK. BUILD also requires a locator and a specialized program that compresses the ROM-DOS data. Datalight provides these programs, named LOC.EXE and COMPRESS.EXE, with this Software Development Kit. These programs must be available in the current directory or in the specified path.

BUILD performs the following operations:

- Assembles the SYSGEN (ROM-DOS configuration) file
- Links the ROM-DOS kernel
- Compresses ROM-DOS data
- Locates the ROM-DOS kernel

BUILD creates only the ROM-DOS kernel. In the case of a ROM-DOS kernel that is bootable from a floppy disk, the file created is ROM-DOS.SYS. In the case of a ROM-DOS kernel that is bootable from a ROM, the file created is ROM-DOS.IMG or ROM-DOS.HEX.

Most programs, such as the command interpreter and DOS utilities (FORMAT, SYS, and so on), never need to be configured; they are standard across all systems.

Note: Under some circumstances BUILD may not be flexible enough to meet the special needs of your system. For instance, ROM-DOS in ROM normally gains control via BIOS extension, and it may be necessary for ROM-DOS to receive control via a direct jump rather than using a BIOS extension. In this case, a manual build of ROM-DOS is required. Refer to 'Chapter 9, Creating ROM-DOS Without BUILD' for a complete description of manually building the ROM-DOS kernel.

BUILD Command Line Options

Normally, BUILD can be run without command-line options. BUILD determines the appropriate display colors, and finds the assembler and linker. BUILD provides several command-line options described in the following table.

Option	Description
/L	Causes BUILD to locate ROM-DOS, without assembling or linking. Refer to the section 'Chapter 9, Creating ROM-DOS Without BUILD' for information on the use of this option.
/N	Causes BUILD to use monochrome. Non-color displays that appear to be color displays to BUILD, such as LCD displays, may not be readable in full color.
/P	Causes BUILD to pause after running each sub-program. This option allows you to observe what command-lines BUILD is passing to the assembler, linker and so on.
/T	Causes BUILD to display in TTY mode rather than graphics. This option is necessary for incompatible monitor types.

BUILD can rerun the last session using a configuration file. Each time BUILD runs, it saves a list of your keystrokes in a file named BUILD.CFG. This file can then be piped into BUILD to repeat the last session. For example:

```
C:\>BUILD <BUILD.CFG
```

If a number of standard sessions are planned, copy the file BUILD.CFG to some other name. Then pipe the new filename into BUILD.

BUILD also outputs a file named BUILD.TXT. This file contains a complete list of the questions and the answers you selected during the last BUILD session and is the same information as on BUILD's final confirmation screen. BUILD.TXT can be referenced when calling technical support or saved with your project for future reference.

The third output file from BUILD.EXE is BUILD.BAT. BUILD.BAT outputs a complete set of instructions for assembling, linking, compressing, and locating the version of the ROM-DOS kernel set up in the previous run of BUILD. Executing the file BUILD.BAT generates a copy of the previous ROM-DOS kernel without running the BUILD program. BUILD.BAT relies on the existence of two other files, ROM-DOS.LNK (linking command line) and ROM-DOS.LOC (location configuration file). Both files are generated during the BUILD session. To run BUILD.BAT, specify the .BAT extension, otherwise the .EXE extension is assumed and BUILD.EXE runs.

Datalight recommends saving a copy of BUILD.BAT, ROM-DOS.LNK, ROM-DOS.LOC and BUILD.TXT under different names or in a separate directory when you successfully create a working ROM-DOS kernel. This ensures that you can always re-create the same working ROM-DOS kernel configured for your exact needs.

Note: Each revision of BUILD may change, so don't use old configuration files on a new BUILD.

If you want to change the default colors, specify the new colors in a text file named BUILD.COL The colors must be listed as four comma-separated integers, on the first line of the file. The numbers represent the background, window, error, and question colors. For example, to set a gray background with white text, a blue text window with white text, a red error window with white text, and a blue question prompt with yellow text, enter:

```
C:\>COPY CON BUILD.COL
127, 31, 79, 30 <Ctrl+Z>
```

BUILD Sample Sessions

BUILD allows you to create both a floppy/hard disk bootable version of ROM-DOS and a ROM bootable version. You must have the assembler and linker in your path (Borland's TASM/TLINK combination) for BUILD to complete its process. If BUILD does not find an available assembler/linker it warns you and gives you an option to proceed anyway or quit the BUILD process. A pair of examples are shown below. The output from BUILD is shown in block letters. The user-entered responses are shown using large, bold, letters. You can press Esc to exit BUILD at any time.

Example 1: Creating a Disk-Bootable Version

This example uses the concise Quick-Build (as opposed to Custom-Build) and creates a disk-bootable version of ROM-DOS.

```
c:\ROMDOS> BUILD
Build Utility for ROM-DOS v6.22 (Revision 3.00)
Copyright (c) 1989-1999 Datalight, Inc.

Do you wish to Quick-Build or Custom Build ROM-DOS: Q
Will ROM-DOS boot from Floppy/Hard disk? Y

TASM.EXE /DBCHECK=1 /DBEXT=1 /DBOOTDRV=0 /DBOOTMENU=1 /Mx SYSGEN.ASM;

Turbo Assembler Version 4.1 Copyright (c) 1988, 1993 Borland International
Assembling file:    SYSGEN.ASM
Error messages:     None
Warning messages:   None
Passes:             1
Remaining memory    412k

TLINK.EXE @ROM-DOS.LNK
Turbo Line Version 7.1.32.2 Copyright (c) 1987, 1996 Borland International
Warning:  No stack

COMPRESS ROM-DOS

LOC @ROM-DOS.LOC
DATALIGHT 80x86 Locator v2.0
(c) Copyright 1988-1992, Datalight Inc.
52K ROM at D000:0000 is 99% full (File 'ROM-DOS.img').
Program entry point at D000:0000
ROM-DOS has now been built for a Floppy/Hard disk system.
It take 53K (54032 bytes) of disk space.
```

```
The files ROM-DOS.SYS and COMMAND.COM (ROM-DOS.SYS was
created by this build, COMMAND.COM should be found in the
current directory) are used by Datalight's SYS and FORMAT
utilities to create bootable Floppy/Hard disks.

Type "SYS A:" or "FORMAT A:/S" to make a bootable floppy.
```

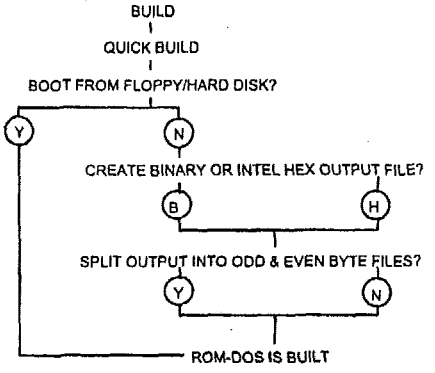
Example 2: Creating an Executable-within-ROM ROM-DOS Kernel

This BUILD example produces a ROM-DOS kernel that resides in (and is executed from) ROM at segment C800. The example uses the Custom-Build option instead of the Quick-Build option.

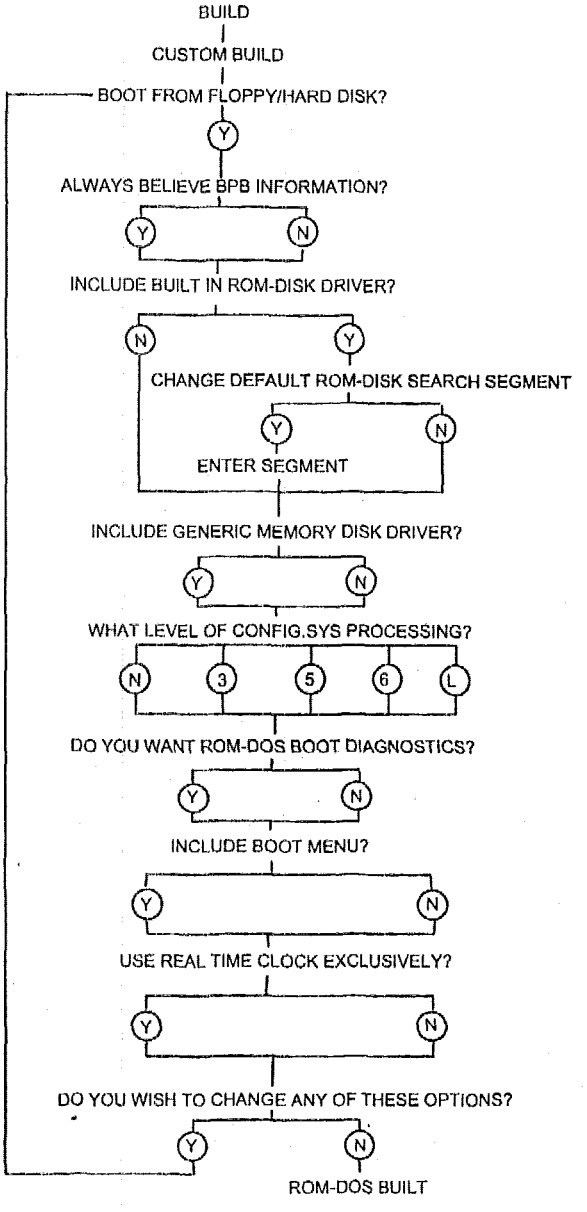
```
C:\ROMDOS> BUILD

Do you wish to Quick-Build or Custom-Build ROM-DOS:          c
Will ROM-DOS boot from Floppy/Hard disk?                     N
Copy ROM-DOS to RAM?                                         N
Where in ROM shall ROM-DOS code reside                       C800
Where shall ROM-DOS data reside                              70
Can a Floppy DOS superscede ROM-DOS in ROM?                  Y
Do you want to include the Floppy/Hard disk driver?          Y
Always believe the BPB information?                          N
Include the Generic Memory Disk Driver?                      N
Include the built-in ROM-DISK driver in ROM-DOS?             N
Read CONFIG.SYS from a specific drive letter?                N
Read CONFIG.SYS from which device (ROM, Floppy, Hard)?       R
What level of CONFIG.SYS processing (None, 3, 5, 6)?        6
Do you want ROM-DOS boot diagnostics?                        Y
Include the Boot Menu?                                       N
Use Real Time Clock Exclusively?                             N
Create Binary or Intel HEX file(s) as output (B/H):          H
Place an Extended Address in the Intel HEX file?             N
Split the output into Odd byte and Even byte files?          N

The file ROM-DOS.HEX has been created.
ROM-DOS has now been Built to boot from ROM.  This
version of ROM-DOS requires 53K (54176 bytes) of ROM and
can reside at any address between C000:0 and F000:0.
The ROM-DOS data will reside at 70:0.
ROM-DOS is 100% relocatable so the PROM can be moved
to any address without burning another image.
However ROM-DOS in ROM boots via BIOS extension so
it MUST reside between the address C000:0 and F000:0
on a 2K boundary for the BIOS to recognize it.  See
the manual about BIOS extensions.
```

Flowcharts for the BUILD Program

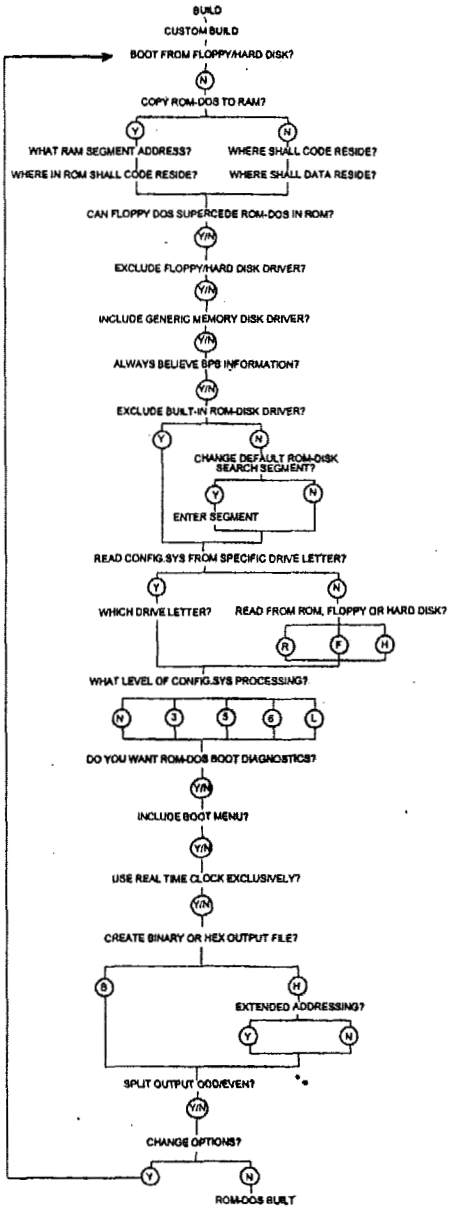
The BUILD program displays decision prompts when creating a ROM-DOS kernel with a brief description of ramifications and recommended responses. Flowcharts provided on the following pages show the decision prompts for Quick-Build, for Custom-Build disk boot, and Custom-Build ROM boot versions of the ROM-DOS kernel. For a complete list of command line options, refer to the earlier section, 'BUILD Command Line Options' on page 20.



Quick-Build Option Flow Diagram



Custom-Build option, Disk Boot Flow Diagram



Custom-Build option, ROM Boot Flow Diagram

Creating a Custom Sign-on Message

ROM-DOS allows for flexible sign-on messages. The standard "Starting ROM-DOS..." message can be customized for use with special evaluation kits, or to allow alternative sign-on screens. To make your own sign-on, follow these steps:

1. Modify the "starting-msg" string in DOSIGNON.C, found in the root directory of the installed SDK.
2. Compile DOSIGNON.C to DOSIGNON.OBJ (linking is not necessary), for example
   ```
   bcc -c dosignon.c
   ```
3. Place DOSIGNON.OBJ in USER.LIB using Borland's TLIB command. Refer to Chapter 6, Including Device Drivers' for an example of using the TLIB command.
4. Run BUILD to create a version of ROM-DOS with the new sign-on message. The new sign-on replaces the "Starting ROM-DOS..." message.

Chapter 5, Creating a ROM Disk

A ROM disk, like a fast write-protected floppy disk, contains all of the parts of a standard disk. Each disk consists of a boot sector, a File Allocation Table (FAT), a root directory, and any files that are to be included on the disk. From the point of view of ROM-DOS and any application, the ROM disk appears as a normal disk drive.

The ROM disk image is built using the ROMDISK.EXE utility which creates ROM disks from a directory tree on your hard disk. The file that the ROMDISK utility outputs is suitable for input to your PROM programmer. The ROMDISK utility can create:

- ROM disks up to 32MB.
- ROM disks with directories and subdirectories.
- ROM disks containing programs that can execute-in-place (refer to Appendix C).

A ROM disk is typically used in diskless systems to hold applications and/or data. A ROM disk is similar to a RAM disk used under DOS, except that it is read-only and always resides in ROM or linear flash memory.

For the ROM disk to be recognized by ROM-DOS, a piece of code called the ROM disk driver must be included within the ROM-DOS kernel. The BUILD program, which creates versions of ROM-DOS specifically for your system, provides the option of including the ROM disk driver. Custom memory disk drives can also be built-in to ROM-DOS or loaded by means of CONFIG.SYS.

Running ROMDISK To Create a Disk in ROM

You can run the ROMDISK utility at the command line by entering ROMDISK with or without command line options. When run without command line options, ROMDISK displays a summary of the available options. ROMDISK allows you to produce a binary image or an Intel hex file, representing the ROM disk. This file is programmed into ROM to create a ROM disk. The size of the ROM disk is only limited by the number and size of the files placed on the ROM disk.

The ROM disk has the contents of a standard disk including a boot record, FAT, root directory and data area. The sector size, which defaults to 128 bytes, may be set by entering the sector size on the ROMDISK command line. There is no limit to the number of files that may be placed on a ROM disk, other than the above mentioned limit of 32MB.

Place all of the files to be included on the ROM disk in a directory (the directory may contain subdirectories). The directory that you create becomes the root directory on the ROM disk. All subdirectories remain at levels below that.

Use the /S option to transfer subdirectories to the ROM disk. Without the /S option, only the files in the root directory are included. The syntax is similar to XCOPY, with the destination always a file containing the ROM disk. The following example places the contents of the TEMPDIR directory, including subdirectories, in the image file DISK.IMG.

Example:

```
C:\ROMDOS> ROMDISK TEMPDIR DISK.IMG /s

\COMMAND.COM
\TRANSFER.EXE
\VDISK.SYS
\CONFIG.SYS
\UTILS\FORMAT.COM
\UTILS\SYS.COM

ROM Disk    Volume 'ROM-DISK    '
Built from C:\DL\ROMDOS\TEMPDIR\*.*
Placed in  disk.IMG

   114944 bytes total ROM disk size
      128 bytes in boot sector
     1408 bytes in 11 FAT sectors
      256 bytes in root directory
      128 bytes in 1 directories
   113024 bytes in 6 user file(s)
        0 bytes available on disk

      128 bytes in each of 898 sectors
```

The file created by ROMDISK.EXE defaults to an image file but can also be an Intel hex file. The created file must be programmed into ROM and located in the 80x86 memory space. The ROM disk must always start on a paragraph boundary.

ROMDISK Options

The ROMDISK utility has the following command line options to configure the output file.

Option	Description
/D<seg>	Specifies an RXE data segment name. The default value for <seg> is DATA.
/E	Prevents extended records from being placed in the Intel hex output. These records are placed in a Hex file by default.
/F#	Sets the fill bytes. The default is a fill byte of 0xFF. The number following the /F option is assumed to be in hexadecimal format.
/H[#]	Produces an Intel Hex file. An optional number following the /H option specifies the actual address of the start of the ROM disk. The start address is required for EXE files that are to be placed in and executed from ROM. The default address is 0x000.
/I[#]	Produces an image file. The optional number following the /I option specifies the actual address of the start of the ROM disk. The start address argument is required for EXE files that are to be placed in and executed from ROM. The default address is 0x0000.

Option	Description
/O	Omits the timestamp from the volume label.
/R#	Specifies the RXE interrupt number. The default number is 0x90.
/S	Includes subdirectories found within the source subdirectory selected to include in the ROM disk image.
/T	Displays statistics on the ROM disk but does not actually create the image or hex file. This is useful when you need to make sure that the required files fit into the available space.
/V"str"	Sets the volume label to something other than ROMDISK. The volume label string can be up to 11 characters and must be in quotes.
/Z#	Specifies the sector size of the ROM disk. The default sector size is 128 bytes. Legal values for this option are 128, 256, and 512.

Placing Execute-in-place Files in ROM

Any Datalight ROM disk (excluding ROM disks compressed with Stacker) can include programs that execute in-place on the ROM disk. This type of executable program is referred to as an RXE (for ROM executable).

Note: Refer to 'Appendix C, Creating RXE-Compatible Programs' for instructions on preparing your application for use as an RXE.

When DOS (ROM-DOS or any other DOS) loads an EXE program, certain values within the EXE image are changed or altered. These values reflect the location of the CODE and DATA within the program. Programs that execute from within ROM cannot be altered, so the altering process must take place before the file is added to the ROM disk. This operation is performed by the RXE converter program.

This SDK contains a program named RXE_CVT.EXE. The ROM disk builder, ROMDISK.EXE, calls RXE_CVT for each program that is to execute in-place from ROM. You must tell ROMDISK.EXE which programs are RXEs and which programs are programs that execute from RAM.

To tell ROMDISK to convert an .EXE program (one that executes in RAM) into an .RXE program (a program that executes in-place from ROM), place the .EXE, and its .MAP file into a subdirectory which contains all the files for the ROM disk. Then, rename the .EXE file to .RXE. This informs the ROM disk builder that the .RXE program should be converted and that its .MAP file should not be included in the ROM disk.

The following example shows all the steps necessary to create a ROM disk with one RXE program.

```
C:\DATALGHT\ROMDOS> MKDIR TEMP
C:\DATALGHT\ROMDOS> COPY COMMAND.COM TEMP
```

```
C:\DATALGHT\ROMDOS> COPY TEST.EXE TEMP\TEST.RXE
C:\DATALGHT\ROMDOS> COPY TEST.MAP TEMP
C:\DATALGHT\ROMDOS> ROMDISK /iE000 TEMP

\COMMAND.COM
\TEST.EXE -- RXE (uses interrupt 90H)
ROM disk   Volume 'ROM-DISK    '
Built from C:\ROMDOS\DEVKIT\TEMP\*.*
Placed in  ROM-DISK.IMG
    33408 bytes total ROM disk size
      128 bytes in boot sector
      512 bytes in 4 FAT sectors
      128 bytes in root directory
     6784 bytes in 1 RXE files
    25856 bytes in 2 user files
        0 bytes available on disk
      128 bytes in each of 261 sectors
```

The .EXE file renamed TEST.RXE in the TEMP directory (in the preceding example) is converted into a true RXE, but is included on the ROM disk under the name TEST.EXE. The file extension must be .EXE on the ROM disk so that COMMAND can run the program. The .MAP file, TEST.MAP, is not included on the ROM disk. The .MAP file was used to convert the TEST program into an EXE that runs from within ROM, but is not needed to run the EXE.

When the EXE within the ROM runs, its behavior differs from a normal EXE. COMMAND tells ROM-DOS to run the program, but the program remains in ROM and is not loaded into RAM.

Each RXE program uses one interrupt vector to help the program load and run. The interrupt vector is setup and used only while the RXE program is actually running, although it can be used for other purposes at other times. Multiple interrupts are used for multiple RXEs; that is, each RXE on the ROM disk has its own interrupt. The default interrupt is 0x90 for the first RXE, 0x91 for the second, and so on. This default interrupt can be changed with the /R# command line option.

It is extremely important to specify the segment (paragraph) address where the ROM disk starts. Failure to do this causes the RXE program to fail. Use the following command line template for ROMDISK.EXE whenever including EXE programs on the ROM disk:

```
C:\DATALGHT\ROMDOS> ROMDISK /iE000 TEMPDIR
```

or

```
C:\DATALGHT\ROMDOS> ROMDISK /hE000 TEMPDIR
```

Configuring the ROM Disk Device Driver

The ROM disk image produced by ROMDISK.EXE can be placed in a system in several ways; in conventional memory (under the 1MB boundary), paged into a window in the 80x86 address space, or in extended memory.

The standard ROM disk device driver provided in ROM-DOS supports a ROM disk in conventional memory, between the addresses of 40:0 and FFFF:0. This standard ROM disk

device driver searches for the start of the disk beginning at a specified segment (usually C000:0). The starting search segment can be specified while building ROM-DOS.

A ROM disk device driver that searches for a ROM disk in paged memory, or extended memory, may be developed by using the code templates provided in the MEMDISK subdirectory. Refer to 'Chapter 7, Using a Custom Memory Disk' on page 43 for more details. If the standard built-in ROM disk driver must be modified, the code can be found in DEVROM.ASM in the DEVSRC subdirectory.

Compressing ROM Disks

The ROM disk device driver is capable of reading ROM disks compressed with the Stac Electronics STACKER utility which can achieve a disk-space savings of 25 to 50 percent or more with little or no speed degradation. A compressed ROM disk is created differently than a standard ROM disk in that the ROMDISK.EXE program cannot be used for the ROM disk. Instead, a RAM disk is created, compressed, and loaded with the desired files and programs. The RAM disk is then converted with the DISK2IMG (disk-to-image) utility into a binary image file for application to your PROM programmer.

Note: RXE programs cannot be placed on a compressed disk.

Creating a Bootable Floppy Disk

The first step for the compressed ROM disk is to make a bootable floppy disk as described under 'Example 1: Creating a Bootable Version of ROM-DOS on a Floppy Disk' on page 13. The bootable disk should contain the Datalight RAM disk driver, VDISK.SYS, the Stac utility STACKER.COM, and a CONFIG.SYS file with the following lines:

```
device=vdisk.sys 128 /e
device=stacker.com @
```

The STACKER.COM utility is explained in detail in 'Appendix B, Stacker Data Compression.' The Stacker utilities can be found on Disk 2 of the SDK. Run the Installation program to install the Stacker utilities on your system. Refer to the ROM-DOS User's Manual for information on the VDISK.SYS driver and its options.

The most significant option for a compressed disk is the size. The disk in the preceding example is 128KB when installed. The VDISK size is usually a multiple of the EPROM size used by the completed ROM disk. For example, two 64KB EPROMS hold a 128KB ROM disk image. The VDISK size does not have to be as large as the sum of the files to be placed onto it since compression allows for more available space. The smallest size VDISK recognized by Stacker is 70KB.

Note: When calculating the size, remember that 1KB is equivalent to 1,024 bytes. Consequently, an image file of 131,072 bytes is 128KB, not 131KB in size.

When the bootable disk is complete, use it to reboot your system. Make note of the drive letter of the installed VDISK. Using the incorrect drive letter with the Stacker commands in the following instructions could cause your hard disk or disk partition to be compressed accidentally.

Your VDISK can now be prepared in one of two methods:

- Create a non-default disk driver; compress the entire disk. Another ROM disk or physical drive must be available for the boot files. A fully-compressed disk cannot be used as the default drive.
- Create a default disk drive; compress a portion of the disk and install boot files on the uncompressed portion of the disk. This method allows the compressed disk to be the default drive and contain the command interpreter and AUTOEXEC.BAT files.

Create a Non-Default Disk Drive - Compress the Entire Disk

The paragraphs that follow provide the details of these steps to create a non-default ROM disk drive using the Stacker utilities.

1. Compress the VDISK.
2. Copy files to the VDISK.
3. Unmount the VDISK (so the real disk is visible).
4. Copy disk image to hard disk with DISK2IMG.
5. PROGRAM image file into ROM or flash memory.
6. Boot ROM disk system with a configuration of ROM-DOS that includes the built-in ROM disk device driver.

The next step in creating a non-default drive is to compress the VDISK. To compress the disk, run the following commands from the directory where you installed STACKER.EXE, where d: is the designation of the VDISK.

```
create d:
stacker d:
```

When the compression process is completed, copy all of the files to be included on the compressed ROM disk to the compressed VDISK RAM drive. When all of the files have been transferred, the VDISK must be unmounted so that it is no longer recognized as a compressed disk and the real disk is visible. To unmount, enter the following command, substituting the appropriate drive letter:

```
stacker -d:
```

The unmounted VDISK can now be used to create a binary image file for your ROM disk. Use DISK2IMG.EXE to convert the ROM disk into a binary image. The syntax for DISK2IMG is:

```
DISK2IMG d: cmpdisk.img
```

where d: is the RAM disk to convert and **cmpdisk.img** is the name of the binary image file.

Refer to your PROM programmer documentation for information on programming the image file into ROM or flash memory. Most PROM programmers can split your image for programming onto a set of EPROMS if needed. For example a 128KB image file can be programmed into two 64KB EPROMS or one 128KB EPROM.

Note: The DISK2IMG utility can convert any disk to an image file. The Datalight ROM disk driver can only recognize a disk image that is created using the instructions in this chapter. The ROM disk device searches memory for a specific signature to identify a ROM disk. The VDISK device driver uses the same signature for marking the disk. A standard floppy disk does not have this signature.

If not already completed, the next step is to BUILD a version of ROM-DOS that includes the built-in ROM disk device driver. ROM-DOS can then be placed into ROM or flash memory or onto a floppy or hard disk drive for booting. The CONFIG.SYS file on the default drive should contain the following instruction to mount the compressed ROM disk drive:

```
DEVICE=stacker.com @ D:
```

The drive letter in the DEVICE statement depends on your system configuration. The drive letter may be different from the designated drive used when creating and compressing the VDISK in earlier steps. For example, if you create your VDISK on a PC with two floppy drives and one hard drive, the VDISK installs as drive D. If the compressed ROM disk is placed on a system having no floppy or hard disks or drivers (only another ROM disk), the compressed ROM disk is installed as drive B:. After booting, the compressed ROM disk is available for use.

Creating a Default Drive ROM Disk

The paragraphs that follow provide the details of these steps to create a default ROM disk drive using the Stacker utilities.

Copy CONFIG.SYS and device files to VDISK. CONFIG.SYS must have Stacker device statement.

1. Compress the remainder of the VDISK.
2. Copy files (including command interpreter) to compressed portion of VDISK.
3. Unmount the VDISK (so the real disk is visible).
4. Copy disk image to hard disk with DISK2IMG.
5. PROGRAM image file into ROM or flash memory.
6. Boot ROM disk system with a configuration of ROM-DOS that includes the built-in ROM disk device driver.

Although the process is similar to creating a non-default drive disk, a few extra steps must be taken to prepare a disk as a default drive ROM disk with compressed areas. The first step is to

place a CONFIG.SYS file on the VDISK which contains the following instruction to mount the compressed ROM disk drive at boot time:

```
DEVICE=stacker.com @ D:
```

The drive letter in the device statement is dependent on your system configuration. The drive letter may be different from the designated drive used when creating and compressing the VDISK in upcoming steps. For example, if you create your VDISK on a PC with two floppy drives and one hard drive, the VDISK installs as drive D:. If the compressed ROM disk will be placed on a system with only another ROM disk (no floppy or hard disks or drivers), the compressed ROM disk will be installed as drive B:.

Next, device drivers, such as STACKER.COM, can be placed on the VDISK. If other devices need to be loaded prior to the Stacker drivers, they must be placed on the uncompressed portion of the VDISK, unless they can be located by an explicit path on another drive available at boot time.

The device statements for other devices should be placed prior to the Stacker device statement in the CONFIG.SYS file. The STACKER.COM device driver can be placed on the VDISK or located by an explicit path as with other drivers. It is best to limit the number of drivers and files placed on the uncompressed VDISK since the less spaced used leaves more available for compression.

When working with smaller VDISKs, select a minimum of files for the uncompressed disk. Stacker requires a large part of the disk to be available for compression and produces error messages if the disk is already too full. If this happens, create a larger VDISK or remove some of the files.

The next step is to compress the VDISK by running the following commands from your STAC directory, substituting the appropriate drive letter for the example drive d.

```
create d:
stacker d:
```

After the compression process is completed, copy the command interpreter COMMAND.COM and, optionally, an AUTOEXEC.BAT file to the VDISK RAM drive. Any application or utility files to be included on the compressed ROM disk should also be copied at this time. When all of the files have been transferred, the VDISK must be unmounted so that it is no longer recognized as a compressed disk and the real disk is visible. To unmount, enter the following command.

```
stacker -d:
```

Note: 'Appendix B, Stacker Data Compression' contains additional information on files you may want to include.

The unmounted VDISK can now be used to create a Binary image file for your ROM disk. To convert the RAM disk into an image, use the DISK2IMG.EXE utility. For example, to convert the ROM disk d: to an image file named **cmpdisk.img** in the current directory, enter:

```
DISK2IMG d: cmpdisk.img
```

The image file is now ready to be programmed into ROM or flash memory. Refer to your PROM programmer documentation for programming ROM's or flash memory. Most PROM programmers can split your image for programming onto a set of EPROMS if needed. For example a 128KB image file can be programmed onto two 64KB EPROMS, or one 128KB EPROM.

Note: The DISK2IMG utility can convert any disk to an image file. The Datalight ROM disk driver can only recognize a disk image that is created using the instructions in this chapter. The ROM disk device searches memory for a specific signature to identify a ROM disk. The VDISK device driver uses the same signature for marking the disk. A standard floppy disk does not have this signature.

If not already completed, the next step is to BUILD a version of ROM-DOS that includes the built-in ROM disk device driver. The BUILD option for reading the CONFIG.SYS file should be set to read from ROM for a ROM system. If ROM-DOS is to boot from a floppy disk but use the ROM disk as a default drive, modification to some system files is necessary. Contact Datalight for assistance with this procedure.

ROM-DOS can then be placed into ROM or flash memory or onto a floppy or hard disk drive for booting. After booting, the compressed ROM disk is available for use. Note that files placed on the uncompressed portion of the ROM disk are not available unless the disk is unmounted. Refer to 'Appendix B, Stacker Data Compression' for more information on unmounting a compressed drive.

Chapter 6, Including Device Drivers

ROM-DOS communicates to hardware through built-in and installable device drivers. These drivers process all the low-level I/O and hardware-related functions such as setting the system clock, reading from a disk or writing to the display. This processing frees ROM-DOS from the task and, more importantly, from any knowledge of the hardware platform.

This chapter provides an overview of the device drivers that are built-in to the ROM-DOS kernel, a list of required and optional device drivers, as well as the methods for including new or modified device drivers in your ROM-DOS installation. This chapter also assumes that you understand the term "device driver" and are familiar with how to install a device driver by statements in the CONFIG.SYS file.

ROM-DOS Device Drivers

ROM-DOS includes all the device drivers necessary to start a system from ROM, floppy disk, or hard disk. As well as these built-in device drivers, this SDK includes sample device drivers that can be installed from CONFIG.SYS or built-in to the ROM-DOS kernel.

Built-in device drivers are those drivers that have been linked in with ROM-DOS. They are initialized before installed device drivers get initialized, and are generally those devices that will remain standard and constant on your system.

Since ROM-DOS does all of its communication to the hardware through device drivers, a few built-in drivers are mandatory to start a system. These built-in device drivers include the console, the clock, and at least one disk device (either the ROM disk device driver, floppy disk device driver, or the hard disk device driver).

The console is required to display error messages. In systems with no console (or those with a serial port acting as a console), the console driver may be modified to display no output. The clock driver is needed to update the date and time of files as well as to provide the DOS date and time functions.

A disk driver is required in any system. At least one disk must be available to ROM-DOS to find and process CONFIG.SYS and/or load the command interpreter or initial application, whichever applies. ROM-DOS halts if there are no disk devices. The other built-in device drivers, listed below, can be helpful in many systems, but are not required by ROM-DOS.

- Console (CON)
- Clock (CLOCK$)
- Printer (PRN)
- Serial (AUX)
- Com port (COM)

- Null (NUL)
- Floppy/hard disk
- ROM disk

The ROM-DOS SDK includes the source code for a variety of configurable device drivers. These drivers, such as those in the MEMDISK directory, can be compiled and installed in CONFIG.SYS or built-in to the ROM-DOS kernel to add functionality to the operating system. No attempt is made in this section to describe the contents of any of these drivers or how they operate.

Writing Device Drivers

Typically, the only type of device driver you need to write for ROM-DOS is an installable driver, the type loaded from CONFIG.SYS. Installable device drivers are the most flexible way to create a driver under ROM-DOS. There is no limitation to the size of the code of the device driver, other than available RAM. Character devices can even override the built-in character devices by using exactly the same name as the built-in counterpart.

There is only one drawback to making a device driver installable from CONFIG.SYS instead of built-in. The driver must either be present on one of the built-in disks or on a disk device that has been previously installed. The device drivers provided in the MEMDISK subdirectory can be used as templates for writing your own custom device drivers.

If you want to write a built-in device driver (one that is linked in with ROM-DOS and not installed through CONFIG.SYS), there are some special considerations:

- The segment nomenclature must agree with ROM-DOS.
- The total ROM-DOS code must be less than 64KB.
- The total ROM-DOS data and stack must be less than 64KB.
- The return address upon initialization is ignored. Built-in devices cannot allocate memory.
- Multiple devices in one file are allowed, but must be treated specially.
- The code cannot modify itself in any way.
- ROM-DOS supported math functions need to be used instead of compiler math function. Refer to 'Chapter 7, Using a Custom Memory Disk.'

One reason for adding a new built-in device into ROM-DOS is to provide a new type of disk device from which CONFIG.SYS is processed or the starting application is loaded. ROM-DOS already has built-in floppy, hard, and ROM disk drivers.

The command interpreter COMMAND.COM can typically be loaded from a disk device installed during CONFIG.SYS processing, so COMMAND.COM does not need to reside on a built-in disk.

Adding New Device Drivers

Installable device drivers are included in your ROM-DOS installation by using a DEVICE= statement in the file CONFIG.SYS. For example:

```
DEVICE=VDISK.SYS 1024 /e
```

The standard built-in device drivers are located in the object library file named ROMDOS.LIB. When ROM-DOS is built, the linker (TLINK) loads each device specified in SYSGEN.ASM from the ROMDOS.LIB file. If you have created your own built-in driver, you can either add the driver to the ROM-DOS library or make a new library named USER.LIB. Datalight recommends adding new or replacement drivers into USER.LIB to make sure the integrity of the original ROM-DOS library file is not compromised.

If the driver you created is intended to replace a standard built-in driver, ROM-DOS uses the driver in the USER.LIB file instead of the driver of the same name in the file ROM-DOS.LIB. USER.LIB drivers always take precedence over ROM-DOS.LIB drivers when there are two drivers with the same name.

Note: If you are using Datalight's MEMDISK.ASM and a client code module, no changes to the SYSGEN.ASM file need to be made. Follow the instructions outlined in 'Chapter 7, Using a Custom Memory Disk' for creating a built-in device and using the BUILD.EXE utility. For all other custom devices, the following instructions should be used.

The new device driver source files must be compiled or assembled into object files. This can be done with Borland language tools and with reference to the compiler manuals.

Examples:

```
Tasm /Mx devprn.asm
```

Place the resulting object files into the ROMDOS.LIB file using a library maintenance utility (TLIB). The next time ROM-DOS.SYS is linked, the new device drivers will be included in that version of ROM-DOS.SYS

Using Borland's library maintenance utility (TLIB), the following command replaces the ROM disk device driver object file (.OBJ) in the original ROMDOS.LIB file. It also produces a file named ROMDOS.LST that lists the object files in the library and all public labels.

```
C:\>TLIB ROMDOS.LIB -+devrom.obj, romdos.lst
```

To add DEVROM.OBJ to the USER.LIB library file instead, type:

```
C:\>TLIB USER.LIB +devrom.obj,  user.lst
```

If the file USER.LIB did not previously exist, TLIB creates it.

Adding Device Drivers to SYSGEN

Once a built-in device driver has been compiled (or assembled) and added to one of the library files (ROMDOS.LIB or USER.LIB), update SYSGEN.ASM to include the device driver. If the built-in driver is a replacement for a standard built-in driver (such as _comx for COM1-COM4), no changes need to be made to SYSGEN.ASM. If the built-in driver is new, such as a special built-in disk driver, then SYSGEN.ASM must be informed. To do this, locate the section "A NULL Terminated Array of Built-in Devices" in SYSGEN that lists all built-in drivers. Add the following lines to the appropriate group of definitions:

```
extrn _newdiskx:byte
dw    OFFSET _DEVDATA:_newdiskx
```

Substitute the name of your driver for _newdiskx. These lines inform the linker (TLINK) that a label by the name of _newdiskx should be linked into the ROM-DOS program. This label is found in either the ROM-DOS or USER library (ROMDOS.LIB or USER.LIB).

Chapter 7, Using a Custom Memory Disk

ROM and RAM disks are the basis of diskless systems and the possible configurations for such disks are as varied as the systems using them. A RAM disk may be implemented as battery-backed static RAM, a ROM disk as paged EPROMs, or a disk could be created using flash memory which can later be updated in the field.

A ROM disk is necessarily built into a diskless system, enabling ROM-DOS to read CONFIG.SYS and/or load the first program. A built-in RAM disk may also be required, in some cases. Installable devices are desirable for some systems due to the advantage of command line options over built-in device drivers.

ROM-DOS includes complete source code for several types of memory disks. Look in the MEMDISK subdirectory for examples. The examples include a paged ROM disk, a RAM disk, a ROM disk, and an extended memory disk driver.

The ROM-DOS configurable memory disk allows you to build a custom memory disk device driver by modifying only the initialize, read and, write functions. This disk can be configured to be either built-in (loaded by CONFIG.SYS) or as a terminate-stay-resident program loaded from the command line.

The custom memory disk is made up of two modules, the base module and the client code module. The memory disk base module is named MEMDISK.ASM and handles all of the direct interaction with DOS. The client code modules are named to reflect their function, such as MEMPAGED.C and MEMROM.C.

Creating a Custom-Memory Disk

The process for creating a custom memory disk consists of the following basic steps:

- Review the source code modules of the intended driver in the MEMDISK subdirectory. For example, for a fixed address disk, review MEMFIXED; for a paged memory disk, review MEMPAGED.C and SC400PAG.C. (SC400PAG.C is an example paging algorithm set up for an Elan SC400 platform.) The header of each module describes the driver type and uses. Make appropriate modifications to the source code based on the commented sections in the beginning of each module. For example, for a paged disk, modify the paging routines to match your hardware paging mechanism; for a fixed address disk, supply the destination address for the memory disk.
- Review the MAKEFILE section appropriate to the driver you are building. Most sections contain two target output files. For example, , there are instructions for building MEMPAGED.SYS (the CONFIG.SYS installable version of the driver) and MEMPAGED.LIB (the built-in version of the driver) in the paged memory disk section. The target name is MEMPAGED.LIB, however, the actual output file is named USER.LIB.

- Switch to the MEMDISK subdirectory and run Borland's MAKE utility with the appropriate target name. For example, to create a built-in paged memory disk, run the following:

```
MAKE MEMPAGED.LIB
```

- To create a stand-alone installable version, run:

```
MAKE MEMPAGED.SYS
```

Note: If the driver is prepared as built-in, a new USER.LIB file is created in the ROM-DOS root directory. If you already had other items in a USER.LIB file, you may want to make a copy of the original file. You may also have to use the TLIB command to add individual object modules into the library file if more than one memory disk or other custom code is to be included in the USER.LIB file.

The BUILD program handles the remainder of the process. Refer to 'Chapter 4, Building ROM-DOS' for more information on the BUILD program. Part of the BUILD process involves linking in library modules. Your new memory disk code will be placed in a library file called USER.LIB. The BUILD program will look for this file at Link time and include your drivers into the ROM-DOS kernel.

When you run the BUILD program, you are prompted to include a custom memory disk. Answer yes for this prompt to add the new built-in driver. You must have a USER.LIB file present in the same directory as the BUILD program when including a custom memory disk, or errors will be generated during the compilation processes.

Memory Disk Base

The memory disk base module is the non-changeable part of the memory disk. The base module may be configured using assembler options. Available options for the base are placed in the top lines of code in MEMDISK.ASM. The makefile already defines the necessary options for both stand-alone and built-in versions of the drivers. Additional options can be added to the makefile or entered on the command line. If you customize the options, the options set in the MEMDISK module must be in agreement with the options set in the client code module. These configuration options all follow the standard conventions of undefined = false or off, and defined = true or on.

About Client Code Functions

The client code module of the memory disk must be ported to the different environments in which it is to be used. The functions can be coded in any language but they must conform to the C-language calling sequences and conventions. The client code module(s) must supply some of the functions listed below, depending on their purpose.

A ROM disk requires only the **meminit** and **memread** functions while a RAM disk requires these plus **memwrite**. All other functions are optional.

TSR disks must support **memuninit**, removable disks must support **memchanged**, while disks that support IOCTL must support the **memioctl** function.

meminit

The **meminit** function is called once during disk initialization.

```
int meminit(struct BPB bpb[], char far *cmdline, unsigned *endseg, int drv);
```

The BIOS parameter block, pointed to by the argument *bpb*, should be filled in during **meminit**. The *cmdline* argument is used for parsing the device command line. This pointer points to the start of the memory disk file name in the CONFIG.SYS device line. The *cmdline* argument has no meaning for built-in devices, but can be useful for devices installed in CONFIG.SYS. The return value of **meminit** is the number of drives.

The *endseg* argument, when first called, points to a value that is the next available segment, the segment used for a RAM disk if DOS RAM is to be used for the disk. If memory at the *endseg* is being used, *endseg* must be updated to account for the amount of memory used. If DOS RAM is not to be used, or this device is to be built-in, then this argument is ignored.

A built-in RAM disk must allocate memory using the DOS memory allocate function (Int 21h, AH=48), in the **meminit** function. This is how a built-in RAM disk would get DOS RAM for disk storage.

Note: This is used only for built-in devices; not those that are CONFIG.SYS loadable. Installable devices should update *endseg*.

---

memread and memwrite

```
int memread(long offset, unsigned len, char far * buffer);
int memwrite(long offset, unsigned len, char far * buffer);
```

The **memread** and **memwrite** functions use a 32-bit value, named *offset*, to specify where on the disk a read or write operation occurs. This value is usually, but not always, a multiple of the sector size. The *len* argument is a 16-bit unsigned number that defines the size of the read or write in bytes. The *buffer* argument is a 32-bit far pointer that defines where the disk data is read from or written to. The return value from **memread** and **memwrite** is a non-zero for success and zero for failure. A failed return value causes a critical error.

memchanged

The **memchanged** function notifies DOS that a removable disk has been removed.

```
int memchanged(struct BPB *bpb)
```

memchanged is a C-callable device driver media-check.

The value returned from the **memchanged** function indicates if the disk has changed or not. A return value of -1 indicates that the disk has changed while a return value of 1 indicates that the disk has not changed. If the memory disk is not a removable disk, **memchanged** should always

return 1. If the memory disk is a PCMCIA memory card that can be removed, this function is critical.

A return value of 0 from **memchanged** indicates that the disk may have changed. A 0 return value is acceptable but not recommended as most modern BIOSs today support a change-line-status-function that returns a definite status. (The 0 return value was used on the original PC, which did not support such a BIOS call.)

memunitit and memioctl

memuninit and **memioctl** have the following syntax.

```
id memuninit(void)
bool memioctl(unsigned category, char far * buffer);
```

Public Fields

There are also two data fields are defined in MEMDISK.ASM that are intended to be used or set by the client code:

```
unsigned char memerr;
unsigned char memunit;
```

The client code should set *memerr* when an error occurs during any of the **memread**, **memwrite** or other client functions. *memerr* is ignored if no error occurs so there is no need to reset or clear it. The *memunit* field is set by MEMDISK.ASM before passing control to the client code. The unit number will be in the range of 0 to *n* where *n* is the number of drives returned by **meminit** upon driver initialization.

Terminate-and-stay-resident (TSR) Drivers

You can configure the custom disk driver so that it can load from the DOS prompt or from a batch file, as well as from CONFIG.SYS. The TSRDEV switch in MEMDISK.ASM should be set to true for this option. A custom disk driver installed at the DOS prompt can also be unloaded from memory, thereby freeing the memory and drive letters occupied by that driver. The following functions and data are defined only in the stand-alone TSR-enabled custom disk driver:

```
void tsr_setidstr(char * tsr_id);
int tsr_uninstall(void);
int tsr_already_loaded(void);
int tsr_drives_available(void);
```

The function **tsr_setidstr**(char * tsr_id) allows your custom memory disk to have a unique identifying sequence of bytes. You can also retain the default identification sequences. Each custom memory disk driver should have a unique identifier.

Use **tsr_uninstall** to remove the latest driver from memory. **tsr_uninstall** fails if the device is not found or was loaded from CONFIG.SYS instead of from the command line.

Use **tsr_already_loaded** to determine if a previous copy of the custom disk driver has been installed. Multiple copies are allowed as a convenience in the event your driver implementation needs to check for another disk already resident.

The **tsr_drives_available** function returns the current number of drive letters available. That is, if LASTDRIVE is set to E: and drives A:, B:, and C: are used, there are two drive letters available (D: and E:).

Memory Disk Math Routines

For built-in device drivers, ROM-DOS includes a set of supported math functions. The MEMDISK.H module includes those functions not already built-in to ROM-DOS. The built-in ROM-DOS functions, plus the math routines in MEMDISK.H, replace the math library normally linked in at compilation time. Memory disk client code that will be built-in should not use long math but equivalent functions. The supported math functions and their definition prototypes include:

```
/* File DOSMATH.ASM */
typedef unsigned long ulong;
typedef unsigned char uchar;
ulong pascal _lshru(ulong l, unsigned u);
ulong pascal _lshlu(ulong l, unsigned u);
ulong pascal _lmulu(ulong l, unsigned u);
unsigned pascal _ldivu(ulong ul, unsigned u);
void pascal memmove(void *to,void *from, unsigned len);
void pascal fmemmove(void far *dst, void far *src, unsigned len);
void pascal memset(void *dst, uchar val, unsigned len);
void pascal fmemset(void far *dst, uchar val, unsigned len);
        // Note: Unlike the standard C memcmp, this memcmp returns
        //       TRUE if strings are identical, FALSE if different.
        //       Same for fmemcmp.
bool pascal memcmp(char *str1, char *str2, unsigned len);
bool pascal fmemcmp(char far *str1, char far *str2, unsigned len);
int pascal strlen(char *s);
int pascal fstrlen(char far * s);
```

Chapter 8, Making Special Configuration Changes

While the BUILD program allows complete and easy configuration of ROM-DOS for most installations, there may be designs that require you to make changes to SYSGEN.ASM prior to running the BUILD program. This chapter covers the areas of system configuration that may need special attention to accommodate your design and are described under the following section, 'Configuring ROM-DOS Through SYSGEN.ASM.'

In addition, this chapter describe how boot-time configuration can be controlled through the standard CONFIG.SYS file or by reconfiguring the BIOS to change the way ROM-DOS operates.

Configuring ROM-DOS Through SYSGEN.ASM

The file SYSGEN.ASM allows you to configure the operation of ROM-DOS at compile time. Most of the options in SYSGEN are configured by answering prompts when running the BUILD program as described in 'Chapter 4, Building ROM-DOS.' The following configuration options are described in subsequent sections; you can modify their behavior without having to read the source code.

- Assembly defines
- List of built-in devices
- Power save option
- CONFIG.SYS defaults
- Initial environment
- ROM disk search address

Assembly Defines

The assembly defines configure ROM-DOS in SYSGEN.ASM. The BUILD program normally sets these as it assembles the SYSGEN.ASM file. The defines are described in the following table.

Option	Description
BCHECK=1	Displays boot diagnostics. BUILD will set or clear this option as appropriate.
BEXT=1	Boot from BIOS extension; otherwise it will boot from disk. BUILD always sets this option.
BOOTDEV=id	ROM-DOS can boot from any device (floppy, hard or ROM disk). The id code

Option	Description
	specifies which one: 00=floppy, 80H=hard, 10H= ROM disk.
BOOTDRV=n	ROM-DOS can boot from a specific driver letter. Choose the boot drive letter where 0=A, 1=B, 2=C, and so on.

BOOTMENU=1	Display the following menu upon boot time. The menu lets the user choose from where to load DOS, and/or where to read CONFIG.SYS.
	`ROM-DOS boot options:`
	`1. Load DOS off floppy`
	`2. Load DOS off hard disk`
	`3. Make floppy default drive`
	`4. Make hard disk default drive`
	`5. Make ROM disk default drive`
	`6. Continue as if Alt key not pressed`

DATASEG=seg	Hard-code the ROM-DOS DATA segment (no fix-ups). Otherwise LOC will set the DATA segment address. Set seg equal to the destination segment in RAM. This value can be either hex or decimal.
FLOPCFG=1	Regardless of boot drive, ROM-DOS looks for CONFIG.SYS on the floppy disk before going to the ROM disk.
FLOPCHK=1	A floppy disk DOS can supersede ROM-DOS in ROM.
GENERIC=1	Include generic memory disk driver.
HARDCHK=1	Bootable hard disk partition can supersede ROM-DOS in ROM.
NOFLOP=1	No floppy or hard disk is needed (ROM disk only). This can save approximately 3KB of ROM.
NOROM=1	No ROM disk is desired This can save approximately 1KB of ROM/disk.
RAMBOOT=seg	Copy ROM-DOS from ROM to RAM upon boot for faster execution. Set seg equal to the destination segment in RAM. Default for BUILD.EXE is 70h. This value can be either hexadecimal or decimal.
RDISK=seg	Choose the segment where the built-in ROM disk driver starts searching for the ROM disk.
USERTC=1	Use the real-time clock exclusively instead of the BIOS ticks for the time.

The following example shows the use of some assembly defines/options.

```
TASM /Mx /DBEXT=1 /DBOOTMENU=1 SYSGEN;
```

The BUILD program automatically generates the above assembly command-line given the appropriate options. There is usually no need to assemble SYSGEN.ASM manually.

The example configures ROM-DOS to boot from a BIOS extension. It also causes ROM-DOS to display a boot menu if the user holds down the Alt-key during boot up.

Listing Built-in Devices

Built-in device drivers can be added to ROM-DOS by placing the name of the device driver header in a list of block device drivers terminated by a NULL (0) character. A new device may be added by placing its PUBLIC label name in the list.

Note: The block device drivers list must have at least one disk device entry so that the file CONFIG.SYS and the initial program can be read from disk at boot time.

The only disk device required in this list, for an embedded system, is the ROM disk driver. This device driver is supplied in source form on the distribution disk. The PUBLIC name for this device is *~romx* and it is found in the file DEVROM.ASM.

The code and data size of ROM-DOS can be decreased by commenting-out external references to unused devices in SYSGEN.ASM. If no references are made to a device in SYSGEN.ASM, the device will not be linked in from the library. The size of ROM-DOS decreases by the amount of code/data space occupied by that device driver.

The following example shows the list of built-in devices listed in SYSGEN.ASM:

```
public      _built_in
built_in    LABEL WORD
; built-in character devices
  dw OFFSET DGROUP:_nulx     ; MUST be 1st
  dw OFFSET _DEVDATA:_conx   ; MUST be 2nd
  dw OFFSET _DEVDATA:_clkx   ; MUST be 3rd
  dw OFFSET _DEVDATA:_comx   ; not required
  dw OFFSET _DEVDATA:_prnx   ; not required
  ; warning: while the COM and PRN(LPT) drivers are not required, the
  ; absence of them can cause somewhat odd behavior in some programs.
  ; built-in disk devices (at least 1 disk required)
IFNDEF NOFLOP
  dw  OFFSET _DEVDATA:_fdhdx ;  optional
ENDIF
IFNDEF NOROM
  dw  OFFSET _DEVDATA:_romx  ;  optional
ENDIF
IFDEF GENERIC
  dw  OFFSET _DEVDATA:_memx  ; optional
ENDIF

  dw  0  ; NULL terminator for list
```

Note: SYSGEN.ASM has two complete lists of the device drivers. One list is defined with the 'dw OFFSET' syntax as shown in the above listing. The other list is defined with the 'extrn' syntax and appears in SYSGEN.ASM immediately before the above list. You must add or remove drivers of the same name in both the 'dw OFFSET' and 'extrn' lists.

Power Save Option

ROM-DOS, when not actively performing an application function, spends much of its time waiting for user input. During that time, ROM-DOS checks the BIOS for another character, performs an Int 28h, and then goes back to checking the BIOS for a character. Even when there

is no user input, the computer is using electrical power. It is possible to avoid this waste of power on computers that support a static state or a slower processor speed.

To use the power save option, the BIOS Int 16h, function 00h is modified to switch into low power mode until a key is pressed. A *powersave* flag causes ROM-DOS to either poll the BIOS or call it and wait. If *powersave* is 0 then ROM-DOS polls the BIOS (Int 16h Function 1). If *powersave* is set to 1, ROM-DOS cals the BIOS and waits (Int 16h Function 0).

CONFIG.SYS Defaults

This section of SYSGEN.ASM allows you to define the default number of FILES, BUFFERS, the status of BREAK, and most options you normally set in CONFIG.SYS. However, Datalight does not recommend modifying this section since it is easier to modify CONFIG.SYS. If your system does not use a CONFIG.SYS file, it may be necessary to set these values in the SYSGEN.ASM file.

One such CONFIG.SYS default is the command interpreter COMMAND.COM, the first program executed by ROM-DOS. The initial program for an embedded system could be your application program. SYSGEN.ASM allows you to set the program name and the initial command line argument string. Both strings must be null terminated.

```
c_public        init_break,init_files,init_buffers
c_public        init_fcbs,init_lastdrive,init_shell
_init_break     db 0                    ; BREAK=OFF
_init_files     dw 8                    ; FILES=8 (0=calc)
_init_buffers   dw 0                    ; BUFFERS= (0=calc)
_init_fcbs      dw 2                    ; FCBS=2
_init_lastdrive db 'E'                  ; LASTDRIVE=E
_init_shell     db "COMMAND.COM /P",0   ; SHELL=COMMAND.COM /P
```

The Initial Environment

The initial environment string and maximum size (in paragraphs) are set in SYSGEN.ASM. There may be multiple environment variables, each separated by a zero-byte. The end of the environment is determined by a second zero-byte. The variable *env_para*, is where the number of environment paragraphs is specified. This value must be larger than the space required to hold the initial environment string.

```
c_public        env_para,env_string
_env_para       dw 10H
_env_string     db "PATH=",0
                db "PROMPT=$p$g",0
                db 0
```

The ROM Disk Start Address

The start address of the ROM disk is specified with a 16-bit segment value named *romdisk*. This value represents the first segment at which the ROM disk driver looks for a valid ROM disk. The driver searches ROM for the ROM disk until it finds a valid disk or reaches the end of

memory (segment 0xFFFF). The following example causes the ROM disk driver to begin its search at segment C000 and search until it reaches 0xFFFF.

```
IFDEF RDISK
romdisk   dw   RDISK ; segment of ROM disk
ELSE
romdisk   dw   0C000H ; segment of ROM disk
ENDIF
```

The BUILD program prompts for a change in the default search segment. Datalight recommends that you make no change to the segment in SYSGEN.ASM but let BUILD handle it.

Ordering Floppy and Hard Disk Drives

ROM-DOS can support disk drives in almost any order or configuration. For example, ROM-DOS can accommodate a ROM disk as drive A to boot the system, a hard disk as drive C, and a floppy disk as drive D. Disk drive flexibility allows your system to be configured as needed.

The following paragraphs describe how to split the floppy disk/hard disk driver so that floppy and hard drives do not need to be contiguous. The floppy/hard disk driver can be split into two separate drivers: flopx and hardx. You can add these drivers to the list in SYSGEN.ASM to allow floppy and hard drives to appear as any drive. For example, assume your system needs the following drives:

A: floppy disk

B: floppy disk (optional)

C: ROM disk

D: RAM disk

E: hard disk (optional)

With this configuration, the software can always rely on the RAM and ROM disks, without the drive letters changing. To use these new disk drivers:

1. Copy the FLOPHARD.LIB to USER.LIB. Keep in mind that if other drivers have already been placed in USER.LIB, you may need to manually add them again to the library using TLIB.
2. Update SYSGEN.ASM to include _flopx and _hardx in the driver list. Refer to 'Listing Built-in Devices' on page 50 for more details.
   Drivers are initialized in order, so be sure to place them in the order you want drive letters assigned. Also change all other references to _fdhx (in the SYSGEN.ASM file) to _flopx or _hardx, as appropriate.
3. Update SYSGEN.ASM to comment-out the fdhdx driver in the driver list.
4. Run the BUILD program to create ROM-DOS.
5. Double-check the .MAP file to make sure there is a _flopx public, a _hardx public, and no _fdhdx publics.

Configuring Through CONFIG.SYS

ROM-DOS supports the standard system configuration file, CONFIG.SYS. This file contains commands that reconfigure the system during boot-up. The CONFIG.SYS commands supported by ROM-DOS include:

```
BREAK=              MENUDEFAULT=
BUFFERS=            MENUITEM=
COUNTRY=            NEWFILE=
DEVICE=             NUMLOCK=
DEVICEHIGH=         REM=
DOS=                SET=
FCBS=               SHELL=
FILES=              STACKS=
INCLUDE=            SUBMENU=
INSTALL=            SWITCHES=
LASTDRIVE=          ;
MENUCOLOR=          ?
```

The NEWFILE command is unique to ROM-DOS and allows CONFIG.SYS to transfer control to another CONFIG.SYS file, possibly on some other drive or in a subdirectory. Use the NEWFILE command as follows.

```
NEWFILE=<filespec>
```

You can use this command to pass control to a new drive installed via CONFIG.SYS as shown in the following example.

```
DEVICE=NEWDISK.SYS
REM NEWDISK installed as drive E:
NEWFILE=E:\CONFIG.SYS
```

The REM command is used to place remarks in CONFIG.SYS and may also be used to comment-out commands.

When ROM-DOS is configured with the BUILD program, you can select from the following three levels of CONFIG.SYS processing.

DOS Level	Commands Included
3.31	Commands include: BREAK, BUFFERS, COUNTRY, DEVICE, FCBS, FILES, LASTDRIVE, NEWFILE, REM and SHELL.
5.0	Includes all commands available with DOS 3.31, plus DOS, INSTALL, and STACKS.
6.0	Includes all commands from both the DOS 3.31 and 5.0 levels with the addition of DEVICEHIGH, INCLUDE, MENUCOLOR, MENUDEFAULT, MENUITEM, NUMLOCK, SET, SUBMENU, and SWITCHES, the semicolon (;), and the question mark (?).

Configuring Through the BIOS

Another possible method of configuring ROM-DOS is to change the BIOS to handle new hardware while leaving the normal device drivers intact. For example, you could modify Int 13h of the BIOS so that that the floppy and hard disk drivers operate on some different disk hardware, such a PCMCIA memory cards.

The advantage of modifying the BIOS, especially if you have your own BIOS that you're familiar with, is the time it can save in writing and debugging a new device driver. In many cases, the standard drivers will work normally through a modified BIOS.

The Command Interpreter

The command interpreter loaded by ROM-DOS may be specified with the SHELL command in the SYSGEN.ASM or CONFIG.SYS file. The command interpreter is normally the COMMAND.COM program.

```
SHELL=COMMAND.COM /p /e:512
```

For many embedded systems, a command interpreter may not be required. Any program can start at boot time and have full use of ROM-DOS, as is usually the case with single-application systems.

By specifying a command interpreter other than COMMAND.COM, the ROM or disk space (about 28KB) and RAM space (about 3KB) required by COMMAND.COM can be saved. However, without COMMAND.COM loaded, the DOS prompt and the processing of batch files (including AUTOEXEC.BAT) are not available. Optionally, Datalight provides a mini-command interpreter that supports a limited command line and batch processing.

The command interpreter loaded at boot time must adhere to the following rules.

- It must never terminate. If it does terminate, ROM-DOS prints a message indicating that the command interpreter has quit and then halts the system.
- It must handle Ctrl+C if it could occur. If Ctrl+C is encountered and it is not handled, then the shell program is terminated and the system halts.
- It must handle Int 24h (critical error handler) if it can occur. If a critical error is encountered that is not handled, the shell program is terminated and the system halts.

Chapter 9, Creating ROM-DOS Without BUILD

The BUILD program is flexible enough to cover most systems on which ROM-DOS may need to be run. However, there may be situations where BUILD does not meet the needs of the particular system you are designing. If you need an option BUILD does not provide, refer to the information provided in this chapter. Refer also to 'Chapter 4, Building ROM-DOS' for information regarding the build options provided by the BUILD program.

There are four steps, described in the following sections, to manually create a ROM-DOS kernel:

- Assemble SYSGEN.ASM
- Link ROM-DOS
- Compress ROM-DOS Data
- Locate ROM-DOS

Assembling SYSGEN

The first step in manually building ROM-DOS is to configure and assemble the file SYSGEN.ASM. 'Chapter 8, Making Special Configuration Changes' provides a detailed description of the options available in SYSGEN.ASM.

The proper assembly-time defines need to be determined prior to assembling the file. You may find it helpful to run the BUILD program and answer the prompts to suit your intended installation.

Note: Use the /P option with BUILD to pause the display after each compilation step and record the options, or refer to the file BUILD.BAT created during the BUILD session.

SYSGEN.ASM can be assembled with Borland's TASM.

```
C:\>TASM /DBCHECK=1 /DBEXT=1 /DBOOTDRV=0 /DBOOTMENU=1 /DRAMBOOT=070h
         /Mx sysgen.asm
```

The preceding example configures SYSGEN.ASM and assumes ROM-DOS has the following features:

- Boot diagnostics available
- BIOS-extension boot (rather than from a disk)
- Default drive is the A drive
- The boot menu option is available
- ROM-DOS is copied to RAM address 70:0 at runtime.

Linking ROM-DOS

The second step to create ROM-DOS is to link the ROM-DOS kernel files together. As in the first step, the output from the BUILD program can be used as a reference. BUILD generates a linker response file named ROM-DOS.LNK. You can use this response file or create your own. You can also reference the file BUILD.BAT for a complete listing of the linker command line syntax.

The standard link line for ROM-DOS is:

```
C:\>TLINK  DOSASM+SYSGEN+DOSCFG6,ROM-DOS,, ROMDOS.LIB+DOSINIT.LIB/m/c
```

It is also necessary to generate a .MAP file at link time using the /m option to TLINK. The .MAP file is essential for the locating process explained later in this chapter.

If you have created some built-in device drivers and have modified SYSGEN.ASM to include those drivers, make sure they have been placed in the file USER.LIB. Refer to 'Chapter 6, Including Device Drivers' for more details.

The linker command is (TLINK requires a @ symbol before a linker response file to process it):

```
C:\>TLINK @ROM-DOS.LNK
```

Compressing ROM-DOS

The third step in building ROM-DOS, compressing the ROM-DOS data, is optional but highly recommended. Running the COMPRESS.EXE program, located in the root ROMDOS directory, creates a smaller ROM image.

```
C:\>COMPRESS ROM-DOS
```

Note: COMPRESS is a specialized tool for shrinking the size of ROM-DOS data and will not work on other programs or data. COMPRESS.EXE does not compress a ROM disk. For compression of applications, see 'Appendix B, Stacker Data Compression' on page 81.

Locating ROM-DOS

Locating is the final step in creating a new ROM-DOS kernel. This step converts an executable (.EXE) file into a format that can run from within ROM. In addition to the ROM-DOS.EXE file, locating ROM-DOS requires a map (ROM-DOS.MAP) file and a special location (ROM-DOS.LOC) file. Also, the executable file must have startup code that can be placed in and run from ROM. The startup code is assembled in the file DOSASM.OBJ.

Running the Locator LOC.EXE

The program LOC.EXE translates the ROM-DOS.EXE program into a fully-located DOS that can be placed in and run from ROM. Run LOC.EXE with no command line options for a quick help screen. Location is performed automatically by the ROM-DOS BUILD program. The discussion here is for situations that are outside of the scope of the BUILD program.

The locator requires an executable (.EXE) file, a map (.MAP) file and a location (.LOC) file to produce an output file suitable for a PROM programmer. The locator can produce Intel hex files and binary images.

Options to the locator may appear on the location command line or may be placed in the beginning of a location command file.

```
LOC [options] @loc-file
```

The locator command line options that are pertinent to ROM-DOS are described in the following table. These options may also be placed in the location file.

Option	Description
/b	Specify the /b BIOS extension option during the location process when creating a BIOS extension version of a program. This option checksums the first 2KB bytes of your program and adds a fix-up byte into the BIOS extension area so a BIOS will checksum it to be zero. The BIOS calculates a checksum of the first 2KB bytes to be zero.
/i	Specify the /i option when your PROM programmer takes image files as input. If the /i option is not used, LOC produces Intel hex output files.
/s	Specify the /s split option on the locator command line to create two files instead of one. When creating ROMs for a 186, or 286 system, the PROMs must be split into even and odd PROMs. The files have extensions of .EVN and .ODD.
/e	Specify the /e option to prevent extended addresses from being placed in your Intel hex files.

Standard Location Files

There are usually only three ways to locate ROM-DOS. You may locate ROM-DOS

- so that it can boot from a floppy disk,
- to boot from within ROM, but from a standard BIOS via BIOS extension, or
- to start directly from a BIOS that jumps to it.

The locator requires a special location file that describes the segment location of any program. The segments and classes listed in the first portion of the .MAP file provide a framework and reference for creating the .LOC instruction file. The following sections describe the location file for the three ways to locate ROM-DOS.

BUILD.EXE generates its own .LOC locator command file named ROM-DOS.LOC. By answering the prompts displayed by BUILD as closely as possible to your intended application, you create the basis for your .LOC command file. You can also compare the segments and classes in the ROM-DOS.MAP file and the .LOC file. Modify the file ROM-DOS.LOC to specify exact location requirements.

Floppy Disk Location

When creating a version of ROM-DOS that boots from a floppy or hard disk, all ROM-DOS kernel code and data is loaded starting at segment 70 or address 0070:0. The following location command file will locate ROM-DOS.EXE starting at segment 70.

```
/inf          # Image file No Fill
rom-dos,      # Name of EXE file,
rom-dos,      # Name of MAP file
rom-dos,      # Absolute MAP file
BIOSEXT       @       0xF000 +
CODE          @       $ +
DEVDATA       @       $ (0x70) +
DATA          @       $ ($) +
BSS           @       ($) +
HEAP          @       ($) ,
40KB ROM @ 0xF000 +
10KB RAM @ 0x70
```

By means of a text editor, this location file can be created and saved on disk under the name of RD-FLOP.LOC. For example, to use the above RD-FLOP.LOC file, enter the following on the command line:

```
C:\>LOC  @RD-FLOP
C:\>COPY  ROM-DOS.IMG ROM-DOS.SYS
C:\>SYS  A:
```

ROM Location with BIOS Extension

ROM-DOS can be booted from ROM using a BIOS extension. Creating a BIOS extension requires a small change when assembling the SYSGEN.ASM file and a different location file. The location file in the following example locates ROM-DOS at E000:0, a typical address for a BIOS extension, and also creates an Intel hex file named ROM-DOS.HEX.

To make a ROM-DOS BIOS extension bootable, the variable BEXT must be defined. BEXT is defined on the TASM command line using the option /dBEXT as shown in the following example.

```
C:\>TASM /Mx /dBEXT=1 SYSGEN.ASM;
```

The BIOS extension must be located in the address range of C000:0 to EFFF:0.

```
/B            # Fixup BIOS Extension
rom-dos,      # Name of EXE file,
rom-dos,      # Name of MAP file
rom-dos,      # Absolute MAP file
BIOSEXT       @       0xE000 +
CODE          @       $ +
DEVDATA       @       $ (0x70) +
DATA          @       $ ($) +
BSS           @       ($) +
HEAP          @       ($) ,
40KB ROM @ 0xE000 +
32KB RAM @ 0x70
```

After creating a ROM-DOS that boots from BIOS extension, remember to create a checksum byte. The whole BIOS extension (which is actually defined to be 2KB) must checksum to zero. Some BIOSs do not rely on this checksum, but it is good practice to ensure that this checksum is always valid. The /B option in the LOC file causes LOC to place the checksum in the BIOS extension for you.

ROM Location with BIOS Jump

ROM-DOS can be booted from ROM using a direct jump from the BIOS or other calling program. The TASM option /dBEXT should not be included on the assembly command line (or it should be set to 0). The location file displayed below locates ROM-DOS at F000:0. It creates 32KB split image files named ROM-DOS.EVN and ROM-DOS.ODD.

```
/I    # Binary (Image) file requested
/S    # Create split output files
rom-dos,      # Name of EXE file,
rom-dos,      # Name of MAP file
rom-dos,      # Absolute MAP file
BIOSEXT    @  0xF000 +
CODE       @  $ +
DEVDATA    @  $ (0x70) +
DATA       @  $ ($) +
BSS        @  ($) +
HEAP       @  ($) ,
64K ROM @ 0xF000 +
32K RAM @ 0x70
```

Chapter 10, Debugging and Troubleshooting

ROM-DOS provides standard MS-DOS functionality in the ROM environment. This allows most of the actual program development to be performed on a desktop PC running DOS. The remainder of the development can be done on the target hardware under ROM-DOS. The routines that most likely require debugging under ROM-DOS are those device drivers and other program segments that access non-standard, non-PC hardware. This chapter provides information and solutions regarding problems that may occur during the startup process for ROM-DOS on your system. Such problems may also be attributable to your BIOS.

Print Statements

The simplest method for debugging your program is running your program with embedded print statements at meaningful points. This method of debugging requires a console available on your target system. The console may be a serial port or a display/keyboard combination.

The program can be uploaded to a target system RAM disk using the COMM or TRANSFER programs. The TRANSFER program takes a file from the host PC, across the console, and places the file on a RAM disk or other disk device on the target system. Refer to the ROM-DOS User's Guide for more information on the TRANSFER and COMM programs.

Remote Debugging

The Borland Turbo Debugger provided in the Datalight STDKcan be used in the remote mode if there are COM1 or COM2 serial ports available on the system. TD-REMOTE provides an ideal interface and flexible debugging. For more information, refer to the Borland online documentation.

Local Debugging

If your target hardware has a PC-compatible display and keyboard, you can use your normal debugger on the target hardware under ROM-DOS. Some debuggers check for and require a particular DOS version number. You can use the VER command (refer to the ROM-DOS User's Guide) to change the version number reported by ROM-DOS to that required by your debugger.

Troubleshooting with Boot Diagnostics

ROM-DOS has the ability to display special characters at each stage of the boot process. These characters provide a method for determining where in the boot process an error occurs. These

characters are referred to as boot diagnostics and are included in the ROM-DOS kernel if you answer Y to the following prompt displayed by the BUILD program.

```
Do you want boot Diagnostics (Y/N): Y
```

To perform a manual link of ROM-DOS, assemble the file SYSGEN.ASM with the option /DBCHECK=1 enabled. See 'Linking ROM-DOS' on page 58 for details on manually linking ROM-DOS.

The boot diagnostics are displayed (via BIOS Int 10h, function 0Eh) to indicate completion of each step of the boot process. The boot process steps are listed below.

Boot Diagnostic	Description
B	BIOS extension has gained control. This is only displayed when ROM-DOS is placed in ROM. When booted from a disk, this boot diagnostic is not shown.
0	Interrupts are enabled, ROM-DOS has control, and the first instructions have been executed.
1	Startup code (decompress) has completed. The startup code copies the DOS data into RAM. The DOS code is also copied for a disk boot of ROM-DOS or a ROM boot with the copy to RAM feature enabled. To make room for data decompression, the startup code relocates the ROM-DOS code to the top of memory. The data is decompressed to its full size in lower memory. The stack is also set up and uninitialized data is zeroed. Boot failures at this point are typically due to insufficient RAM to accommodate the code and full data size, or an incomplete ROM-DOS image in ROM.
2	Minimum DOS structures allocated. The memory pool is set up and the default structures are at the top of RAM. The DOS interrupts are also set up.
3	Interrupts have been initialized. Boot failures at this point may be due to another process using an interrupt that ROM-DOS has set up for its own use. An example of this is a watchdog timer that traps Int 21h.
4	Built-in devices have been initialized. BIOS interrupt calls are made during the initialization (Int 13h for disk drive support, Int 10h for video). Failures may be due to incomplete BIOS interrupt support or a failure to find a disk of any type in the system.
5	Root PSP is now in existence.
6	Default drive has been determined.
7	The first pass of CONFIG.SYS processing is complete. All CONFIG.SYS statements except the INSTALL= are processed (device drivers are loaded). Standard handles such as PRN, AUX, and CON are opened.
8	All internal structures allocated. TSR programs listed in CONFIG.SYS INSTALL= statements are loaded. Failure at this point may indicate a faulty TSR program.
9	ROM-DOS has been loaded high (if DOS=HIGH). The DOS buffers have been created and copied to the HMA area if sufficient space.
DOS prompt or application start	The standard handles have been re-opened and the final program has been called via Int 21h. This program is typically COMMAMD.COM or an application program. Failure to reach the DOS prompt after boot diagnostic 9 is usually caused by not finding the program (not on the disk), a corrupted file, a command interpreter from a different operating system, or a faulty application program. Failure may also be due to insufficient RAM to run the command

Boot Diagnostic	Description
	interpreter or application program.

Some Common Problems

The following paragraphs list some of the more commonly-encountered problems. Refer to the Support section on our Datalight website for additonal information, white papers, and links to our technical support.

Problem: During the boot process the following error message appears:

```
No Disk Devices System Halted
```

Solution: The ROM-DOS kernel did not find a disk containing of any sort in the system. Check the SYSGEN.ASM file for the block device list. Be sure that there is at least one disk in the list.

If the system has only a ROM disk, then the ROM disk driver was unable to find it in memory. Check the SYSGEN.ASM file for the segment address from which the ROM disk started its search for. Look at variable *–romdisk* in file SYSGEN.ASM.

Problem: During the boot process the following error message appears:

```
ROM-DOS Not Found, System Halted
```

Solution: The BIOS did not find the ROM-DOS BIOS Extension. This is the message the Datalight BIOS provides when ROM-DOS appears unavailable to boot.

The ROM-DOS BIOS extension must be placed in the addressable memory. If this is not the case, the BIOS does not search low enough in memory for the ROM-DOS BIOS extension.

Problem: During the boot process the following error message appears:

```
BAD or Missing <ProgramName>
```

Solution: The <ProgramName>was not found on the default disk or it was not loadable. Check the *init–prog* variable in the file SYSGEN.ASM and check that the file is on the disk.

Chapter 11, Programming ROM-DOS into ROM

Once all options and devices have been set and ROM-DOS has been linked and located, it is time to program ROM-DOS into ROM. There may be up to three files connected with ROM-DOS that are programmed into ROM:

ROM-DOS.HEX - the ROM-DOS kernel

ROMDISK.HEX - the ROM disk image

DLBIOS.HEX - the BIOS (optional)

The three files listed above have the .HEX extension for Intel hex files, but could also be .IMG binary image files. The first file, ROM-DOS.HEX, is usually around 54KB unless significant device support has been added or removed. The second file, ROMDISK.HEX, allows for booting on a completely diskless system and can range from 1KB to just under 1MB in size. A practical limit is usually 512KB for conventional memory installation.

The optional BIOS is designed to fit in a minimum of ROM along with ROM-DOS. Although the DLBIOS.HEX is optional, ROM-DOS does require a BIOS to run. Since some hardware has a pre-installed BIOS, you may not have to program an EPROM with the BIOS. For a pre-installed BIOS, it is helpful to know its size and memory location so that you can properly locate ROM-DOS.

A typical BIOS starts searching for a BIOS extension signature starting at address C000:0H and then continues the search on every 2KB boundary up to address F000:0H or F800:0H. This allows ROM-DOS to be placed in ROM anywhere between C000:0H and F000:0H. Under unusual circumstances, ROM-DOS can be placed outside this address range and be initiated via a special BIOS extension program.

As ROM-DOS boots, it searches for Datalight ROM disk signatures in memory (assuming you have included a ROM disk driver in your configuration). By default, ROM-DOS searches for ROM disks within the same search range as the standard BIOS. ROM-DOS searches only on 16-byte paragraph boundaries. A 16-byte boundary is represented by an address *nnnn*:0 where *nnnn* indicates a value in the default range of C000H to F000H. However, ROM-DOS is flexible enough to allow the search range to be expanded.

You can specify a new starting point for the search when you run the BUILD program to configure ROM-DOS. The ROM disk can be located either above or below ROM-DOS in conventional memory or, with the addition of specially modified drivers, in extended or paged memory.

Chapter 12, Mini-Command Interpreter

Some ROM-DOS applications run on standard PCs and require full command interpreter functionality. Some ROM-DOS applications do not require a command interpreter; such as when the application is loaded at boot time and runs until the power is turned off.

The mini-command interpreter (MINICMD.COM) suits applications that require a limited and possibly tailored command interpreter. MINICMD.COM contains only the basic functions of COMMAND.COM and is only 4KB as opposed to the 34KB of COMMAND.COM. MINICMD.COM supports program launching, limited batch file execution, echo control, and environment management. MINICMD.COM can also be used to load applications specified in the AUTOEXEC.BAT file at boot time.

/P (permanent) is the only option to MINICMD.COM. The permanent option causes MINICMD.COM to remain loaded at all times and is used by ROM-DOS when initially loading MINICMD.COM. The permanent option causes MINICMD.COM to ignore the EXIT command. The environment size (/E) option is not implemented. The initial environment size can be specified when building ROM-DOS using the *_env_para* variable in the SYSGEN.ASM file.

External Commands

MINICMD.COM executes applications and batch files in the same manner as COMMAMD.COM. MINICMD.COM searches in the local directory for the program or batch file and then uses the directories specified in the PATH environment variable. The search precedence is .COM file, .EXE file, then .BAT file.

Internal Commands

MINICMD.COM supports several internal commands. Each internal command can be enabled or disabled by means of a #define statement in the file MINICMD.H. A description of each internal command is given in the following table.

Command	Description	Example
CD	Displays or changes the current working directory.	`CD \database\new` Changes the current working directory to DATABASE\NEW
COPY	Copies one file to another. This is only a single file copy. Wild cards are not supported. You must provide the full path (if not the current directory) and file name for both the source and the destination file.	`COPY FILE1.DAT \BIN\BACKUP.DAT` Copies one file to the \BIN directory with the new name of BACKUP.DAT.

Command	Description	Example
DEL	Deletes a single file. Wild cards are not supported.	`DEL file1.dat` Deletes the file named FILE1.DAT.
DIR	Displays a list of the files in a specified or the current directory.	`DIR \ROMDOS\DEVSRC` Displays the contents of the \DEVSRC directory.

ECHO [ON | OFF | Text]	The ECHO command followed by text displays the text to the console.
	The ECHO command followed by OFF disables echoing commands and prompts. Commands and prompts will not resume until ECHO ON is entered.
	The ECHO command followed by ON enables echoing of commands and prompts after they were disabled using an ECHO OFF command.
		`ECHO ON`
		Echoes each subsequent command, causing the prompt and command to be displayed
		`ECHO OFF`
		Disables echoing for each subsequent command, preventing display of prompt and command.
		`ECHO This is an ECHO Example`
		Displays "This is a ECHO Example" to be displayed regardless of whether ECHO is on or off.

EXIT	Terminates the mini-command interpreter. If the /P option was specified to mini-command interpreter during spawning, then the EXIT command has no effect	`EXIT`
GOTO	Transfers control to another line in the batch file.	`GOTO MESSAGE` Moves the control of execution within the batch file to the line labeled :MESSAGE
IF	Allows conditional execution of commands.	Refer to the ROM-DOS User's Guide.
PAUSE	Causes the batch file processing to stop and wait for the user to press a key.	`PAUSE`
REM	Causes the subsequent command-line characters to be ignored; usually used to place comments in a batch file.	`REM This is a comment.`
SET [ARG= [Text]]	Sets, displays, or removes environment variables. SET without arguments displays the current environment variable settings.	`SET Path=f:\bin` Initializes the PATH environment variable to F:\BIN
TYPE	Displays a text file on the console.	`TYPE File1.dat` Displays the file FILE1.DAT on the console.
VER	Displays the version number of MINICMD.COM	`VER`
[DRIVE]	Specifies a new current drive with a single	`A:`

Command	Description	Example
	letter followed by a colon.	Sets the current drive to the A drive.

Configuring the Mini-Command Interpreter

MINICMD.COM is configurable and commands can be added to or removed from it. After modifications, MINICMD.COM must be rebuilt using the Borland C compiler and assembler provided in the Datalight SDTK.

The *command–table* structure contained in CMDMAIN.C lists the internal commands and the associated C-function called when that command is entered on the command line. Removing an entry from the *command–table* structure makes that command unavailable.

A new internal command can be added by adding the command name, along with the name of the C-function into the *command–table* structure. Each new C-function is placed into a separate file so it can be added or removed easily.

The C-prototype for functions implementing a mini-COMMAND is:

```
void do_cmd(char *args);
```

When an internal command is entered, the function in the *command–table* is called with the text options following the command on the command line. For example:

```
C:\>DIR \bin\abc
```

The function to implement the DIR command is called with the *args* option equal to \bin\abc.

Chapter 13, Power Management

As more and more computers become mobile, batteries become the desired power source. Power management is a critical issue for battery-powered systems. Power management requires cooperation between applications, DOS, the BIOS and the hardware. Intel and Microsoft have defined a DOS/BIOS level of cooperation that an application can query. This specification, named Advanced Power Management (APM), involves a TSR program (POWER.EXE, hereafter referred to as POWER) provided by Datalight which communicates to the BIOS and applications.

POWER provides a real mode APM 1.1 connection to the APM BIOS and complies with all requirements of Appendix D – APM Driver Considerations of the APM 1.2 Specification. The specific functionality of both the application and BIOS interfaces are explained in the paragraphs that follow. Copies of the APM 1.2 BIOS interface specification may be obtained by searching for "APM 1.2" at the following locations on the World Wide Web:

```
http://www.microsoft.com
http://www.intel.com
```

Operation of POWER.EXE and the Application Interface

As an APM driver, POWER allows application programs to notify it when they are idle and to reject requests by other applications, by POWER itself or by an APM BIOS to transition the system into a lower power state. POWER also provides a mechanism by which application programs can be notified of changes in system power availability.

POWER interfaces with DOS application programs using a two-phase software interface. This interface employs interrupt 2Fh and allows a DOS program to:

- Notify the system that it is idle and initiate a request to reduce the system power level.
- Receive notification of changes in the system power availability.
- Reject requests by other applications, by POWER or by the APM BIOS to reduce the system power level.

While POWER does monitor a number of devices for activity that precludes reducing system power availability, it does not provide power management for individual devices. POWER always reduces the available power state for the CPU and for all devices which may be controlled by the APM BIOS. This feature does not preclude an APM BIOS from reducing the power level of individual devices (either automatically or cooperatively), but POWER does not initiate such power reductions. If an APM BIOS generates a request to reduce a specific device's power level, POWER broadcasts the event to any cooperating applications. If no applications reject the device specific power down request, POWER calls the BIOS to change the devices power state.

Notifying the System that the Application is Idle.

Applications notify POWER that they are idle by loading the AX register with the value 1680h and then generating Int 2Fh. No status is returned by the POWER interrupt handler and no registers are changed by POWER.

Note: Int 2Fh is an often-called interrupt that may be used by many programs. POWER cannot guarantee that another program in the Int 2Fh chain will not change any register values.

Receiving Notification of System Power Changes

To receive notification of APM events, an application must hook Int 2Fh and process calls in which the AX register contains the value 530Bh.. On receipt of such an interrupt, the BX register contains an APM event code. POWER acts only on those events listed below.

Event	Event Type	Event Code
SYSTEM_STANDBY	Request	0001h
SYSTEM_SUSPEND	Request	0002h
END_NORMAL_SUSPEND	Notification	0003h
END_CRITICAL_SUSPEND	Notification	0004h
BATTERY_LOW	Notification	0005h
POWER_STATUS_CHANGE	Notification	0006h
UPDATE_TIME	Notification	0007h
CRITICAL_SUSPEND	Notification	0008h
USER_STANDBY	Request	0009h
USER_SUSPEND	Request	000Ah
END_NORMAL_STANDBY	Notification	000Bh

POWER responds to each of the above APM events with the actions defined in Appendix D of the APM 1.2 specification. POWER transmits any event code returned by the APM BIOS to any APM applications that have hooked the Int 2F chain. Applications and device drivers may augment the functionality of POWER by processing these additional events and either controlling the availability of specific devices directly or initiating application-idle signals as appropriate.

Whenever an application receives a notification event or accepts a power change request it should pass the event onto the next handler in the Int 2Fh chain without altering any of the original register values.

Rejecting Requests to Change the Current Power State

Applications may reject any of the request event types defined in the table on page 74. To reject an APM request event, the application sets the BH register to 80h and immediately returns from the Int 2Fh call without altering any other register values.

It is possible for other APM-aware applications to be running that have early positions in the Int 2Fh interrupt chain. It is also possible for such applications to have already saved their operating state in response to a power-down request before a subsequent APM application rejects the request. POWER generates a resume notification event following any rejected power-down request to allow any applications, which may have acted on the request, to return to a fully-operational state.

The BIOS Interface to POWER

When POWER detects an APM BIOS that supports version 1.1 or later of the APM BIOS Specification, it initiates a version 1.1 connection to the APM BIOS in real mode. As a client of the APM BIOS, POWER acts only on those event codes defined in the table on page 74. POWER complies with Appendix D – APM Driver Considerations of the APM 1.2 Specification and uses the following APM BIOS functions via the software Int 15h interface.

APM BIOS Function	Function Number
APM_INSTALLATION_CHECK	5300h
APM_REAL_MODE_CONNECT	5301h
APM_INTERFACE_DISCONNECT	5304h
APM_SET_POWER_STATE	5307h
APM_RESTORE_DEFAULTS	5309h
APM_GET_PWR_STATUS	530Ah
APM_GET_PWR_MGMT_EVENT	530Bh
APM_GET_DRIVER_VERSION	530Eh

All events initiated by POWER are system level power requests, no specific devices are supported and consequently, neither is the APM_ENABLED state. POWER monitors the BIOS software interface to the disk, serial ports, keyboard, printer ports and display, as well as the hardware keyboard interrupt, for activity. POWER uses the periodic software Int 1Ch to measure the time since the last user activity and to poll the APM BIOS for pending events. If no events occur within the least-time-out value specified on the POWER.EXE command line, POWER generates a power-down request.

POWER also generates a power-down request in response to an application-idle request. In either case, if no applications reject the power-down request, POWER calls the APM BIOS to set the appropriate system power state.

Note: POWER only calls the BIOS to reduce the current power availability, never to increase it. The BIOS is responsible for increasing the power availability and for notifying POWER (by posting an APM event) that the power availability has changed.

If the system power state is initially in the ready or full-power state, POWER attempts to set the system power state to standby. If the system state is already in the standby state and a time-out occurs due to no user activity (or if a subsequent application idle event is received), POWER attempts to further reduce the current power state to the suspend state.

The APM BIOS must notify POWER of any increase in power availability by posting an APM event. Whenever POWER processes such events, it automatically sets the timer tick count to the time kept by the CMOS real-time clock (if one is available). This strategy of directly setting the system power state is the only method POWER employs to control power consumption. No CPU_IDLE or CPU_BUSY calls are generated by POWER.

POWER transmits any APM BIOS event code (supported or not) to any APM applications that have chained into Int 2Fh. DOS applications and device drives may extend POWER's functionality by processing these additional event codes and either controlling the availability of specific devices directly or generating application-idle signals.

Installation and Usage

POWER can be loaded either at system startup or when the system is running. Once loaded, POWER remains in memory and active until the system is turned off. POWER can be loaded at system startup by placing an INSTALL command in the CONFIG.SYS file as shown in the following example.

```
INSTALL=C:\BIN\POWER.EXE
```

POWER can also be installed by means of a statement in the AUTOEXEC.BAT file.

Operation of POWER can be configured using the following command line options. The # sign defines the number of seconds that a device may be inactive before it is powered down.

/C#	Set the inactive time for COMM ports.
/D#	Set the inactive time for disks.
/H	Display the basic help screen.
/K#	Set the inactive time for the keyboard
/P#	Set the inactive time for printers.
/S#	Set the inactive time for the display.
/ADV:MIN	Provide minimum power reduction (most responsive).
/ADV:REG	Provide standard power reduction.
/ADV:MAX	Provide maximum power reduction (least responsive).
/STD	Provide standard power reduction.

/OFF Turn power management off.

The following example shows power being installed at boot time from CONFIG.SYS. The COM ports are not monitored and the disk inactive time is set to 25 seconds.

```
INSTALL = POWER /C0 /D25
```

Note: To prevent a particular device from being monitored by POWER, enter a zero value for the inactive time. This feature enables an application to access a device directly without the possibility of the device being powered-down as the application was about to use it.

Systems Without APM

A system not equipped with an APM BIOS can still take advantage of the ROM-DOS power management functionality. The only hardware requirement for implementation of power management is a battery-backed real-time clock.

The most basic of support provided is the placing of the CPU in a static or halt state when the system is not in use. This is done by executing a STOP or HLT instruction when requested. When the next interrupt occurs, system operation resumes. At this time, the real-time clock is read and the BIOS tick count set, so that both of these time bases are in sync.

Non Standard Platforms/Pen Based Systems

The Power Management software was written with a standard Palm-Top PC in mind. The power management software detects system-idle when no keys are being pressed and there is no other activity in the background.

If a platform is to use a pen-based system without a keyboard, the system appears to not receive user input because keys are never pressed. Such configurations require a special idle detection.

Note: Special input devices generally require more coordination between Datalight and individual OEM's. Please contact Datalight with your specific requirements.

Appendix A, Booting with a BIOS Extension

The actual booting and initialization of the processor is handled by the BIOS. A BIOS will only start executing ROM-DOS from ROM if a BIOS extension has set up Int 19h (warm boot) to point to ROM-DOS.

A BIOS performs its own initialization, then searches and executes BIOS extensions, and finally executes the warm boot interrupt (Int 19h). ROM-DOS uses the BIOS Extension to install its own boot address in the Int 19h vector.

A BIOS extension is a routine that is placed in ROM and that contains a special header recognized by the BIOS. If the BIOS finds a BIOS extension header it calls the code following that header. The BIOS searches from C000:0 to F000:0 for a BIOS extension header. The format of this header is:

```
DW    0AA55H        ; Extension ID
DB    1             ; Size of BIOS
                    ; Extension in 512
                    ; byte blocks
JMP   SHORT start
DB    0             ; Checksum byte
start:
MOV   AX,0          ; Beginning of Code
```

Some BIOSs require the checksum of the whole extension to equal 0. The checksum is computed by adding the bytes of the BIOS extension.

The LOC program can perform the fix-up on the BIOS extension using the /B option to the locator. The BUILD program causes LOC to handle this if you are booting from ROM.

Appendix B, Stacker Data Compression

Datalight has licensed Stacker for the benefit of our customers. The inclusion of Stacker requires a supplement to the standard license agreement and an additional license fee. Please contact your Datalight representative for additional information.

The Stac Electronics Stacker Data Compression utilities allow you to safely double your disk capacity. Data compression makes a file use less space by eliminating repetitive bits of information. After a disk is compressed, you can access the disk as always, but with one difference. You will have up to twice as much available disk space.

Datalight provides a subset of the full Stacker 3.0 utilities, the essential programs for compressing floppy disks, RAM disks, and ROM disks. If you need a copy of the complete Stacker disks and documentation, contact Datalight.

The Stacker files on your disk are listed below, along with a brief description of their function. A complete description of these utilities is included later in this chapter.

File Name	Description
CREATE.EXE	Creates an empty compressed drive from the available free space on a disk. Create can be used in an AUTOEXEC.BAT file or directly on the command line.
SCREATE.SYS	The device driver used to compress RAM disks. This device driver is loaded along with your Ram disk device driver in CONFIG.SYS.
SCREXEC.EXE, SCREXEC2.EXE	Used by the CREATE.EXE program. Neither of these two programs should be run alone.
STACKER.COM	The standard Stacker device driver. When loaded from the CONFIG.SYS file, it sets automatic recognition of Stacker drives on your system. After drives are recognized, STACKER.COM can be used on the command line to unmount or remount compressed drives or obtain system information on available compressed drives.
STACKER.EXE	The Stacker Anywhere program. This program allows you to access a compressed disk even on systems that are not running Stacker data compression. Only a single Stacker Anywhere drive can be installed at one time.

Compressing a Non-Bootable Floppy Disk

Compressing a floppy disk is an easy process. You can compress a freshly formatted (empty) floppy disk or one that has information on it. To achieve maximum space availability, it is best to compress an empty disk since the Stacker utilities only maximize the available free space, not the portions already used for file storage.

To compress a floppy disk, insert the disk into the disk drive (drive A: is used in following example) and enter the following on the command line.

```
CREATE A:
```

When the CREATE utility is finished, several new files are on the floppy disk. The file STACVOL.DSK is the compressed portion of this disk. The disk is now divided into two portions. What you are looking at is the uncompressed portion. When the compressed disk is recognized, you will see only the files that have been placed by you on the compressed disk. The uncompressed portion will not be accessible until the compressed disk is unmounted. The CREATE program also places a file named README.STC which contains instructions for manually installing the compressed disk. The third new file is the program STACKER.EXE. This is the Stacker Anywhere program which allows you to load your compressed disk on a system that is not running Stacker already.

To use the new compressed disk, you must mount it. This can be done two ways. The first is to have the device driver STACKER.COM in your CONFIG.SYS file. For complete instructions of the STACKER.COM device driver, refer to the complete command descriptions later in this chapter. If you add this statement to CONFIG.SYS after compressing your floppy, you will have to reboot your system so the compressed drive will be recognized. If the Stacker device is already running when you created the compressed floppy, the compressed disk is automatically recognized. You cannot see the files listed in the previous paragraph since they reside on the uncompressed portion of your disk. To temporarily unmount this disk, the STACKER.COM program can be used from the command line. To unmount, type:

```
STACKER -A:
```

To remount, enter:

```
STACKER A:
```

The second way to recognize your compressed disk is to use the program STACKER.EXE. If you are already using STACKER.COM, you will not need STACKER.EXE. STACKER.EXE lets you mount your compressed disk on any system that is not already running compressed drives. You can run STACKER.EXE directly from the floppy disk or from another location on your system.

```
STACKER A:
```

Note: .COM files take precedence over .EXE files. If you are running STACKER.EXE from a directory where both STACKER.COM and STACKER.EXE are present, you will have to specify the .EXE extension to run the program.

To unmount your Stacker Anywhere drive, enter EXIT. To remount, run STACKER.EXE again.

Compressing a Bootable Floppy Disk

Compressing a bootable floppy disk requires a few more steps than a non-booting disk. Start with a SYS formatted disk that contains the operating system files. Remove the file COMMAND.COM from the formatted disk.

```
DEL A: COMMAND.COM
```

Next, create a CONFIG.SYS file for your disk. The following lines are an example of a CONFIG.SYS file. Only the STACKER.COM device statement is required, all other lines are optional.

```
BREAK=ON
FILES=30
BUFFERS=30
DEVICE=A:\STACKER.COM A:
```

Copy STACKER.COM to drive A:. Note that any other device drivers that will be installed via CONFIG.SYS need to be copied to the A: drive before the disk is compressed. The DEVICE= statement for the device driver should be placed in CONFIG.SYS prior to the STACKER.COM device statement.

Run CREATE.EXE to compress the remaining free space on the disk.

```
CREATE A:
```

CREATE places the files STACKER.EXE and the STACVOL.DSK file on the disk. If you already have Stacker running on your system and have allowed for replaceable drives, the newly compressed disk is automatically mounted. If Stacker is not running, mount the drive using Stacker Anywhere. (See previous section on Compressing a Non-Bootable Floppy Disk for more on using Stacker Anywhere).

Once the drive is mounted, copy the file COMMAND.COM to the compressed A: drive. An AUTOEXEC.BAT file can also be added. If you need to unmount the compressed disk after booting and the file STACKER.COM will not be available from another drive on your system, copy STACKER.COM to the compressed disk as well. When all of the files are copied, unmount the disk using Stacker Anywhere or STACKER.COM (depending on how the disk was mounted).

You now have a bootable compressed disk. When you boot this disk, the compressed A: drive will be mounted and can be used just like a standard disk.

Compressing a RAM Disk

A pre-existing RAM drive can be compressed by following the instructions in either your CONFIG.SYS file or your AUTOEXEC.BAT file. If your RAM drive can be installed from AUTOEXEC.BAT, the drive can be compressed using the CREATE.EXE utility as described for floppy disks. Place the instructions listed below in your AUTOEXEC.BAT file immediately following the device statement for your RAM drive:

```
CREATE drive: /B
STACKER drive:
```

The CREATE command creates the Stacker drive and the STACKER command recognizes the drive as a Stacker drive and allows access to it. The STACKER.COM device statement in your CONFIG.SYS file should also be changed. The @ symbol should be added to the end of the device statement to reserve a Stacker drive to mount for the RAM disk. For example:

```
DEVICE=STACKER.COM C:\STACKVOL.DSK D: @
```

After restarting your system, the compressed RAM drive will be accessible.

To create a compressed RAM drive from CONFIG.SYS, use the SCREATE.SYS device driver along with your RAM disk device driver. The RAM disk device must be first in the CONFIG.SYS file, followed by the SCREATE device statement. The last statement needed is the STACKER.COM device which makes the compressed RAM disk accessible. Using Datalight's RAM disk driver VDISK.SYS, add these three lines to CONFIG.SYS in the order shown:

```
DEVICE=C:\ROMDOS\VDISK.SYS 1028 /E
DEVICE=SCREATE.SYS D:
DEVICE=STACKER.COM C:\STACKVOL.DSK D:
```

The VDISK.SYS statement creates a 1024KB RAM disk. SCREATE.SYS compresses the disk to make available 2048KB of compressed space. The STACKER.COM device makes the compressed RAM disk, drive D: in this example, available for use as a compressed disk. After system restart, the Stacker RAM disk is available for immediate use.

Compressing a ROM disk

For compressing a ROM disk, refer to 'Chapter 5, Creating a ROM Disk' for complete details.

Stacker Utilities

The following sections contain descriptions of the provided Stacker utilities, including syntax and options.

CREATE.EXE

CREATE creates an empty Stacker drive from the available free space on a disk. CREATE.EXE can be run from the DOS prompt or from within your AUTOEXEC.BAT file.

Syntax

CREATE drive:[\STACVOL.*xxx*][*/options*]

Remarks

The *drive:* argument specifies the drive to compress. If no other options are provided on the command line, the default filename for the compressed disk is STACVOL.DSK and all available free disk space is used.

The optional STACVOL.*xxx* argument sets the file extension for the Stacker drive file. The default extension is .DSK. The STACVOL file must reside in the root directory of the uncompressed drive.

Options

The /S=*sss.s*[K|M]option sets the amount of unused disk space to be used for the STACVOL file. The optional specifiers K and M indicate a size entered in thousands of bytes (KB) or millions of bytes (MB). Using /S=0 or not using the /S switch will cause all available space to be dedicated to the Stacked drive. The resulting Stacker drive will have twice as much disk space as the amount you specify when the compressed disk is mounted. Refer also to the /R option.

The /R=*n.n* option specifies the maximum size of the Stacker drive by providing an expected compression ratio. The argument *n.n* indicates the ratio. The values can range from 1.0 to 8.0. If you were to use the options /S=20 /R=2.5 the maximum space available on the compressed drive would be 50MB, 2.5 times the size of the available uncompressed space of 20MB. The default ration is 2:1 or /R=2.0. Some data, such as dictionaries and .ZIP files, will not compress well. For this type of data the recommended selection is /R=1.5.

The /C=*n* option sets the cluster size for the disk. The *n* argument can be set to 4,8,16, or 32; this represents 4KB, 8KB, 16KB or 32KB clusters. The default is 8KB clusters. 16KB or 32KB clusters are recommended for large disks up to 2GB. 4KB clusters should be used when you will be storing a large number of small files which could cause more files on the disk than clusters.

The /B option runs the CREATE program in batch mode. In batch mode, no prompts are displayed to the display. Use this option when running CREATE from within your AUTOEXEC.BAT file.

The /M option changes the display for monochrome monitors. This switch should also be used for Laptops with LCD displays.

Examples

To compress a disk in drive A: using the default filename for the STACVOL file, and a compression ration of 1 to 1.5, use the following CREATE command:

```
CREATE A: /R=1.5
```

When compressing a RAM drive from your AUTOEXEC.BAT file, the /B option is used so that CREATE.EXE will run in batch mode. This example also changes the extension on the file STACVOL:

```
CREATE D:\STACKVOL.RD1 /B
```

SCREATE.SYS

Purpose

SCREATE.SYS device driver is used to compress a RAM drive which is installed via CONFIG.SYS. The device statement for SCREATE.SYS must be placed after the device statement for the RAM drive. The STACKER.COM device statement will be placed after SCREATE in the CONFIG.SYS file. If your RAM drive is installed from your AUTOEXEC.BAT file, use the CREATE.EXE program for compressing the drive. SCREATE.SYS does not use any resident conventional memory, so it does not need to be loaded high.

Syntax

DEVICE=[path]SCREATE.SYS drive: [*/options*]

Remarks

The optional *path* argument specifies the exact drive and directory path to find the file SCREATE.SYS.

The *drive:* argument specifies the letter identifier for the installed RAM drive.

Options

The /S=*sss.s*[K|M]option sets the amount of unused disk space to be used for the STACVOL file. The optional specifiers K and M indicate a size entered in thousands of bytes (KB) or millions of bytes (MB). Using /S=0 or not using the /S switch will cause all available space to be dedicated to the Stacked drive. The resulting Stacker drive will have twice as much disk space as the amount you specify when the compressed disk is mounted. Refer also to the /R option.

The /R=*n.n* option specifies the maximum size of the Stacker drive by providing an expected compression ratio. The argument *n.n* indicates the ratio. The values can range from 1.0 to 8.0. If you were to use the options /S=20 /R=2.5 the maximum space available on the compressed drive would be 50MB, 2.5 times the size of the available uncompressed space of 20. The default ration is 2:1 or /R=2.0. Some data, such as dictionaries and .ZIP files, will not compress well. For this type of data the recommended selection is /R=1.5.

The /C=*n* option sets the cluster size for the disk. The *n* argument can be set to 4,8,16, or 32 represent 4KB, 8KB, 16KB or 32KB clusters. The default is 8KB clusters. 16KB or 32KB clusters are recommended for large disks up to 2GB. 4KB clusters should be used when you will be storing a large number of small files which could cause more files on the disk than clusters.

Examples

To install a compressed RAM disk from CONFIG.SYS you will need to add three device statements to the CONFIG.SYS file. The devices should be entered in the same order as this example, first the RAM disk driver, then the SCREATE.SYS driver, then the STACKER.COM driver.

```
DEVICE=C:\ROMDOS\VDISK.SYS 64 /E
DEVICE=C:\STACKER\SCREATE.SYS D:
DEVICE=C:\STACKER\STACKER.COM D:
```

This example sets up a 64KB RAM disk using Datalight's VDISK.SYS RAM disk driver. The RAM disk will be installed as drive D:. SCREATE.SYS will then compress the RAM disk, doubling the available disk space. The final device, STACKER.COM will make available the compressed drive D: at boot time so that it is immediately available for use.

STACKER.COM

Purpose

STACKER.COM is a dual purpose device driver. It can be loaded from the CONFIG.SYS file to allow access to compressed drives immediately at boot time. STACKER.COM can also be used on the command line to obtain information or to mount and unmount compressed drives.

Syntax

```
STACKER
```

or

```
STACKER [-] DRIVE:
```

or

```
STACKER drive1:=drive2:\STACVOL.xxx
```

or

```
STACKER @drive:\STACVOL.xxx
```

or

```
DEVICE=[drive:path]STACKER.COM[options] [drive:[\STACVOL.xxx]]...
```

Remarks

The first four syntax options are for use on the command line. There are no additional options other than those defined in the following paragraphs. The fifth syntax option is used in the CONFIG.SYS file. The options listed in the next section apply only to using STACKER.COM as a device driver.

If you type STACKER by itself, a report on the status of each system drive at startup time will be displayed. The complete STACVOL filename associated with each Stacker drive will also be displayed.

The *[-]drive:* argument can be used when running STACKER.COM from the command line. Specifying only the *drive:* name argument instructs Stacker to look on the indicated *drive:* for the STACVOL.DSK file and mount that file as Stacker drive *drive:*. The optional minus sign tells the Stacker device driver to unmount the Stacker drive *drive:*. Unmounting a compressed drive makes the uncompressed portion accessible and can be accessed as a standard disk for procedures such as disk formatting.

The third argument syntax, *drive1:=drive2:\STACVOL.xxx*, will mount the *drive2:\STACVOL.xxx* file as drive *drive1:*. In order for this option to work, you must have specified a reserve drive letter in your Stacker device statement in CONFIG.SYS. To specify a reserve drive letter, place an asterisk (*) at the end of the DEVICE= statement for STACKER.COM.

The syntax option, *@drive:\STACVOL.xxx*, mounts the specified *drive:\STACVOL.xxx* file using the same drive letter. In other words, the compressed drive will replace the uncompressed drive that is referenced by the same drive letter. The uncompressed drive will be inaccessible until the Stacker drive using the drive letter is unmounted. The at sign (@) must be placed on the DEVICE= statement for STACKER.COM in CONFIG.SYS. The at sign allocates a single replaceable drive.

The final syntax option is used only in the CONFIG.SYS file. The optional *drive:path* designates the complete path for locating the file STACKER.COM if it is not in the root directory. The *drive:[\STACVOL.xxx]* argument specifies the drive letter of the uncompressed drive. The optional STACVOL file name should always be included when the default filename STACVOL.DSK has not been used. The STACVOL file must be in the root directory. If the STACVOL filename is not included in the argument a new drive letter will not be assigned. The same drive letter as the uncompressed portion of the drive will be used. The entire *drive:[\STACKVOL.xxx]* argument will be repeated for each Stacker drive on the system.

Options

The asterisk (*) option reserves a drive letter. The reserved drive letter will hold the drive letter associated with its position in the STACKER.COM Device= statement.

The /-AUTO option turns off auto-mounting for all replaceable drives. This saves about 2.7 KB in the driver.

The /B option sets hardware information. /B=1 indicates that the MC/16 coprocessor card is installed in the system. /B=*nnn* sets the base address of the AT/16 coprocessor card (valid values are 200, 220, 240, 260, 280, 2A0, and 2E0). /B=*nnnn* indicates the address for the XT/8 coprocessor card (valid values are C400, C800, CC00, D000, D400, D800, and DC00).

The /EMS option specifies that Stacker's disk cache (up to 64KB) should be stored in expanded memory if it is available. Datalight does not provide an expanded memory driver with the ROM-DOS utilities..

The /M=*nn* switch sets the cache memory size to *nn* KB. Any value between 1 and 64 can be entered. The driver memory requirements increase by the amount specified up to the maximum of 64KB for the cache. The /M switch has no effect if used with the /EMS switch.

The /P=*n* option sets the compression tuning. Values between 0 and 9 can be entered. The value 0 sets compression tuning off which decreased memory requirements by 4.4KB. The default is P=1. When P is set to 0 or 1, Stacker runs as fast as it can but achieves only standard compression. When P is set to 9, Stacker achieves the best compression it can, using all of the time it needs. The best way to find the optimal setting for your system is to experiment.

The /UM option places the Stacker cache in upper memory, in the first space large enough to hold it. You must have Upper Memory Block support available to use this option. The /EMS option takes precedence over the /UM option. If both are used, the cache will be placed in extended memory. Datalight provides support for Upper Memory Block support with HIMEM.SYS (included in the ROM-DOS SDK).

Examples

To obtain information on each system drive, use the STACKER command without any options:

```
STACKER
```

To unmount a compressed floppy disk on your B: drive so that you can access the uncompressed portion of the disk, type:

```
STACKER -B:
```

To install the STACKER.COM device driver via the CONFIG.SYS file so that it will recognize and make available a compressed C:, A:, and D: drive, and with the maximum amount of data compression (but slower access time), use the STACKER command as follows:

```
DEVICE=STACKER.COM /P=9 C:\STACVOL.DSK A: D:
```

STACKER.EXE

Purpose

STACKER.EXE is the Stacker Anywhere utility. This utility allows the use of a compressed removable disk anywhere. It can be used on another computer that has Stacker installed or on systems that do not use Stacker data compression.

Syntax

STACKER[.EXE][drive:]

Remarks

The optional .EXE extension is only necessary when you run the STACKER.EXE program from a directory where both STACKER.COM and STACKER.EXE are present. This is necessary since the .COM file extension takes precedence over the .EXE extension.

When you have used the CREATE.EXE utility to compress a disk, the STACKER.EXE program is placed on the uncompressed portion of the disk. The Stacker Anywhere program can be run directly from the compressed disk or from any other drive on the system. If you are running Stacker Anywhere from the compressed disk, the *drive:* argument is not needed. Stacker Anywhere assumes the current drive.

To unmount a Stacker Anywhere disk, type EXIT and press enter. To re-mount the disk, run the Stacker Anywhere program again.

Examples

To mount a compressed floppy disk in drive B:, using the Stacker Anywhere utility, enter:

B:STACKER	to run STACKER.EXE located on the B: drive.
STACKER	to run STACKER.EXE from the B: drive.
STACKER B:	to run STACKER.EXE from a location other than the B: drive.

To unmount the Stacker Anywhere disk mounted as drive B:, enter EXIT.

Appendix C, Creating RXE-Compatible Programs

An executable file format, named RXE for relocatable executable, minimizes RAM usage and is available in addition to the standard EXE and COM executable formats. An RXE program is placed in a ROM disk and then executed in-place directly from the ROM.

A standard EXE file is a single program block loaded sequentially into RAM prior to execution. An RXE program is similar to an EXE program but has two distinct program blocks: the code block and the data block. The code block is fixed-up to reside and execute in an absolute area of ROM. The data block is relocatable and is loaded into RAM by ROM-DOS before program execution begins.

An RXE file is converted from a DOS EXE file. During conversion, you (or ROMDISK building utility) specify the absolute segment address where the RXE file is to reside and the relative offset of the data block within the EXE. During the conversion of standard EXE programs to RXE programs, the code, written in a high level language such as Borland C, converts without difficulty. The parts of a program written in assembly language may require some modification.

The loading and execution of an RXE follows the standard operations of a program that is contained in and executed from ROM. The code block executes from ROM while the program data resides in RAM. If any of the data is initialized, it is copied from ROM to RAM before program execution. The RXE scheme presents considerable RAM savings over an EXE file format which must be loaded completely into RAM. Some additional ROM space is preserved because the EXE header is considerably reduced when converted to an RXE.

About RXE

The RXE is a standard EXE file with several extensions. An RXE can be executed from the ROM-DOS command line. If the code of the RXE in not found in memory during the startup sequence, an error message is displayed and the program exits to DOS.

The RXE file has four main parts: the EXE header, the code block, the RXE initialization routine, and the data block. The RXE code block and the RXE data block comprise the EXE program text. The RXE initialization routines provide some extra initialization and run-time support for the RXE.

The RXE initialization is a small piece of user-configurable code that must be placed into the RXE file during conversion, and that is loaded into RAM by the DOS loader. The RXE initialization code contains a table of RXE information, a short initialization routine, and the RXE interrupt interpreter. This code is provided in source (RXE–INIT.ASM) for configuration to a particular environment.

The division between the code block and the data block is specified at conversion-time. This division can be specified as a segment name, segment class, or an absolute number. If a segment name is used, then this number is looked up in the .MAP file (created during the program

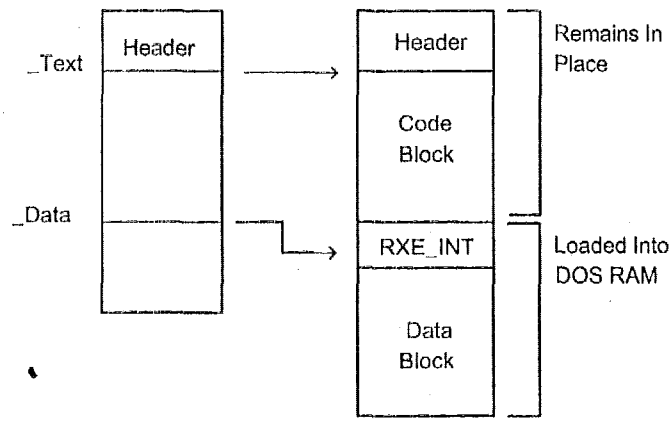
compilation) for the physical location. *Data–Seg* is a name used hereafter to refer to the segment name starting at the dividing line between the code block and the data block.

The code block is converted from the first part of the EXE program text, the part of the EXE file that remains in ROM and executes in-place. The code block portion of the RXE program does not change during the execution of the program (as it is expected to be placed in ROM). The data block is the second part of the EXE program text and is loaded into RAM by the DOS program loader.

As an example of and RXE, the following .MAP file listing is the MAP file of a simple "Hello World!" program compiled with Borland C.

The code block includes only the segment called –TEXT. The data block includes the segments –DATA, –EMUSEG, –CRTSEG, –SCNSEG, –BSS, –BSSEND, and –STACK. The dividing line or the *Data–Seg* is the segment named –DATA.

```
Start     Stop      Length    Name      Class
00000H    01356H    01357H    _TEXT     CODE
01360H    01781H    00422H    DATA      DATA
01782H    01785H    00004H    EMUSEG    DATA
01786H    01787H    00002H    CRTSEG    DATA
01788H    01789H    00002H    CVTSEG    DATA
0178AH    0178FH    00006H    SCNSEG    DATA
01790H    017D5H    00046H    BSS       BSS
017D6H    017D6H    00000H    BSSEND    STACK
017E0H    0185FH    00080H    STACK     STACK
The program entry point is 0000:0000.
```



RXE Format File

During the conversion from EXE to RXE, each fix-up is modified as follows: All code block fix-ups are resolved, and then removed from the fix-up list. The code block is placed at a known fixed-address since the actual fix-up value is known at conversion time.

Data block fix-ups in the code block are emulated or flagged as an error. The run-time address of the data block is not known until ROM-DOS loads the data block into memory. Consequently, references to the data block segments are not known at conversion time. If an instruction is

encountered in the code block that loads a register with data block segment values, the converter emulates this load with an Int 18h. The Int 18h loads the correct register with appropriate segment value at run-time, and then returns to the program. For example,

```
MOV  AX, SEG DGROUP
```

is changed to

```
INT  18H
DB   0
```

If there is a place in the code block that references a data block segment but does not load a register with it, then it cannot be emulated. These references are flagged as an error. The following is an example:

```
MOV  AX, DS_var
  .
  .
DS_var:  DW  DGROUP
```

Although the previous example is quite simple, the code can be changed, in most cases, to support a RXE compatible method of obtaining the segment value.

Data block fix-ups in the data block are adjusted and left in the fix-up list.

An RXE is prepared for execution differently than an EXE program. Perform the following steps before executing an RXE program.

Load the RXE INIT code and RXE program data into DOS RAM.
Start RXE_INIT code running.
Checksum the RXE code block to verify that it resides in linear memory and is contiguous. If not, print error message and exit.
Set RXE interrupt to point to handler in RXE_INIT.
Go to program startup in ROM.

Rules For An RXE File

Programs can be built with Borland C++ and converted to RXE programs. There are several problems and considerations to be aware of in the conversion process using either compiler. A data word that contains the segment value of a segment (that is part of the data block) is invalid within the code block of an RXE program. A common example of this is a DW Seg DGROUP in the code block. For example:

```
TEXT Segment ...
mov  AX, CS:DATA_SEG
  .
  .
.DATA_SEG  DW DGROUP
_TEXT Ends
```

Note: The Borland C compiler uses this in its startup code. A simple change is required in the start up code to make it RXE-compatible. Refer to 'Build an RXE using Borland C++' on page 99 for details on these changes.

Another problem in making a program RXE-compatible is data or self modifying code in the code block. One example of this is the use of floating point in your program with the C-compiler 8087 emulator library code. Another example is library routines that store data in the code segment such as the Borland **malloc** routine in large model.

An RXE-compatible program that uses floating point (8087) emulation has self-modifying code. The 8087 emulation instruction is an interrupt, whose handler emulates the 8087 instruction. If an 8087 is present, some of these handlers change the emulated instruction (the interrupt) to be an actual 8087 instruction, and then return to and execute that instruction. The utility RXECHECK is used to determine if a program has any areas of self-modifying code.

The RXE-compatible program must be an EXE file with at least one fix-up for the DATA segment.

The RXE-compatible program must not use overlays.

All of the program CODE must be addressable in memory at all times. A problem can occur when using a paged EPROM device to implement the ROM disk.

The RXE-compatible program must not assume that the DATA segment follows immediately after the CODE. The following two assembly language files assemble and operate correctly from an EXE but not from an RXE.

```
File1.ASM
CODE Segment "CODE"
     extrn data_item
     mov   AX,data_item
             .
             .
CODE  Ends
      END
File2.ASM
DATA  Segment "DATA"
      public  data_item
data_item   dw   0
DATA  Ends
      END
```

Note: The external word *data–item* is defined in the segment DATA, but is declared external from the segment CODE. The linker fixes the item assuming that DATA immediately follows CODE. To fix this so it works in an RXE setting, declare the segment DATA in File1.ASM and declare *data–item* as an external within the DATA segment.

References within the code block to segments in the data block must be immediate loads of the segment values. For example,

```
Code segment    "Code"
mov ax, seg data     ; OK
mov  AX,DS_val       ; Not for RXE
          .
          .
          .
ds_val:   dw seg data    ; not for RXE
```

The RXE scheme allows only thirty-two unique data segment values to be referenced from within the program's code block. If your application must reference many different data segments, consolidate (make contiguous) segments whenever practical.

RXE Conversion Steps

The following steps provide a guide for conversion of an EXE program to an RXE program.

- Develop and test your application for operation as a standard .EXE.
- Build the RXE library files for the specific memory mode and compiler tools in use.
- Determine the code/data blocks of the RXE.
- Check program for RXE ability using the RXECHECK utility and refer to the previously defined 'Rules For An RXE File' on page 93 for proper RXE coding practices.
- Configure the RXE initialization (RXE–INIT.ASM) code. This step is only required if using a paged memory disk or have other unique requirements.
- Run and debug the RXE on a PC.
- Create a ROM disk.
- Program a ROM disk from the ROM disk image and test the program as an RXE.

Develop and Test Application

If not already completed, you can use one of the sample applications from the \RXE\EXAMPLES directory to practice developing an RXE program. Select the example that most closely resembles the type of application you intend to create.

Prepare Compiler Startup Code Library Files

Prepare the Datalight RXE library for placement in ROM and specifically for the compiler and memory model you plan to use for your application program. Instructions for preparing the code for Borland C are presented in the section titled 'Build an RXE using Borland C++' on page 99.

Determine Code/data blocks

Next, determine how much of the program comprises the code block. The code block starts with the first segment of the program and ends at the start of the data block. This should obviously include the CODE or TEXT segments of the program, but could also include segments that contain constant data. The programs provided in the \RXE\EXAMPLES directory, when compiled, can be used for studying this process.

First, link the program and produce a link MAP. The MAP file is used by the RXE tools and also by yourself for viewing the segments that make up the program. With our examples, after running the MAKEFILE or batch files, you can view the .MAP files and the data segment selections used for running RXECHECK.

Next, review the MAP file and look for the last CODE segment. See if the next segments are FAR–DATA segments. If they are, then they may include constant data, and may remain in the code block; that is, in ROM.

You can test different points in the segment list to determine how much of the program you can make into the code block. Begin testing at the ENDCODE class and continue until you encounter a lot of changed memory or until you get to the DATA class. For a C-program there may not be any segments between these two. To test a segment for the *mem–seg*, just run RXECHECK and see if there are any changes to the code block. Please note that certain compiler versions produce a segment called EMULATOR_TEXT. This segment needs to be part of the DATA block rather than code block since it contains references to DGROUP.

Check Program for RXE-ability

The RXECHECK utility is used to check a program for self-modifying code data in the code block or invalid Data segment references. This utility executes your program and determine if any of these non-RXE coding practices is used in your program. RXECHECK is also useful for validating your code block selection. Example output for RXECHECK is shown below:

```
C:> RXECHECK BEGDATA VBASIC
Datalight RXE Check Utility v6.22
Copyright (c) 1989-1998 Datalight, Inc.
RXE Test Program!
CS = 0x3648 DS = 365E Stack = 36AB:0FFE
Appears RXE-Test has NOT been converted to an RXE
RXE-Test compiled with Borland C using Large Memory Model on Oct 25 1996
Malloc(2048) = 37EE:0004
Malloc(2048) = 386F:0004
Malloc(2048) = 38F0:0004
The Program 'basic' can NOT be made into an RXE
   1) The code block has been modified during
      execution. See file CODE.DIF for a list of
      code block modifications during execution!
```

Modify RXE Initialization Code

For most configurations, modification of this code is unnecessary, however, the RXE initialization code (RXE–INIT.ASM) may be configured to a hardware environment if required. An example reason for modifying this code is setting up to use LIM or other paged memory devices to contain the RXE program. As distributed, the file RXE–INIT.ASM assumes that the RXE program is in memory at all times. This code switches it into memory if required.

Before using the RXE–INIT code it must be assembled and linked into RXE–INIT.EXE. Either MASM/LINK or TASM/TLINK combination may be used for this purpose.

```
TASM /mx RXE_INIT.ASM
TLINK RXE_INIT.OBJ
```

Test and Debug The RXE From RAM

Debugging an RXE program is difficult because there is limited debugging available once an application is converted into an RXE.. It is recommended that debugging occur in the EXE phase of an application destined to be an RXE.

An RXE program runs directly from memory. The memory must either be a ROM disk or a RAM disk. This SDK contains an absolute RAM disk (ABSDISK) which is used for testing the RXE without the need to program a ROM each time you perform a test. ABSDISK allows placement of an RXE program or file in a known absolute address in CPU memory space. This tool fulfills the requirement that an RXE's code block reside at its executable (absolute) address, while necessarily allowing the debug process to operate in RAM, thereby simplifying RXE development.

The ABSDISK.ASM source file contains two equate-variables that govern the ABSDISK—ABS–ON and ABS–SEG. When ABS–ON is set to 1 (equ 1), an ABSDISK device driver is created such that the first file copied to it is placed at the segment address specified by the ABS–SEG equate, with an offset value of zero. The ABSDISK has the ABS–SEG starting at 1000H.

The first step in testing an RXE running in RAM is loading the ABSDISK.SYS into your system using CONFIG.SYS.

```
Device = ABSDISK.SYS 64
```

Note: The number 64 specifies the ABSDISK size in kilobytes. Increase this value if your application requires more code space.

This ABSDISK loads the first file at memory location based on the address set in the device (default is 1000:0h). Using this information, you can convert your program to an RXE that executes at segment 1000h. For example:

```
RXE_CVT program data 1000
```

Then boot ROM-DOS with the ABSDISK.SYS in the CONFIG.SYS file. Copy the converted PROGRAM.EXE onto the ABSDISK, and execute the program. Note that RXE–CVT does not change the EXE to RXE.

If you need to test a new version of your application on your ABSDISK, use the RESET.EXE utility. This utility allows a file to be copied starting at the first available data sector of a blank disk.

MS-DOS and ROM-DOS copy files onto a disk starting at the next-available-sector. If a file has been copied onto the disk and then deleted, then the next-available-sector pointer points to the next sector past the end of the previous file even though it has been deleted.

Since an RXE program is required to reside contiguously at the memory segment specified to RXE–CVT, the RESET utility must be used during a typical debug session where several iterations of your program may be copied to the ABSDISK. Without RESET, the only alternative would be to reboot ROM-DOS for each fresh copy of your RXE.

Run the RESET program with the ABSDISK drive designation immediately following.

```
RESET D:
```

Create A ROM Disk With An RXE

The next step is to create a ROM disk with your RXE in it. The ROM disk build program ROMDISK.EXE performs the RXE conversion automatically. An RXE program resides in a ROM disk and appears to DOS and the user as a standard EXE program. This section explains how to use the ROMDISK utility to build a ROM Disk image that contains RXE programs.

Note: It is assumed that before attempting to build a ROM Disk, the EXE program has been successfully tested for RXE using the RXECHECK utility.

The ROMDISK utility program is used to build RXE programs on the ROMDISK image. The ROMDISK utility is run in the standard manner, however, there are three additional requirements for a ROM disk containing an RXE program:

- The starting address for the ROM disk is required.
- The data block segment should be listed on the ROMDISK command line.
- The EXE program is renamed to RXE and the MAP file is placed in the directory containing the ROM disk root directory.

The absolute address is specified on the ROMDISK utility command line using the /H (create an Intel Hex output file) option or the /I (create image file) option. The absolute segment address follows both of these command line options.

The data block segment name is placed on the command line with the /D option as -DDATA for Borland tools.

The following example builds a ROM disk that is placed at D000:0000 in the 80x86 address space. The RXE data block in the File TEST.EXE begins at the DATA segment. The ROMDISK output file is image/binary format.

```
C>MD TMPDISK
C>Copy TEST.EXE TMPDISK\TEST.RXE
C>Copy TEST.MAP TMPDISK\TEST.MAP
C>Copy AUTOEXEC.BAT TMPDISK
C>Copy CONFIG.SYS  TMPDISK
C>ROMDISK -ID000 -DDATA TMPDISK
```

The next example builds a ROM disk that is placed at A800:0000 in the address space. The output format is Intel hex.

```
C>ROMDISK -HA800 -DBEGDATA
```

Program A ROM Disk With An RXE

The last step is to program the ROMDISK ROMs. Refer to the documentation provided with your EPROM programmer or flash loader software. Boot ROM-DOS, change the current disk to the ROM disk with the RXE files and execute them.

Build an RXE using Borland C++

It is strongly recommended that you have a copy of the source for the startup code and the library for the language you choose.

Compiling and Linking

Compiling with Borland C++ is given for the command line only. Compile and link your program for a DOS environment only. Extended memory is not supported for RXE programs.

It is possible to build using the IDE, but it is not recommended. The need for changed startup code and possible extra floating point libraries may require the extra flexibility afforded by the command line compiler. Example:

```
bcc -ls -ml -M Basic.c ..\libsrc\rxe_b_l.lib
```

Startup Code

The Borland C++ startup code contains an invalid data segment reference. To speed development of your RXE program, this SDK includes pre-modified startup code files for Borland C/C++ versions 2.0, 3.0, 4.0, and 4.5. You can use the pre-modified files directly from the RXE directories without the need to copy them to your compiler directories.

We recommend that you use the pre-modified files whenever possible. If you are using a compiler version that is not provided, you may need to make these changes yourself. As mentioned, the startup code does contain an invalid data segment reference. The name of that reference is DGROUP@@. The DGROUP@@ data item is found in the startup file CRT0.ASM which can be found in the directory.

```
BC45\LIB\SOURCE\LIB\C0.ASM
```

All references to DGROUP@@ are coded as follows:

```
MOV   ES, CS:DGROUP@@
```

They need to be changed to the following. These changes do not change the execution of the program, but make it an error-free RXE conversion.

```
PUSH   AX
MOV    AX, DGROUP
MOV    ES, AX
POP    AX
```

Library

The Borland C++ library has functions that save data in the code segment. The most common function **malloc** (and C++ **new** is included in this as it references it). The LIBSRC directory contains ready-to-link library code, including a **malloc** function that can be used with RXE code.

To prepare these files, change to the LIBSRC directory. Modify the MAKEFILE to recognize the appropriate path and compiler version. Also, make sure the Borland compiler tools in your environment path. In the file MAKEFILE, locate the following two lines:

```
BCC_VER   =5
BCC_ROOT  =C:\bc5
```

Change BCC_VER to the appropriate version number, such as 2 for Borland C/C++ version 2.0. Set the BCC_ROOT variable to the correct path for the root directory of your compiler tools. For example, if your Borland 2.0 tools are located on your E: drive, change the path to E:\BC2.

Next, run MAKEALL.BAT. MAKEALL.BAT confirms your compiler and version choices and calls the BORLAND.BAT file to process the files. MAKEALL generates the new library files RXE_B_S.LIB, RXE_B_M.LIB, RXE_B_C.LIB, and RXE_B_L.LIB. The S, M, C, and L in the library name represent the memory model for the code.

The MAKEFILE instruction file in the EXAMPLES subdirectory demonstrates linking in this code with the Borland compilers. If you encounter any other problems with the Borland Libraries, please contact Datalight.

Finding the code block or <DataSeg>

Borland C++ creates a minimal number of segments during the compile process. The first data segment is called "DATA" and would most often be the best choice for being the DataSeg when developing an RXE or when running the RXECHECK program.

Floating Point

The Borland's C++ floating point feature cannot be placed in ROM and so is not RXE-compatible. A solution to this problem is to use the floating point package named GOFAST from US Software (www.ussoftware.com). Systems using an 80x87 cannot use the Microsoft floating point startup code.

The EXAMPLES directory contains a floating point code example. The MAKEFILE instruction file demonstrates linking in the GOFAST libraries properly.

```
bcc -c -ml floatpt.c
bcc -M -ml floatpt.obj rxe_b_l.lib emjir7.obj usemund.lib
```

The Examples

The EXAMPLES directory contains three source code examples plus compiling/linking instructions. The examples are as follows:

BASIC.C	A basic program
FLOATPT.C	A modest floating point example
CPP.CPP	A simple C++ example

The batch files and Borland MAKEFILE assume you have your path and environment set up for the compiler tools to be used. Neither the batch files or the MAKEFILE can do this. The floating point example assumes that you have the US Software GOFAST libraries available.

Before compiling the code, edit the file MAKEFILE to set the version number and correct path for your Borland compiler tools. In the file MAKEFILE, locate the following two lines:

```
FP_LIB_DIR=C:\GOFAST
BCC_ROOT  =C:\bc$(BCC_VER)
```

If you have the US Software GOFAST libraries available, set the correct path for their location on your drive. The floating point example will fail to complete if the GOFAST libraries are not present. Set the BCC_ROOT path and directory to match the location on your drive, for example E:\BC2.

The default memory model is l (large). If you plan to use a different memory model, also edit the SHOWALL.BAT batch file. Locate the correct section for your compiler and compiler version. Set the variable MEM_MODEL equal to s, m, c, or l for small, medium, compact, or large memory model respectfully.

After making these changes, run SHOWALL.BAT and answer the prompts for compiler type and version. The batch file makes the appropriate calls to the Borland MAKEFILE and other batch files to compile and link the code and run it through the RXECHECK process to confirm the RXE-compatible code.

RXECHECK.EXE Utility

The two most common problems in converting an EXE program to an RXE are the code blocks containing data or self modifying code, and an invalid data segment reference. Both conditions can be detected using the RXECHECK utility.

The RXECHECK utility loads the EXE program and calculates a checksum both before and after code execution. If the two checksums differ, then code was modified during execution. RXECHECK also searches the code block for invalid data segment references.

RXECHECK produces one or more data files to help with the RXE process if the check indicates problems. The file CODE.DIF provides a list of code block modifications found during execution of your program. The file FIX-UP.BAD lists invalid code block fix-ups.

RXECHECK executes your program to determine if it can be made into an RXE or not. If your program contains hardware specific code, you may be unable to execute RXECHECK on your development system. On some hardware platforms, you might be able to execute RXECHECK on the actual target platform. If this is not possible, other debugging methods must be considered.

Command Line

```
RXECHECK Data-Seg  Prog Prog-args
```

Data-seg is the name of a segment or class from the MAP file. *Data-seg* is the relative offset from the start of the EXE program text that marks division between the code block and the data block. *Prog* is the name of the EXE program. *Prog_args* are any arguments required by the EXE program.

Example Usage

```
C>rxecheck DATA BASIC.EXE

Datalight RXE Check Utility Version  v6.22
```

```
Copyright (c) 1989-1998 Datalight, Inc.

RXE Test Program!
CS = 0x34CD DS = 3607 Stack = 3607:FFF4

Appears RXE-Test has NOT been converted to an RXE
RXE-Test compiled with Borland C using Small Memory Model on Oct 25 1997

Malloc(2048) = 085C
Malloc(2048) = 1060
Malloc(2048) = 1864

The Program 'BASIC.EXE' can be made into an RXE
```

RXE_CVT.EXE Conversion Utility

The RXE-CVT utility converts an EXE format file to an RXE format file. The EXE file name does not change, only the format of the executable program. If execution of an RXE-converted program is attempted under MS-DOS, an error message is displayed and the operation cancelled.

Command Line

```
RXE-CVT "{options}" EXE-file Data-seg Mem-Seg
```

Data-seg may be a named segment, a segment class, or a hexadecimal number. *Data-seg* is the relative offset from the start of the EXE program text that marks the division between the code block and the data block. If *Data-seg* is a hexadecimal value, then it is assumed to be a count of paragraphs (16-byte increments). Because the *Data-seg* argument can either be a name or a hexadecimal value, a leading zero must be used when a hex value begins with an alpha (A-F) character.

Mem-seg is a hexadecimal value that specifies the absolute physical segment number where the RXE file is located in physical memory. This value is usually the segment where the file starts on the ROM disk. For example, if a value of 1234 is specified, the RXE file starts in the absolute memory location 1234:0h.

Mem-seg is provided automatically by the ROMDISK utility when placing a RXE in a ROM disk. *Mem-seg* is fixed at 1000h when using the ABSDISK.SYS device driver.

Options

The following table list the options to RXE-CVT (options are case-insensitive).

Option	Description
/C	Detect all occurrences of an error. This option is usually used to find all the places where a DW DGROUP is in the code block of a program.
/I<int#>	Specifies the RXE interrupt. This interrupt is used in the interpretation of data block segment loads in the code block.
/Q	RXE_CVT executes without messages. The ROMDISK utility uses this option.
/R	Provides a .RXE extension for the output filename. The default output filename is *filename*.EXE. The /R option preserves the original *filename*.EXE file.

Example

In the following example, the RXE–CVT utility creates a file named PROG.EXE. The code block begins at the start of the PROG file and continue until the data segment. The data block starts at the beginning of the data segment and continues until the last segment of the input file PROG.EXE. The resulting RXE is fixed up for execution at segment 1234h. The absolute start address is 1234:0000.

```
C>RXE–CVT  PROG.EXE  data  1234
```

The next example creates the file named TEST.RXE. The data block is assumed to begin at 400h bytes from the start of the EXE program text image. The code is located in memory starting at C000:0000h.

```
C>RXE–CVT  -R  TEST 0400 C000
```

The last example creates TEST.EXE located for execution from the ABSDISK device driver. The *Mem–seg* value 1000h is the segment address of the start of a loaded ABSDISK device driver. When TEST.EXE is copied to ABSDISK, its image resides at 1000:0000h.

```
C>RXE–CVT  TEST  0400 1000
```

Some Common EXE-to-RXE Problems

Common problems occurring when converting an EXE to RXE include:

- Storing data in the code block.
- Referencing a data block segment in a manner other than loading it's value into an 8086 general purpose register. A data segment reference is made by performing a far call to a data segment. Another method for referencing a data segment is having a DW seg DGROUP type reference in the code block.

Data Stored in the Code Block

The problem of storing data in the code block must be corrected by different program methods. This problem should not occur in high level language code, only in assembly code. The data can be saved on the stack or in the data segment.

Data Block Reference with a Far Call

Correcting the far call to a segment in the data block must be rewritten as follows. These instructions may be put into an assembly language macro.

```
CALLF   DGROUP:data_func
```

Is changed to:

```
PUSH    CS
PUSH    Offset L1 ; Push Return Addr
MOV     AX, DGROUP
PUSH    AX
PUSH    Offset DGROUP:data_func  ; Push calling Addr
RETF                             ; Fake the far call

L1:
```

Assembly code with the DW DGROUP in the code block must be removed. If the intent of the code is to load the value of DGROUP, then it can be modified as follows:

```
MOV     AX, DGROUP
```

Error Messages

Fatal Error: RXE code not in memory!

This is a run-time error message indicating that the RXE initialization code determined that the program code was not in the specified memory location during conversion from an EXE to an RXE.

Note: The following error messages may be produced by either RXE–CVT or RXECHECK. Each of these error messages are fatal, meaning the conversion (RXE–CVT) or checking process (RXECHECK) has been terminated without, in the case of RXE–CVT, generation of a valid RXE file. Also, 80x86 machine instruction MOV SP, Seg... is not supported in an RXE. The RXE converter does not support loading SP with a segment value. All other registers (AX, BX, CX, DX, SI, DI, and BP) are supported. Code using SP for segment loading must be rewritten.

Cannot execute '*FileName*'

RXECHECK attempted to execute a program. The EXE file was not found or it has already been converted to an RXE program.

Cannot find the RXE loader (RXE_INIT.EXE)

The RXE–CVT program could not find and load the file RXE–INIT.EXE. This file is a startup file for the RXE program. Assemble and link the file RXE–INIT.ASM.

Cannot open '*FileName*'

The file named *FileName* could not be opened. Check the name and determine why it is bad.

Could not find segment '*SegName*' in the file '*FileName*'

The data block identifier specified by *SegName* was not found in a segment in the MAP file named *FileName*. Check the MAP file to determine the correct identifier of the program's data block.

data block segment is not on a segment boundary it must be

The segment of the specified data block, was not on a paragraph boundary. Standard compilers always start DGROUP on a paragraph boundary.

This may indicate a bad choice for the data block segment.

Data_Seg value <value> is too large; greater than EXE size

This error indicates that the value entered for the *Data–Seg* value in RXE–CVT was too large. The value is larger than the size of the EXE file.

This error most often means that the *Data–Seg* parameter and the *Mem–Seg* parameter were mixed up.

EXE and MAP files were not created by the same LINK run

The EXE and the MAP files must have the same date and time stamp. If a MAP file is out of date with the EXE file, unexpected errors can occur.

File '*FileName*' is already an RXE format file

This message is issued when an attempt is made to convert a file that is already in RXE form.

File '*FileName*' not an EXE file

The file *FileName* is not a valid EXE file.

Invalid parameter '<parameter>'

The parameter is not used by the program. This is a fatal error. Change the parameter and re-run the program.

MAP File has already been converted to RXE format

The MAP file has been changed by the last RXE conversion process. The program must be linked and a new MAP file must be produced.

Non-convertible EXE file:
Appears to be "DW DGROUP" in code block at *????:????*

This error occurs when a *Data–Seg* variable has been detected in the EXE's code block. Conversion from an EXE to an RXE can only be performed on EXE files that load a data block segment value directly into a register. The loading method is determined by the byte immediately before the data block fix-up to verify that it is an 8086 load-immediate instruction.

Legal example:

```
mov   AX, DGROUP
```

Illegal example:

```
_Text Segment Byte Public 'code'
  MOV    AX, CS:DGROUP@
          .
          .
          .
DGROUP@  dw   DGROUP
_Text Ends
```

Non-convertible EXE file:EXE does not have any fixups

An EXE file must have at least one fix-up specifying the start of the data block in order to build an RXE from it. This EXE does not have any fix-ups.

Non-convertible EXE file; too many data segments

The EXE file has more than 32 different references to the data block.

Not enough memory

The RXE converter ran out of memory. Remove RAM disks or TSR programs to increase the available memory.

Read/Write failure

An error occurred during the reading or writing of files.

RXE Specified interrupt value invalid

The interrupt specified was not in the range of 0 through 0xFF. These are the only legal values.

Appendix D, Implementing ROM-DOS Super-boot

Double-booting a System Using Hidden Files

Most disk-based computer systems boot up with their primary operating system, such as Windows 95, Windows 98, Windows NT, LINUX, and others for access to the available disk drives and to establish the system environment. In some systems, it may be beneficial to employ a special boot-up mode to a different environment and a secondary operating system. A secondary operating system can be used to run special diagnostic programs that need to be kept hidden from the end-user. Such diagnostics, accessed by a hot-key combination, may be used by service personnel or by the end user under the direction of technical support personnel.

The need to run the computer under a secondary operating system or under its primary operating system, can be met by selectively booting the computer to either of two disk drives/disk partitions included in the system. The ability to double-boot the system is provided by a special version of Datalight's ROM-DOS that includes Super-boot capability. A double-boot arrangement may also be needed when it is necessary to run disk-recovery utilities or applications that are specific to the respective primary or secondary operating systems.

The remainder of this appendix describes the procedure for installing ROM-DOS as the secondary operating system on the system hard disk designated as the boot disk drive.

About the Boot Disk

Implementation of Super-boot must be accomplished on the hard disk designated as the boot disk (that is, the first hard disk in the system). The secondary operating system (ROM-DOS) must be installed prior to the primary operating system and the boot disk drive must be new or one that can be reformatted without regard for the data which it contains. If partitions already exist on the drive, they must be removed prior to implementing the Super-boot partition if there is insufficient free non-partitioned space on the drive to accommodate the new partition. For example, if you already have an NT partition on your disk, you can still add a Super-boot partition if there is non-partitioned space on the first hard-drive (BIOS drive 80h). Prior to implementing Super-boot, the hard disk must contain a low-level format as normally required before running FDISK, the partitioning utility included with ROM-DOS.

Implementation Procedure

The following procedure illustrates how to construct a Super-boot partition and two normal ROM-DOS partitions on a hard disk. It assumes that you have installed the ROM-DOS SDK and are able to produce a normal bootable floppy disk. In addition to the development machine on which ROM-DOS is installed, you will need a second machine with a hard drive on which you can run FDISK and FORMAT. You will also need two floppy disks which you can reformat.

This example creates three partitions. One is a Super-boot partition, the others are DOS partitions. Although two DOS partitions is not a requirement for Super-boot, the two DOS partitions are suggested because it better demonstrates the change in drive ordering when booting using different Super-boot libraries as a starting point. This example only proceeds through the steps using one of the Super-boot library options. The process can be repeated with other library choices.

1. **Prepare a normal ROM-DOS bootable floppy disk**
   On your development system, prepare a normal ROM-DOS bootable floppy disk. Copy the ROM-DOS FDISK and FORMAT utilities onto this floppy disk.

2. **Prepare a Super-boot bootable floppy disk**
   If you have an existing copy of USER.LIB in the root of your ROM-DOS directory tree, rename it to USER.BAK (or other convenient name) before continuing with this procedure.

   From the \DATALGHT\ROMDOS\SUPRBOOT directory, issue the command:

   ```
   COPY LASTB.LIB ..\USER.LIB
   ```

   Once the copy is complete, change back to the root of the ROM-DOS directory and run BUILD, selecting the quick build option (as outlined in the Chapter 4 of the ROM-DOS Developer's Guide). Build will produce a special Super-boot version of ROM-DOS that can be used to produce a bootable floppy disk. Format the second floppy disk using the newly created ROM-DOS.SYS file and then copy the ROM-DOS FORMAT utility onto this bootable Super-boot floppy disk.

3. **Partition the hard drive**
   Reboot the test machine off of the normal ROM-DOS boot disk. From the DOS command prompt, issue the following commands:

   ```
   FDISK 80 /I98 /S5 /C
   FDISK 80 /B /S15 /C
   FDISK 80 /B /C
   ```

   These commands do not work properly if the drive is already fully-partitioned, in which case, use the FDISK menu interface to remove all of the partitions on the disk before proceeding. This example assumes the hard drive has no existing partitions. These commands create a 5MB Super-boot partition, a second 15MB partition, and a third partition that fills the remainder of the disk (or stops at the 2GB partition size limit of DOS).

4. **Reboot from the normal ROM-DOS floppy disk and format the bootable DOS partition**
   Reboot from the normal ROM-DOS boot disk and then issue the following DOS commands:

   ```
   FORMAT C: /s
   FORMAT D:
   ```

   These commands prepare the second (15MB) partition as a bootable normal DOS partition using the normal version of ROM-DOS and make the third partition a normal non-bootable DOS partition. The Super-boot partition will not be visible to the normal ROM-DOS version nor to other operating systems.

5. **Reboot from the Super-boot floppy disk and format the Super-boot partition**
   Reboot the test machine using the Super-boot version of ROM-DOS constructed in step 2.

   The 5MB Super-boot partition appears as drive E while the 15MB normal DOS partition (made bootable in the previous step) is drive C. The third partition is drive D. The order in which DOS lists partitions is (with this particular version of the Super-boot kernel):

   1. DOS bootable partitions
   2. DOS primary non-bootable partitions
   3. DOS extended partitions
   4. Super-boot partitions

   If a different library from the SUPRBOOT sub-directory is chosen, then the relative order of the Super-boot partition may have been different. The implications of selecting other SUPRBOOT libraries with which to build ROM-DOS are discussed later. For now, prepare the Super-boot partition for use and continue. To complete this step, run FORMAT from the Super-boot bootable floppy disk as shown below:

   ```
   FORMAT E: /S
   ```

6. **Reboot and activate the Super-Boot partition**
   Remove the Super-boot floppy disk and reboot the test machine. When you see the lights on the keyboard flash, immediately press Alt-F2 to activate the Super-boot partition code. When you have finished booting to the COMMAND prompt, issue the DIR command to verify that you have booted from the 5MB Super-boot partition, and also note that the default drive is drive E. View a directory of drives C and D and notice that their drive order has not changed and that drive C is still the bootable DOS partition.

   While the Super-boot kernel is active, you can issue the VER command with the /R option to display the Super-boot options. Display of the Super-boot options are not available when booting from the normal ROM-DOS kernel.

7. **Reboot into the normal DOS partition**
   Reboot again, but this time allow the normal boot process to continue without activating the Super-boot partition using the Alt-F2 hotkey. When the boot sequence is complete, view a directory of drives C and D and verify that their order is preserved. Note that the Super-boot partition is no longer visible and that the VER /R command no longer lists the ROM-DOS Super-boot options.

Super-boot Partition Order

As previously mentioned, DOS will list the drives according to their partition type. The order in which ROM-DOS presents these drives depends on whether booting from a normal DOS kernel or a Super-boot kernel. A normal ROM-DOS kernel will assign drive letters to partitions in the following order:

1. DOS bootable partitions

2. DOS primary non-bootable partitions
3. DOS extended partitions

ROM-DOS provides several options in selecting the relative order of a Super-boot partition in the list above. These options are:

FIRST-The Super-boot partition will be assigned a drive letter before any of the other drive types (typically drive C).

MIDDLE - The Super-boot partition will always appear after the first bootable partition (typically drive D).

LAST - The Super-boot partition will appear as the last hard drive in the system (as illustrated in the preceding example).

For each of these drive letter assignments, except First and FirstB, you also have the option of reading CONFIG.SYS and AUTOEXEC.BAT (the boot files) from either the Super-boot drive or the normal boot drive (typically drive C). Super-boot library files that force the kernel to read these files from the Super-boot drive are given a 'B' suffix. Note that the ROM-DOS Super-boot kernel is only loaded from the Super-boot partition when pressing Alt-F2 during the boot process. The drive from which the boot files are read, once the Super-boot kernel gets control, depends on which Super-boot library was included in the ROM-DOS build. The First and FirstB libraries are identical and both boot from the first drive letter and read the boot files from the same drive.

The following table summarizes the affects of including the various Super-boot libraries in the ROM-DOS build. For simplicity, it is assumed that first available hard disk drive is drive C, although ROM-DOS allows for many configuration options that might change the first available hard disk drive.

Summary of Super-boot Libraries

Super-boot .LIB file	*Super-boot drive letter*	*Boot files read from drive*
FIRST	C:	C:
FIRSTB	C:	C:
MIDDLE	D:	C:
MIDDLEB	D:	D:
LAST	Last hard disk drive	C:
LASTB	Last hard disk drive	Last hard disk drive

Using Win95 or Win98 as Primary Operating System

One additional step to the process outlined above needs to be taken if Win95/98 is to be used as the primary partition operating system. After running FDISK on the drive, formatting and placing the system files on p and Super-boot partitions as instructed, you can install Win95/98 onto the primary partition. Create a partition large enough to accommodate the installion of the software and provide CD-ROM drivers as necessary for loading the Win95/98 software from a CD.

When the Win95/98 installation is completed, the Super-boot partition may no longer be accessible. To correct this, re-boot from the normal ROM-DOS floppy disk prepared in step 1. Rerun FDISK using the menu method (run FDISK without command line arguments). Select "V" to view the partitions and verify that the Super-boot partition is still present. Press Esc to return to the main FDISK menu, then run the "M" option to re-write the master boot record code. Press Esc to return to the main FDISK menu, then chose "Save and Exit". When the system reboots, pressAlt-F2 to activate the Super-boot partition.

Appendix E, Dynamic System Configuration

Introduction

When installing software on a system or configuring a system to accommodate its current operating environment, it may be beneficial to install only certain software. For example, when booting (or installing software) from a CD-ROM, it may be desirable to install only those software components, such as device drivers, required for the particular system instead of loading the system with unnecessary software. When installing device drivers at boot time, system resources can be maximized by installing only those drivers required for the installed hardware and omitting those for which no hardware exists.

How Does Dynamic System Configuration Work?

In a system where the exact hardware configuration is unknown, following procedure is performed at boot time to determine the need for and to load only the required device drivers. This procedure, which relies on the Dynamic Driver Loader program and the NEWFILE feature of Datalight's ROM-DOS, programmatically determines the need for specific device drivers and subsequently loads them from the installing media.

1. Run the Dynamic Driver Loader program. This program detects the installed hardware and writes a configuration file that reflects the detected hardware.
2. Process the configuration file stored on the RAM disk to install the needed device drivers from the installation media, such as a CD-ROM.

Note: Because the dynamic configuration process requires the creation of a configuration data file (performs disk I/O), a writeable disk must be available in the target system. If no such disk is present, such as when booting from CD-ROM or read-only memory, a temporary RAM disk must be created during the configuration process.

Using the Dynamic Driver Loader – An Example

The example of dynamic system configuration presented in this document describes a method of automatically configuring a typical x86-based system. This particular example checks the system for the presence of extended memory (XMS) and if found, installs two RAM disks above the 1MB boundary. This example is used because it can be implemented in any system equipped with extended memory and so serves as working example.

Configuration is performed by loading a small program (the Dynamic Driver Loader) during the processing of CONFIG.SYS. This program examines the system for extended memory and, when found, creates a pair of RAM disks for use by installed applications. While the following

example loads drivers that address memory as disk drives, the same concept can be used to load drivers for hardware components other than extended memory.

Examining the Example CONFIG.SYS File

In a ROM-DOS system, boot-up includes the processing of the CONFIG.SYS file located in the root directory of the default drive, which may be an installation CD-ROM or floppy disk. The following statement loads the HIMEM extended memory device driver, needed to use extended memory.

```
DEVICE=HIMEM.SYS
```

The next statement loads the VDISK.SYS RAM-disk driver. The Dynamic Driver example requires the use of a RAM disk on which to place the new configuration file. A customized Dynamic Driver could place the new file onto an alternate read/write drive. A 16KB RAM disk will be created on conventional memory.

```
DEVICE=VDISK.SYS 16
```

The next statement in CONFIG.SYS loads the Dynamic Driver Loader program, DYNDRVR.SYS.

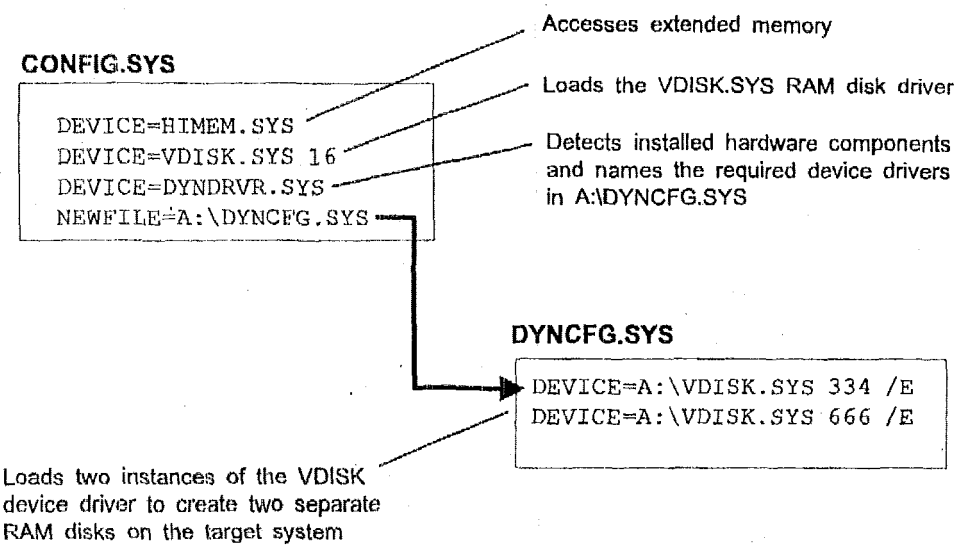
```
DEVICE=DYNDRVR.SYS
```

When loaded, DYNDRVR.SYS examines the target system and detects the hardware components. DYNDRVR.SYS then lists the required device driver for any hardware it detects as a DEVICE= statement in the file DYNCFG.SYS file. In its default form, DYNDRVR.SYS detects extended memory and writes the DEVICE=VDISK statements to DYNCFG.SYS.

```
NEWFILE=A:\DYNCFG.SYS
```

The last statement in this CONFIG.SYS file uses the NEWFILE command to access the DEVICE= statements specified by the DYNDRVR.SYS program and stored in the DYNCFG.SYS file on the disk drive. In this example, the DEVICE= statements in DYNCFG.SYS establish a pair of RAM disks if extended memory is available in the system.

For the example DYNDRVR.SYS driver to work, the NEWFILE= statement must be placed in the CONFIG.SYS file exactly as shown above. The statement must be uppercase. The reference to the A drive is adjusted by DYNDRVR.SYS to reflect the correct RAM driver letter.



About the Dynamic Driver Loader

The Dynamic Driver Loader is provided in C source code and can be adjusted to accommodate a wide range of hardware. When run (during the processing of CONFIG.SYS) in its default form, DYNDRVR.SYS detects extended memory and, if present, creates a pair of RAM disks. If extended memory is not present in the target system, DYNDRVR.SYS reports an error and terminates. The basic steps taken by DYNDRVR.SYS, as provided, include:

- Examine the target hardware for the existence of extended memory (XMS).
- If XMS is found, determine which drive to write the DYNCFG.SYS file.
- Update the NEWFILE command in CONFIG.SYS to point to the DYNCFG.SYS file.
- Create the DYNCFG.SYS file with the DEVICE= statements needed establish the two RAM disks.

The source code for DYNDRVR.SYS is a template that can be altered and added to as needed to suit a particular target system. The source code files are located in the ROM-DOS\MEMDISK subdirectory. Functionality can be added to detect any number and type of hardware components and then programmatically insert the appropriate DEVICE= statement(s) into DYNCFG.SYS. When DYNDRVR.SYS terminates, processing of CONFIG.SYS continues with the NEWFILE command, described below. Refer to "Chapter 7, Using a Custom Memory Disk" and the Makefile in the MEMDISK directory for instructions on compiling the source code. To create the driver using the supplied Makefile, use the command:

```
Make dyndrvr.sys
```

To manually compile the code:

```
TASM /z /mx /zd /I /DROMDISK=1 memdisk.asm
BCC -c -w -O -Z -I dyndrvr.c
TLINK /s /m /c /l memdisk+dyndrvr,dyndrvr.sys,,memutil.lib/m;
```

About the NEWFILE Command

Another key to this automated system configuration is the use of the NEWFILE command. NEWFILE allows normal CONFIG.SYS processing to take place within a separate file, named in the NEWFILE statement. When the NEWFILE command is processed by ROM-DOS, control passes from the current file (containing the NEWFILE statement) to the file named in the statement. Any commands in the current file that follow the NEWFILE statement are ignored. In the preceding example, NEWFILE causes the file DYNCFG.SYS to be processed and load the appropriate device drivers for the hardware detected by the driver DYNDRVR.SYS.

NEWFILE commands can be nested. That is, the original CONFIG.SYS file can contain a NEWFILE statement which names and executes a second file. The second file can include a NEWFILE statement to call a third file, and so on. Each file named in the successive NEWFILE statements must have a unique name, or be located on a different drive or directory, otherwise, an infinite loop results as control is repeatedly passed back to the same file.

If the filename specified in the NEWFILE statement cannot be located, processing continues with the original CONFIG.SYS file and the next statement is processed. The next statement can be a another NEWFILE= command, allowing flexibility for systems which may not have certain drives and files installed at boot time.

In summary, the first valid NEWFILE statement is processed, transferring control to the specified filename. The remaining instructions in the original CONFIG.SYS file, including any NEWFILE statements, are not processed unless the file named in the first NEWFILE statement is not found.

Appendix F, Supplemental ROM-DOS Utilities

General Information

The following utilities are a small collection of programs used by the BUILD utility during the building of ROM-DOS. Some of these tools are useful for other development purposes, and so have been included in this SDK.

DISK2IMG.EXE

DISK2IMG.EXE takes a disk and saves all of the contents in a raw binary format. This is used to make compressed ROM Disk images.

Syntax

DISK2IMG <d:> <filespec>

Remarks

Refer to the instructions in Chapter 5 on Compressing ROMDisks for more information.

Options

The <d:> argument specifies the drive letter of the disk to be compressed. Typically this will be a floppy drive, but could be any drive.

The <filespec> argument is the output file name for the disk image to be stored in. The output file name must be provided. No default file name is used.

Examples

```
DISK2IMG a: flop.img
```

This example creates an image file, called flop.img, from the contents of the a: drive.

```
DISK2IMG e: c:\temp\compress.bin
```

This example creates a file, called compress.bin, located in the c:\temp directory, with an image of the e: drive.

DUMP.EXE

DUMP allows you to show the contents of memory or a file in both hex and ASCII format to the display.

Syntax

DUMP <filename> [<offset>][<length>]
or
DUMP /<address> [<length>]

Remarks

The <offset>, <address>, and <length> arguments represent hexadecimal numbers.

Options

The <filename> argument is the name of the file that you want to dump to the screen.

The <offset> argument is the offset into the file that you want to start viewing data from.

The <address> argument is the memory address that you want to view. The address should be a two word value, for example: 03F8:0010.

The <length> argument specifies the length or amount of data to display from the specified file or memory address. If no length is specified for a file, the entire file will be displayed to the screen. For memory addresses, 70h bytes will be displayed.

Examples

```
DUMP TEST.EXE
```

Dump the entire file, test.exe to the screen. You can pipe this command through MORE to view one page at a time.

```
DUMP /40:0 100
```

Dump 100h bytes from memory, starting at addres 40:0.

EXE2BIN.COM

EXE2BIN attempts to convert an executable file into a com file.

Syntax

```
EXE2BIN <infile> <outfile>
```

Remarks

The following must be true for the conversion to be successful: CS=DS-SS (Tiny Model) and there cannot be any segment fixups. Segment fixup addresses are contained in the .exe header so that variable sin the code will be set to contain the segment address which can only be known when DOS loads the file. COM files do not have a mechanism to do fixups.

Examples

```
EXE2BIN test.exe test.com
```

Convert the .exe file test.exe into test.com.

HEXCAT.COM

HEXCAT.COM concatenates multiple HEX files into a single file.

Syntax

```
HEXCAT [/N<hexfilename>] <filespecs>
```

Remarks

The individual files are concatenated one after another. HEXCAT does not allow positional offsets or spacing between the files. If more precise placement is needed, the software provided with most PROM Programmer's will allow this kind of file location.

Options

The /N option allows you to specify the output file name for the concatenated files. If /N is not used, the default file name will be ROM.HEX.

The <filespecs> argument is the individual file names to be concatenated together into one file.

Examples

```
HEXCAT /Nbigfile.hex fileone.hex filetwo.hex
```

Concatenate the files fileone.hex and filetwo.hex into the single output file, bigfile.hex.

```
HEXCAT test1.hex test2.hex
```

Concatenate the files test1.hex and test2.hex into the default output file name of ROM.HEX.

PROTO.EXE

PROTO creates function prototypes in C language and Assembly language files.

Syntax

```
PROTO <filename> [<filenames>]
```

Remarks

The start of the function must begin with the first character in a new line. A simple space can cause the function prototype for this function to be overlooked. Which can be useful under certain circumstances. In order to have proto create a prototypes for the assembly files there must be a multi-line comment (see below) with a c-style function description.

```
    C language file :
unsigned MultiplyTwoBytes( unsigned char ucValue1, unsigned char ucValue2)
{
    ...
    }
    Assembly Language (ASM)file:
    comment ~
```

```
    unsigned MultiplyTwoBytes( unsigned char ucValue1, unsigned char
    ucValue2) {}
    -
    Public MultiplyTwoBytes
    MultiplyTwoBytes proc
    ...
MultiplyTwoBytes endp
```

The prototypes are only displayed to the screen so the user can redirect them where they want.

Proto takes every option on the command line and assumes that it is a file name. Therefore you can specify multiple file names on the command line. Wildcard characters are allowed in the file names.

Examples

```
PROTO hello.c hello.asm >hello.h
```

Create a prototype listing in the output file hello.h from the files hello.c and hello.asm.

```
PROTO test*.c >test.h
```

Create a prototype listing in the output file test.h from all of the files matching the name mask of test*.c in the current directory.

Appendix G, Licensed Utilities

Datalight offers the following utilities, licensed from other companies for the benefit of its customers. These utilities are not included in the SDK, but are available upon request on a "Supplemental Utilities Disk". For more information contact your Datalight representative or see our web site. The following overviews provide a basic description of these utilities.

Datalight Cache

An efficient method of enhancing system performance through disk drive access is to use a disk-caching program. Disk caching takes advantage of the fact that, during the course of normal operation, most users access the same program code and data repeatedly within a short period of time. The mechanics of a disk caching program involves the allocation of some system memory for temporary disk data storage. When the running application accesses data from the disk driver, the disk caching program saves the data to the temporary storage area in memory in the event it will be accessed again. If the application attempts to read the data again, the disk-caching program intercepts the disk access and provides the data stored in the temporary storage area established in system memory. Because the data is provided immediately by the disk-caching program instead of the operating system reading it from the disk drive, system speed is effectively improved.

Datalight Cache provides a boost in overall system performance. Tests show marked improvement in disk read operations when using Datalight Cache, the results of which are listed below. The following table compares the times required to read several files of differing sizes from a hard disk drive, and with buffers of varying sizes. While disk write operations undergo little or no improvement, read operations benefit significantly from the use of Datalight Cache.

File Size	Buffer Size	Read Time - No Cache	Read Time - Datalight Cache
5000KB	50KB	4.09sec	0.739sec
5000KB	25KB	4.14sec	0.79sec
2500KB	10KB	2.13sec	0.46sec
500KB	5KB	0.49sec	0.12sec
100KB	5KB	0.14sec	0.08sec
100KB	2KB	0.16sec	0.05sec

Based on internal Datalight tests, individual results may vary.

386MAX with ROM-DOS

While Datalight's EMM386 meets the needs of most OEM's products, there are customers that require a full-featured memory-management utility to support applications that use upper memory. 386MAX, licensed from Qualitas, Inc., meets the needs of those products that require more complete memory management support.

386MAX automatically configures the target system so that best use is made of available memory. In addition, 386MAX determines whether the monochrome display adapter (MDA) region of memory is available for use by other programs.

386MAX includes the following features, some of which are not included, or included to a lesser degree, in Datalight's EMM386:

No HIMEM.SYS — 386MAX does not require that HIMEM.SYS be loaded to access memory above the 1MB boundary.

Increased XMS Handles — 386MAX provides up to 65,535 XMS handles as compared to 128 with EMM386.

Supports XMS v3.x — 386MAX supports XMS v3.x. Datalight HIMEM.SYS supports XMS v2.x, which has a 32MB limit on memory allocation.

Small Memory Model — 386MAX uses significantly less conventional memory than Datalight EMM386.

Larger UMBs — 386MAX provides 84KB of upper memory blocks (UMBs) as compared to 28KB for Datalight EMM386, depending on the target system configuration.

Increased AMRs — 386MAX handles up to 255 alternate mapping register (AMR) sets as compared to seven for Datalight EMM386. (AMRs, traditionally implemented in hardware, are emulated in software by the memory manager.)

Increased Memory Pages — 386MAX can handle up to 60 mapable pages of memory depending on the hardware configuration. Datalight EMM386 handles four mapable pages and does not support the Reallocate function.

Glossary

BIOS

Basic Input Output System -- software that interfaces directly with the hardware.

BIOS Extension

A short program that the BIOS recognizes and executes as the BIOS initializes the system.

Boot

Booting is restarting and reloading DOS. A PC can be booted by turning it off and then turning it on or by pressing the Ctrl, Alt, and Del keys simultaneously.

Bootable disk

A system disk that contains the files necessary to start and run the computer.

Built-in Device

Built-in Device is an input/output device which is part of the DOS kernel.

FAT

File Allocation Table -- a data table which allows DOS to keep track of file location on the disk so that they can be accessed by programs running on DOS.

DOS

Disk Operating System -- an operating system that relies on disks for file storage.

DOS kernel

The DOS kernel is the part of DOS that handles a standard DOS call (Int 21h). It handles opening, reading/writing of files, loads programs, and manages memory.

Memory Disk

A disk that uses either ROM or RAM for the disk media. The memory disk has a FAT, directories, and file data.

PCMCIA

Personal Computer Memory Card Interface Association. PCMCIA is a group that defined the standard for a credit-card size card that may act as a memory RAMDISK, ROMDISK, or FLASHDISK. These cards are commonly referred to as PC cards.

POST

Power On Self Test -- a test performed by the BIOS that checks the computer hardware for problems before fully initializing the computer.

RAM Disk

A disk drive that uses RAM for the media in place of the usual rotating disk drive.

ROM-DOS

Datalight operating system that can be placed in and execute from within a ROM.

ROM disk

A disk drive that uses ROM for the media in place of the usual rotating disk drive.

ROM Scan

The scanning of the ROM area for BIOS extensions performed by the BIOS at initialization time.

ROM

Read Only Memory. This is memory that is not changeable once placed in a computer.

Shell

The Shell is the command interpreter, usually COMMAND.COM. The shell takes text commands and calls the DOS kernel to implement them.

System disk

See Bootable disk.

Index

D

S

T

6.1.12 Commented Source Code for ROM-DOS

The commented source code can be purchased from Datalight and requires a non-disclosure agreement to obtain.

6.1.13 Third-Party ROM-DOS References

Third-party references do not directly address ROM-DOS in the manner they address Microsoft DOS. Datalight makes the following recommendations in the READ.ME file included in the ROM-DOS Software Development Kit.

The following are all helpful texts on the topic of DOS and its internals (we use them at Datalight). The knowledge in them applies to Datalight's ROM-DOS as well as other DOS compatibles. They come highly recommended by Datalight.

1. Microsoft MS-DOS Programmer's Reference
   Microsoft Press, 1993
   ISBN 1-55615-546-8

2. DOS Internals
   by Geoff Chappell, Addison-Wesley 1994
   ISBN 0-201-60835-9

3. Undocumented DOS, 2nd Edition
   by Andrew Schulman, Addison-Wesley, 1993
   ISBN 0-201-63287-X

4. PC Interrupts, 2nd Edition
   by Ralph Brown, Addison Wesley, 1994
   ISBN 0-201-62485-0

The following books were also useful.

1. PC Intern 6th Edition
   by Michael Tischer and Bruno Jennrich,
   Abacus, 1996
   ISBN 1-55755-304-1

2. MS-DOS 6.2 Quick Reference
   by Sally Neuman, QUE, 1993
   ISBN 1-56529-655-9

3. The Indispensable PC Hardware Book, 3rd Edition
   by Hans-Peter Messmer, Addison-Wesley, 1997
   ISBN 0-201-40399-4

6.2 Phoenix, Award BIOS – System BIOS

The basic input/output software (BIOS) is loaded into every computer by the hardware vendor. This software handles the hardware specific instructions for the operating system. The computational block uses an Award BIOS. However, Phoenix now owns Award so information resides on the Phoenix Internet site.

6.2.1 Narrative Description of Award ROM BIOS

The following vendor description was downloaded from the Phoenix Internet site at:

http://www.phoenix.com/platform/awardbios.html.

AwardBIOS™

System Foundation Software

The full-featured AwardBIOS delivers the superior performance, compatibility, and functionality that your PC motherboard, low-end desktop, or embedded systems require. Its many options and extensions let you customize your products to a wide range of designs and target markets.

Broad Support

The modular, adaptable AwardBIOS supports the broadest possible range of third-party peripherals and all popular chipsets plus Intel, AMD, Cyrix, IBM and x86 compatible CPUs from 386 through Pentium Pro and Pentium II.

With Phoenix customer engineering facilities in North America, Asia, and Europe, we're always nearby to support new hardware designs, optimizing your time to market. Our BIOS modification utilities help you easily tailor BIOS features to suit your designs and your customers.

Latest Technology Advances

Phoenix Technologies consistently works with industry leaders and standards associations to ensure support of the latest technology standards:

- ACPI/APM. AwardBIOS complies with the Advanced Configuration and Power Interface (ACPI) specification, revision 1.0, and supports APM BIOS specifications through 1.2.
- NetPC. NetPC support in AwardBIOS lowers the Total Cost of Ownership of computer systems.
- PC 97. AwardBIOS complies with Microsoft® PC 97 BIOS requirements.
- Managed PC. Phoenix is implementing software to support the Managed PC initiative.
- I_2O (Intelligent I/O). Phoenix Technologies is a member of the I2O SIG and supports I2O architecture in the AwardBIOS.
- System Management (SM) BIOS. Phoenix Technologies has

Related Pages

Products
- for Desktops

Solutions
- 1394
- ACPI
- AGP
- BIOS
- PCI
- Plug & Play
- Power Management
- USB

participated actively in development of the SMBIOS specification, successor to the Desktop Management BIOS specification, and has implemented support in the AwardBIOS.
- Plug and Play. AwardBIOS detects and configures PnP devices during POST and, through its PnP extensions, communicates with Windows to determine system resources. Support of Microsoft's AML permits compatibility with PnP capability in future Windows operating systems (e.g., MS Windows 2000).
- PCI. AwardBIOS supports Intel's Peripheral Component Interconnect (PCI) bus specification ver. 2.1, including PCI-to-PCI and PCI-to-ISA bridging.
- USB. AwardBIOS USB module supports both Universal and Open HCI standards, maintaining full core compatibility and providing Legacy support for USB HID (human input device) hardware and multi-layered USB hubs.

AwardBIOS continues to support standard AT-compatible features, including Legacy PC AT interrupt service routines and compatibility with device service routines, to ensure a clear interface with existing software for your designs.

Preboot Management Solutions

Preboot Manager™ application and Preboot Agent™ BIOS extension let centrally located support personnel administer, diagnose and troubleshoot remote systems, without even loading an operating system. Preboot Agent redirects floppy drive calls (INT13h) to the Preboot Manager, permitting Manager operators to load applications onto the Agent from the Manager floppy drive.

Embedded & Internet Devices

Phoenix Technologies can work with you to design your individual solution. The flexible AwardBIOS adapts to support a wide variety of commercial and industrial applications, from uninterruptible process monitoring to consumer products.

AwardBIOS Features

Easy Customization

- Worldwide Customer Engineering
- Elimination of unnecessary modules
- Support of third-party peripheral ROMs

Industry-Standards Compliant

- ACPI, PCI, SMBIOS (DMI), USB, EISA
- BIOS Boot Specification
- Plug and Play
- PC Card
- Legacy PC AT ISRs & DSRs

CPU Support

- Intel, AMD, Cyrix, IBM, TI, IDT, etc.
- Low-power SMM & SMI

Chipset Support

- ACC, ALI, AMD, Intel, ITE, NSC, OPTi, SiS, UMC, VIA, etc.

Bus Support

- PCI, ISA, EISA, VL-Bus
- PCI-to-PCI & PCI-to-ISA bridging

Power Management

- ACPI 1.0, APM 1.2

Plug and Play Ready

- Resource allocation for PCI & PnP/Legacy ISA

Operating System Ready

- DOS/MS Windows 3.x
- MS Windows NT
- MS Windows 95/98
- OS/2 Warp
- NetWare
- SCO UNIX
- RTOS

HDD/FDD/CD-ROM ATAPI

- Selectable boot drive options, including CD-ROM, LS-120, SCSI, Iomega Zip, and network cards
- Auto-IDE detection
- "Fast" DMA transfer/Ultra DMA
- PIO modes 0-4
- High-performance LBA transfers
- Hard drives over 8.4 GB

Extensive POST

- Test and initialize all system components
- Quick POST option

Option ROM Support

- SCSI
- Video
- PC Card boot
- Ethernet

Flash Support

- Boot block BIOS for fault recovery

Memory Management

- RAM support up to 4 GB
- Auto-memory chipset sizing
- Auto-sizing for cache mapping
- System and video BIOS shadowing
- Memory parity check
- ECC support

Options

- Bus-mastering support
- Fast gate A20 support
- PS/2-style mouse support
- Combo-I/O controller support
- HDD LBA mode support
- Tick timer support
- Watchdog timer support
- CMOS backup
- Award Preboot Agent for remote system management and troubleshooting
- LM78 support and GUI utility

Utilities

- MODBIN for OEM changes to binary files
- AWDFLASH flash update
- CBROM binary combiner
- NVRAM archive/restore
- PCI configuration manager
- EISA configuration utility

Security Products

- Boot sector virus protection
- Multi-level password protection

Other Features

- ROM compression

6.2.2 BIOS Function Information and Calling Protocols

The following is the Phoenix BIOS 4.0 Release 6 users manual, which is very similar to the Award BIOS 4.0 in the Ampro CPU. This manual was downloaded from the Phoenix Internet site at:

http://www.phoenix.com/pcuser/userman.pdf.

Other BIOS information can be found on the Phoenix site's download menu at

http://www.phoenix.com/pcuser/index.html

PhoenixBIOS 4.0
Revision 6

User's Manual

Copyright

PhoenixBIOS 4.0 User's Manual

6 January 2000

Disclaimer

The programs are provided "as is" without warranty of any kind either expressed or implied, including but not limited to the implied warranties of merchantability and fitness for a particular purpose. This publication could contain technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of this publication. Phoenix Technologies Ltd. is without obligation to notify any person of such revisions or changes.

Purpose of Document

This guide explains how to configure your PC and optimize its performance using the Setup program. It also explains how to use the BIOS function calls in writing computer programs.

PB4.0 UM 01.06.00

Contents

Intro

About This Manual

This manual is divided into the following chapters:

Chapter 1 - The Setup Guide

This chapter describes a typical menu-driven Phoenix Setup program, which allows you to specify changes in the computer hardware (e.g. add a new diskette drive) and optimize system performance. Setup maximizes your control over your system's features and performance.

This Setup Guide is only an example. The Setup menus on your computer may be quite different. Consult the Setup manual supplied with your computer.

Chapter 2 - PhoenixBIOS Utilities

This chapter describes two new programs that give you more control over the boot process:

Phoenix QuietBoot
Phoenix MultiBoot

Chapter 3 - Phoenix Phlash

This chapter describes how to use the Phoenix Phlash utility for upgrading your BIOS without having to replace the BIOS ROM chip.

Chapter 4 - Programmer's Guide

This chapter gives programmers and expert PC users a detailed description of *Phoenix*BIOS. It contains the following sections:

Overview
Hardware Requirements
Fixed Disk Tables
Function Keys
POST Errors and Beep Codes
BIOS Services
BIOS Data Area
Interrupt Vector Table

Chapter 1 The Setup Guide

With the **PhoenixBIOS Setup** program, you can modify BIOS settings and control the special features of your computer. The Setup program uses a number of menus for making changes and turning the special features on or off.

Note: The menus shown here are from a typical system. The actual menus displayed on your screen may be quite different and depend on the hardware and features installed in your computer. For more accurate information about your BIOS Setup program, consult your system manual or contact the manufacturer.

The Main Menu

To start the ***Phoenix*BIOS** Setup utility:

Turn on or reboot your system. PhoenixBIOS displays this message:

```
Press <F2> to enter SETUP
```

2. Pressing <F2> displays the Main Menu, which looks like this:

```
                    PhoenixBIOS Setup Utility
  Main    Advanced    Security    Power    Boot    Exit
                                                  Item Specific Help
  System Time              [16:19:20]
  System Date:             [03/02/1994]           <Tab>, <Shift-Tab>, or
                                                  <Enter> selects field
  Legacy Diskette A:       [1.44/1.25  MB 3½"]
  Legacy Diskette B        [Not Installed]

▶ Primary Master           6449 MB
▶ Secondary Slave          None
▶ Secondary Master         CD-ROM
▶ Secondary Slave          None

  Numlock:                 [Disabled]

▶ Memory Cache             [Enabled]
▶ System Shadow            [Enabled]
▶ Video Shadow             [Enabled]

  System Memory              640 kB
  Extended Memory          31744 kB

  F1  Help    ↕ Select Item    -/+ Change Values        F9  Setup Defaults
  ESC Exit    ↔Select Menu     Enter Select  ▶Sub-Menu  F10 Save and Exit
```

See p. 4 for a description of the fields on this menu.

The Menu Bar

The Menu Bar at the top of the window lists these selections:

Main	Use this menu for basic system configuration.
Advanced	Use this menu to set the Advanced Features available on your system's chipset.
Security	Use this menu to set User and Supervisor Passwords and the Backup and Virus-Check reminders.
Power	Use this menu to configure Power-Management features.
Exit	Exits the current menu.

Use the left and right ↔ arrow keys to make a selection.

See the section below, "Exiting Setup," for a description on exiting the Main Menu.

The Legend Bar

Use the keys listed in the legend bar on the bottom to make your selections or exit the current menu. The chart on the following page describes the legend keys and their alternates:

Key	Function
<F1> or <Alt-H>	General Help window (See below).
<Esc>	Exit this menu.
↔ arrow keys	Select a different menu.
or arrow keys	Move cursor up and down.
<Tab> or <Shift-Tab>	Cycle cursor up and down.
<Home> or <End>	Move cursor to top or bottom of window.
<PgUp> or <PgDn>	Move cursor to next or previous page.
<F5> or <->	Select the Previous Value for the field.
<F6> or <+> or <Space>	Select the Next Value for the field.
<F9>	Load the Default Configuration values for this menu.
<F10>	Save and exit.
<Enter>	Execute Command or Select *P* Submenu.
<Alt-R>	Refresh screen.

To select an item, use the arrow keys to move the cursor to the field you want. Then use the plus-and-minus value keys to select a value for that field. The Save Values commands in the Exit Menu save the values currently displayed in all the menus.

To display a sub menu, use the arrow keys to move the cursor to the sub menu you want. Then press **<Enter>**.
A pointer (▸) marks all sub menus.

The Field Help Window

The help window on the right side of each menu displays the help text for the currently selected field. It updates as you move the cursor to each field.

The General Help Window

Pressing **<F1>** or **<Alt-H>** on any menu brings up the General Help window that describes the legend keys and their alternates:

```
                        General Help

Setup changes system behavior by modifying the BIOS
Configuration parameters.  Selecting incorrect values
may cause system boot failure; load Setup Default values
to recover.

<Up/Down> arrows select fields in current menu.
<PgUp/PgDn> moves to previous/next page on scrollable menus.
<Home/End> moves to top/bottom item of current menu.

Within a field, <F5> or <-> selects next lower value and
<F6>, <+>, or <Space> selects next higher value.

<Left/Right> arrows select menus on menu bar.
<Enter> displays more options for items marked with a ▶,
<Enter> also displays an option list on some fields.

<F9> loads factory-installed Setup Default values.
<F10> restores previous values from CMOS.

<ESC> or <Alt-X> exits Setup: in sub-menus, pressing these
keys returns to the previous menu.

<F1> or <Alt-H> displays General Help (this screen).

                        [Continue]
```

The scroll bar on the right of any window indicates that there is more than one page of information in the window. Use **<PgUp>** and **<PgDn>** to display all the pages. Pressing **<Home>** and **<End>** displays the first and last page. Pressing **<Enter>** displays each page and then exits the window.

Press **<Esc>** to exit the current window.

Main Menu Selections

You can make the following selections on the Main Menu itself. Use the sub menus for other selections.

Feature	Options	Description
System Time	HH:MM:SS	Set the system time.
System Date	MM/DD/YYYY	Set the system date.
Diskette 1 Diskette 2	360 kB, 5 ¼" 1.2 MB, 5 ¼" 720 kB, 3 ½" 1.44/1.25 MB, 3 ½" 2.88 MB, 3 ½" Not installed Disabled	Select the type of floppy-disk drive installed in your system. 1.25 MB is a Japanese media format that requires a 3½" 3-Mode Diskette drive.
System Memory	N/A	Displays amount of conventional memory detected during boot up.
Extended Memory	N/A	Displays the amount of extended memory detected during boot up.

You can set the boot sequence of the bootable drives by selecting **Boot Sequence** on the Main Menu or opening the **Boot Menu**.

Master and Slave Sub-Menus

The **Master** and **Slave** sub-menus accessed from the Main Menu control these types of devices:

Hard-disk drives
Removable-disk drives such as Zip drives
CD-ROM drives

*Phoenix*BIOS 4.0 supports up to two **IDE disk adapters**, called **primary** and **secondary** adapters. Each adapter supports one **master drive** and one optional **slave drive** in these possible combinations:

1 Master
1 Master, 1 Slave
2 Masters
2 Masters, 1 Slave
2 Masters, 2 Slaves

There is one IDE connector for each adapter on your machine, usually labeled "Primary IDE" and "Secondary IDE." There are usually two connectors on each ribbon cable attached to each IDE connector. When you have connected two drives to these connectors, the one on the end of the cable is the Master.

If you need to change your drive settings, selecting one of the Master or Slave drives on the Main Menu displays a sub-menu like this:

```
                          PhoenixBIOS Setup Utility
    Main
                  Primary Master                          Item Specific Help

  Type:                        [Auto]                  Select the drive type of
  Cylinders:                   [ 13328]                the fixed disk installed
  Heads:                       [   15]                 in your system. If type
  Sectors/Track:               [   63]                 User is selected,
  Maximum Capacity:            6449 MB                 Cylinders, Heads, and
                                                       Sectors can be edited
  Landing Zone:                [ 762]                  directly.
  Write Precomp:               [None]                  Auto attempts to
                                                       automatically detect the
  Multi Sector Transfer;       [16 Sectors]            drive type for drives
  LBA Mode Control:            [Enabled]               that comply with ANSI
  32-bit I/O:                  [Enabled]               specifications.
  Transfer Mode:               [Fast PIO 4]
  SMART Monitoring             [Enabled]

  F1  Help     ↕ Select Item     -/+ Change Values          F9  Setup Defaults
  ESC Exit     ↔Select Menu     Enter Select  ▶ Sub-Menu    F10 Save and Exit
```

Use the legend keys listed on the bottom to make your selections and exit to the Main Menu. Use the following chart to configure the hard disk.

Feature	Options	Description
Type	None 1 to 39 User Auto IDE Removable CD-ROM ATAPI Removable	None = Autotyping is not able to supply the drive type or end user has selected None, disabling any drive that may be installed. User = You supply the hard-disk drive information in the following fields. Auto = Autotyping, the drive itself supplies the correct drive information. IDE Removable = Removable read-and-write media (e.g., IDE Zip drive). CD-ROM = Readable CD-ROM drive. ATAPI Removable = Read-and-writea media (e.g., LS120, USB Floppy, USB Zip).
Cylinders	1 to 65,536	Number of cylinders.
Heads	1 to 16	Number of read/write heads.
Sectors/Track	1 to 63	Number of sectors per track.
Landing Zone*	1 to 2048	Number of the cylinder specified as the landing zone for the read/write heads.

Feature	Options	Description
Write Precomp*	1 to 2048 None	Number of the cylinder at which to change the write timing.
Multi-Sector Transfers	Disabled Standard 2 sectors 4 sectors 8 sectors 16 sectors	Any selection except Disabled determines the number of sectors transferred per block. Standard is 1 sector per block.
LBA Mode Control	Enabled Disabled	Enabling LBA causes Logical Block Addressing to be used in place of Cylinders, Heads, & Sectors.
32-Bit I/O	Enabled Disabled	Enables 32-bit communication between CPU and IDE card. Requires PCI or local bus.
Transfer Mode	Standard Fast PIO 1 Fast PIO 2 Fast PIO 3 Fast PIO 4	Selects the method for transferring the data between the hard disk and system memory. The Setup menu only lists those options supported by the drive and platform.
SMART Monitoring	Enabled Disabled	Turn on Self-Monitoring Analysis-Reporting Technology, which monitors condition of the hard drive and reports when a catastrophic IDE failure is about to happen.

* IDE drives do not require setting Landing Zone and Write Precomp.

When you enter Setup, the Main Menu usually displays the results of **Autotyping**– information each drive provides about its own parameters (e.g., cylinders, heads, and sectors)–and how the drives are arranged as Masters or Slaves on your machine.

Some older drives, however, do not use Autotyping and require selecting type User and entering a pre-defined fixed-disk type value (e.g., 1 to 39) or specifying the drive parameters separately with the User type selected. You can find the correct parameters for hard-disk drives in the drive manual or written on the casing of the drive itself.

Note: Exiting this menu keeps your selections but loses internal autotyping information, which may not be selected. If you exit this menu and re-enter it, press <Enter> on Autotype again to restore the Autotype information.

Note: Do not attempt to change these settings unless you have an older drive that does not support autotyping.

Note: Before changing the contents of this menu, **write them down**. Once you have established correct parameters for your drive, **write them down and store them in a safe place** (e.g., tape them to the disk drive) for use in case these values are lost in CMOS or if autotyping fails. If these hard-disk parameters are not correctly entered in CMOS, you cannot access the data on your drive.

WARNING: Incorrect settings can cause your system to malfunction. To correct mistakes, return to Setup and restore the Setup Defaults with <F9> and re-enter the correct drive parameters.

Memory Cache

Enabling **cache** saves time for the CPU by holding data most recently accessed in regular memory (dynamic RAM or DRAM) in a special storage area of static RAM (SRAM), which is faster. Before accessing regular memory, the CPU first accesses the cache. If it does not find the data it is looking for there, it accesses regular memory. Selecting "Memory Cache" from the Main menu displays a menu like the one shown here. The actual features displayed depend on your system's hardware.

```
                           PhoenixBIOS Setup Utility
     Main
                       Memory Cache                          Item Specific Help

  External cache:          [Enabled]                   Sets the state of the
                                                       external system memory
  Cache Interleave:        [Disabled]                  cache.
  Cache Write Back:        [Disabled]
  Cache Read Cycles:       [2T]

  Cache System BIOS:       [Disabled]
  Cache Video BIOS:        [Enabled]
  Cache E800 - EFFF:       [Disabled]
  Cache E000 - E7FF:       [Disabled]
  Cache D800 - DFFF:       [Disabled]
  Cache D000 - D7FF:       [Disabled]
  Cache C800 - CFFF:       [Disabled]
  Cache C800 - CFFF:       [Disabled]

  Non-cacheable Regions
    Region 0, start:   [     0 kB]
    Region 0, size:    [Disabled]
    Region 1, start:   [     0 kB]
    Region 1, size:    [Disabled]

  F1  Help    ↕ Select Item    -/+ Change Values        F9  Setup Defaults
  ESC Exit    ↔Select Menu     Enter Select  ▸ Sub-Menu  F10 Save and Exit
```

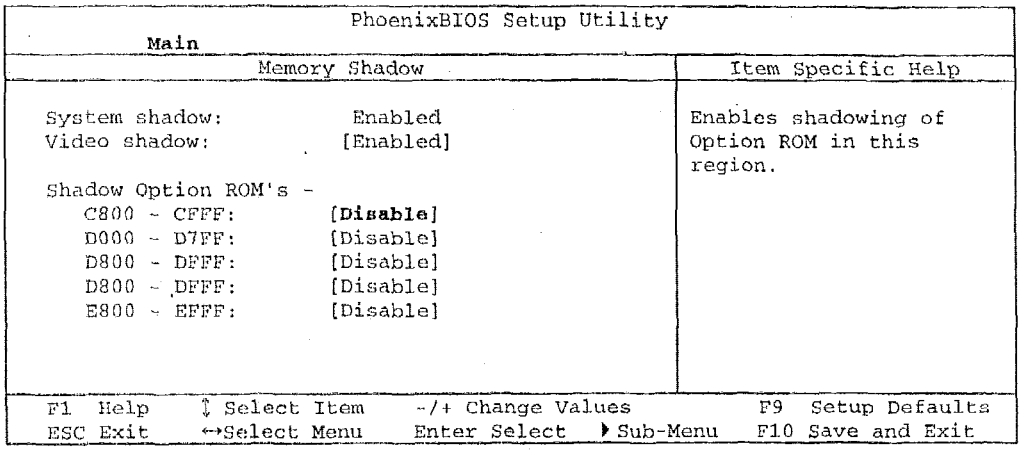
Use the legend keys listed on the bottom to make your selections and exit to the Main Menu. Use this chart to configure the memory cache.

Feature	Options	Description
External Cache	Enabled Disabled.	Generally enables or disables all memory caching.
Cache Interleave	Enabled Disabled	Interleaves multiple banks of static RAM. Improves CPU access.
Cache Write Back	Enabled Disabled	Enables caches to both read and write to memory. Disabled caches reads only.
Cache Read Cycles	Chipset Dependent	Sets the number of clock pulses for reading from the cache. Shorter number of pulses improves performance.
Cache Write Cycles	Chipset Dependent	Sets the number of clock pulses for writing to the cache. Shorter number of pulses improves performance.
Cache System BIOS	Enabled Disabled	Caches the system BIOS and improves performance.
Cache Video BIOS	Enabled Disabled	Caches the video BIOS and improves performance.
Cache segments, e.g., E800-EFFF	Enabled Disabled	Controls caching of individual segments of memory usually reserved for shadowing system or option ROMs
Non-cacheable regions:		Specifies areas of regular and extended memory as non-cacheable regions.
Region 0, start	0 Multiples of 64	Multiples of 64 define start of non-cacheable region 0 in kilobytes.
Region 0, size	Disabled Multiples of 64	Disabling makes this region available for cache. Multiples of 64 define size of non-cacheable region 0 in kilobytes.
Region 1, start	0 Multiples of 64	Multiples of 64 define start of non-cacheable region 1 in kilobytes.
Region 1, size	Disabled Multiples of 64	Disabling makes this region available for cache. Multiples of 64 define size of non-cacheable region 1 in kilobytes.

WARNING: Incorrect settings can cause your system to malfunction. To correct mistakes, return to Setup and restore the Setup Defaults with <F9>.

Memory Shadow

Selecting "System Shadow" or "Video Shadow" from the Main Menu displays a menu like the one shown here. The actual features displayed depend on the capabilities of your system's hardware.



Use the legend keys to make your selections and exit to the Main Menu. Use the following chart to configure memory shadowing.

WARNING: Incorrect settings can cause your system to malfunction. To correct mistakes, return to Setup and restore the Setup Defaults with <F9>.

Feature	Options	Description
System shadow	N/A	Usually permanently enabled.
Video shadow	Enabled Disabled	Shadows video BIOS and improves performance.
Shadow Option ROM	Enabled Disabled	Shadows option ROM located in the specified segments of memory and can improve performance. WARNING: Some option ROMs do not work properly when shadowed.

Boot Sequence

Selecting "Boot Sequence" on the Main Menu displays the Boot Options menu.

```
                          PhoenixBIOS Setup Utility
       Main
                      Boot Options                          Item Specific Help

 Boot sequence:    [Disabled]                             Order in which the
 SETUP prompt:     [Enabled]                              system searches for a
 POST Errors:      [Enabled]                              boot disk.
 Floppy check:     [Enabled]
 Summary screen:   [Enabled]

   F1  Help     ↕ Select Item    -/+ Change Values          F9   Setup Defaults
   ESC Exit     ↔Select Menu     Enter Select  ▶Sub-Menu    F10 Save and Exit
```

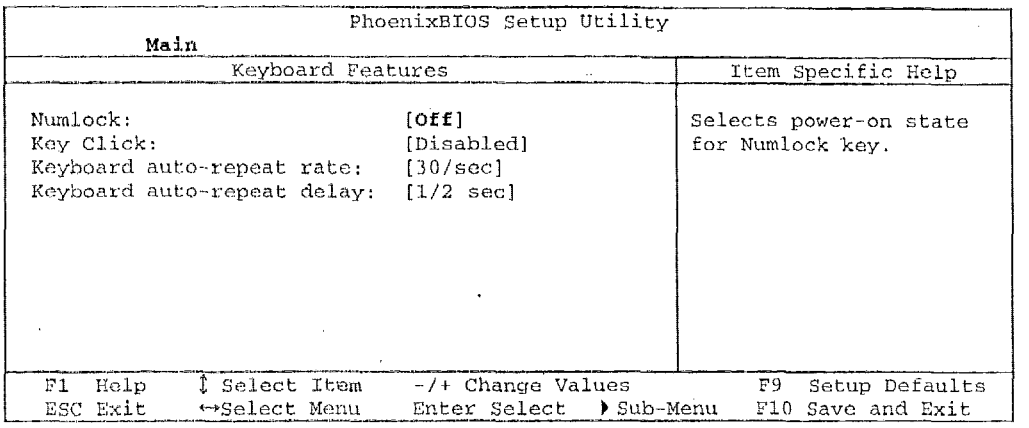
Use the legend keys to make your selections and exit to the Main Menu.

Use the following chart to select your boot options.

Feature	Options	Description
Boot sequence	A: then C; C: then A: C: only	The BIOS attempts to load the operating system from the disk drives in the sequence selected here. See also the Boot Menu on p. 11.
Setup prompt	Enabled Disabled	Displays "Press <F2> for Setup" during boot up.
POST errors	Enabled Disabled	At boot error, pauses and displays "Press <F1> to resume, <F2> to Setup".
Floppy seek	Enabled Disabled	Seeks diskette drives during boot up. Disabling speeds boot time.
Summary screen	Enabled Disabled	Displays system summary screen during boot up.

Keyboard Features

Selecting "Numlock" on the Main Menu displays the Keyboard Features menu:



Use the legend keys to make your selections and exit to the Main Menu.

Use the following chart to configure the keyboard features:

Feature	Options	Description
Numlock	Auto On Off	On or Off turns NumLock on or off at boot up. Auto turns NumLock on if it finds a numeric key pad.
Key Click	Enabled Disabled	Turns audible key click on.
Keyboard auto-repeat rate	2/sec 6/sec 10/sec 13.3/sec 21.8/sec 26.7/sec 30/sec	Sets the number of times a second to repeat a keystroke when you hold the key down.
Keyboard auto-lag delay	¼ sec ½ sec ¾ sec 1 sec	Sets the delay time after the key is held down before it begins to repeat the keystroke.

Boot Menu

After you turn on your computer, it will attempt to load the operating system (such as Windows 98) from the device of your choice. If it cannot find the operating system on that device, it will attempt to load it from one or more other devices in the order specified in the Boot Menu. Boot devices (i.e., with access to an operating system) can include: hard drives, floppy drives, CD ROMs, removable devices (e.g., Iomega Zip drives), and network cards.

Note: Specifying any device as a boot device on the Boot Menu requires the availability of an operating system on that device. Most PCs come with an operating system already installed on hard-drive C:.

Selecting "Boot" from the Menu Bar displays the Boot menu, which looks like this:

```
                    PhoenixBIOS Setup Utility
  Main    Advanced    Security    Power    Boot    Exit
                                                  Item Specific Help

QuickBoot Mode:              [Enabled]
Display OPROM Messages:      [Enabled]
Preferred Video:             [AGP]
Summary Screen:              [Enabled]

     Removable Devices
     ATAPI CD-ROM Drive
    -Hard Drive
        Primary Master
        Bootable Add-in Card
     Network Boot

                                          Use these keys to set
                                          the boot order in which
                                          the BIOS attempts to
                                          boot the OS:
                                          <+> or <-> moves device
                                          up or down.
                                          <Enter> expands or
                                          collapses devices marked
                                          with + or -.
                                          <Ctrl+Enter> expands all
                                          <Shift+1> enables or
                                          disables a device.
                                          <n> moves a removable
                                          device between hard or
                                          removable disk.

F1  Help   ↕ Select Item   -/+ Change Values      F9  Setup Defaults
ESC Exit   ↔Select Menu    Enter Select  ▶Sub-Menu  F10 Save and Exit
```

Use this menu to arrange to specify the priority of the devices from which the BIOS will attempt to boot the Operating System. In the example above, the BIOS will attempt first to boot from the CD-ROM drive (the only Removable Device listed). Failing that, it will attempt to boot from the Primary Master hard disk, and so on down the list.

Removable Devices, **Hard Drive**, and **Network Boot** are the generic types of devices on your system from which you can boot an operating system. You may have more than one device of each type. If so, the generic type is marked with a plus or minus sign. Use the <Enter> key to expand or collapse the devices marked with <+> or <->. Press <Ctrl+Enter> to expand all such devices.

Note: Floppy drives are not managed on this menu as part of Removable Devices.

To change a device's priority on the list, first select it with the up-or-down arrows, and move it up or down using the <+> and <-> keys. Pressing <n> moves a device between the Removable Devices and Hard Drive. Pressing <Shift+1> enables or disables a device.

Feature	Options	Description
QuickBoot Mode	Enabled Disabled	Enabled skips some POST tests, speeding boot time
Display OPROM Messages	Enabled Disabled	Displays boot messages of add-on cards. Recommended for newly installed cards. May be disabled later.
Preferred Video	AGP PCI	If you have more than one video card, select one to be used at boot.
Summary Screen	Enabled Disabled	Display system configuration screen during POST.

The Advanced Menu

Selecting "Advanced" from menu bar on the Main Menu displays a menu like this:

```
                        PhoenixBIOS Setup Utility
  Main    Advanced    Security    Power    Boot    Exit
                                                    Item Specific Help
Setting items on this menu to incorrect values
may cause your system to malfunction.

Installed Operating System          [Other]         Select the operating
Reset Configuration Data:           [No]            system installed on you
▶PCI Configuration                                  system that you use most
                                                    often.
PS/2 Mouse                          [Enabled]
Secured Setup Configurations        [No]            Note: An incorrect
▶Peripheral Configuration                           setting can cause
                                                    unexpected behavior in
Large Disk Access Mode:             [DOS]           some operating systems.
Local Bus IDE adapter:              [Both]
SMART Device Monitoring:            [Enabled]

▶Advanced Chipset Control
▶I/O Device Configuration

F1  Help    ↕ Select Item    -/+ Change Values        F9  Setup Defaults
ESC Exit    ↔Select Menu     Enter Select  ▶Sub-Menu  F10 Save and Exit
```

Use the legend keys to make your selections and exit to the Main Menu.

Feature	Options	Description
Installed Operating System	Other Win95 Win98/NT	Select the operating system you use most often.
Reset Configuration Data	Yes No	Yes erases all configuration data in a section of memory for **ESCD** (Extended System Configuration Data) which stores the configuration settings for non-PnP plug-in devices. Select Yes when required to restore the manufacturer's defaults.
PS/2 Mouse	Enabled Disabled Auto OS Controlled	Disabled disables any installed PS/2 mouse, but frees up IRQ 12 for use by another device. Auto lets the BIOS control the mouse. OS Controlled lets the operating system control the mouse.
Secured Setup Configurations	Yes No	Yes prevents the Operating System from overriding selections you have made in Setup.
Large Disk Access Mode	DOS Other	Select DOS if you have DOS. Select Other if you have another operating system such as UNIX. A large disk is one that has more than 1024 cylinders, more than 16 heads, or more than 63 tracks per sector.
SMART	Enabled Disabled	Enabled installs SMART (Self-Monitoring Analysis-Reporting Technology), which issues a warning if an IDE failure is imminent.

WARNING: Incorrect settings can cause your system to malfunction. To correct mistakes, return to Setup and restore the Setup Defaults with <F9>.

Advanced Chipset Control (No PCI)

In a system with no PCI, selecting "Advanced Chipset Control" from menu bar on the Advanced menu displays a menu like this:

```
                    PhoenixBIOS Setup Utility
                Advanced
                    Warning!                          Item Specific Help
  Setting items on this menu to incorrect values
  may cause your system to malfunction.               Controls system memory
                                                      parity through the
Parity check:              [Enabled]                  chipset.
Hidden refresh:            [Enabled]
Slow refresh:              [Disabled]
Read wait states:          [0]
Write wait states:         [0]
Extra bus wait states:     [0]
Multiple ALE:              [Enabled]
Keyboard reset delay:      [Disabled]

 F1  Help    ↕ Select Item    -/+ Change Values          F9  Setup Defaults
 ESC Exit    ↔Select Menu     Enter Select  ▸Sub-Menu    F10 Save and Exit
```

The chipset consists of one or more integrated circuits that act as an interface between the CPU and much of the system's hardware. You can use this menu to change the values in the chipset registers and optimize your system's performance.

Use the legend keys to make your selections, display the sub menus, and exit to the Main Menu.

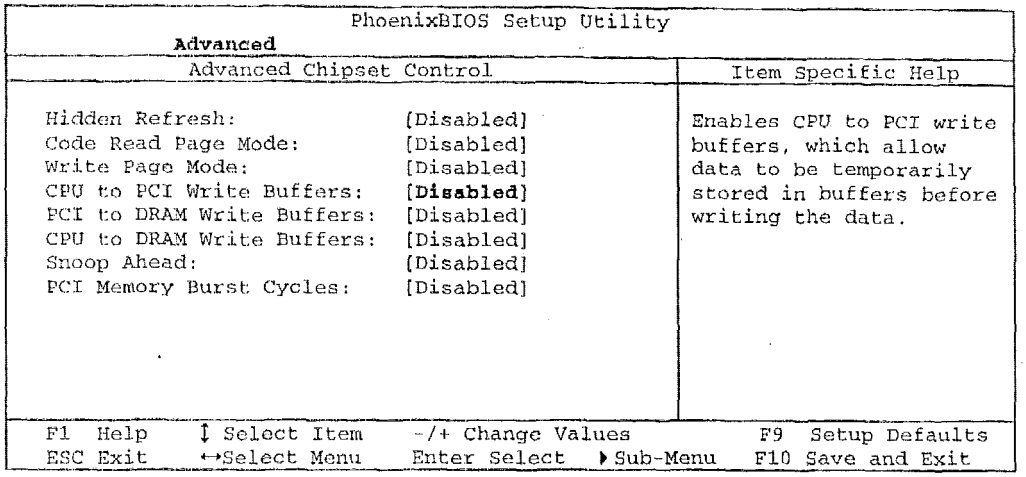
Use the following chart in configuring the chipset:

Feature	Options	Description
Parity check	Enabled Disabled	Controls system memory parity checking.
Hidden refresh	Enabled Disabled	Refreshes regular memory without holding up the CPU.
Slow Refresh	Enabled Disabled	Slows memory refresh by a factor of 4.
Read wait states	0 to n	Sets the number of wait states added to reads from system memory. Chipset dependent.
Write wait states	0 to n	Sets the number of wait states added to writes to system memory. Chipset dependent.
Extra bus wait states	0 to n	Sets the number of wait states added to accesses of the AT bus. Chipset dependent.
Multiple ALE	Enabled Disabled	Determines whether to use single or multiple ALEs during cycle conversion.
Keyboard reset delay	Enabled Disabled	Enabled adds a 2 microsecond delay before resetting the system.

NOTE: The contents of this menu depend on the chipset installed on your motherboard, and chipsets vary widely. Consult your dealer or the chipset manual before changing the items on this menu. **Incorrect settings can cause your system to malfunction.**

Advanced Chipset Control Menu (PCI BIOS)

If the system has a PCI chipset, selecting "Advanced Chipset Control" from the Advanced menu displays a menu like this:



The chipset is one or more integrated circuits that act as an interface between the CPU and the system's hardware. It manages such things as memory access, buses, and caching. You can use this menu to optimize the performance of your computer.

Use the legend keys to make your selections and exit to the Main Menu.

Use the following chart in configuring the chipset:

Feature	Options	Description
Hidden Refresh	Disabled Enabled	Refreshes regular memory without holding up the CPU
Code Read Page Mode	Disabled Enabled	Improves performance when code contains mainly sequential instructions.
Write Page Mode	Disabled Enabled	Improves performance when data is written sequentially.
CPU to PCI Write Buffers	Disabled Enabled	Stores CPU data in buffers before writing to PCI.
PCI to DRAM Write Buffers	Disabled Enabled	Stores PCI data in buffers before writing to DRAM.
CPU to DRAM Write Buffers	Disabled Enabled	Stores CPU data in buffers before writing to DRAM.
Snoop Ahead	Disabled Enabled	Improves PCI bus master access to DRAM.
PCI Memory Burst Cycles	Disabled Enabled	Enables PCI memory burst write cycles.

NOTE: The contents of this menu depend on the chipset installed on your motherboard, and chipsets vary widely. Consult your dealer or the computer manual before changing the items on this menu. **Incorrect settings can cause your system to malfunction.**

PCI Devices Menu

If the system has a PCI bus, selecting "PCI Devices" from menu bar on the Advanced menu displays a menu like this:

```
                    PhoenixBIOS Setup Utility
Advanced
PCI Devices                                    Item Specific Help

PCI Device Slot #1:                            Initialize device
  Option ROM Scan:                [Enabled]    expansion ROM
  Enable Master:                  [Disabled]
  Latency Timer:                  [0040h]

PCI Device Slot #2:
  Option ROM Scan:                [Disabled]
  Enable Master:                  [Disabled]
  Latency Timer:                  [0000]

PCI Device Slot #3
  Option ROM Scan:                [Disabled]
  Enable Master:                  [Disabled]
  Latency Timer:                  [0000]

Shared PCI IRQs:                  [Auto]

F1  Help   ↕ Select Item   -/+ Change Values        F9  Setup Defaults
ESC Exit   ↔Select Menu    Enter Select  ▸Sub-Menu  F10 Save and Exit
```

PCI Devices are devices equipped for operation with a **PCI** (Peripheral Component Interconnect) **bus**, a standardized Plug-and-Play hardware communication system that connects the CPU with other devices. Use this menu to configure the PCI devices installed on your system.

Use the legend keys to make your selections and exit to the Advanced menu. Use the following chart in configuring the PCI devices:

Feature	Options	Description
PCI Device Slots 1-n:		
Option ROM Scan	Disabled Enabled	Initialize device expansion ROM.
Enable Master	Disabled Enabled	Enables selected device as a PCI bus master. Not every device can function as a master. Check your device documentation.
Latency Timer	0000h to 0280h	Bus master clock rate. A high-priority, high-throughput device may benefit from a greater value.
Shared PCI IRQs	Share One IRQ Share Two IRQs Share Three IRQs Auto	Share *n* IRQs: Forces PCI devices to use at most *n* IRQs. Auto: Minimizes PCI IRQ Sharing.

NOTE: The contents of this menu depend on the devices installed on your system. **Incorrect settings can cause your system to malfunction. To correct mistakes, return to Setup and restore the System Defaults (F9).**

I/O Device Configuration Menu

The CPU communicates with external devices such as printers through devices called **Input/Output (I/O) ports** such as serial and parallel ports. These I/O devices require the use of system resources such as I/O addresses and interrupt lines. If these devices are Plug and Play, either the BIOS can allocate the devices during POST, or the operating system can do it. If the I/O devices are not Plug and Play, they may require manually setting them in Setup.

On some systems, the **chipset** manages the communication devices. Other systems have, instead, a separate **I/O chip** on the motherboard for configuring and managing these devices.

Many systems allow you to control the configuration settings for the I/O ports. Select "I/O Device Configuration" on the Advanced Menu to display this menu and specify how you want to configure these I/O Devices:

```
                    PhoenixBIOS Setup Utility
Advanced
I/O Device Configuration                    Item Specific Help

Serial Port A:          [Enabled]           Set Serial Port A:
  Base I/O address/IRQ  [3F8/IRQ4]          Using options:
Serial Port B:          [OS Controlled]
Parallel Port:          [User]              Disabled
   Mode:                [Bi-directional]      [No configuration]
   Base I/O address     [378]
   Interrupt            [IRQ5]              Enabled
                                              [User configuration]
Diskette Controller     [Enabled]
    Base I/O address:   [Primary]           Auto
                                              [BIOS configuration]

                                            OS Controlled
                                              [OS configuration]

F1  Help   ↕Select Item   -/+ Change Values        F9  Setup Defaults
ESC Exit   ↔Select Menu   Enter Select  ▸Sub-Menu  F10 Save and Exit
```

Use the legend keys to make your selections and exit to the Main Menu.

Use the following chart to configure the Input/Output settings:

Feature	Options	Description
Serial port A: Serial port B:	Disabled Enabled Auto OS Controlled	Disabled turns off the port. Enabled requires you to enter the base Input/Output address and the Interrupt number on the next line. Auto makes the BIOS configure the port automatically during POST. OS Controlled lets the PnP Operating System (such as Windows 95) configure the port after POST.
Base I/O Address/IRQ	3F8, IRQ 4 2F8, IRQ 3	If you select Enabled, choose one of these combinations.
Parallel Port:	Disabled Enabled Auto OS Controlled	Disabled turns off the port. Enabled requires you to enter the base Input/Output address and the Interrupt number below. Auto makes the BIOS auto configure the port during POST. OS Controlled lets the PnP Operating System (such as Windows 95) configure the port after POST.
Mode	Output only Bi-directional	Output only is standard one-way protocol for a parallel device. Bi-directional uses two-way protocol of an Extended Capabilities Port (ECP).
Base I/O Address	378 278 3BC	If you select Enabled for the Parallel Port, choose one of these I/O addresses.
Interrupts	IRQ5 IRQ7	If you select Enabled for the Parallel Port, choose one of these interrupt options.
Diskette Controller	Disabled Enabled	Enables the on-board legacy diskette controller. Disabled turns off all legacy diskette drives.
Base I/O Address	Primary Secondary	If you select Enabled for the Diskette Controller, choose Primary for one diskette drive installed or Secondary for two diskette drives installed.

Use this menu to specify how the I/O (Input and Output) ports are configured:

Manually by you.

Automatically by the BIOS during POST (See "ROM BIOS Functions" on page 32)

Automatically by a PnP Operating System such as Windows 95 after the Operating System boots.

Warning: If you choose the same I/O address or Interrupt for more than one port, the menu displays an asterisk (*) at the conflicting settings. It also displays this message at the bottom of the menu:

```
* Indicates a DMA, Interrupt, I/O, or memory resource
conflict with another device.
```

Resolve the conflict by selecting another settings for the devices.

The Security Menu

Selecting "Security" from the Main Menu displays a menu like this:

```
                    PhoenixBIOS Setup Utility
  Main    Advanced    Security    Power    Boot    Exit

                                                  Item Specific Help

Set User Password            [Enter]              Supervisor password
Set Supervisor Password      [Enter]              controls access to Setup
                                                  utility.
Virus Check Reminder:        [Disabled]
System backup Reminder:      [Disabled]

Password on boot:            [Disabled]
Diskette access:             [Disabled]
Fixed disk boot sector:      [Normal]

F1  Help   ↕ Select Item   -/+ Change Values        F9  Setup Defaults
ESC Exit   ↔Select Menu    Enter Select  ▸Sub-Menu  F10 Save and Exit
```

Use the legend keys to make your selections and exit to the Main Menu.

Enabling "Supervisor Password" requires a password for entering Setup. The passwords are not case sensitive.

Pressing <Enter> at either Set Supervisor Password or Set User Password displays a dialog box like this:

```
          Set Password
  Enter new password:  [        ]
  Confirm new password: [        ]
          Enter: Accept
```

Type the password and press <Enter>. Repeat.

Note: In some systems, the User and Supervisor passwords are related; you cannot have a User password without first creating a Supervisor password. In other systems, you can create and use them independently.

Use the following chart to configure the system-security and anti-virus options.

Feature	Options	Description
Set User Password	Up to seven alphanumeric characters	Pressing <Enter> displays the dialog box for entering the user password. In related systems, this password gives restricted access to SETUP menus.
Set Supervisor Password	Up to seven alphanumeric characters	Pressing <Enter> displays dialog box for entering the supervisor password. In related systems, this password gives full access to Setup menus.
Password on boot	Enabled Disabled	Enabled requires a password on boot. Requires prior setting of the Supervisor password. If supervisor password is set and this option disabled, BIOS assumes user is booting.
Diskette access	Enabled Disabled	Enabled requires a password to boot from or access the floppy disk.
Fixed disk boot sector	Normal Write Protect	Write protects the boot sector on the hard disk for virus protection. Requires a password to format or Fdisk the hard disk.
System backup reminder Virus check reminder	Disabled Daily Weekly Monthly	Displays a message during boot up asking (Y/N) if you have backed up the system or scanned it for viruses. Message returns on each boot until you respond with "Y". Daily displays the message on the first boot of the day, Weekly on the first boot after Sunday, and Monthly on the first boot of the month.

The Power Menu

Selecting "Power" from the menu bar displays a menu like this:

```
                        PhoenixBIOS Setup Utility
    Main    Advanced    Security    Power    Boot    Exit

                                                  Item Specific Help

Power Savings                   [Customize]

Standby Timeout:                [15 sec]          Select Power Management
  Auto Suspend Timeout:         [15 sec]          Mode. Choosing modes
                                                  changes system power
Hard Disk Timeout:              [10 min]          management settings.
Video Timeout:                  [ 5 min]          Maximum Power Savings
                                                  conserves the greatest
Resume On Modem Ring:           [Off]             amount of system power
Resume On Time:                 [Off]             while Maximum
                                                  Performance conserves
                                                  power but allows
                                                  greatest system
                                                  performance. To alter
▶ Advanced Options                                these settings, choose
                                                  Customize. To turn off
                                                  power management, choose
                                                  Disable.

F1  Help    ↕ Select Item    -/+ Change Values        F9  Setup Defaults
ESC Exit    ↔Select Menu     Enter Select  ▶Sub-Menu  F10 Save and Exit
```

Use this menu to specify your settings for Power Management. Remember that the options available depend upon the hardware installed in your system. Those shown here are from a typical system.

A power-management system reduces the amount of energy used after specified periods of inactivity. The Setup menu pictured here supports a **Full On** state, a **Standby** state with partial power reduction, and a **Suspend** state with full power reduction.

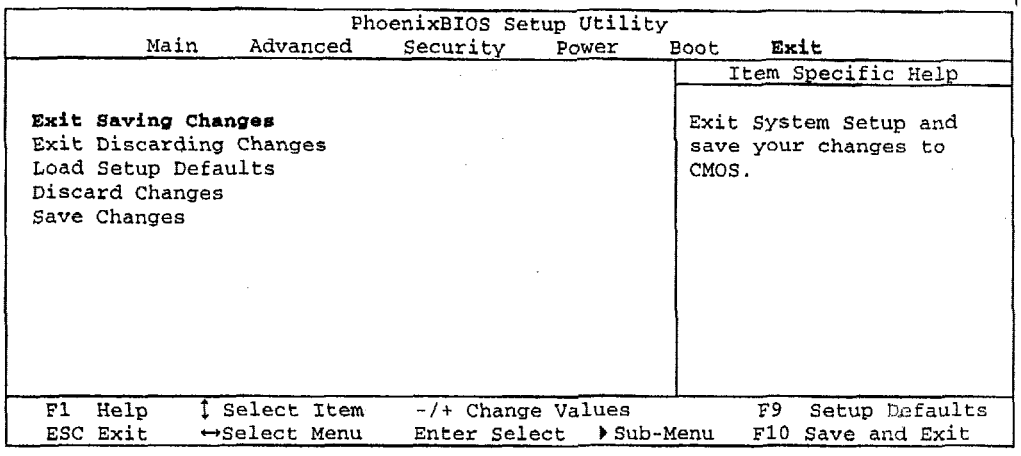
Use the Advanced Options on this menu to specify whether or not the activity of interrupts can terminate a Standby or Suspend state and restore Full On. Do not change these settings without knowing which devices use the interrupts.

Use the legend keys to make your selections and exit to the Main Menu. Use the following chart in making your selections:

Feature	Options	Description
Power Management Mode	Disabled Customize Maximum Power Savings Maximum Performance	Maximum options: pre-defined values. Select Customize to make your own selections from the following fields. Disabled turns off all power management.
Standby Timeout	Off 1 min 2 min 4 min 6 min 8 min 12 min 16 min	Inactivity period required to put system in Standby (partial power shutdown).
Auto Suspend Timeout	Disabled 5 min 10 min 15 min 20 min 30 min 40 min 60 min	Inactivity period required after Standby to Suspend (maximum power shutdown).
Hard Disk Timeout	Disabled 1 min 2 min 4 min 8 min 12 min 16 min	Inactivity period of hard disk required before standby (motor off).
Video Timeout	Disabled 10 sec 15 sec 20 sec 30 sec 45 sec 1 min to 15 min	Set inactivity period required before independently turning off monitor. Disabled turns CRT off in Standby.
Resume On Modem Ring	Off On	Wakes up system when an incoming call is detected on the modem.
Resume On Time	Off On	Wakes up system at predetermined time.
IRQ0...IRQ15 SMI NMI	Disabled Enabled	Enabling interrupt causes it to restore Full On during Standby or Suspend. SMI = System Management Interrupt. NMI = Non-Maskable Interrupt.

The Exit Menu

Selecting "Exit" from the menu bar displays this menu:



The following sections describe each of the options on this menu. Note that <Esc> does not exit this menu. You must select one of the items from the menu or menu bar to exit.

Saving Values

After making your selections on the Setup menus, always select either "Saving Values" or "Save Changes." Both procedures store the selections displayed in the menus in **CMOS** (short for "battery-backed CMOS RAM") a special section of memory that stays on after you turn your system off. The next time you boot your computer, the BIOS configures your system according to the Setup selections stored in CMOS.

After you save your selections, the program displays this message:

```
Values have been saved to CMOS!
    Press <space> to continue
```

If you attempt to exit without saving, the program asks if you want to save before exiting.

During boot up, *Phoenix*BIOS attempts to load the values saved in CMOS. If those values cause the system boot to fail, reboot and press **<F2>** to enter Setup. In Setup, you can get the Default Values (as described below) or try to change the selections that caused the boot to fail.

Exit Discarding Changes

Use this option to exit Setup without storing in CMOS any new selections you may have made. The selections previously in effect remain in effect.

Load Setup Defaults

To display the default values for all the Setup menus, select "Load Setup Defaults" from the Main Menu. The program displays this message:

```
ROM Default values have been loaded!
    Press <space> to continue
```

If, during boot up, the BIOS program detects a problem in the integrity of values stored in CMOS, it displays these messages:

```
System CMOS checksum bad - run SETUP
Press <F1> to resume,  <F2> to Setup
```

The CMOS values have been corrupted or modified incorrectly, perhaps by an application program that changes data stored in CMOS.

Press **<F1>** to resume the boot or **<F2>** to run Setup with the ROM default values already loaded into the menus. You can make other changes before saving the values to CMOS.

Discard Changes

If, during a Setup Session, you change your mind about changes you have made and have not yet saved the values to CMOS, you can restore the values you previously saved to CMOS.

Selecting "Discard Changes" on the Exit menu updates all the selections and displays this message:

```
CMOS values have been loaded!
    Press <space> to continue
```

Save Changes

Selecting "Save Changes" saves all the selections without exiting Setup. You can return to the other menus if you want to review and change your selections.

PhoenixBIOS Messages

The following is a list of the messages that the BIOS can display. Most of them occur during POST. Some of them display information about a hardware device, e.g., the amount of memory installed. Others may indicate a problem with a device, such as the way it has been configured. Following the list are explanations of the messages and remedies for reported problems.

*If your system displays one of the messages marked below with an asterisk (*), write down the message and contact your dealer. If your system fails after you make changes in the Setup menus, reset the computer, enter Setup and install Setup defaults or correct the error.

0200 Failure Fixed Disk
Fixed disk is not working or not configured properly. Check to see if fixed disk is attached properly. Run Setup. Find out if the fixed-disk type is correctly identified.

0210 Stuck key
Stuck key on keyboard.

0211 Keyboard error
Keyboard not working.

***0212 Keyboard Controller Failed**
Keyboard controller failed test. May require replacing keyboard controller.

0213 Keyboard locked - Unlock key switch
Unlock the system to proceed.

0220 Monitor type does not match CMOS - Run SETUP
Monitor type not correctly identified in Setup

0230 Shadow Ram Failed at offset: *nnnn
Shadow RAM failed at offset ***nnnn*** of the 64k block at which the error was detected.

***0231 System RAM Failed at offset: nnnn**
System RAM failed at offset ***nnnn*** of in the 64k block at which the error was detected.

0232 Extended RAM Failed at offset: nnnn** Extended memory not working or not configured properly at offset ***nnnn.

0250 System battery is dead - Replace and run SETUP
The CMOS clock battery indicator shows the battery is dead. Replace the battery and run Setup to reconfigure the system.

0251 System CMOS checksum bad - Default configuration used
System CMOS has been corrupted or modified incorrectly, perhaps by an application program that changes data stored in CMOS. The BIOS installed Default Setup Values. If you do not want these values, enter Setup and enter your own values. If the error persists, check the system battery or contact your dealer.

***0260 System timer error**
The timer test failed. Requires repair of system board.

***0270 Real time clock error** Real-Time Clock fails BIOS hardware test. May require board repair.

0271 Check date and time settings BIOS found date or time out of range and reset the Real-Time Clock. May require setting legal date (1991-2099).

0280 Previous boot incomplete - Default configuration used
Previous POST did not complete successfully. POST loads default values and offers to run Setup. If the failure was caused by incorrect values and they are not corrected, the next boot will likely fail. On systems with control of **wait states**, improper Setup settings can also terminate POST and cause this error on the next boot. Run Setup and verify that the wait-state configuration is correct. This error is cleared the next time the system is booted.

0281 Memory Size found by POST differed from CMOS
Memory size found by POST differed from CMOS.

02B0 Diskette drive A error

02B1 Diskette drive B error
Drive A: or B: is present but fails the BIOS POST diskette tests. Check to

see that the drive is defined with the proper diskette type in Setup and that the diskette drive is attached correctly.

02B2 Incorrect Drive A type - run SETUP
Type of floppy drive A: not correctly identified in Setup.

02B3 Incorrect Drive B type - run SETUP
Type of floppy drive B: not correctly identified in Setup.

02D0 System cache error - Cache disabled
RAM cache failed and BIOS disabled the cache. On older boards, check the cache jumpers. You may have to replace the cache. See your dealer. A disabled cache slows system performance considerably.

02F0: CPU ID:
CPU socket number for Multi-Processor error.

***02F4: EISA CMOS not writeable**
ServerBIOS2 test error: Cannot write to EISA CMOS.

***02F5: DMA Test Failed**
ServerBIOS2 test error: Cannot write to extended **DMA** (Direct Memory Access) registers.

***02F6: Software NMI Failed**
ServerBIOS2 test error: Cannot generate software NMI (Non-Maskable Interrupt).

***02F7: Fail-Safe Timer NMI Failed**
ServerBIOS2 test error: Fail-Safe Timer takes too long.

device Address Conflict
Address conflict for specified ***device.***

Allocation Error for: *device*
Run ISA or EISA Configuration Utility to resolve resource conflict for the specified ***device.***

CD ROM Drive
CD ROM Drive identified.

Entering SETUP ...
Starting Setup program

Failing Bits: *nnnn
The hex number ***nnnn*** is a map of the bits at the RAM address which failed the memory test. Each 1 (one) in the map indicates a failed bit. See errors 230, 231, or 232 above for offset address of the failure in System, Extended, or Shadow memory.

Fixed Disk *n*
Fixed disk ***n*** (0-3) identified.

Invalid System Configuration Data
Problem with NVRAM (CMOS) data.

I/O device IRQ conflict
I/O device IRQ conflict error.

PS/2 Mouse Boot Summary Screen:
PS/2 Mouse installed.

nnnn kB Extended RAM Passed
Where ***nnnn*** is the amount of RAM in kilobytes successfully tested.

nnnn Cache SRAM Passed
Where ***nnnn*** is the amount of system cache in kilobytes successfully tested.

nnnn kB Shadow RAM Passed
Where ***nnnn*** is the amount of shadow RAM in kilobytes successfully tested.

nnnn kB System RAM Passed
Where ***nnnn*** is the amount of system RAM in kilobytes successfully tested.

One or more I2O Block Storage Devices were excluded from the Setup Boot Menu
There was not enough room in the IPL table to display all installed I_2O block-storage devices.

Operating system not found
Operating system cannot be located on either drive A: or drive C:. Enter Setup and see if fixed disk and drive A: are properly identified.

Parity Check 1 *nnnn
Parity error found in the system bus. BIOS attempts to locate the address and display it on the screen. If it cannot locate the address, it displays ????. Parity is a method for checking errors in binary data. A parity error indicates that some data has been corrupted.

Parity Check 2** ***nnnn
Parity error found in the I/O bus. BIOS attempts to locate the address and display it on the screen. If it cannot locate the address, it displays **????.**

Press <F1> to resume, <F2> to Setup, <F3> for previous
Displayed after any recoverable error message. Press <F1> to start the boot process or <F2> to enter Setup and change the settings. Press <F3> to display the previous screen (usually an initialization error of an **Option ROM**, i.e., an add-on card). Write down and follow the information shown on the screen.

Press <F2> to enter Setup
Optional message displayed during POST. Can be turned off in Setup.

PS/2 Mouse:
PS/2 mouse identified.

Run the I2O Configuration Utility
One or more unclaimed block storage devices have the Configuration Request bit set in the LCT. Run an I2O Configuration Utility (e.g. the SAC utility).

System BIOS shadowed
System BIOS copied to shadow RAM.

UMB upper limit segment address: ***nnnn***
Displays the address ***nnnn*** of the upper limit of **Upper Memory Blocks**, indicating released segments of the BIOS which can be reclaimed by a virtual memory manager.

Video BIOS shadowed
Video BIOS successfully copied to shadow RAM.

Chapter 2 Boot Utilities

Phoenix Boot Utilities are:

Phoenix QuietBoot™
Phoenix MultiBoot™

Phoenix QuietBoot displays a graphic illustration rather than the traditional POST messages while keeping you informed of diagnostic problems.

Phoenix MultiBoot is a boot screen that displays a selection of boot devices from which you can boot your operating system.

Phoenix QuietBoot

Right after you turn on or reset the computer, **Phoenix QuietBoot** displays the QuietBoot Screen, a graphic illustration created by the computer manufacturer instead of the text-based POST screen, which displays a number of PC diagnostic messages.

To exit the QuietBoot screen and run Setup, display the MultiBoot menu, or simply display the PC diagnostic messages, you can simply press one of the hot keys described below.

The QuietBoot Screen stays up until just before the operating system loads unless:

1. You press <Esc> to display the POST screen.
2. You press <F2> to enter Setup.
3. POST issues an error message.
4. The BIOS or an option ROM requests keyboard input.

The following explains each of these situations.

Press <ESC>

Pressing <Esc> switches to the POST screen and takes one of two actions:

1. If MultiBoot is installed, the boot process continues with the text-based POST screen until the end of POST, and then displays the **Boot First Menu**, with these options:
   a. Load the operating system from a boot device of your choice.
   b. Enter Setup.
   c. Exit the Boot First Menu (with <Esc>) and load the operating system from the boot devices in the order specified in Setup.
2. If MultiBoot is not installed, the boot process continues as usual.

Press <F2>

Pressing <F2> at any time during POST switches to the POST screen (if not already displayed) and enters Setup.

POST Error

Whenever POST detects a non-fatal error, QuietBoot switches to the POST screen and displays the errors. It then displays this message:

```
Press <F1> to resume, <F2> to Setup
```

Press <F1> to continue with the boot. Press <F2> if you want to correct the error in Setup.

Keyboard Input Request

If the BIOS or an **Option ROM** (add-on card) requests keyboard input, QuietBoot switches over to the POST screen and the Option ROM displays prompts for entering the information. POST continues from there with the regular POST screen.

Phoenix MultiBoot

Phoenix MultiBoot expands your boot options by letting you choose your boot device, which could be a hard disk, floppy disk, or CD ROM. You can select your boot device in Setup, or you can choose a different device each time you boot during POST by selecting your boot device in **The Boot First Menu**.

MultiBoot consists of:

The Setup Boot Menu
The Boot First Menu

See the Setup Boot menu on p. 11. The following describes the Boot First Menu.

The Boot First Menu

Display the Boot First Menu by pressing <Esc> during POST. In response, the BIOS first displays the message, "Entering Boot Menu ..." and then displays the Boot Menu at the end of POST. Use the menu to select any of these options:

1. Override the existing boot sequence (for this boot only) by selecting another boot device. If the specified device does not load the operating system, the BIOS reverts to the previous boot sequence.
2. Enter Setup.
3. Press <Esc> to continue with the existing boot sequence.

```
Boot Menu

Select boot device or Setup.
Use the Up and Down arrows to
select the Boot First device,
then <Enter> or press <Esc> to
exit.

  1.  Hard Drive
  2.  ATAPI CD-ROM
  3.  Diskette Drive
  4.  Removable Devices
  5.  Network Boot

      <Setup>
```

If there is more than one bootable hard drive, the first one in the Setup Boot menu is the one represented here.

Chapter 3 Phoenix Phlash

Phoenix Phlash gives you the ability to update your BIOS from a floppy disk without having to install a new ROM BIOS chip.

Phoenix Phlash is a utility for "flashing" (copying) a BIOS to the Flash ROM installed on your computer from a floppy disk. A Flash ROM is a Read-Only Memory chip that you can write to using a special method called "flashing." Use Phoenix Phlash for the following tasks:

Update the current BIOS with a new version.

Restore a BIOS when it has become corrupted.

Installation

Phoenix Phlash is shipped on a floppy disk with your computer as a compressed file called CRISDISK.ZIP that contains the following files:

CRISDISK.BAT	Executable file for creating the Crisis Recovery Diskette.
PHLASH.EXE	Programs the flash ROM.
PLATFORM.BIN	Performs platform-dependent functions.
BIOS.ROM	Actual BIOS image to be programmed into flash ROM.
MINIDOS.SYS	Allows the system to boot in Crisis Recovery Mode.
MAKEBOOT.EXE	Creates the custom boot sector on the Crisis Recovery Diskette.

To install Phoenix Phlash on your hard disk, follow this simple procedure:

1. Insert the distribution diskette into drive A:
2. Unzip the contents of CRISDISK.ZIP into a local directory, presumably `C:\PHLASH`.
3. Store the distribution diskette in a safe place.

Create the Crisis Recovery Diskette

If the OEM or dealer from whom you purchased your system has not provided you with one, then you should create a **Crisis Recovery Diskette** before you use the Phlash utility. If you are unable to boot your system and successfully load the Operating System, the BIOS may have been corrupted, in which case you will have to use the Crisis Recovery Diskette to reboot your system. There are several methods that you can use to create the Crisis Recovery Diskette. Below is one recommended procedure.

1. Be sure you have successfully installed the Phlash Utility onto your hard disk.
2. Insert a clean diskette into drive A: or B:
3. From the local directory, enter the following:

   CRISDISK [drive]:

   where [*drive*] is the letter of the drive into which you inserted the diskette. For help, type **/?** or **/h**.

   CRISDISK.BAT formats the diskette, then copies MINIDOS.SYS, VGABIOS.EXE (if available), PHLASH.EXE, PLATFORM.BIN and BIOS.ROM to the diskette, and creates the required custom boot sector.

4. Write protect and label the Crisis Recovery Diskette.

NOTE: You can only supply a volume label after the Crisis Recovery Diskette has been formatted and the necessary files copied because MINIDOS.SYS must occupy the first directory entry for the diskette to boot properly.

Updating the Crisis Recovery Diskette

If the BIOS image (BIOS.ROM) changes due to an update or bug fix, you can easily update the Crisis Recovery Diskette. Simply copy the new BIOS.ROM image onto the Crisis Recovery Diskette. No further action is necessary.

Executing Phoenix Phlash

You can run Phoenix Phlash in one of two modes:

Command Line Mode

Crisis Recovery Mode

WARNING! For your own protection, be sure you have a Crisis Recovery Diskette ready to use before executing Phlash.

Command Line Mode

Use this mode to update or replace your current BIOS. To execute Phlash in this mode, move to the directory into which you have installed Phoenix Phlash and type the following:

```
phlash
```

Phoenix Phlash will automatically update or replace the current BIOS with the one which your OEM or dealer supplies you.

Phlash may fail if your system is using memory managers, in which case the utility displays the following message:

```
Cannot flash when memory managers are present.
```

If you see this message after you execute Phlash, you must disable the memory manager on your system. To do so, follow the instructions in the following sections.

Disabling Memory Managers

To avoid failure when flashing, you must disable the memory managers that load from CONFIG.SYS and AUTOEXEC.BAT. There are two recommended procedures for disabling the memory managers. One consists of pressing the <F5> key (only if you are using DOS 5.0 or above), and the other requires the creation of a boot diskette.

DOS 5.0 (or later version)

For DOS 5.0 and later, follow the two steps below to disable any memory managers on your system. If you are not using at least DOS 5.0, then you must create a boot diskette to bypass any memory managers (See Create a Boot Diskette, below).

1. Boot DOS 5.0 or later version. (In Windows 95, at the boot option screen, choose Option 8, "Boot to a previous version of DOS.")
2. When DOS displays the "Starting MS-DOS" message, press <F5>.

After you press <F5>, DOS bypasses the CONFIG.SYS and AUTOEXEC.BAT files, and therefore does not load any memory managers.

You can now execute Phlash.

Create a Boot Diskette

To bypass memory managers in DOS versions previous to 5.0, follow this recommended procedure:

1. Insert a diskette into your A: drive.
2. Enter the following from the command line:

   ```
   Format A: /S
   ```

3. Reboot your system from the A: drive.

Your system will now boot without loading the memory managers, and you can then execute Phlash.

NOTE: The boot diskette you create here is distinct from a *Crisis Recovery Diskette*. See page 409 for details about creating the Crisis Recovery Diskette.

Crisis Recovery Mode

You should only have to operate Phoenix Phlash in this mode only if your system does not boot the operating system when you turn on or reset your computer. In these cases, the BIOS on the Flash ROM has probably been corrupted. Boot your system with the Crisis Recovery Diskette taking these steps:

1. Insert the Crisis Recovery diskette (which your dealer supplied or one that you should have created from the instructions above) into drive A:.
2. Reset your computer, power off-on, or press <Ctrl> <Alt> <Del> to reboot the system.
3. When your system reboots, Phoenix Phlash will restore the BIOS from the diskette and successfully boot the operating system.

Chapter

4 Programmer's Guide

This chapter of the User's Manual gives application developers, programmers, and expert computer users a detailed description of the BIOS.

This chapter describes the following subjects:

- What is a ROM BIOS?
- System Hardware Requirements
- Fixed-Disk Tables
- PhoenixBIOS Function Keys
- POST Errors
- Run-Time Services

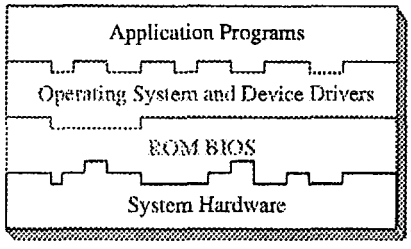
What is a ROM BIOS?

This section briefly explains the function of a BIOS in managing the special features of your system.

A **ROM BIOS (Basic Input/Output System)** is a set of programs permanently stored in a **ROM** (Read-Only Memory) chip located on the computer motherboard. These programs micro-manage the hardware devices installed on your computer. When you turn on your computer, the ROM BIOS initializes and tests these devices. During run-time, the ROM BIOS provides the Operating System and application programs with access to these devices. You can also use the BIOS **Setup** program to change your computer's hardware or behavior.

Software works best when it operates in layers, and the ROM BIOS is the bottom-most software layer in the computer. It functions as the interface between the hardware and the other layers of software, isolating them from the details of how the hardware works. This arrangement enables you to change hardware devices without having to install a new operating system.

The following diagram shows the function of the ROM BIOS as the interface between the hardware and other layers of software:



ROM BIOS Functions

The *PhoenixBIOS* software performs these functions:

The Setup Program	Using the Setup program, you can install, configure, and optimize the hardware devices on your system (clock, memory, disk drives, etc.).
Initialize Hardware at Boot	At power-on or reset, perform Power-On Self Test (POST) routines to test system resources and run the operating system.
Run-Time Routines	Basic hardware routines that can be called from DOS and Windows applications.

Initialize and Configure the computer

The first job of a ROM BIOS is to initialize and configure the computer hardware when you turn on your computer (system boot). The BIOS runs a series of complex programs called the **Power On Self Test** (POST), which performs a number of tasks, including:

Test Random Access Memory (RAM)

Conduct an inventory of the hardware devices installed in the computer

Configure hard and floppy disks, keyboard, monitor, and serial and parallel ports

Configure other devices installed in the computer such as CD-ROM drives and sound cards

Initialize computer hardware required for computer features such as Plug and Play and Power Management

Run Setup if requested

Load and run the Operating System such as DOS, OS/2, UNIX, or Windows 95 or NT.

BIOS Services

The second task of the ROM BIOS is to provide the Operating System, device drivers, and application programs with access to the system hardware. It performs this task with a set of program routines called **BIOS Services**, which are loaded into high memory at boot time.

The number of BIOS Services is always changing. The BIOS Services of PhoenixBIOS 4.05 provide precise control of hardware devices such as disk drives, which require careful management and exhaustive checking for errors. They also help manage new computer features such as Power Management, Plug and Play, and MultiBoot.

System Hardware Requirements

PhoenixBIOS 4.0 requires the following hardware components on the motherboard:

System Board Requirements
1. CPU (486 or later)
2. AT-compatible and MC146818 RTC-compatible chipset.
3. AT or PS/2-compatible Keyboard controller
4. At least 1 MB of system RAM

The Power On Self Test (POST) of the BIOS initializes additional ROM BIOS extensions (Option ROMs) if they are accessible in the proper format. The requirements are:

Adapter ROM Requirements
1. The code must reside in the address space between C0000H and F0000H.
2. The code must reside on a 2K boundary.
3. The first two bytes of the code must be 55H and AAH.
4. The third byte must contain the number of 512-byte blocks.
5. The fourth byte must contain a jump to the start of the initialization code.
6. The code must checksum to zero (byte sum).

NOTE: The address space from C0000H to C8000H is reserved for external video adapters (e.g. EGA, VGA). Part of the address space from D0000H to E0000H is typically used by expanded memory (EMS).

Fixed Disk Tables

PhoenixBIOS 4.0 supports up to four fixed-disk drives. For each drive, it supports 39 pre-defined drive types and four user-defined types (40-43). Below is a table of the pre-defined drive types and their default values.

End users can modify the user-defined drive type for each fixed disk listed in Setup by using the menus of the Setup program. This feature avoids the need for customized software for non-standard drives.

NOTE: Since most hard drives are autotyped (i.e., automatically determined by the BIOS or Operating System), there is usually no need to set the drive geometry manually.

Type	Cylinders	Heads	Sectors	Wrt Pre	Landing
1	306	4	17	128	305
2	615	4	17	300	615
3	615	6	17	300	615
4	940	4	17	512	940
5	940	6	17	512	940
6	615	4	17	-1	615
7	462	8	17	256	511
8	733	5	17	-1	733
9	900	15	17	-1	901
10	820	3	17	-1	820
11	855	5	17	-1	855
12	855	7	17	-1	855
13	306	8	17	128	319
14	733	7	17	-1	733
15	Reserved				
16	612	4	17	0	633
17	977	5	17	300	977
18	977	7	17	-1	977
19	1024	7	17	512	1023
20	733	5	17	300	732
21	733	7	17	300	732
22	733	5	17	300	733
23	306	4	17	0	336
24	612	4	17	305	663
25	612	2	17	300	612
26	614	4	17	-1	614
27	820	6	17	-1	820
28	977	5	17	-1	977
29	1218	15	36	-1	1218

Type	Cylinders	Heads	Sectors	Wrt Pre	Landing
30	1224	15	17	-1	1224
31	823	10	17	512	823
32	809	6	17	128	809
33	830	7	17	-1	830
34	830	10	17	-1	830
35	1024	5	17	-1	1024
36	1024	8	17	-1	1024
37	615	8	17	128	615
38	1024	8	26	-1	1024
39	925	9	17	-1	925
40	User def.				
41	User def.				
42	User def.				
43	User def.				

PhoenixBIOS Function Keys

The following are the special PhoenixBIOS function keys:

<F2>	Enter SETUP program during POST
Ctrl-Alt-<->	Switch to slow CPU speed
Ctrl-Alt-<+>	Switch to fast CPU speed

The speed switching keys are only operational when speed switching is available.

POST Errors and Beep Codes

Recoverable POST Errors

Whenever a recoverable error occurs during POST, *Phoenix*BIOS displays an error message describing the problem.

*Phoenix*BIOS also issues a beep code (one long tone followed by two short tones) during POST if the video configuration fails (no card installed or faulty) or if an external ROM module does not properly checksum to zero.

An external ROM module (e.g. VGA) can also issue audible errors, usually consisting of one long tone followed by a series of short tones.

Terminal POST Errors

There are several POST routines that issue a **POST Terminal Error** and shut down the system if they fail. Before shutting down the system, the terminal-error handler issues a beep code signifying the test point error, writes the error to port 80h, attempts to initialize the video, and writes the error in the upper left corner of the screen (using both mono and color adapters).

The routine derives the beep code from the test point error as follows:

1. The 8-bit error code is broken down to four 2-bit groups (Discard the most significant group if it is 00).
2. Each group is made one-based (1 through 4) by adding 1.
3. Short beeps are generated for the number in each group.

   Example:

   Test point 01Ah = 00 01 10 10 = 1-2-3-3 beeps

Test Points and Beep Codes

At the beginning of each POST routine, the BIOS outputs the test point error code to I/O address 80h. Use this code during trouble shooting to establish at what point the system failed and what routine was being performed.

Some motherboards are equipped with a seven-segment LED display that displays the current value of port 80h. For production boards that do not contain the LED display, you can purchase a card that performs the same function.

If the BIOS detects a terminal error condition, it halts POST after issuing a terminal error beep code (See above) and attempting to display the error code on upper left corner of the screen and on the port 80h LED display. It attempts repeatedly to write the error to the screen. This may cause "hash" on some CGA displays.

If the system hangs before the BIOS can process the error, the value displayed at the port 80h is the last test performed. In this case, the screen does not display the error code.

The following is a list of the checkpoint codes written at the start of each test and the beep codes issued for terminal errors. Unless otherwise noted, these codes are valid for PhoenixBIOS 4.0 Release 6.x.

Code	Beeps	POST Routine Description
02h		Verify Real Mode
03h		Disable Non-Maskable Interrupt (NMI)
04h		Get CPU type
06h		Initialize system hardware
07h		Disable shadow and execute code from the ROM.
08h		Initialize chipset with initial POST values
09h		Set IN POST flag
0Ah		Initialize CPU registers
0Bh		Enable CPU cache
0Ch		Initialize caches to initial POST values
0Eh		Initialize I/O component
0Fh		Initialize the local bus IDE
10h		Initialize Power Management
11h		Load alternate registers with initial POST values
12h		Restore CPU control word during warm boot
13h		Initialize PCI Bus Mastering devices
14h		Initialize keyboard controller
16h	1-2-2-3	BIOS ROM checksum
17h		Initialize cache before memory Auto size
18h		8254 timer initialization
1Ah		8237 DMA controller initialization
1Ch		Reset Programmable Interrupt Controller
20h	1-3-1-1	Test DRAM refresh
22h	1-3-1-3	Test 8742 Keyboard Controller
24h		Set ES segment register to 4 GB
28h		Auto size DRAM
29h		Initialize POST Memory Manager
2Ah		Clear 512 kB base RAM
2Ch	1-3-4-1	RAM failure on address line *xxxx**
2Eh	1-3-4-3	RAM failure on data bits *xxxx** of low byte of memory bus
2Fh		Enable cache before system BIOS shadow
32h		Test CPU bus-clock frequency
33h		Initialize Phoenix Dispatch Manager
36h		Warm start shut down
38h		Shadow system BIOS ROM
3Ah		Auto size cache
3Ch		Advanced configuration of chipset registers
3Dh		Load alternate registers with CMOS values
41h		Initialize extended memory for RomPilot
42h		Initialize interrupt vectors
45h		POST device initialization

Code	Beeps	POST Routine Description
46h	2-1-2-3	Check ROM copyright notice
47h		Initialize I20 support
48h		Check video configuration against CMOS
49h		Initialize PCI bus and devices
4Ah		Initialize all video adapters in system
4Bh		QuietBoot start (optional)
4Ch		Shadow video BIOS ROM
4Eh		Display BIOS copyright notice
4Fh		Initialize MultiBoot
50h		Display CPU type and speed
51h		Initialize EISA board
52h		Test keyboard
54h		Set key click if enabled
55h		Enable USB devices
58h	2-2-3-1	Test for unexpected interrupts
59h		Initialize POST display service
5Ah		Display prompt "Press F2 to enter SETUP"
5Bh		Disable CPU cache
5Ch		Test RAM between 512 and 640 kB
60h		Test extended memory
62h		Test extended memory address lines
64h		Jump to UserPatch1
66h		Configure advanced cache registers
67h		Initialize Multi Processor APIC
68h		Enable external and CPU caches
69h		Setup System Management Mode (SMM) area
6Ah		Display external L2 cache size
6Bh		Load custom defaults (optional)
6Ch		Display shadow-area message
6Eh		Display possible high address for UMB recovery
70h		Display error messages
72h		Check for configuration errors
76h		Check for keyboard errors
7Ch		Set up hardware interrupt vectors
7Dh		Initialize Intelligent System Monitoring
7Eh		Initialize coprocessor if present
80h		Disable onboard Super I/O ports and IRQs
81h		Late POST device initialization
82h		Detect and install external RS232 ports
83h		Configure non-MCD IDE controllers
84h		Detect and install external parallel ports
85h		Initialize PC-compatible PnP ISA devices
86h		Re-initialize onboard I/O ports.
87h		Configure Motherboard Configurable Devices (optional)
88h		Initialize BIOS Data Area
89h		Enable Non-Maskable Interrupts (NMIs)
8Ah		Initialize Extended BIOS Data Area
8Bh		Test and initialize PS/2 mouse
8Ch		Initialize floppy controller
8Fh		Determine number of ATA drives (optional)
90h		Initialize hard-disk controllers
91h		Initialize local-bus hard-disk controllers
92h		Jump to UserPatch2
93h		Build MPTABLE for multi-processor boards
95h		Install CD ROM for boot
96h		Clear huge ES segment register

Code	Beeps	POST Routine Description
97h		Fix up Multi Processor table
98h	1-2	Search for option ROMs. One long, two short beeps on checksum failure
99h		Check for SMART Drive (optional)
9Ah		Shadow option ROMs
9Ch		Set up Power Management
9Dh		Initialize security engine (optional)
9Eh		Enable hardware interrupts
9Fh		Determine number of ATA and SCSI drives
A0h		Set time of day
A2h		Check key lock
A4h		Initialize typematic rate
A8h		Erase F2 prompt
AAh		Scan for F2 key stroke
ACh		Enter SETUP
AEh		Clear Boot flag
B0h		Check for errors
B1h		Inform RomPilot about the end of POST.
B2h		POST done - prepare to boot operating system
B4h	1	One short beep before boot
B5h		Terminate QuietBoot (optional)
B6h		Check password (optional)
B7h		Initialize ACPI BIOS
B9h		Prepare Boot
BAh		Initialize SMBIOS
BBh		Initialize PnP Option ROMs
BCh		Clear parity checkers
BDh		Display MultiBoot menu
BEh		Clear screen (optional)
BFh		Check virus and backup reminders
C0h		Try to boot with INT 19
C1h		Initialize POST Error Manager (PEM)
C2h		Initialize error logging
C3h		Initialize error display function
C4h		Initialize system error handler
C5h		PnPnd dual CMOS (optional)
C6h		Initialize note dock (optional)
C7h		Initialize note dock late
C8h		Force check (optional)
C9h		Extended checksum (optional)
CAh		Redirect Int 15h to enable remote keyboard
CBh		Redirect Int 13h to Memory Technologies Devices such as ROM, RAM, PCMCIA, and serial disk
CCh		Redirect Int 10h to enable remote serial video
CDh		Re-map I/O and memory for PCMCIA
CEh		Initialize digitizer and display message
D2h		Unknown interrupt
		The following are for boot block in Flash ROM
E0h		Initialize the chipset
E1h		Initialize the bridge
E2h		Initialize the CPU
E3h		Initialize system timer
E4h		Initialize system I/O
E5h		Check force recovery boot
E6h		Checksum BIOS ROM
E7h		Go to BIOS

Code	Beeps	POST Routine Description
E8h		Set Huge Segment
E9h		Initialize Multi Processor
EAh		Initialize OEM special code
EBh		Initialize PIC and DMA
ECh		Initialize Memory type
EDh		Initialize Memory size
EEh		Shadow Boot Block
EFh		System memory test
F0h		Initialize interrupt vectors
F1h		Initialize Run Time Clock
F2h		Initialize video
F3h		Initialize System Management Manager
F4h		Output one beep
F5h		Clear Huge Segment
F6h		Boot to Mini DOS
F7h		Boot to Full DOS

* If the BIOS detects error 2C, 2E, or 30 (base 512K RAM error), it displays an additional word-bitmap (*xxxx*) indicating the address line or bits that failed. For example, "2C 0002" means address line 1 (bit one set) has failed. "2E 1020" means data bits 12 and 5 (bits 12 and 5 set) have failed in the lower 16 bits. Note that error 30 cannot occur on 386SX systems because they have a 16 rather than 32-bit bus. The BIOS also sends the bitmap to the port-80 LED display. It first displays the checkpoint code, followed by a delay, the high-order byte, another delay, and then the low-order byte of the error. It repeats this sequence continuously.

PhoenixBIOS 4.0 Services

The ROM BIOS contains a number of useful run-time **BIOS Services** that are easily called by an outside program. As a programmer, you can execute these services, which are nothing more than subroutines, by invoking one of the BIOS interrupt routines (or, when specified, calling a protected-mode entry point and offset). Invoking a software interrupt causes the CPU to fetch an address from the **interrupt table** in low memory and execute the service routine. Some services return exit values in certain registers. All registers are preserved unless they return data or status.

Generally, a Carry flag set on exit indicates a failed service. A zero on exit in the AH register usually indicates no error; any other value is the service's **exit status code.**

BIOS32 Service Directory

While the standard BIOS services are accessed through the interrupt table, newer services are accessed by a FAR CALL to a service entry point. Programmers can determine the entry point by searching for a particular signature (such as "$PnP") in the BIOS range and finding the entry point in the header.

The **BIOS32 Service Directory** (standard in PhoenixBIOS 4.0) provides a single entry point for all those services in the BIOS that are designed for BIOS clients running in a 32-bit code segment, such as 32-bit operating systems and 32-bit device drivers. The BIOS32 Service Directory itself is a 32-bit BIOS service that provides a single entry point for the other 32-bit services. For a full description of this service, see the **Standard BIOS 32-Bit Service Directory Proposal, Rev 0.4** published by Phoenix and available on the Phoenix Web site at:

```
http://www.phoenix.com/products/specs.html
```

Programs calling the 32-bit BIOS services should scan 0E0000h to 0FFFF0h on the 16-byte boundaries for the contiguous 16-byte data structure beginning with the ASCII signature "_32_".

If they do not find this data structure, then the platform does not support the BIOS32 Service Directory. The following chart describes the data structure.

Offset	Size	Description
0h	4 bytes	ASCII signature "_32_" Offset 0 = underscore Offset 1 = "3" Offset 2 = "2" Offset 3 = underscore
4h	4 bytes	Entry point for the BIOS32 Service Directory, a 32-bit physical address
8h	1 byte	Revision level. Currently 00h.
9h	1 byte	Length of this structure in 16-byte units. This structure is 16 bytes long, so the field = 01h.
0Ah	1 byte	Checksum of whole data structure. Result must be 0.
0Bh	5 bytes	Reserved. Must be zero.

Once the data structure is found and verified, the program can do a FAR CALL to the entry point specified in the above structure. The calling environment requires:

1. The CS code segment selector and the DS data segment selector must encompass the physical page of the entry point as well as the following page.
2. The SS stack segment selector must have available 1 kB of stack space.
3. Access to I/O space.

The BIOS32 Service Directory provides a single call that:

1. Determines if the called 32-bit service is available, and, if it is available,
2. Returns three values:
   a) Physical address of the base of the BIOS service.
   b) Length of the BIOS service.
   c) Entry point into the BIOS service (offset of the base).

BIOS32 Service Directory

Entry:

EAX	Service Identifier. Four-character string identifying the 32-bit service requested (e.g., "$PCI").
EBX	Low-order byte [BL] is the BIOS32 Service Directory Function Selector. Currently, zero supplies the values described below. Upper three bytes are reserved and must be zero on entry.

Exit:

AL	Return code: 00h = Service corresponding to the Service Identifier is present. 80h = Service corresponding to the Service Identifier is not present. 81h = Function Selector specified not supported.
EBX	Physical address of base of 32-bit service.
ECX	Length of BIOS service.
EDX	Entry point of BIOS service (offset to base in EBX).

Interrupt 10h–Video Services

The INT 10h software interrupt handles all video services. The results of some of these functions may depend on the active video mode and the particular video controller installed.

Interrupt 10 Video Services

AH = 00h Set video mode
Entry:
AL Mode value (0-7):
0 = 40x25 Black & White
1 = 40x25 Color
2 = 80x25 Black & White
3 = 80x25 Color
4 = 320x200 Color
5 = 320x200 Black & White
6 = 640x200 Black & White
7 = Monochrome only

AH = 01h Set cursor size
Entry:
CH Bits 4-0 = Cursor top scan line
CL Bits 4-0 = Cursor bottom scan line

AH = 02h Set cursor position
Entry:
BH Page to set cursor
DL Character column position
DH Character row position

AH = 03h Get cursor position of page
Entry:
BH Page to return cursor
Exit:
DL Character column position
DH Character row position
CL Cursor top scan line
CH Cursor bottom scan line

AH = 05h Change displayed (active) page
Entry:
AL Page number to display

AH = 06h Scroll active page up
Entry:
CL Upper left column to scroll up
CH Upper left row to scroll up
DL Lower right column to scroll up
DH Lower right row to scroll up
BH Attribute for blanked space
AL Number of lines to scroll up
0 = Blank screen

AH = 07h Scroll active page down
Entry:
CL Upper left column to scroll down
CH Upper left row to scroll down
DL Lower right column to scroll down
DH Lower right row to scroll down
BH Attribute for blanked space
AL Number of lines to scroll down
0 = Blank screen

AH = 08h Read character and attribute
Entry:
BH Video page to read character
Exit:
AL Character
AH Character attribute

Continued

Interrupt 10h–Video Services, Continued

AH = 09h Write character and attribute
Entry:
AL Character to write
BL Character attribute (alpha)
Character color (graphics)
BH Page to write character
CX Count of characters to write

AH = 0Ah Write character at cursor
Entry:
BH Page to write character
AL Character to write
CX Count of characters to write

AH = 0Bh Set color palette
Entry:

BH = 00 Set colors:
If mode = 4 or 5, BL = background color
If mode = 0-3, BL = border color
If mode = 6 or 11, BL = foreground color
BL 0-31 = Intense versions of colors 0-15

BH = 01 Set palette for mode 4 or 5
BL 00 Palette = Green (1), Red (2), Yellow (3)
01 Palette = Cyan (1), Magenta (2), White (3)

AH = 0Ch Write graphics pixel
Entry:
AL Color value for pixel
(XORed if bit7=1)
CX Column to write pixel
DX Row to write pixel

AH = 0D Read graphics pixel
Entry:
CX Column to read pixel
DX Row to read pixel
Exit:
AL Value of pixel read

AH = 0E Teletype write character
Entry:
AL Character to write
BL Foreground color (graphics only)

AH = 0F Return Current Video Parameters
Exit:
AL Current video mode
AH Number of character columns
BH Active page

AH = 13h Write string
Entry:
ES:BP Pointer to string
CX Length of string to display
DH Character row for display
DL Character column for display
BL Display attribute
AL *Write string mode*
0 = Chars only, no cursor update
1 = Chars only, update cursor
2 = Char, Attrib, no cursor update
3 = Char, Attrib, update cursor

Interrupt 11h–Return System Information

This service returns the equipment installed as determined by the BIOS on power-up diagnostics and stored in the BIOS Data Area.

Interrupt 11 Return System Information

Exit:	
AX	Equipment information:
Bit	**Definition**
0	Not used
1	Math coprocessor installed
2	PS/2 mouse installed
3	Not used
4,5	Initial video mode: 00 = EGA/VGA 01 = 40x25 CGA 10 = 80x25 CGA 11 = Monochrome
6,7	Diskette drives: 00 = 1 drive 01 = 2 drives 10 = 3 drives 11 = 4 drives
8	Not used
9-11	Number of serial adapters
12	Game Adapter installed
13	Not used
14,15	Number of parallel adapters

Interrupt 12h–Return Memory Size

Returns up to 640 kB of the amount of system memory determined by early POST diagnostics.

Interrupt 12 Return System Memory Size

Exit:	
AX	Number of 1-kilobyte memory blocks

Interrupt 13h–Diskette Services

Interrupt 13 is the BIOS software interface for access to the 5-¼" and 3-½" inch diskette drives. When there is a fixed disk in the system, the BIOS assigns Interrupt 13h to the fixed disk and routes diskette calls to Interrupt 40h.

The following table lists the AH error codes.

Int 13 Diskette Exit Status Codes

AH	00h = No error If Carry = 1:
AH	01h = Illegal BIOS command 02h = Bad address mark 03h = Write-protect occurred 04h = Sector not found 06h = Media changed 09h = DMA crossed 64K boundary 08h = DMA failed 0Ch = Media not found 10h = CRC failed 20h = NEC failed 30h = Drive does not support media sense 31h = No media in drive 32h = Drive does not support media type 40h = Seek failed 80h = Time out occurred

The following table contains the combinations of drive types and media types supported by the INT 13 services 02h to 05h.

Diskette Types

Media	Drive	Sec/Trk	Tracks
360 kB	360 kB	8-9	40
360 kB	1.2 MB	8-9	40
1.2 MB	1.2 MB	15	80
720 kB	720 kB	9	80
720 kB	1.44 MB	9	80
1.44 MB	1.44 MB	18	80
720 kB	2.88 MB	9	80
1.44 MB	2.88 MB	18	90
2.88 MB	2.88 MB	36	80

The following describes the diskette services with their entry and exit values.

Interrupt 13h Diskette Services

AH = 00h Reset diskette system

AH = 01h Return diskette status

Exit:

AH 00h = No error
01h = Illegal BIOS command
02h = Address mark not found
03h = Write-protect error
04h = Sector not found
06h = Media has been changed
08h = DMA overrun
09h = DMA boundary error
0Ch = Media not found
10h = CRC error
20h = NEC error
40h = Seek error
80h = Time out occurred

AH = 02h Read diskette sectors

Entry:

ES:BX Buffer address
DL Drive number (0-1)
DH Head number (0-1)
CH Track number (0-79)
CL Sector number (8-36)
AL Number of sectors (1-15)

Exit:

AL Number of sectors transferred

AH = 03h Write diskette sectors

Entry:

ES:BX Buffer address
DL Drive number (0-1)
DH Head number (0-1)
CH Track number (0-79)
CL Sector number (8-36)
AL Number of sectors (1-15)

Exit:

AL Number of sectors transferred

AH = 04h Verify diskette sectors

Entry:

DL Drive number (0-1)
DH Head number (0-1)
CH Track number (0-79)
CL Sector number (8-36)
AL Number of sectors (1-15)

Exit:

AL Number of sectors verified

Continued

Interrupt 13h–Diskette Services, Continued

AH = 05h Format diskette track

Entry:
ES:BX Buffer address
DL Drive number (0-1)
DH Head number (0-1)
CH Track number (0-79)
CL Sector number (8-36)
AL Number of sectors (1-15)
Exit:
AL Number of sectors formatted

AH = 08h Read drive parameters

Entry:
DL Drive number
Exit:
ES:DI Pointer to parameter table
DH Maximum head number
DL Number of diskette drives present
CH Maximum track number
CL Drive capacity:
Bits 0-5 Maximum sector number
Bits 6-7 Maximum track number
BL Diskette drive type from CMOS:
Bits 0-3:
00 = CMOS not present or invalid
01 = 360 kB
02 = 1.2 MB
03 = 720 kB
04 = 1.44 MB
06 = 2.88 MB
Bits 4-7: 0

AH = 15h Read drive type

Entry:
DL Drive number
Exit:
AH 00 = Drive not present
01 = Drive cannot detect media change
02 = Drive can detect media change
03 = Fixed disk

AH = 16h Detect media change

Entry:
DL Drive Number (0-1)
Exit:
If Carry = 0:
AH 00 = Disk change not active
01 = Invalid drive number
06 = Either disk change line active or
change line not supported
80h = Drive not ready or no drive present:
(timeout)

AH = 17h Set diskette type

Entry:
AL Format:
00 = Invalid Request
01 = 360kB floppy in 360kB drive
02 = 360kB floppy in 1.2MB drive
03 = 1.2MB floppy in 1.2MB drive
04 = 720kB floppy in 720kB (1.44MB not supported)
DL Drive Number (0-1)

AH = 18h Set media type for format

Entry:
CH Maximum track number
CL Diskette parameters:
Bits 0-5: Maximum sector number
Bits 6-7: Maximum track number
DL Drive Number (0-1)
Exit:
ES:DI Pointer to parameter table

Continued

Interrupt 13h–Diskette Services, Continued

AH = 20h Get media type

Entry:
DL Drive number (0-1)
Exit:
AL Type of media installed:
00h = 720 kB diskette
01h = 1.44 MB diskette
02h = 2.88 MB diskette
03h = 1 MB diskette
04h = 2 MB diskette
06h = 4 MB diskette

Interrupt 13h–Fixed Disk Services

Interrupt 13h accesses these Services:
Standard Fixed-Disk Services, 00h-15h
Enhanced Disk Drive Services, 41h -48h
Bootable CD-ROM Services, 4Ah-4Dh

The following box lists the error codes:

Int 13h Fixed-Disk Exit Codes

AH 00h = No error
If Carry = 1:
AH 01 = Bad command or parameter
02h = Address mark not found
04h = Sector not found
05h = Reset failed
07h = Drive parameter activity failed
0Ah = Bad sector flag detected
10h = ECC data error
11h = ECC data corrected
20h = Controller failure
40h = Seek failed
80h = Time out occurred
AAh = Drive not ready
BBh = Undocumented controller error
CCh = Controller write fault
E0h = Unrecognized controller error

The following describes the Standard Fixed-Disk services of PhoenixBIOS 4.0:

Interrupt 13 Standard Fixed Disk Services

AH = 00 Reset diskette and fixed-disk systems

AH = 01h Read disk status

Entry:
DL Drive number (80h-81h)
Exit:
AH 001h = Bad command
002h = Bad address mark
004h = Record not found
005h = Controller reset error
007h = Drive initialization error
00Ah = Bad sector
010h = ECC data error
020h = Controller failed
040h = Seek error
0AAh = Drive not ready
0BBh = Invalid controller error
0CCh = Controller write fault
0E0h = Unrecognized controller error

Continued

Interrupt 13h–Fixed Disk Services, Continued

AH = 02h Read disk sectors

Entry:
ES:BX Buffer address
DL Drive number (80h-81h)
DH Head number (0-15)
CH Track number (0-1023)
Put the two high-order bits (8 and 9)
in the high-order bits of CL
CL Sector number (1-17)
AL Number of sectors (1-80h for read)
(1-79h for long read, includes ECC)
Exit:
AL Number of sectors transferred

AH = 03h Write disk sectors

Entry:
ES:BX Buffer address
DL Drive number (80H-81H)
DH Head number (0-15)
CH Track number (0-1023)
Put the two high-order bits (8 and 9)
in the high-order bits of CL
CL Sector number (1-17)
AL Number of sectors (1-80h for write)
(1-79h for long write, includes ECC)
Exit:
AL Number of sectors transferred

AH = 04h Verify disk sectors

Entry:
ES:BX Buffer address
DL Drive number (80h-81h)
DH Head number (0-15)
CH Track number (0-1023)
Put the two high-order bits (8 and 9)
in the high-order bits of CL
CL Sector number (1-17)
AL Number of sectors (1-80h for write)
(1-79h for long write, includes ECC)
Exit:
AL Number of sectors verified

AH = 05h Format disk cylinder

Entry:
ES:BX Pointer to table containing the
following byte pair for each sector
in the track:
Byte 0: 00h if sector is good
80h if sector is bad
Byte 1: Sector Number (0-255)
DL Drive number (80H-81H)
DH Head number (0-15)
CH Track number (0-1023)
Put the two high-order bits (8 and 9)
in the high-order bits of CL
CL Sector number (1-17)
AL Number of sectors (1-80h for write)
(1-79h for long write, includes ECC)
Exit:
AL Number of sectors formatted

Continued

Interrupt 13h–Fixed Disk Services, Continued

AH = 08h Read drive parameters
Entry:
DL Drive number (80H-81H)
Exit:
CL Maximum sector number
CH Maximum cylinder number
(High bits in CL)
DH Maximum head number
DL Number of responding drives (0-2)
If Carry = 1:
AH 07h = Invalid drive number
AL 0 = Error
CX 0 = Error
DX 0 = Error

AH = 09h Initialize drive parameters
Entry:
DL Drive number (80H-81H)

AH = 0Ah Read long sector
Entry:
ES:BX Buffer address
DL Drive number (80H-81H)
DH Head number
CH Cylinder number
CL Sector number/Cyl high
AL Number of sectors
Exit:
AL Number of sectors transferred

AH = 0Bh Write long sector
Entry:
ES:BX Buffer address
DL Drive number (80H-81H)
DH Head number
CH Cylinder number
CL Sector number/Cyl high
AL Number of sectors
Exit:
AL Number of sectors transferred

AH = 0Ch Seek drive
Entry:
ES:BX Buffer address
DL Drive number (80H-81H)
DH Head number
CH Cylinder number
CL Cylinder high

AH = 0Dh Alternate disk reset
Entry:
DL Drive number (80H-81H)

AH = 10h Test drive ready
Entry:
DL Drive number (80H-81H)

AH = 11h Recalibrate drive
Entry:
DL Drive number (80H-81H)

AH = 14h Controller diagnostic
Entry:
DL Drive number (80H-81H)

AH = 15h Read drive type
Entry:
DL Drive number (80H-81H)
Exit:
AH 00 = Drive not present
01 = Drive cannot detect media change
02 = Drive can detect media change
03 = Fixed disk
CX High word of number of 512-byte blocks
DX Low word of number of 512-byte blocks

Interrupt 13h–Extended Fixed Disk Services

The following describes the Interrupt 13h Extended Fixed Disk Services, including the *Phoenix*BIOS **Enhanced Disk Drive (EDD)** services:

Int 13h Extended Fixed Disk Services

AH = 41h Check Extensions Present

Entry:
BX 55AAh
DL Drive Number
Exit:
AH Major version number (20h)
AL Internal use only
BX 55AAh = Extensions present
CX Feature support map:
Bit 0: 1 = Extended disk access
Bit 1: 1 = Removable drive control
Bit 2: 1 = Enhanced Disk Drive Extensions
Bits 3-7, Reserved, must be 0

AH = 42h Extended Read

Entry:
DL Drive Number
DS:SI Disk address packet

AH = 43h Extended Write

Entry:
AL Verify Bits:
Bit 0: 0 = Write with verify off
1 = Write with verify on
Bits 1-7 Reserved, set to 0
DL Drive number
DS:SI Disk address packet

AH = 44h Verify Sectors

Entry:
DL Drive number
DS:SI Disk address packet

AH = 47h Extended Seek

Entry:
DL Drive number
DS:SI Disk address packet

AH = 48h Get Drive Parameters

Entry:
DL Drive Number
DS:SI Address of Result Buffer
Exit:
DS:SI Pointer to Result Buffer:

```
info_size  dw 30     ;size of this buffer
flags  dw ?          ;info flags (See below)
cylinders  dd ?      ;cylinders on disk
heads  dd ?          ;heads on disk
sec_per_track  dd ?  ;sectors per track
sectors  dq ?        ;sectors on disk
sector_size  dw ?    ;bytes per sector
extended_table dd?   ;extended table ptr
                     ; (See below)
```

info flags:

Bit 0 0 = DMA boundary errors possible
1 = DMA errors handled
Bit 1 0 = CHS info not supplied
1 = CHS info valid
Bit 2 0 = Drive not removable
1 = Drive removable
Bit 3 0 = No write with verify
1 = Write with verify
Bit 4 0 = No change-line support
1 = Change-line support
Bit 5 0 = Drive not lockable
1 = Drive lockable
Bit 6 0 = CHS values for installed media
1 = Maximum CHS values for drive (media absent)

Continued

Interrupt 13h–Extended Fixed Disk Services, Continued

Extended Fixed Disk Parameter Table

Byte	Type	Description
0-1	Word	I/O port address
2-3	Word	Control port address
4		Bit 0-3 Reserved, must be 0
		Bit 4 0 = Master, 1 = Slave
		Bit 5 Reserved, must be 0
		Bit 6 1 = LBA enabled
		Bit 7 Reserved, must be 1
5		Bits0-3 Phoenix Proprietary
		Bits 4-7 Reserved, must be 0
6		Bits 0-3 IRQ for this drive
		Bits 4-7 Reserved, must be 0
7	Byte	Sector count for multi-sectored transfers
8		Bits 0-3 DMA channel
		Bits 4-7 DMA type
9		Bits 0-3 PIO type
		Bits 1-7 Reserved, must be 0

Byte	Type	Description
10-11		Bit 0 1 = Fast PIO access enabled
		Bit 1 1 = DMA access enabled
		Bit 2 1 = Block PIO access enabled
		Bit 3 1 = CHS translation enabled
		Bit 4 1 = LBA translation enabled
		Bit 5 1 = Removable media
		Bit 6 1 = CD ROM
		Bit 7 1 = 32-bit transfer mode
		Bit 8 1 = ATAPI Device uses Interrupt DRQ
		Bits 9-10 CHS Translation Type
		Bits 11-15 Reserved, must be 0
12-13	Byte	Reserved, must be 0
14	Byte	Extension Revision number
15	Byte	Checksum, 2s complement of the sum of bytes 0-14

Interrupt 13h–Bootable CD-ROM Services

Bootable CD-ROM Services 4Ah-4Ch use a pointer to the **Specification Packet**, described here:

Bootable CD-ROM Specification Packet

Offset	Type	Description
0h	Byte	Packet size, currently 13h
1h	Byte	Boot media type: Bits 0-3: 00h = No emulation 01h = 1.2 MB diskette 02h = 1.44 MB diskette 03h = 2.88 MB diskette 04h = Hard disk (drive C:) Bits 05h-07h: Reserved Bit 6: 01h = System has ATAPI driver with 8 & 9 below describing IDE interface. Bit 7: 01h = System has SCSI drivers with 8 & 9 below describing SCSI interface
2h	Byte	Drive number: 00h = Floppy image 80 = Bootable hard disk 81h -FFh = "Non-bootable" or "No emulation"

Continued

Interrupt 13h–Bootable CD-ROM Services, Continued

Offset	Type	Description
3h	Byte	Controller index of CD drive
4h-7h	Dword	Logical Block Address
8h-9h	Word	Device specification: For SCSI: Byte 8: LUN and PUN of CD drive Byte 9: Bus number For IDE: Byte 8 LSB: 0 = Master, 1 = Slave
Ah-Bh	Word	User buffer segment
Ch-Dh	Word	Load segment (only for Int 13h 4Ch): 00h = 7C0h
Eh-Fh	Word	Virtual sector count (only for Int 13h 4Ch)
10h	Byte	Low-order bits (0-7) of the cylinder count (Matches returned CH of Int 13h 08h)
11h	Byte	Bits 0-5: Sector count Bits 6-7: High order 2 bits of cylinder count (Matches returned CL of Int 13h 08h)
12h	Byte	Head count (Matches returned DH of Int 13h 0h)

Bootable CD-ROM Service 4Dh uses a pointer to the **Command Packet**, described here:

Bootable CD-ROM Command Packet

Offset	Type	Description
0h	Byte	Packet size in bytes, currently 08h
1h	Byte	Count of sectors in boot catalog to transfer
2-h	Dword	Pointer to destination buffer for boot catalog
6-7h	Word	Beginning sector to transfer, relative to start of the boot catalog. Int 14 4Dh should set this value to 00h.

The following describes the Interrupt 13 Bootable CD-ROM Services of PhoenixBIOS 4.0:

Int 13 Bootable CD-ROM Services

AH = 4Ah Initiate disk emulation

Entry:
AL 00
DS:SI Pointer to Specification Packet (See above)
CF 0 = Specified drive emulating
1 = System not in emulation mode

AH = 4Bh Terminate disk emulation

Entry:
AL 00h = Return status and terminate emulation
01h = Return status only, do not terminate
DL Drive number to terminate
7Fh = Terminate all
DS:SI Empty Specification Packet
Exit:
DS:SI Completed Specification Packet (See above)
AX Exit status codes
CF 0 = System released
1 = System not in emulation mode

AH = 4Ch Initiate disk emulation and boot

Entry:
AL 00h
DS:SI Specification Packet (See above)

AH = 4Dh Return boot catalog

Entry :
AL 00h
DS:SI Point to Command Packet (See above)

Interrupt 14h–Serial Services

The INT 14 software interrupt handles serial I/O service requests. Use the AH register to specify the service to invoke. This describes the UART Modem and Line Status returned by these services. It also includes two services, 04h and 05h, that support the extended communication capabilities of PS/2.

The following describes the modem status returned by serial services.

Modem Status

AL	Description
Bit 0	1 = Delta clear to send
Bit 1	1 = Delta data set ready
Bit 2	1 = Trailing edge ring indicator
Bit 3	1 = Delta data carrier detect
Bit 4	1 = Clear to send
Bit 5	1 = Data set ready
Bit 6	1 = Ring indicator
Bit 7	1 = Received line signal detect

The following describes the line status returned by Int 14h Serial Services.

Line Status

AH	Description
Bit 0	1 = Data ready
Bit 1	1 = Overrun error
Bit 2	1 = Parity error
Bit 3	1 = Framing error
Bit 4	1 = Break detect
Bit 5	1 = Trans holding register empty
Bit 6	1 = Trans shift register empty
Bit 7	1 = Time out error

The following describes the serial communication services of *Phoenix*BIOS 4.0:

Interrupt 14h Serial Services

AH = 00 Initialize Serial Adapter

Entry:

AL Init parameters:

Bit 1,0 10 = 7 data bits
11 = 8 data bits

Bit 2 0 = 1 stop bit
1 = 2 stop bits

Bit 4,3 00 = No parity
10 = No parity
01 = Odd parity
11 = Even parity

Bit 7-5 000 = 110 Baud- 417 divisor
001 = 150 Baud-300 divisor
010 = 300 Baud-180 divisor
011 = 600 Baud-0C0 divisor
100 = 1200 Baud-060 divisor
101 = 2400 Baud-030 divisor
110 = 4800 Baud-018 divisor
111 = 9600 Baud-00C divisor

DX Serial port (0-3)

Exit:

AL Modem status

AH Line status

AH = 01h Send character

Entry:

AL Character to transmit

DX Serial port (0-3)

Exit:

AH Line status

AH = 02h Receive character

Entry:

DX Serial port (0-3)

Exit:

AL Character received

AH Line Status

AH = 03h Return serial port status

Entry:

DX Serial port (0-3)

Exit:

AH Line status

AL Modem status

Continued

Interrupt 14h–Serial Services, Continued

AH = 04h Extended Initialize (PS/2)

Entry:
DX 0-3 = Communications adapter
AL 00 = Break
01 = No break
BH Parity:
00 = None
01 = Odd
02 = Even
03 = Stick parity odd
04 = Stick parity even
BL Stop bits:
00 = One
01 = Two if 6,7, or 8-bit word length
One and one-half if 5-bit word length
CH Word length:
00 = 5 bits
01 = 6 bits
02 = 7 bits
03 = 8 bits
CL Baud rate:
00 = 110 baud
01 = 150 baud
02 = 300 baud
03 = 600 baud
04 = 1200 baud
05 = 2400 baud
06 = 6000 baud
07 = 9600 baud
08 = 19200 baud
Exit:
AL Modem status
AH Line status

AH = 05h Extended Communications Port Control (PS/2)

AL = 00 Read modem control register

Entry:
DX Serial port (0-3)
Exit:
BL Modem control register

AL = 01 Write modem control register

Entry:
DX Serial port (0-3)
BL Modem control register
Exit:
AL Modem status
AH Line status

Interrupt 15h–System Services

The INT 15 software interrupt handles a variety of system services:

Multi-tasking–80h, 81h, 82h, 85h, 90h, and 91h
Joystick support–84h
Wait routines–83h and 86h
Protected-mode support–87h and 89h
Report extended memory to 64 kB–88h
System information–C0h
Advanced Power Management (optional)–53h
Report extended memory above 64 kB (optional)–8Ah and E8h
PS/2 Mouse support (optional)–C2h
EISA Support (optional)–D8h

The first section describes the standard Interrupt 15 services, followed by separate sections describing each of the optional services.

Interrupt 15h System Services

AH = 00-03h Cassette services
Entry:
No longer supported
Exit:
Carry 1 = Not supported

AH = 80h Device open
Entry:
BX Device identifier
CX Process identifier

AH = 81h Device close
Entry:
BX Device identifier
CX Process identifier

AH = 82h Program termination
Entry:
BX Device identifier

AH = 83h Event wait

AL 00 = Set interval
Entry:
ES:BX Pointer to byte in caller's memory that will have bit 7 set when interval expires.
CX Microseconds before post (high byte)
DX Microseconds before post (low byte)
Exit:
AH 83h
AL A value written to CMOS register B
00h = Function busy

AL 01 = Cancel set interval
Exit:
AH 83
AL 00

AH = 84h Joystick support
Entry:
DL 00 = Read switch settings
Exit:
AL Switch settings
DL 01 Return resistive inputs
Exit:
AX Input bit 0 (Joystick A, x coordinate)
BX Input bit 1 (Joystick A, y coordinate)
CX Input bit 2 (Joystick B, x coordinate)
DX Input bit 3 (Joystick B, y coordinate)

AH = 85h System request key pressed
Entry:
AL 00 System request key pressed
AL 01 System request key released

AH = 86h Wait
Entry:
CX Number of microseconds to wait (high byte)
DX Number of microseconds to wait (low byte)

Continued

Interrupt 15h–System Services, Continued

AH = 87h Extended memory move block
Entry:
CX Number of words to move
ES:SI Pointer to Global Descriptor
Byte 0-1 Bits 0-15 of Segment Limit
Byte 2-3 Bits 0-15 of Base Address
Byte 4 Bits 16-23 of Base Address
Byte 5 Access Rights
Byte 6 Bits 7-4 more Access Rights
Bits 3-0 upper 4 bits of Segment Limit
Byte 7 Bits 24-31 of Base Address
(See Intel programmer's reference)

AH = 88h Extended memory size
Exit:
AX Amount of Extended memory less 1 kB up to 64 MB, in 1 kB blocks (FFFCh implies 64 MB or greater. Use INT 15 Big Memory Services for further information).

AH = 89h Enter protected mode
Entry:
ES:SI Pointer to Global Descriptor (See service 87)
BH Offset in IDT for IRQ 00-07
BL Offset in IDT for IRQ 08-0F

AH = 90h Device busy
Entry:
AL Type code:
00h = Fixed disk (May time out)
01h = Diskette (May time out)
02h = Keyboard (No time out)
03h = Pointing device (May time out)
80h = Network (No time out)
FCh = Fixed disk reset (May time out)
FDh = Diskette drive motor start (May time out)
FEh = Printer (May time out)
ES:BX Points to request block if AL = 80h-FFh
Exit:
Carry 0 = No wait performed
(Driver must perform own wait)
1 = Wait performed (I/O complete or time out)

AH = 91h Interrupt complete
Entry:
AL Type code: See service 90h

AH = C0h Return system parameters
Exit:
ES:BX Pointer to System Configuration
Bytes 1-2 Length of table in bytes (8)
Byte 3 Model (FCh = AT
Byte 4 Sub model (01h = AT)
Byte 5 BIOS revision level (0)
Byte 6 Feature information:
Bit 0 0 = Reserved
Bit 1 0 = ISA-type I/O channel
Bit 2 0 = EDBA not allocated
Bit 3 0 = Wait for external event supported
Bit 4 1 = Keyboard intercept (INT 154F) called by INT 09h
Bit 5 1 = Real time clock present
Bit 6 1 = Second PIC present
Bit 7 0 = Fixed disk BIOS does not use DMA channel 3
Byte 7 Reserved
Byte 8 Reserved

AH = C1h Return Extended BIOS Data Area Address
Exit:
ES Extended BIOS Data Area Segment Address
If Carry = 1
AH 86 = Invalid BIOS routine call (No EBDA)

Interrupt 15h–APM Services

The INT 15 software interrupt optionally handles the calls supporting APM (Advanced Power Management).

The following are the APM exit status codes:

APM Service Exit Status Codes

AH 00h = No error
If Carry = 1:
AH 01h = Power Management disabled
02h = Real Mode interface already connected
03h = Interface not connected
05h = 16-bit protected mode interface already connected
06h = 16-bit protected mode interface not supported
07h = 32-bit protected mode interface already connected
08h = 32-bit protected mode interface not supported
09h = Unrecognized Device ID
0Ah = Parameter value out of range
0Bh = Interface not engaged
60h = Unable to enter requested state
80h = No PM events pending
86h = No APM present

The following are the Interrupt 15 APM Services of *Phoenix*BIOS 4.0:

Interrupt 15h APM Services

AH = 53h APM 1.0 and APM 1.1 BIOS Services

AL = 00h Installation Check
Entry:
BX 0000h = Power Device ID (APM BIOS)
All other values reserved
Exit:
AH APM major revision in BCD
AL APM minor revision in BCD
BH ASCII "P"
BL ASCII "M"
CX APM information:
Bit 0 1 = 16 bit Prot Mode supported
Bit 1 1 = 32 Bit Prot Mode supported
Bit 2 1 = CPU IDLE slows down CPU speed.
Requires APM CPU Busy service
Bit 3 1 = BIOS Power Management is disabled
Bit 4 1 = APM disengaged

AL = 01h Interface Connect
Entry:
BX 0000h = Power Device ID (APM BIOS)
All other values reserved

AL = 02h Protected-mode 16-bit interface connect
Entry:
BX 0000h = Power Device ID (APM BIOS)
All other values reserved
Exit:
AX APM 16-bit code segment (real mode segment base address)
BX Offset of entry point into the BIOS
CX APM 16-bit data segment (real mode segment address)
SI BIOS code segment length
DI BIOS data segment length

Continued

Interrupt 15h–APM Services, Continued

AL = 03h Protected-mode 32-bit interface connect

Entry:

BX Power Device ID, 0000h
All other values reserved

Exit:

AX APM 32-bit code segment (real mode segment base address)
EBX Offset of entry point into the BIOS
CX APM 16-bit data segment (real mode segment address)
DX APM data segment (real mode segment address)
SI BIOS code segment length
DI BIOS data segment length

AL = 04h Protected-mode 32-bit interface connect

Entry:

BX 0000h = Power Device ID (APM BIOS)
All other values reserved

AL = 05h CPU Idle

AL = 06h CPU busy

AL = 07h Set Power State

Entry:

BX Power Device ID:
0001h = All PM devices managed by the BIOS
01XXh = Display
02XXh = Secondary Storage
03XXh = Parallel Ports
04XXh = Serial Ports
05XXh = Network Adapters
06XXh = PCMCIA Sockets
E000h-EFFFh = OEM-defined power-device IDs
where:
XXh = Unit Number (0 based)
Unit Number FFh = all units in this class

CX Power State:
*0000h = APM enabled
0001h = Standby
0002h = Suspend
0003h = Off
**0004h = Last Request Processing Notification
**0005h = Last Request Rejected
0006h-001Fh = Reserved system states
0020h-003Fh = OEM-defined system states
0040h-007Fh = OEM-defined device states
0080-FFFFh = Reserved device states
* Not supported for Power Device ID 0001h
**Only supported for Power Device ID 0001h

AL = 08h Enable/disable power management

Entry:

BX Power Device ID:
0001h = All PM devices controlled by the BIOS
FFFFh = All PM devices controlled by the BIOS (For compatibility with APM 1.0)
All other values reserved

CX Function code:
0000h = Disable power management
0001h = Enable power management

AL = 09h Restore Power-On Defaults

Entry:

BX Power Device ID:
0001h = All PM devices managed by the BIOS
FFFFh = All PM devices managed by the BIOS (For compatibility with APM 1.0)
All other values reserved

Continued

Interrupt 15h–APM Services, Continued

AL = 0Ah Get Power Status

Entry:

BX Power Device ID, 0000h = APM BIOS
All other values reserved

Exit:

BH AC line status:
00h = Off line
01h = On line
02h = On backup power
FFh = Unknown
All other values reserved

BL Battery status:
00h = High
01h = Low
02h = Critical
03h = Charging
FFh = Unknown

CL Percentage of charge remaining:
0-100 = Percentage of full charge
FFh = Unknown
All other values reserved

AL = 0Bh Get PM Event

Exit:

BX PM event code

AL = 0Ch Get Power State

Entry:

BX Power Device ID:
0001h = All PM devices managed by the BIOS
01XXh = Display
02XXh = Secondary Storage
03XXh = Parallel Ports
04XXh = Serial Ports
05XXh = Network Adapters
06XXh = PCMCIA Sockets
E000h-EFFFh = OEM-defined power-device IDs
All other values reserved
where:
XXh = Unit Number (0 based)

AH = 53h APM 1.1 BIOS Services

AL = 0Dh Enable/Disable power management (APM 1.1 only)

Entry:

BX Power Device ID:
0001h = All PM devices managed by the BIOS
01XXh = Display
02XXh = Secondary Storage
03XXh = Parallel Ports
04XXh = Serial Ports
05XXh = Network Adapters
06XXh = PCMCIA Sockets
E000h-EFFFh = OEM-defined power-device IDs
All other values reserved
where:
XXh = Unit Number (0 based)

AL = 0Eh APM Driver Version (APM 1.1 only)

Entry:

BX 0000h = BIOS device
CH APM Driver major version number (BCD)
CL APM Driver minor version number (BCD)

Exit:

AH APM Connection major version number (BCD)
AL APM Connection minor version number (BCD)

Continued

Interrupt 15h–APM Services, Continued

AL = 0Fh Engage/disengage power management (APM 1.1 only)

Entry:
BX Power Device ID:
0001h = All PM devices managed by the BIOS
01XXh = Display
02XXh = Secondary Storage
03XXh = Parallel Ports
04XXh = Serial Ports
05XXh = Network Adapters
06XXh = PCMCIA Sockets
E000h-EFFFh = OEM-defined power-device IDs
All other values reserved
where:
XXh = Unit Number (0 based)
Unit Number FFh = all devices in this class
CX Function code:
0000h = Disengage power management
0100h = Engage power management

Interrupt 15h–Big Memory Services

The INT 15 software interrupt is an installable option that handles the calls reporting extended memory over 64 MB.

Interrupt 15h Big Memory Services

AH = 8Ah Big Memory size, Phoenix definition

Entry:
AX Low 16-bit value
DX High 16-bit value
= amount of memory above 64 MB in 1 kB blocks

AH = E8h Big Memory size

AL = 01h Big Memory Size, 16 Bit

Exit:
Carry 0 = E801 Supported
AX Memory 1 MB to 16 MB, in 1 kB blocks
BX Memory above 16 MB, in 64 kB blocks
CX Configured memory 1 MB to 16 MB, in 1 kB blocks
DX Configured memory above 16 MB, in 64 kB blocks

AL = 20h System Memory Map

Entry:
EBX Continuation value
ES:DI Address of Address Range Descriptor
ECX Length of Address Range Descriptor
(=> 20 bytes)
EDX "SMAP" signature
Exit:
Carry 0 = E820 Supported
EAX "SMAP" signature
ES:DI Same value as entry
ECX Length of actual reported information in bytes
EBX Continuation value

Structure of Address Range Descriptor:
Bytes 0-3 Low 32 bits of Base Address
Bytes 4-7 High 32 bits of Base Address
Bytes 8-11 Low 32 bits of Length in bytes
Bytes 12-15 High 32 bits of Length in bytes
Bytes 16-20 Type of Address Range:
1 = AddressRangeMemory, available to OS
2 = AddressRangeReserved, not available
3 = AddressRangeACPI, available to OS
4 = AddressRangeNVS, not available to OS
Other = Not defined, not available

Continued

Interrupt 15h–Big Memory Services, Continued

NOTE: Each call of this service defines a descriptor buffer and requests the memory status of the address range specified by the continuation value, where zero = first address range. The function fills the buffer and returns the continuation value for the next address range, where zero = last address range.

AL = 81h Big Memory Size, 32-Bit Protected Mode

Exit:
Carry 0 = E881 supported
EAX Memory 1 MB to 16 MB, 1 kB blocks
EBX Memory above 16 MB, 64 kB blocks
ECX Configured memory 1 MB to 16 MB, 1 kB blocks
EDX Configured memory above 16 MB, 64 kB blocks

Interrupt 15h–PS/2 Mouse Services

The INT 15 software interrupt optionally supports systems with the PS/2 mouse or similar devices installed on the motherboard. The following table describes the exit status codes:

PS/2 Mouse Exit Status Codes

AH 00h = No error
01h = Invalid function call
02h = Invalid input value
03h = Interface error
04h = Request for resend received from 8042
05h = No driver installed (i.e., Function C207 has not been called)

The following table describes the Interrupt 15h PS/2 mouse services of *Phoenix*BIOS 4.0:

Interrupt 15h PS/2 Mouse Services

AH = C2h PS/2 Mouse Support

AL 00 = Enable/Disable PS/2 Mouse
Entry:
BH 00h = Disable
01h = Enable

AL 01 = Reset PS/2 Mouse
Exit:
BH Device ID

AL 02 = Set Sample Rate
Entry:
BH Sample rate:
00h = 10 reports per second
01h = 20 reports per second
02h = 30 reports per second
03h = 40 reports per second
04h = 60 reports per second
04h = 80 reports per second
05h = 100 reports per second
06h = 200 reports per second

AL 03h = Set resolution
Entry:
BH Resolution value:
00h = 1 count per millimeter
01h = 2 counts per millimeter
02h = 4 counts per millimeter
03h = 8 counts per millimeter

AL 04h = Read Device Type
Exit:
BH Device ID

AL 05h = Initialize PS/2 mouse
Entry:
BH Data package size (01-08h, in bytes)

Continued

Interrupt 15h-PS/2 Mouse Services, continued

AL 06h = Set Scaling or Get Status

Entry:

BH 00 = Return status (See Exit Status below)
01 = Set Scaling Factor to 1:1
02 = Set Scaling Factor to 2:1

Exit:

If Entry BH = 00:

BL Status byte 1:
Bit 0 1 = Right button pressed
Bit 1 0 = Reserved
Bit 2 1 = Left button pressed
Bit 3 0 = Reserved
Bit 4 0 = 1:1 Scaling
1 = 2:1 Scaling
Bit 5 0 = Disable
1 = Enable
Bit 6 0 = Stream mode
1 = Remote mode
Bit 7 0 = Reserved

CL Status byte 2:
00h = 1 count per millimeter
01h = 2 counts per millimeter
02h = 4 counts per millimeter
03h = 8 counts per millimeter

DL Status byte 3:
0Ah = 10 reports per second
14h = 20 reports per second
28h = 40 reports per second
3Ch = 60 reports per second
50h = 80 reports per second
64h = 100 reports per second
C8h = 200 reports per second

AL 07 = Set PS/2 mouse driver address

Entry:

ES:BX Pointer to mouse driver

Interrupt 15h–EISA Services

The INT 15 software interrupt optionally supports systems with EISA (Extended Industry Standard Architecture) with these services:

Read slot configuration information–D800h, D880h

Read function configuration information–D801h, D881h

Clear EISA CMOS–D802h , D882h

Write slot configuration information to EISA CMOS–D803h, D883h

Read physical slot information–D804, D884h

The EISA BIOS services accommodate real and protected mode and 16 and 32-bit addressing. See the EISA specifications for descriptions of these services.

The following are the exit status codes for the Int 15 EISA services:

Int 15 EISA Exit Status Codes

AH 00h = No error
If Carry = 1
AH 80h = Invalid slot number
81h = Invalid function number
82h = Extended CMOS corrupted
83h = Empty slot specified
84h = Error writing to CMOS
85h = CMOS is full
86h = Invalid BIOS routine call
87h = Invalid system configuration
88h = Configuration utility not supported

The following are the Interrupt 15 EISA services of *Phoenix*BIOS 4.0:

Interrupt 15h EISA Services

AH = D8h Access EISA System Information

AL 00h = Read slot config information
80h = Read slot config information, 32 bit

Entry:
CL Slot number (0-63)
Exit:
AL Vendor information byte:
Bits 3-0 Duplicate ID number:
0000 = No duplicate ID
0001 = First duplicate ID
Bits 5-4 Slot type:
00 = Expansion slot
01 = Embedded device
10 = Virtual device
11 = Reserved
Bit 6 Product ID:
00 = Readable
01 = Not readable
Bit 7 Duplicate ID:
00 = No duplicate ID
01 = Duplicate IDs
BH Major revision level of config utility
BL Minor revision level of config utility
CH MSbyte of checksum of config file
LSbyte of checksum of config file
DH Number of device functions
DL Combined function information byte:
Bit 7 Reserved
Bit 6 Slot has free-form data entries
Bit 5 Slot has port initialization entries
Bit 4 Slot has port range entries
Bit 3 Slot has DMA entries
Bit 2 Slot has IRQ entries
Bit 1 Slot has memory entries
Bit 0 Slot has function type entries
DI First word of compressed device ID
SI Second word of compressed device ID
(See "Read physical slot information" below)

AL 01h = Read function config information
81h = Read function config information, 32 bit

Entry:
CH Function number (0 to n-1)
CL Slot number (0-63)
DS:SI Pointer to output data buffer
Exit:
DS Segment for return data buffer
SI Offset to return data buffer (16 bit)
ESI Offset to return data buffer (32 bit)

AL 02h = Clear EISA CMOS configuration
82h = Clear EISA CMOS configuration 32 bit

Entry:
BH Configuration utility major revision level
BL Configuration utility minor revision level

AL 03h = Write slot config information
83h = Write slot config information, 32 bit

Entry:
CX Length of data structure in bytes
DS Segment of data table
SI Offset of data table (16-bit call)
ESI Offset of data table (32-bit call)

Continued

Interrupt 15h–EISA Services, Continued

AL 04h = Read board ID registers
84h = Read board ID registers, 32 bit

Entry:
CL Slot number (0-63)
Exit:
DI First word of compressed ID:
Byte 0:
Bits 1-0 2nd character of manufacturer code
Bits 6-2 1st character of manufacturer code
Bit 7 Reserved
Byte 1:
Bits 4-0 3rd character of manufacturer code
Bits 5-7 2nd character of manufacture code, cont.
SI Second word of compressed ID:
Byte 0:
Bits 3-0 2nd hex digit of product number
Bits 7-4 1st hex digit of product number
Byte 1:
Bits 3-0 Hex digit of revision number
Bits 7-4 3rd hex digit of product number
If Carry = 1:

Interrupt 16h–Keyboard Services

The INT 16 software interrupt handles keyboard I/O services. The following describes the keyboard services of *Phoenix*BIOS 4.0:

Interrupt 16h Keyboard Services

AH = 00h Read keyboard input
Exit:
AL ASCII keystroke pressed
AH Scan code of key

AH = 01h Return keyboard status
Exit:
AL ASCII keystroke pressed
AH Scan code of key
ZF No keystroke available
NZ Keystroke in buffer

AH = 02h Return shift-flag status
Exit:
AL Current shift status

AH = 03h Set typematic rate and delay.
Entry:
AL 05 (sub function number)
BL 00H through 1FH, typematic rate
(30 chars/sec to 2 char/sec)
BH Delay rate:
00h = 250 ms
01h = 500 ms
02h = 750 ms
03h = 1000 ms
04h to 07h = Reserved

AH = 05h Add key to Keyboard buffer.
Entry:
CL ASCII code
CH Scan code
Exit:
If Carry = 1:
AL Keyboard buffer full

AH = 10h Read extended character from buffer.
Exit:
AL ASCII keystroke pressed
AH Scan code of key

Continued;

Interrupt 16h–Keyboard Services, Continued

AH = 11h Return extended buffer status.

Exit:

AL	ASCII keystroke pressed
AH	Scan code of key
ZF	No keystroke available
NZ	Keystroke in buffer

AH = 12h Return extended shift status.

Exit:

AL Shift status:

Bit 7	1 = Sys Req pressed
Bit 6	1 = Caps Lock active
Bit 5	1 = Num Lock active
Bit 4	1 = Scroll Lock active
Bit 3	1 = Right Alt active
Bit 2	1 = Right Ctrl active
Bit 1	1 = Left Alt active
Bit 0	1 = Left Ctrl active

AH Extended shift status:

Bit 7	1 = Insert active
Bit 6	1 = Caps Lock active
Bit 5	1 = Num Lock active
Bit 4	1 = Scroll Lock active
Bit 3	1 = Alt pressed
Bit 2	1 = Ctrl pressed
Bit 1	1 = Left Shift pressed
Bit 0	1 = Right Shift pressed

Interrupt 17h–Parallel Printer Services

The INT 17 software interrupt supports up to 4 parallel adapters. The BIOS stores the standard base addresses for three parallel adapters in the BIOS Data Area at 3FCh, 378h, and 278h. These services use the I/O ports 0278h-027Ah, 0378h-037Ah, and 03BCh-03BEh.

Interrupt 17h Parallel Printer Services

AH = 00h Print character

Entry:

AL	Character to print
DX	Printer port (0-3)

Exit:

AH Printer Status (see below)

AH = 01h Initialize printer port

Entry:

DX	Printer port (0-3)

Exit:

AH Printer Status (see below)

AH = 02h Return printer status

Entry:

DX Printer port (0-3)

Exit:

AH Printer Status:

Bit 0	1 = Time-out error
Bit 1	Reserved
Bit 2	Reserved
Bit 3	1 = I/O error
Bit 4	1 = Printer selected
Bit 5	1 = Out of paper
Bit 6	1 = Acknowledgment from printer
Bit 7	1 = Printer not busy

Interrupt 17h–EPP Services

Use Interrupt 17h 02h to obtain the BIOS entry point (also called the EPP Vector) to Enhanced Parallel Printer (EPP) Services. To use the other EPP services, load AH with an appropriate function value and Far call the EPP Vector.

The following are the EPP exit status codes:

EPP Services Exit Status Codes

AH 00h = No error
01h = Failed I/O function
02h = Invalid function
03h = EPP not supported
04h = Not an EPP port
20h = Multiplexor not present
40h = Multiplexor already locked

The following are the Int 17 EPP services of *Phoenix*BIOS 4.0:

Interrupt 17h EPP Service

AH = 02h EPP Installation check
Entry:
DX EPP printer port (0-2)
AL 0
CH 45h = "E"
BL 50h = "P"
BH 50h = "P"
Exit:
AL 45h
CX 5050h
DX:BX EPP BIOS entry point

Vectored EPP Services
(Call entry point)
AH = 00h Query EPP port configuration
Entry:
DL EPP printer port (0-2)
Exit:
AL Interrupt level of EPP port (00-15h)
FFh = Interrupts not supported
BH EPP BIOS revision (MMMMnnnn or M.n)
BL I/O capabilities:
Bit 0 Multiplexor present
Bit 1 PS/2 bi-directional capable
Bit 2 Daisy chain present
Bit 3 ECP capable
CX SPP I/O base address
ES:DI FAR pointer to EPP BIOS manufacturer's info/version text string, zero terminated

AH = 01h Set mode
Entry:
DL EPP printer port (0-2)
AL Modes:
Bit 0 Set compatibility mode
Bit 1 Set Bi-directional mode
Bit 2 Set EPP mode
Bit 3 Set ECP mode
Bit 4 Set EPP software emulation (via standard parallel port)

AH = 02h Get mode
Entry:
DL EPP printer port (0-2)
Exit:
AL Modes:
Bit 0 In compatibility mode
Bit 1 In Bi-directional mode
Bit 2 In EPP mode
Bit 3 In ECP mode
Bit 4 In EPP software-emulation mode
Bit 7 EPP port interrupts enabled

AH = 03h Interrupt control
Entry:
DL EPP printer port (0-2)
AL 0 = Disable EPP port interrupts
1 = Enable EPP port interrupts

AH = 04h Reset EPP port
Entry:
DL EPP printer port (0-2)

Continued

Interrupt 17h–EPP Services, Continued

AH = 05h Write address/select device
Entry:
DL EPP printer port (0-2)
AL Device address to write

AH = 06h Read address
Entry:
DL EPP printer port (0-2)
AL Device address to write
Exit:
AL Address/device data returned

AH = 07 Write byte
Entry:
DL EPP printer port (0-2)
AL Data byte

AH = 08 Write block
Entry:
DL EPP printer port (0-2)
CX Number of bytes to write (0 = 64k)
ES:SI Client buffer w/data
Exit:
CX Bytes not transferred (0 = no error)

AH = 09h Read byte
Entry:
DL EPP printer port (0-2)
Exit:
AL Data byte returned

AH = 0Ah Read block
Entry:
DL EPP printer port (0-2)
CX Number of bytes to read (0 = 64k)
ES:DI Client buffer for returned data
Exit:
CX Bytes not transferred (0 = no error)

AH = 0Bh Write address, read byte
Entry:
DL EPP printer port (0-2)
AL Device address
Exit:
AL Data byte returned

AH = 0Ch Write address, write byte
Entry:
DL EPP printer port (0-2)
AL Device address
DH Data byte to write

AH = 0Dh Write address, read block
Entry:
DL EPP printer port (0-2)
AL Device address
CX Number of bytes to read (0 = 64k)
ES:DI Client buffer for data
Exit:
AL Returned byte data
CX Bytes not transferred (0 = no error)

AH = 0Eh Write address, write block
Entry:
DL EPP printer port (0-2)
AL Device address
CX Number of bytes to write
ES:SI Client buffer w/data
Exit:
CX Bytes not transferred (0 = no error)

AH = 0Fh Lock port
Entry:
DL EPP printer port (0-2)
BL Port address:
Bits 7-4 Daisy chain port number (1-8)
Bits 3-0 Mux device port number (1-8)
0 = No multiplexor

AH = 10h Unlock port
Entry:
DL EPP printer port (0-2)

Continued

Interrupt 17h–EPP Services, Continued

AH = 11h Device interrupt

Entry:
- DL EPP printer port (0-2)
- BL The multiplexor device port (1-8)
  0 = No multiplexor
- AL 0 = Disable device interrupts
  1 = Enable device interrupts.
- ES:DI Far pointer to interrupt-event handler

AH = 12h Real time mode

Entry:
- AL 0 = Query if any real-time device present
  1 = Add (advertise) real-time device
  2 = Remove real-time device

Exit:
- AL 0 = No real-time devices present
  1 = One or more real-time devices present

AH = 40h Query multiplexor

Entry:
- DL EPP printer port (0-2)

Exit:
- AL Bit 0 1 = Channel locked
  Bit 1 1 = Interrupt pending
- BL Currently selected port

AH = 41h Query multiplexor device port

Entry:
- DL EPP printer port (0-2)
- BL The multiplexor device port (1-8)
  0 = No multiplexor

Exit:
- AL Status flags:
  Bit 0 1 = Port selected
  Bit 1 1 = Port locked
  Bit 2 1 = Interrupts enabled
  Bit 3 1 = Interrupt pending
- CX EPP product/Device ID
  0 = Undefined

AH = 42h Set product ID

Entry:
- DL EPP printer port (0-2)
- AL Mapped EPP Mux device port (1-8)
- CX EPP Product ID

AH = 50h Rescan daisy chain

Entry:
- DL EPP printer port (0-2)
- **BL The multiplexor device port (1-8)**
  0 = No multiplexor

AH = 51h Query daisy chain

Entry:
- DL EPP printer port (0-2)
- BL The multiplexor device port (1-8)
  0 = No multiplexor

Exit:
- AL Status flags:
  Bit 0 1 = Channel locked
  Bit 1 1 = Interrupt pending
- BL Currently selected device
- CL Depth of daisy chain on this port
  0 = No daisy chain on this port
- ES:DI Pointer to ASCII string, driver vendor ID

Interrupt 1Ah–Time of Day Services

The INT 1Ah software interrupt handles the time of day I/O services. A Carry flag set on exit may indicate the clock is not operating.

Interrupt 1Ah Time-of-Day Services

AH = 00h	**Read current time**
Exit:	
CX	High word of tick count
DX	Low word of tick count
AL	00h = Day rollover has not occurred (Timer count is less than 24 hours since last power on or reset)
AH = 01h	**Set current time (Clear rollover bit)**
Entry:	
CX	High word of tick count
DX	Low word of tick count
AH = 02h	**Read real time clock**
Exit:	
CH	BCD hours
CL	BCD minutes
DH	BCD seconds
DL	00 = Standard Time 01h = Daylight Savings
AH = 03h	**Set the real time clock**
Entry:	
CH	BCD hours
CL	BCD minutes
DH	BCD seconds
DL	01h = Daylight saving 00h = Otherwise
AH = 04h	**Read date from real time clock**
Exit:	
CH	BCD century
CL	BCD year
DH	BCD month
DL	BCD date
AH = 05h	**Set date in real time clock**
Entry:	
CH	BCD century
CL	BCD year
DH	BCD month
DL	BCD date
AH = 06h	**Set real-time alarm**
Entry:	
CH	BCD hours to alarm
CL	BCD minutes to alarm
DH	BCD seconds to alarm
Exit:	
C	1 = Alarm already set
AH = 07h	**Reset real-time alarm**
Exit:	
AL	Value written to CMOS RAM register 0Bh

Interrupt 1Ah–General PCI Services

PhoenixBIOS 4.0 optionally supports General PCI Interrupt 1Ah Services. The following are the exit status codes:

PCI Services Exit Status Codes

AH	00h = Successful
	If Carry = 1:
AH	81h = Function not supported
	83h = Bad vendor ID
	86h = Device not found
	87h = Bad register number
	88h = Set failed
	89h = Buffer too small

The following are the PCI Services:

Interrupt 1Ah General PCI Services

AH = B1h PCI Services

AL 01h = PCI BIOS present

Exit:
EDX "PCI", "P" in [DL], "C" in [DH], etc.
AL Hardware mechanism:

Bit	Description
5	Spec. Cycle–Config Mechanism #2 support
4	Spec. Cycle–Config Mechanism #1 support
1	Config Mechanism #2 support
0	Config Mechanism #1 support

BH Interface level major version
BL Interface level minor version
CL Number of last PCI bus

AL 02h = Find PCI Device

Entry:
CX Device ID (0-65535)
DX Vendor ID (0-65534)
SI Index (0-n)
Exit:
BH Bus number (0-255)
BL Bits 7-3 Device number
Bits 2-0 Function number

AL 03h = Find PCI class code

Entry:
ECX Class code in lower three bytes
SI Index (0-n)
Exit:
BH Bus number (0-255)
BL Bits 7-3 Device number
Bits 2-0 Function number

AL 06h = Generate special cycle

Entry:
BH Bus number (0-255)
EDX Special cycle data

AL 08h = Read configuration byte

Entry:
BH Bus number (0-255)
BL Bits 7-3 Device number
Bits 2-0 Function number
DI Register number (0-255)
Exit:
CL Byte read

AL 09h = Read configuration word

Entry:
BH Bus number (0-255)
BL Bits 7-3 Device number
Bits 2-0 Function number
DI Register number (0, 2, 4,...254)
Exit:
CX Word read

AL 0Ah = Read configuration Dword

Entry:
BH Bus number (0-255)
BL Bits 7-3 Device number
Bits 2-0 Function number
DI Register number (0, 4, 8,...252)
Exit:
ECX Dword read

AL 0Bh = Write configuration byte

Entry:
BH Bus number (0-255)
BL Bits 7-3 Device number
Bits 2-0 Function number
DI Register number (0-255)
CL Byte value to write

Continued

Interrupt 1Ah–General PCI Services, Continued

AL 0Ch = Write configuration word

Entry:
BH Bus number (0-255)
BL Bits 7-3 Device number
Bits 2-0 Function number
DI Register number (0, 2, 4,...254))
CX Word value to write

AL 0Dh = Write configuration Dword

Entry:
BH Bus number (0-255)
BL Bits 7-3 Device number
Bits 2-0 Function number
DI Register number (0, 4, 8,...252)
ECX Dword value to write.

AL 0Eh = Get PCI IRQ routing options

Entry:
DS Segment or Selector for BIOS data
ES Segment or Selector for Route Buffer parameter
DI 16-bit offset for Route Buffer parameter
EDI 32-bit offset for Route Buffer parameter
Exit:
BX Exclusive-PCI IRQ data map:
Bit 0 1 = IRQ0 PCI only
Bit 1 1 = IRQ1 PCI only
...
Bit 15 1 = IRQ15 PCI only

AL 0Fh = Set PCI hardware interrupt

Entry:
BH Bus number (0-255)
BL Bits 7-3 Device number
Bits 2-0 Function number
CL PCI interrupt pin (0Ah...0Dh)
CH IRQ number (0-15)
DS Segment or Selector for BIOS data

PnP Run-Time Services

Plug and Play automatically configures PC hardware and attached devices without requiring you to manually configure the device with jumpers or in Setup. You can install a new device such as sound or fax card ("plug it in") and start working ("begin playing").

To work properly, however, Plug-and-Play must be supported in the hardware and software, including the BIOS, the operating system (such as Microsoft Windows 95), and the hardware drivers.

Each Plug and Play device must have all of the following capabilities:

1. It must be uniquely identified
2. It must state the services it provides and the resources it requires
3. It must allow software to configure it.

Note: To register a new unique vendor ID or manufacturer ID for Plug and Play hardware, please send e-mail to pnpid@microsoft.com.

NOTE: There are a variety of Plug and Play technologies, including BIOS, ISA, SCSI, IDE, CD-ROM, LPT, COM, PCMCIA, and drivers. For complete instructions on using the PnP BIOS Services, consult the *Plug and Play BIOS Specification V. 1.0a.* You can download this specification and other PnP specifications from this Microsoft Web site:

http://www.microsoft.com/hwdev/specs/pnpspecs.htm

PhoenixBIOS 4.0 optionally supports PnP (Plug and Play) Runtime Services in Real and Protected Mode in with the following routines:

PnP Run-Time Services	
00h	Get Number of Device Nodes
01h	Get Device Node
02h	Set Device Node
03h	Get Event
04h	Send Message
05h	Get Docking Station Information
09h	Set Statically Allocated Resources
0Ah	Get Statically Allocated Resources
0Bh	Get APM 1.1 ID Table
40h	Get ISA Configuration Structure
41h	Get ESCD Information
42h	Read ESCD Data Image
43h	Write ESCD Data Image

The following are the exit status codes for the PnP Runtime Services

PnP Runtime Service Exit Status Codes	
AH	00h = No error
	If Carry = 1:
AH	7Fh = Device not set statically
	81h = Unknown or invalid function
	82h = Function not supported
	83h = Handle for Device Node invalid or out of range
	84h = Bad resource descriptors
	85h = Set Device Node function failed
	86h = No events pending
	87h = System currently not docked
	88h = No ISA PnP cards installed
	89h = Cannot determine docking station capabilities
	8Ah = Undocking failed: no battery
	8Bh = Docking failed: conflict with primary boot device
	8Ch = Caller's memory buffer too small
	8Dh = Use ESCD support function instead
	8Eh = Send Message 04h function not supported
	8Fh = Hardware error

To find the PnP entry points, search for the **PnP BIOS Support Installation Check** structure by searching for the "$PnP" signature in system memory staring from F0000h to FFFFFh at every 16-byte boundary. Check the validity of the structure by adding the values of *Length* bytes, including the *Checksum* field, into a 8-bit value. Zero indicates a valid checksum.

The following describes the support structure:

PnP Support Installation Check		
Offset	**Size**	**Description**
00h	4	ASCII "$PnP" signature
04h	1	Version (10h)
05h	1	Length (21h)
06h	2	Control field
08h	1	Checksum
09h	4	Event-notification flag address
0Dh	2	Real Mode 16-bit offset to entry point
0Fh	2	Real Mode 16-bit code segment address
11h	2	16-bit Protected Mode offset to entry point
13h	4	16-bit Protected Mode code segment base address
17h	4	OEM Device Identifier
1Bh	2	Real Mode 16-bit data segment address
1Dh	4	16-bit Protected Mode data segment base address

Call each service by loading the function parameters on the stack and FAR calling the appropriate entry point. The following are the Runtime Services of *Phoenix*BIOS 4.0, in 'C' syntax.

PnP Runtime-Service Function Parameters

00h Get Number of Device Nodes

```
Entry:
int FAR (*entryPoint)(Function, NumNodes, NodeSize,
BiosSelector);
int Function;
unsigned char FAR *NumNodes;
unsigned int FAR *NodeSize;
unsigned int BiosSelector;
```

01h Get System Device Node

```
Entry:
int FAR (*entryPoint)(Function, Node, devNodeBuffer,
Control, BiosSelector);
int Function;
unsigned char FAR *Node;
struc DEV_NODE FAR *devNodeBuffer;
unsigned int Control;
unsigned int BiosSelector;
```

02h Set System Device Node

```
Entry:
int FAR (*entryPoint)(Function, Node, devNodeBuffer,
Control, BiosSelector);
int Function;
unsigned char Node;
struc DEV_NODE FAR *devNodeBuffer;
unsigned int Control;
unsigned int BiosSelector;
```

03h Get Event

```
Entry:
int FAR (*entryPoint)(Function, Message, BiosSelector);
int Function;
unsigned int FAR *Message;
unsigned int BiosSelector;
```

04h Send Message

```
Entry:
int FAR (*entryPoint)(Function, Message, BiosSelector);
int Function;
unsigned int Message:
unsigned int BiosSelector;
```

05h Get Docking Station Information

```
Entry:
int FAR (*entryPoint)(Function, DockingStationInfo,
BiosSelector);
int Function;
unsigned char FAR *DockingStationInfo;
unsigned int BiosSelector;
Exit:
Docking station info buffer:
        Offset 00h Docking station location identifier
        Offset 04h Serial Number
        Offset 08h Docking Capabilities:
          Bits 2-1:
              00 = Cold Docking
              01 = Warm Docking
              10 = Hot Docking
          Bit 0:
              0 = Surprise-style docking
              1 = VCR-style docking
```

09h Set Statically Allocated Resources

```
Entry:
int FAR (*entryPoint)(Function, Resource Block,
BiosSelector);
int Function;
unsigned char FAR *ResourceBlock;
unsigned int BiosSelector;
```

Continued

PnP Run-Time Services, Continued

0Ah Get Statically Allocated Resources

```
Entry:
int FAR (*entryPoint)(Function, Resource Block,
BiosSelector);
int Function;
unsigned char FAR    *ResourceBlock;
unsigned int      BiosSelector;
```

0Bh Get APM ID Table (For APM 1.1 only)

```
Entry:
int FAR (*entryPoint)(Function, BufSize, APMIdTAble
BiosSelector);
int Function;
unsigned int FAR      *BufSize;
unsigned char FAR *APMIdTable;
unsigned int BiosSelector;
Exit:
APM ID table:
```

Length	Description
Dword	Device identifier
Word	APM 1.1 identifier

40h Get PnP ISA Configuration Structure

```
Entry:
int FAR (*entryPoint)(Function, Configuration, BiosSelector);
int Function;
unsigned char      FAR *Configuration;
unsigned int BIOS Selector;
Exit:
PnP ISA Configuration structure:
```

Offset	Description
00h	Structure revision
01h	Number of Card Select Numbers assigned
02h	ISA Read Data port
04h	Reserved

41h Get Extended System Configuration Data (ESCD)

```
Entry:
int FAR (*entryPoint)(Function, MinESCDWriteSize,
ESCDSize, NVStorageBase, BiosSelector);
int Function;
unsigned int      FAR *MinESCDWriteSize;
unsigned int FAR     *ESCDSize;
unsigned long     FAR *NVStorageBase;
unsigned int BiosSelector;
```

42h Read Extended System Configuration Data

```
Entry:
int FAR (*entryPoint)(Function, ESCDBuffer, ESCDSelector,
BiosSelector);
int Function;
char FAR    *ESCDBuffer;
unsigned int ESCDSelector;
unsigned int BiosSelector;
```

43h Write Extended System Configuration Data (ESCD)

```
Entry:
int FAR (*entryPoint)(Function, ESCDBuffer, ESCDSelector,
BiosSelector);
int Function;
char FAR    *ESCDBuffer;
unsigned int ESCDSelector;
unsigned int BiosSelector;
```

SMBIOS Services

The **System Management BIOS (SMBIOS),** one of the components of the Desktop Management Interface (DMI), is a method for managing PCs in an enterprise. Using
SMBIOS, a Manager of Information Systems can access up-to-date information about the hardware and software installed on every computer on a network.

NOTE: For complete instructions on using these services, see the **System Management BIOS Reference Specification** available at the Phoenix Web site: http://www.phoenix.com/products/specs-smbios.pdf

For descriptions of the DMI architecture, see the Web site of the **Desktop Management Task Force** at: http://www.dmtf.org

The SMBIOS Services are functions 50h through 5Fh of the PnP Run Time Services. See "PnP Run-Time Services" above for a description of how to find the PnP entry points to these SMBIOS Services. The following are the SMBIOS services supported in PhoenixBIOS 4.0:

SMBIOS Services	
50h	Get SMBIOS Information
51h	Get SMBIOS Structure
52h	Set SMBIOS Structure
55h	Get GPNV Information
56h	Read GPNV Information
57h	Write GPNV Data

The following are the exit status codes for the SMBIOS Services:

SMBIOS Services Exit Status Codes	
AX	00h = Function Completed Successfully
AX	81h = Unknown, or invalid, function number passed 82h = The function is not supported on this system 83h = SMBIOS Structure number/handle passed is invalid or out of range. 84h = The function detected invalid parameter or, in the case of a "Set SMBIOS Structure" request, detected an invalid value for a to-be-changed structure field 85h = The SubFunction parameter supplied on a SMBIOS control function is not supported by the system BIOS. 86h = There are no changed SMBIOS structures pending notification. 87h = Returned when there was insufficient storage space to add the desired structure. 8Dh = A "Set SMBIOS Structure" request failed because one or more of the to-be-changed structure fields are read-only. 90h = The GPNV functions do not support locking for the specified GPNV handle. 91h = The GPNV lock request failed - the GPNV is already locked. 92h = The caller has failed to present the predefined GPNVLock value which is expected by the BIOS for access of the GPNV area.

Call each service by loading the function parameters on the stack and FAR calling the appropriate entry point. The following are the SMBIOS Services in 'C' syntax.

SMBIIOS Function Parameters

50h Get SMBIOS Information

Entry:

```
short FAR (*entryPoint)(short Function,
        unsigned char  FAR  *dmiBIOSRevision,
        unsigned short FAR *NumStructures,
        unsigned short FAR  *StructureSize,
        unsigned long FAR *dmiStorageBase,
        unsigned short FAR  *dmiStorageSize,
        unsigned short *BiosSelector );
```

51h Get SMBIOS Structure

Entry:

```
short FAR (*entryPoint) (
        short Function;
        unsigned short FAR *Structure;
        unsigned char FAR *dmiStrucBuffer;
        unsigned short dmiSelector;
        unsigned short BiosSelector);
```

52h Set SMBIOS Structure

Entry:

```
short FAR (*entryPoint) (
        short Function;
        unsigned char FAR *dmiDataBuffer,;
        unsigned char FAR *dmiWorkBuffer,
        unsigned char Control,
        unsigned short dmiSelector;
        unsigned short BiosSelector);
```

55h Get General-Purpose NonVolatile Information

Entry:

```
short FAR (*entryPoint) (
        short Function;
        unsigned short FAR *Handle,
        unsigned short FAR *MinGPNVRWSize,
        unsigned short FAR *GPNVSize,
        unsigned long FAR *NVStorageBase,
        unsigned short BiosSelector);
```

56h Read General-Purpose NonVolatile Data

Entry:

```
short FAR (*entryPoint) (
        short Function;
        unsigned short Handle,
        unsigned char FAR *GPNVBuffer,
        short FAR *GPNVLock,
        unsigned short GPNVSelector,
        unsigned short BiosSelector);
```

57h Write General-Purpose NonVolatile Data

Entry:

```
short FAR (*entryPoint)(
        short Function,
        unsigned short Handle,
        unsigned char FAR *GPNVBuffer,
        short GPNVLock,
        unsigned short GPNVSelector,
        unsigned short BiosSelector );
```

MultiBoot III Run-Time Services

An OS or application program can access the features of PhoenixBIOS MultiBoot II during run-time by using the following MultiBoot III Run-Time Services. You can use these services to query the number and type of Initial Program Load (IPL) devices in the system or display an IPL device menu for specifying the boot priority on the next system restart.

MultiBoot II Run-Time Services are extensions to the Plug and Play run-time functions that implement the *BIOS Boot Specification Ver. 1.01*. You can access this specification in Acrobat format from the Phoenix Web site at:
http://www.phoenix.com/desktop/bbs101.pdf

PnP functions 60h through 6Fh are reserved for the BIOS Boot Specification. See Appendix C of the *Plug and Play BIOS Specification* mentioned above for the details of the calling conventions. These functions are available in Real Mode and 16-bit Protected Mode.

MultiBoot III Run-Time Services

60h Get Version and Installation Check

```
Entry:
short FAR (* entryPoint) (Function, Version, BiosSelector);
short Function;
unsigned short FAR *Version;
unsigned short BiosSelector;
```

61h Get Device Count

```
Entry:
short FAR (* entryPoint) (Function, Switch, Count,
MaxCount, StructSize, BiosSelector);
short Function;
short Switch;
unsigned short FAR *Count;
unsigned short FAR *MaxCount;
unsigned short FAR *StructSize;
unsigned short BiosSelector;
```

62h Get Priority and Table

```
Entry:
short FAR (* entryPoint) (Function, Switch, Priority, Table,
 BiosSelector);
short Function;
short Switch;
unsigned char FAR *Priority;
unsigned char FAR *Table;
unsigned short BiosSelector;
```

63h Set Priority

```
Entry:
short FAR (* entryPoint) (Function, Switch, Priority,
BiosSelector);
short Function;
short Switch;
unsigned byte FAR *Priority;
unsigned short BiosSelector;
```

64h Get IPL Device from Last Boot

```
Entry:
short FAR (* entryPoint) (Function, IPLEntry, BiosSelector);
short Function;
unsigned short FAR *IPLEntry;
unsigned short BiosSelector;
```

BIOS Data Area

The BIOS keeps information about the current operating environment of the AT system in the BIOS Data Area. The normal way to access this information is by means of the BIOS Services, described above. The BIOS Data Area is located from physical address 400h to 501h.

BIOS Data Area Description

Offset	Size	Description
00	2	Com1 address
02	2	Com2 address
04	2	Com3 address
06	2	Com4 address
08	2	Lpt1 address
0A	2	Lpt2 address
0C	2	Lpt3 address
0E	2	LPT4/EBDA address*
10	2	Equipment installed:
	Bit	**Definition**
	0	Not used
	1	Math coprocessor installed
	2	PS/2 mouse installed
	3	Not used
	4,5	Initial video mode: 00 = EGA/VGA 01 = 40x25 CGA 10 = 80x25 CGA 11 = Monochrome
	6,7	Diskette drives: 00 = 1 drive 01 = 2 drives 10 = 3 drives 11 = 4 drives
	8	Not used
	9-11	Number of serial adapters
	12	Game Adapter installed
	13	Not used
	14,15	Number of parallel adapters

Offset	Size	Description
12	1	Interrupt flag (POST)
13	2	Memory size (K bytes)
15	1	Reserved
16	1	Control flag

Keyboard Data Area

Offset	Size	Description
17	1	Keyboard flag 0: Bit Definition 0 Right shift key pressed 1 Left shift key pressed 2 Control key pressed 3 Alt key pressed 4 Scroll lock on 5 Num lock on 6 Caps lock on 7 Insert mode on
18	1	Keyboard flag 1: Bit Definition 3 Freeze state 4 Scroll lock pressed 5 Num lock pressed 6 Caps lock pressed 7 Insert mode pressed
19	1	Keypad input byte
1A	2	Key buffer head
1C	2	Key buffer tail
1E	20	Key buffer

Continued

BIOS Data Area, Continued

Diskette Data Area

Offset	Size	Description
3E	1	Seek/recalibrate status
3F	1	Drive motor status
40	1	Motor on time
41	1	Diskette status:

Bit	Definition
7	1 = Drive not ready
6	1 = Seek error occurred
5	1 = Diskette controller failed
4-0	Error codes:
	01h = Illegal function request
	02h = Address mark not found
	03h = Write protected error
	04h = Sector not found
	06h = Diskette change line active
	08h = DMA overrun on operation
	09h = Data-boundary error (64k)
	0Ch = Media type not found
	10h = Uncorrectable ECC or CRC error
	20h = General controller failure
	40h = Seek operation failed
	80h = Device did not respond

42	7	Controller status

Video Data Area

Offset	Size	Description
49	1	Video mode
4A	2	Video columns
4C	2	Video length
4E	2	Video start
50	10	Cursor locations
60	2	Cursor size
62	1	Active page
63	2	6845 address
65	1	Mode register value
66	1	Video palette

Extended Work Area

67	4	ROM check address
6B	1	CPU rate control

Timer Data Area

6C	2	Timer count low word
6E	2	Timer count high word
70	1	Timer overflow byte

System Data Area

71	1	Break pressed flag
72	2	Soft reset flag

Fixed Disk Data Area

74	1	Fdisk status
75	1	Number of fixed disks
76	1	Fixed disk control
77	1	Reserved

Serial and Parallel Timeout Counters

78	4	Lpt1-4 time-out values
7C	4	Com1-4 time-out values

Extended Keyboard Data Area

80	2	Key buffer start
82	2	Key buffer end

EGA/VGA Data Area

84	1	Number of video rows
85	2	Bytes per character
87	1	EGA Status A
88	1	EGA Status B
89	1	VGA Status A
8A	1	Display Combination Code index

Extended Diskette Area

8B	1	Last diskette data rate

Continued

BIOS Data Area, Continued

Extended Fixed Disk Area

8C	1	FDisk status
8D	1	FDisk error value
8E	1	FDisk interrupt flag

Additional Extended Diskette Area

Offset	Size	Description
8F	1	Floppy info nibbles
90	4	Floppy state information
94	2	Floppy cylinder number

Additional Extended Keyboard Data Area

96	1	Keyboard control
97	1	Keyboard flag 2:

Bit	Definition
0	Scroll LED on
1	Num lock LED on
2	Caps lock LED on
4	Ack code received
5	Resend received
6	LED being updated
7	Keyboard error

Real Time Clock Area

Offset	Size	Description
98	4	RTC user flag
9C	2	RTC time low word
9E	2	RTC time high word
A0	1	RTC wait flag

Network Data Area

A1	7	Network work area

Extended EGA/VGA Data Area

A8	4	EGA/VGA environment pointer

Miscellaneous

AC-FF		Reserved
100	1	Print screen flag

* If the BIOS supports the Extended BIOS Data Area, it uses the LPT4 address in the BIOS data area (Offset 0E) for the Extended BIOS Data Area segment.

Extended BIOS Data Area

The Extended BIOS Data Area (EBDA), located in the top 1k of system RAM, contains information about the pointing device (PS/2 mouse).

INT 15h AH = C1h returns the segment starting address of this table.

Extended BIOS Data Area

Offset	Size	Description
00h	1	Size of EBDA in kbytes
01h	33	Reserved
21h	4	Pointer to device routine
25h	1	First byte of pointer information:

Bit	Definition
4	Pointer error
5	Pointer acknowledge
6	Resend request
7	Command in progress

26h	1	Second byte of pointer information

Bit	Definition
6	Enable pointer device
7	Pointer external device

27h	2	Pointer data package

Interrupt Vectors

The following table describes the AT system interrupt vectors. Status indicates whether the BIOS supports the interrupt.

INT	Description	Status
00	Divide by zero	Not Supported
01	Single step	Not Supported
02	Non-Maskable interrupt	Supported
03	Breakpoint	Not Supported
04	Overflow	Not Supported
05	Print Screen Interrupt	Supported
06	286 LoadAll Handler	Supported
07	Reserved	Not Supported
08	IRQ0 - System Timer Interrupt	Supported
09	IRQ1 - Keyboard Interrupt	Supported
0A	IRQ2 - Reserved	Not Supported
0B	IRQ3 - COM2: Interrupt	Supported
0C	IRQ4 - COM1: Interrupt	Supported
0D	IRQ5 - LPT2: Interrupt	Supported
0E	IRQ6 - Floppy Disk Interrupt	Supported
0F	IRQ7 - LPT1: Interrupt	Supported
10	BIOS Video Interface	Supported
11	BIOS Equipment Check	Supported
12	BIOS Memory Request	Supported
13	BIOS Fixed Disk/Diskette Interface	Supported
14	BIOS Serial Interface	Supported
15	BIOS System Functions Interface	Supported
16	BIOS Keyboard Interface	Supported
17	BIOS Parallel Printer Interface	Supported
18	BIOS Secondary Boot Request	Supported
19	BIOS Primary Boot Request	Supported
1A	BIOS System Timer Interface	Supported
1B	BIOS Control Break Interrupt	Supported
1C	BIOS User System Timer Interrupt	Supported
1D	BIOS Video Init Parameters	Supported
1E	BIOS Diskette Parameters	Supported
1F	BIOS Video Graphic Characters	Supported
40	BIOS Diskette (when fixed disk present)	Supported
41	BIOS Fixed disk 0 parameters	Supported
46	BIOS Fixed disk 1 parameters	Supported
70	IRQ8 - Real time clock interrupt	Supported
71	IRQ9 - IRQ2 redirection	Supported
72	IRQ10 - Reserved	Not Supported
73	IRQ11 - Reserved	Not Supported
74	IRQ12 - Available/PS/2 Mouse	Supported
75	IRQ13 - Math coprocessor	Supported
76	IRQ14 - Primary IDE HDD	Supported
77	IRQ15 - Available/Secondary IDE HDD	Supported

Index

6.2.3 Ampro Information on Award ROM BIOS

The following provides the Ampro description of the Award ROM BIOS from the Ampro Internet site with web hooks at

http://www.ampro.com/products/software/sw-bios.htm.

Also printed is the web summary of the Ampro BIOS & Utilities at

http://www.ampro.com/products/software/sw-bios.pdf

When the system powers up, the details of the BIOS version are displayed on the video screen. The version installed is

Award modular BIOS v 3.10 – Copyright 1984-1990 Award Software Inc.
CM/3SXi R1.03 – Copyright 1985-1999 Ampro Computers Inc.

Note no BIOS id string appeared at bottom of screen during the BIOS testing.

Embedded-PC BIOS & Utilities

Enhanced PC-compatible BIOS and utilities for embedded systems

Further information available for this product:

1. True Embedded System Features

- Solid State Disk Support
- Power Management
- Watchdog Timer
- Battery-Free Operation
- No-Fail Startup
- "Instant-On" Support
- Unattended Operation

2. Remote Access Features

- Serial Terminal Support
- Remote Software Updates
- Serial Boot Loader
- Serial Debugging Capabilities

3. OEM Customizable Features

- OEM Hooks
- User EEPROM Space
- Bidirectional Parallel Port
- Extended BIOS Services

Download Embedded-PC BIOS Utilities Datasheet (1790K Acrobat PDF)

Embedded-PC BIOS & Utilities

Remote Access Features

"Black box" applications abound in the embdeed-PC world. While PC compatible, these systems feature none of the video monitors, keyboards, disk drives, or other traditional I/O devices of "normal" PCs. Simply put, "black box" applications offer no clear means for a user to gain access to the system.

The Ampro BIOS provides a complete set of remote features which will allow full access to an Ampro target from a serial host. These features may be invoked automatically using a specially configured serial cable.

Serial Terminal Support

All Ampro embedded-PC modules support the use of a serial terminal in place of a video monitor and keyboard.

Remote Software Updates

Ampro offers a unique combination of BIOS features and utilities for performing software updates from a serial host. The features provide a simple mechanism for upgrading software in field-deployed "black box" systems.

Serial Boot Loader

Ampro's enhanced embedded-PC BIOS includes a unique capability for bootstrapping the system over a serial port. An appropriately configured boot image can be downloaded to the Ampro module remotely and used as the target system's boot software.

Serial Debugging Capabilities

Ampro's serial access features also enable remote software development and debugging. Ampro has teamed with several serial debugger suppliers to create versions of their software which use the Serial Boot Loader feature of the Ampro BIOS.

<-- True Embedded System Features | OEM Customizable Features -->
Return to Embedded-PC BIOS Utilities Page

Ampro Home Page - Products - Support - Partners - University - News - Contact

▲ Back To Top Important Copyright and Disclaimer Email webmaster@ampro.com

Ampro embedded-PC products offer a variety of features which allow the OEM to tailor operation to the needs of their specific application. The OEM may use these customizable features to change the startup process or make use of nonstandard system resources. These features include:

OEM Hooks

The Ampro BIOS provides two separate mechanisms for embedded software to gain control of the system during startup. The OEM can use this feature to initialize custom hardware, install a custom boot loader, or perform some other pre-boot operation specific to the application.

User EEPROM Space

An extra 512-bit area in the configuration EEPROM is available for OEM use. This space is commonly used to store small amounts of application software data.

Bidirectional Parallel Port

Ampro modules provide a PC compatible parallel port which may be used bidirectionally for general purpose digital I/0. This option is commonly used to interface to a custom keypad or low resolution LCD display.

Extended BIOS Services

Ampro enhanced BIOS services provide OEM application software with control over the enhanced hardware and BIOS features of Ampro embedded-PC modules. These services allow the OEM to exploit the bytewide socket, user EEPROM, bidirectional parallel port, serial console and boot loader, and other product features for their own custom purposes.

<-- Remote Access Features | Return to Embedded-PC BIOS Utilities Page -->

Download Embedded-PC BIOS Utilities Datasheet (1790K Acrobat PDF)

Ampro Home Page - Products - Support - Partners - University - News - Contact

▲ Back To Top Important Copyright and Disclaimer Email webmaster@ampro.com

Embedded-PC BIOS & Utilities
True Embedded System Features

Ampro Engineering has invested over a decade of effort in enhancing the PC architecture to satisfy the rigourous demands of embedded systems. The following sections outline some of the key hardware and BIOS enhancements that Ampro has developed for embedded-PC applications requirements.

Most embedded applications have little or nothing in common. Yet there are several embedded-PC features that are commonly required by many different systems, regardless of their target applications. A comprehensive set of these enhanced features are provided on all Ampro embedded-PC products.

Solid State Disk Support

True embedded systems often require extremely strong data integrity and have rigorous specifications for power consumption, weight, and immunity to adverse environmental conditions. These requirements may prohibit the use of mechanical storage media such as floppy and hard disk drives. Ampro provides a complete set of solutions for substituting solid state memory for traditional disk drives. Solid state disks provide all of the functionality of magnetic drives, but feature extremely low power consumption, very light weight, and high durability and data reliability. Removable options are also available.

Power Management

Low power consumption is one of the most common requirements of embedded systems. This is particularly true of portable and other battery-powered applications, which often must run for extended periods of time on a single battery charge. Ampro embedded-PC modules provide comprehensive BIOS support for both automatic and manual power management operations. These features allow dramatic pwower redictions to be achieved during periods of non-operation. Thermal monitoring and control is also offered on products which require CPU thermal conditioning

Watchdog Timer

many mission-critical systems cannot tolerate the down time resulting from crashes due to software problems, power fluctuations, or other abnormal events. The Watchdog timer protects against fatal stoppages by monitoring system operation and resetting in the event of a failure.

Battery-Free Operation

Ampro embedded-PC modules employ a serial EEPROM chip to store a backup copy of the CMOS RAM data. This eliminates any vulnrability of configuration data to battery failure, which is a common problem among desktop systems. Batteryless operation also allows Ampro products to be used in systems which may be exposed to explosive environmental gasses.

No-Fail Startup

Ampro embedded-PC modules employ various techniques for assuring correct system startup even under adverse conditions. Years of effort have gone into fine-tuning the BIOS to intelligently manage startup errors in case user intervention is not possible.

"Instant-On" Support

Many embedded systems are required to boot up and begin processing within seconds of system power-up. This is much different than desktop PCs, which commonly take a minute or more to reach a fully operational state. Ampro embedded-PC modules provide sophisticated capabilities for dramatically reducing startup delays.

Unattended Operation

Some types of embedded systems are designed to operate unattended for weeks, months, or even years at a time wih no user intervention. Ampro products are designed to allow independent operation for extended periods of time.

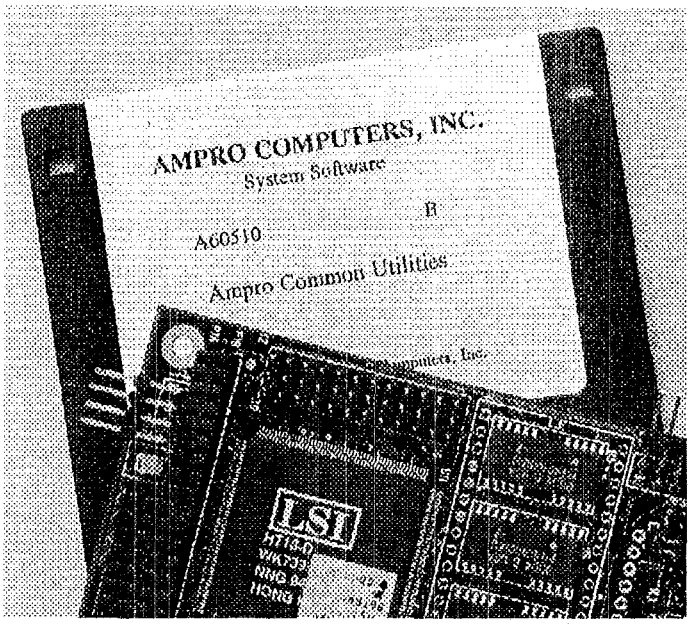
<-- Embedded-PC BIOS Page |Remote Access Features -->

Ampro Home Page - Products - Support - Partners - University - News - Contact

▲ Back To Top Important Copyright and Disclaimer Email webmaster@ampro.com

Embedded-PC BIOS and Utilities

Enhanced PC-compatible BIOS and utilities for embedded systems



- **Proven: field tested in thousands of OEM applications for over seven years**
- **Compatible: based on industry standard Award BIOS core, plus extensively tested Ampro enhancements**
- **Flexible: extended functions and options support the requirements of embedded applications**
- **Reliable: enhanced to support the demands of robust, non-stop, unattended system operation**
- **Complete: includes utilities for extended function support, system development, and field maintenance**
- **Included with all Ampro CPU products**

Ampro's exclusive Embedded-PC BIOS is an enhanced, fully PC-compatible ROM-BIOS that enables reliable, unattended non-stop operation of Ampro Little Board™ and CoreModule™ CPUs in embedded system applications. The Embedded-PC BIOS, included with all Ampro CPU products, greatly increases system reliability, functionality, and flexibility, making it ideal for the broadest possible range of embedded system applications.

Proven Compatibility

To assure the highest degree of PC compatibility, Ampro purchased a source license from Award Software and has used the well-proven Award BIOS as the "kernel" of the Ampro Embedded-PC BIOS. To this fully compatible core, Ampro has added an extensive set of embedded-PC enhancements which have set the standard in the industry for reliability, functionality, flexibility, and support. In all cases, great care in implementation plus thorough internal qualification and field testing have been used to ensure that full compatibility with the standard PC application environment is preserved while offering the highest possible level of reliability and robustness.

Extended Flexibility and Functionality

Embedded systems demand great flexibility in configuring the CPU and system peripherals to support application-specific requirements. Ampro has included BIOS functions which offer a level of system flexibility not found in the normal desktop PC environment. Included are such features as SCSI and solid state disk support, serial console and serial program-download options, non-volatile parameter storage support, and OEM "hooks" that let you customize the system without modifying the BIOS.

Enhanced Reliability

Embedded systems must be able to run reliably in rugged, hostile, and mission-critical environments without operator intervention. A normal PC BIOS, developed for the desktop environment, cannot adequately support these requirements. Over a seven year period, Ampro has evolved a comprehensive set of BIOS enhancements and extensions that increase the robustness and reliability of embedded-PC based system operation. These enhancements include such features as watchdog timer support, fail-safe boot, battery-free boot, and solid state disk support which lets you replace conventional disk drives with rugged and reliable silicon memory devices.

Embedded-PC Utilities

Each Ampro CPU development kit or system includes a set of Embedded-PC Utilities which simplify use of the extended features in the Ampro Embedded-PC BIOS and which assist in system development, operation, and maintenance. Included are utilities for SCSI, watchdog timer control, and support for serially connected remote-hosted system access. A partial list of Ampro's Embedded-PC Utilities appears in the specifications section of this data sheet.

EMBEDDED-PC BIOS EXTENDED FEATURES

- Solid state disk (SSD) support:
  - —Reliability enhancement for rugged, hostile environments
  - —Substitutes EPROM, Flash, or nonvolatile RAM devices for normal floppy drives
  - —Usable as DOS boot device
  - —SSDs may coexist with standard floppy and/or hard disk drives or with PCMCIA devices
  - —NOVRAM and Flash SSDs may be programmed from a resident DOS utility or remotely via a serial access utility

    NOTE: Flash SSD is supported as a (re)programmable read-only DOS drive
- Extended floppy services:
  - —Supports standard PC/AT floppy formats on PC- and XT-compatible CPUs
- SCSI services:
  - —Supports DOS-based system boot and operation using a wide variety of SCSI drives
  - —Extensive configuration options and flexibility
  - —SCSI and IDE drives may coexist
  - —Supports "ASPI" drivers and utilities via a DOS device driver
  - —BIOS SCSI function provides API for application software access and non-standard device support
- Watchdog timer support:
  - —SETUP parameter defines startup WDT time constant, to ensure reliable system BOOT
  - —BIOS function provides API for application software access, to ensure system operation integrity
- Fast boot option:
  - —Offers normal or accelerated power-on self test options for increased system boot speed
  - —Provides microcontroller-like quick startup
- Fail-safe boot support:
  - —Continuously cycles through a defined list of boot devices until a boot device is available
  - —Accommodates slow boot or malfunctioning devices
  - —Supports unattended operation
- Battery-free boot support:
  - —CMOS SETUP data backup is stored in an EEPROM device
  - —Allows system boot and operation with a bad or non-present battery
  - —Supports environments in which batteries are not permitted
  - —Prevents total system failure due to bad batteries (only the real time-of-day function is impaired)
- Serial console option:
  - —Substitutes an external serially-connected device for conventional PC keyboard and video
  - —Saves embedded system cost and space
  - —Allows remote access by users or external computers via serial connection or modem
  - —Useful for diagnostics, system status monitoring, or remote system control
- Serial loader option:
  - —Allows external loading of executable code prior to system boot
  - —Increases embedded system flexibility
  - —Useful for diagnostics, remote system control, and SSD device programming
- EEPROM access function:
  - —BIOS function provides API for application use of 512-bit nonvolatile data area
  - —Useful for implementing features like serialization, copy protection, and passwords
- OEM customization hooks:
  - —Executes custom code prior to system boot via ROM extensions or code "patches"
  - —Allows system customization without BIOS modification
- Advanced Power Management (Refer to Ampro CPU datasheets for further details.)

EMBEDDED-PC UTILITIES

- Serial loader program: DOS utility for transferring a binary executable file from a remote "host" PC to an embedded CPU "target". Useful for a wide variety of purposes including loading of a number of remote debuggers.
- Serial SSD programmer: DOS utility for (re)programming the SSD devices on an embedded CPU "target" under control of a remote serially connected "host" PC
- Setup access utility: DOS program for entering into the in-BIOS SETUP function during system operation
- SCSI support: SCSI device format, initialization, and maintenance utilities
- Watchdog timer utility: DOS program which can be used to "tickle" the CPU's WDT
- Terminal emulation utility: allows a standard PC to emulate an ASCII terminal; for use with the serial console function of an embedded Ampro CPU module
- Advanced Power Management device driver:
  - —DOS device driver supports power management on Ampro CPU modules

    NOTE: APM support is provided as part of the standard embedded-PC BIOS in Ampro's newer products including the Little Board/486*i*.

6.2.4 CMOS Setup Utility User's Guide for ALI M6117 Chipset

The Award CMOS Setup Utility User's Guide for the ALI M6117 Chipset was downloaded from the Phoenix Internet site at:

http://www.phoenix.com/pcuser/bios-award-m6117.pdf.

Award Software International®

CMOS Setup Utility User's Guide

for ALI M6117 Chipset

***Elite*BIOS™ Version 4.51PG**

Return to Award Home Page

Return to BIOS Setup Documents

Table of Contents

Introduction to Setup

This manual describes the Award BIOS Setup program. The Setup program lets you modify basic system configuration settings. The settings are then stored in a dedicated battery-backed memory, called CMOS RAM, that retains the information when the power is turned off.

The Award BIOS in your computer is a customized version of an industry-standard BIOS for IBM PC AT-compatible personal computers. It supports Intel x86 and compatible processors. The BIOS provides critical low-level support for the system central processing, memory, and I/O subsystems.

The Award BIOS has been customized by adding important, but nonstandard, features such as virus and

password protection, power management, and detailed fine-tuning of the chipset controlling the system.

The rest of this manual is intended to guide you through the process of configuring your system using Setup.

Starting Setup

The Award BIOS is immediately activated when you first turn on the computer. The BIOS reads system configuration information in CMOS RAM and begins the process of checking out the system and configuring it through the power-on self test (POST).

When these preliminaries are finished, the BIOS seeks an operating system on one of the data storage devices (hard drive, floppy drive, etc.). The BIOS launches the operating system and hands control of system operations to it.

During POST, you can start the Setup program in one of two ways:

1. By pressing Del immediately after switching the system on, or
2. By pressing Del or by pressing Ctrl-Alt-Esc when the following message appears briefly at the bottom of the screen during POST:

   TO ENTER SETUP BEFORE BOOT PRESS CTRL-ALT-ESC OR DEL KEY

   If the message disappears before you respond and you still wish to enter Setup, restart the system to try again by turning it OFF then ON or by pressing the RESET button on the system case. You may also restart by simultaneously pressing Ctrl-Alt-Del. If you do not press the keys at the correct time and the system does not boot, an error message appears and you are again asked to

   PRESS F1 TO CONTINUE, CTRL-ALT-ESC OR DEL TO ENTER SETUP

Setup Keys

These keys help you navigate in Setup:

Up arrow	Move to previous item
Down arrow	Move to next item
Left arrow	Move to the item in the left hand
Right arrow	Move to the item in the right hand
Esc	Main Menu: Quit and not save changes into CMOS RAM. Other pages: Exit current page and return to Main Menu
PgUp	Increase the numeric value or make changes
PgDn	Decrease the numeric value or make changes
+	Increase the numeric value or make changes
-	Decrease the numeric value or make changes
F1	General help, only for Status Page Setup Menu and Option Page Setup Menu
F2	Change color from total 16 colors. F2 to select Shift-F2 color forward, Shift-F2 to select color backward
F3	Calendar, only for Status Page Setup Menu
F4	Reserved
F5	Restore the previous CMOS value from CMOS, only for Option Page Setup Menu
F6	Load the default CMOS RAM value from BIOS default table, only for Option Page Setup Menu
F7	Load the default
F8	Reserved
F9	Reserved
F10	Save all the CMOS changes, only for Main Menu

Getting Help

Press F1 to pop up a small help window that describes the appropriate keys to use and the possible selections for the highlighted item. To exit the Help Window press Esc or the F1 key again.

In Case of Problems

If, after making and saving system changes with Setup, you discover that your computer no longer is able to boot, the Award BIOS supports an override to the CMOS settings that resets your system to its default configuration.

You can invoke this override by immediately pressing Insert when you restart your computer. You can restart by either using the ON/OFF switch, the RESET button or by pressing Ctrl-Alt-Delete.

The best advice is to alter only settings that you thoroughly understand. In particular, do not change settings in the Chipset screen without a good reason. The Chipset defaults have been carefully chosen by Award or your system manufacturer for the best performance and reliability. Even a seemingly small change to the Chipset setup may causing the system to become unstable.

Setup Variations

Not all systems have the same Setup. While the basic look and function of the Setup program remains the same for all systems, the appearance of your Setup screens may differ from the screens shown here. Each system design and chipset combination require custom configurations. In addition, the final appearance of the Setup program depends on your system designer. Your system designer can decide that certain items should not be available for user configuration and remove them from the Setup program.

Main Setup Menu

When you enter the Award BIOS CMOS Setup Utility, a Main Menu appears on the screen. The Main Menu allows you to select from several Setup functions and two exit choices. Use the arrow keys to select among the items and press Enter to accept and enter the submenu.

A brief description of each highlighted selection appears at the bottom of the screen.

Following is a brief summary of each Setup category.

Standard CMOS	Options in the original PC AT-compatible BIOS.
BIOS Features	Award enhanced BIOS options.
Chipset Features	Options specific to your system chipset.
Power Management	Advanced Power Management (APM) options.
Password Setting	Change, set, or disable a password. In BIOS versions that allow separate user and supervisor passwords, only the supervisor password permits access to Setup. The user password generally allows only power-on access.
IDE HDD Auto Detection	Automatically detect and configure IDE hard disk parameters.
HDD Low Level Format	This option does not appear in many BIOS versions. Most manufacturers of IDE hard drives strongly recommend that you do not run a low-level format on their drives, because of the danger that the bad-track table may be over-written. Award supplies this utility for service personnel only. If you feel that you need to run a low-level format on your hard drive, ***contact your drive manufacturer for instructions!***
Load BIOS Defaults	BIOS defaults are factory settings for the most stable, minimal-performance system operations.
Load Setup Defaults	Setup defaults are factory settings for optimal-performance system operations.
Save & Exit Setup	Save settings in nonvolatile CMOS RAM and exit Setup.
Exit Without Save	Abandon all changes and exit Setup.

Standard CMOS Setup

In the Standard CMOS menu you can set the system clock and calendar, record disk drive parameters and the video subsystem type, and select the type of errors that stop the BIOS POST.

Date

The BIOS determines the day of the week from the other date information. This field is for information only.

Press the right arrow or left arrow key to move to the desired field (date, month, year). Press the PgUp or PgDn key to increment the setting, or type the desired value into the field.

Time

The time format is based on the 24-hour military-time clock. For example, 1 p.m. is 13:00:00. Press the right arrow or left arrow key to move to the desired field . Press the PgUp or PgDn key to increment the setting, or type the desired value into the field.

Daylight Saving

This category may not be present in your Setup program. When enabled, it adds one hour to the clock when daylight-saving time begins. It also subtracts one hour when standard time returns.

Drive C
Drive D

The BIOS supports two IDE drives. This section does not show information about other IDE devices, such as a CD-ROM drive, or about other hard drive types, such as SCSI drives.

NOTE: We recommend that you select type AUTO for all drives.

The BIOS can automatically detect the specifications and optimal operating mode of almost all IDE hard drives. When you select type AUTO for a hard drive, the BIOS detects its specifications during POST, every time the system boots.

If you do not want to select drive type AUTO, other methods of selecting the drive type are available:

1. Match the specifications of your installed IDE hard drive(s) with the preprogrammed values for drive types 1 through 45.
2. Select USER and enter values into each drive parameter field.
3. Use the IDE HDD AUTO DECTECTION function in Setup.

Here is a brief explanation of drive specifications:

- Type: The BIOS contains a table of pre-defined drive types. Each defined drive type has a specified number of cylinders, number of heads, write precompensation factor, landing zone, and number of sectors. Drives whose specifications do not accommodate any pre-defined type are classified as type USER.
- Size: Disk drive capacity (approximate). Note that this size is usually slightly greater than the size of a formatted disk given by a disk-checking program.
- Cyls: Number of cylinders
- Head: Number of heads
- Precomp: Write precompensation cylinder
- Landz: Landing zone
- Sector: Number of sectors
- Mode: Auto, Normal, large, or LBA
    - Auto: The BIOS automatically determines the optimal mode.
    - Normal: Maximum number of cylinders, heads, and sectors supported are 1024, 16, and 63.
    - Large: For drives that do not support LBA and have more than 1024 cylinders.

- LBA (Logical Block Addressing): During drive accesses, the IDE controller transforms the data address described by sector, head, and cylinder number into a physical block address, significantly improving data transfer rates. For drives with greater than 1024 cylinders.

Drive A
Drive B

Select the correct specifications for the diskette drive(s) installed in the computer.

None	No diskette drive installed
360K, 5.25 in	5-1/4 inch PC-type standard drive; 360 kilobyte capacity
1.2M, 5.25 in	5-1/4 inch AT-type high-density drive; 1.2 megabyte capacity
720K, 3.5 in	3-1/2 inch double-sided drive; 720 kilobyte capacity
1.44M, 3.5 in	3-1/2 inch double-sided drive; 1.44 megabyte capacity
2.88M, 3.5 in	3-1/2 inch double-sided drive; 2.88 megabyte capacity

---

Video

Select the type of primary video subsystem in your computer. The BIOS usually detects the correct video type automatically. The BIOS supports a secondary video subsystem, but you do not select it in Setup.

EGA/VGA	Enhanced Graphics Adapter/Video Graphics Array. For EGA, VGA, SEGA, SVGA or PGA monitor adapters.
CGA 40	Color Graphics Adapter, power up in 40 column mode
CGA 80	Color Graphics Adapter, power up in 80 column mode
MONO	Monochrome adapter, includes high resolution monochrome adapters

---

Halt On

During the power-on self-test (POST), the computer stops if the BIOS detects a hardware error. You can tell the BIOS to ignore certain errors during POST and continue the boot-up process. These are the selections:

No errors	POST does not stop for any errors.
All errors	If the BIOS detects any non-fatal error, POST

	stops and prompts you to take corrective action.
All, But Keyboard	POST does not stop for a keyboard error, but stops for all other errors.
All, But Diskette	POST does not stop for diskette drive errors, but stops for all other errors.
All, But Disk/Key	POST does not stop for a keyboard or disk error, but stops for all other errors.

Memory

You cannot change any values in the Memory fields; they are only for your information. The fields show the total installed random access memory (RAM) and amounts allocated to base memory, extended memory, and other (high) memory. RAM is counted in kilobytes (KB: approximately one thousand bytes) and megabytes (MB: approximately one million bytes).

RAM is the computer's working memory, where the computer stores programs and data currently being used, so they are accessible to the CPU. Modern personal computers may contain up to 64 MB, 128 MB, or more.

- **Base Memory**
  Typically 640 KB. Also called conventional memory. The DOS operating system and conventional applications use this area.
- **Extended Memory**
  Above the 1-MB boundary. Early IBM personal computers could not use memory above 1 MB, but current PCs and their software can use extended memory.
- **Other Memory**
  Between 640 KB and 1 MB; often called High memory. DOS may load terminate-and-stay-resident (TSR) programs, such as device drivers, in this area, to free as much conventional memory as possible for applications. Lines in your CONFIG.SYS file that start with LOADHIGH load programs into high memory.

BIOS Features Setup

This screen contains industry-standard options additional to the core PC AT BIOS. This section describes all fields offered by Award Software in this screen. Your system board designer may omit or modify some fields.

Virus Warning

When *Enabled*, you receive a warning message if a program (specifically, a virus) attempts to write to the boot sector or the partition table of the hard disk drive. You should then run an anti-virus program. Keep

in mind that this feature protects only the boot sector, not the entire hard drive.

NOTE: Many disk diagnostic programs that access the boot sector table can trigger the virus warning message. If you plan to run such a program, we recommend that you first disable the virus warning.

CPU Internal Cache

Cache memory is a small area of additional memory that is much faster than conventional DRAM (system memory). When the CPU requests data, the system transfers the requested data from the main DRAM into cache memory, for even faster access by the CPU. (NOTE: This field is moot. 386-type CPUs do not contain internal cache.)

Quick Power On Self Test

Select *Enabled* to reduce the amount of time required to run the power-on self-test (POST). A quick POST skips certain steps. We recommend that you normally disable quick POST. Better to find a problem during POST than lose data during your work.

Boot Sequence

The original IBM PCs loaded the DOS operating system from drive A (floppy disk), so IBM PC-compatible systems are designed to search for an operating system first on drive A, and then on drive C (hard disk). However, modern computers usually load the operating system from the hard drive, and may even load it from a CD-ROM drive.

Swap Floppy Drive

This field is effective only in systems with two floppy drives. Selecting *Enabled* assigns physical drive B to logical drive A, and physical drive A to logical drive B.

Boot Up Floppy Seek

When *Enabled*, the BIOS tests (seeks) floppy drives to determine whether they have 40 or 80 tracks. Only 360-KB floppy drives have 40 tracks; drives with 720 KB, 1.2 MB, and 1.44 MB capacity all have 80 tracks. Because very few modern PCs have 40-track floppy drives, we recommend that you set this field to *Disabled* to save time.

Boot Up NumLock Status

Toggle between *On* and *Off* to control the state of the NumLock key when the system boots. When toggled *On*, the numeric keypad generates numbers instead of controlling cursor operations.

Boot Up System Speed

Select *High* to boot at the default CPU speed; select *Low* to boot at the speed of the AT bus. Some add-in peripherals or old software (such as old games) may require a slow CPU speed.

IDE HDD Block Mode

Block mode is also called block transfer, multiple commands, or multiple sector read/write. If your IDE hard drive supports block mode (most new drives do), select *Enabled* for automatic detection of the optimal number of block read/writes per sector the drive can support.

Gate A20 Option

Gate A20 refers to the way the system addresses memory above 1 MB (extended memory). When set to *Fast*, the system chipset controls Gate A20. When set to *Normal*, a pin in the keyboard controller controls Gate A20. Setting Gate A20 to *Fast* improves system speed, particularly with OS/2 and Windows.

Memory Parity Check

Parity is a testing feature built into some system memory chips (DRAM). Select *Enabled* only if your system DRAM contains parity. This field may not be present in systems without parity DRAM.

Selecting *Enabled* adds a parity check to the boot-up memory tests. If the BIOS detects a parity error, a message appears, describing the problem and, if possible, the location of the problem. The boot process then terminates and you must replace the faulty DRAM.

Selecting *Disabled* omits the memory parity check.

Typematic Rate Setting

When *Disabled*, the following two items (Typematic Rate and Typematic Delay) are irrelevant. Keystrokes repeat at a rate determined by the keyboard controller in your system.

When *Enabled*, you can select a typematic rate and typematic delay.

Typematic Rate (Chars/Sec)

When the typematic rate setting is enabled, you can select a typematic rate (the rate at which character repeats when you hold down a key) of 6, 8, 10,12, 15, 20, 24 or 30 characters per second.

Typematic Delay (Msec)

When the typematic rate setting is enabled, you can select a typematic delay (the delay before key strokes begin to repeat) of 250, 500, 750 or 1000 milliseconds.

Security Option

If you have set a password, select whether the password is required every time the *System* boots, or only when you enter *Setup*.

Shadow

Software that resides in a read-only memory (ROM) chip on a device is called *firmware*. The Award BIOS permits *shadowing* of firmware such as the system BIOS, video BIOS, and similar operating instructions that come with some expansion peripherals, such as, for example, a SCSI adaptor.

Shadowing copies firmware from ROM into system RAM, where the CPU can read it through the 16-bit or 32-bit DRAM bus. Firmware not shadowed must be read by the system through the 8-bit X-bus. Shadowing improves the performance of the system BIOS and similar ROM firmware for expansion peripherals, but it also reduces the amount of high memory (640 KB to 1 MB) available for loading device drivers, etc.

Enable shadowing into each section of memory separately. Many system designers hardwire shadowing of the system BIOS and eliminate a System BIOS Shadow option.

Video BIOS shadows into memory area C0000-C7FFF. The remaining areas shown on the BIOS Features Setup screen may be occupied by other expansion card firmware. If an expansion peripheral in your system contains ROM-based firmware, you need to know the address range the ROM occupies to shadow it into the correct area of RAM.

Chipset Features Setup

ADVANCED OPTIONS. The parameters in this screen are for system designers, service personnel, and technically competent users only. Do not reset these values unless you understand the consequences of your changes.

NOTE: This chapter describes all fields offered by Award Software in this screen. Your system board designer may omit or modify some fields.

DRAM refresh type

The refresh type *(RAS only* or *CAS before RAS*) depends on the specifications of the installed DRAMs. The system board designer must select the correct type. Do not change from the default setting.

Fast RAS precharge time

The RAS precharge time *(1.5T* or *2.5T)* depends on the specifications of the installed DRAMs. The system board designer must select the correct type. Do not change from the default setting.

Memory remap

You can remap the area of memory between the 640 KB application memory boundary and the 1 MB boundary up to the top of installed memory. This technique allows an additional 384 KB of RAM above 1 MB.

RAS active time insert wait

Select *Enabled* if the installed DRAM requires additional wait states. The system board designer must set this option according to the installed DRAM. Do not change from the default setting unless you are experiencing memory errors.

CAS precharge insert wait

Select *Enabled* if the installed DRAM requires additional wait states. The system board designer must set this option according to the installed DRAM. Do not change from the default setting unless you are experiencing memory errors.

Memory CAS read insert wait

Select *Enabled* if the installed DRAM requires additional wait states. The system board designer must set this option according to the installed DRAM. Do not change from the default setting unless you are experiencing memory errors.

Memory write insert wait

Select *Enabled* if the installed DRAM requires additional wait states. The system board designer must set this option according to the installed DRAM. Do not change from the default setting unless you are experiencing memory errors.

Memory miss read insert wait

Select *Enabled* if the installed DRAM requires additional wait states. The system board designer must set this option according to the installed DRAM. Do not change from the default setting unless you are experiencing memory errors.

AT-BUS Clock

You can set the speed of the AT bus in terms of a fraction of the CPU clock speed (PCLK2), or at the fixed speed of 7.16 MHz.

DRAM self refresh

When *Disabled*, DRAM is refreshed by IBM AT methodology, using a CPU cycles for each refresh. When self refresh is *Enabled*, the DRAM controller seeks the most opportune moment for a refresh, regardless of CPU cycles, with least disruption of system activity and least performance penalty. Self refresh is faster and more efficient, and it also allows the CPU to maintain the status of the DRAM even if the system goes into a power management "suspend" mode.

Slow Refresh

Select a longer period to refresh system memory if you are encountering data corruption because your DRAM is not fast enough to handle faster timing.

ISA memory high speed

Select *Enabled* if you have installed high-speed add-on RAM in an ISA expansion slot.

ISA I/O high speed

Select *Enabled* if you have installed a high-speed add-on I/O peripheral in an ISA expansion slot.

ISA write cycle insert wait

If you have add-on RAM in an ISA expansion slot, select *Enabled* to allow additional time for the slower throughput of the ISA bus.

On-chip I/O recovery

Select *Enabled* to allow extra preparation time between I/O cycles controlled by the M6117 chip.

16 Bit ISA insert wait

Your system quite possibly has much higher performance than some of your input/output (I/O) devices. This means that unless the system is instructed to allow more time, more wait states, for devices to respond, it might think the device has malfunctioned and stop its request for I/O. If all your I/O devices

are capable, then disabling this setting could result in greater throughput. Otherwise, data could be lost.

I/O recovery

Select *Enabled* to allow extra preparation time between I/O cycles controlled by add-on peripherals.

I/O recovery period

When I/O recovery (above) is *Enabled*, you can set the recovery period.

Power Management

NOTE: This chapter describes all fields offered by Award Software in this screen. Your system board designer may omit or modify some fields.

Power Management

This option allows you to select the type (or degree) of power saving for Doze, Standby, and Suspend modes. See the section *PM Timers* for a brief description of each mode.

This table describes each power management mode:

Max Saving	Maximum power savings. **Only Available for SL CPUs.** Inactivity period is 9 seconds in each mode.
User Define	Set each mode individually. Each inactivity period ranges from 9 seconds to 3 hours. Select time-out periods in the *PM Timers* section, following.
Min Saving	Minimum power savings. Inactivity period is 3 hours in each mode.

PM Control by APM

If Advanced Power Management (APM) is installed on your system, selecting *Yes* gives better power savings.

Video Off Option

Select the power-saving modes during which the monitor goes blank:

Always On	Monitor remains on during power-saving modes.
Suspend --> Off	Monitor blanked when system enters *Suspend* mode.
Susp,Stby --> Off	Monitor blanked when system enters *Suspend* mode.
All Modes --> Off	Monitor blanked when system enters any power saving mode.

Video Off Method

Determines the manner in which the monitor is blanked.

V/H SYNC+Blank	System turns off vertical and horizontal synchronization ports and writes blanks to the video buffer.
DPMS Support	Select this option if your monitor supports the Display Power Management Signaling (DPMS) standard of the Video Electronics Standards Association (VESA). Use the software supplied for your video subsystem to select video power management values.
Blank Screen	System only writes blanks to the video buffer.

PM Timers

The following modes are Green PC power saving functions. Doze, Standby, and Suspend modes are user-configurable only during User Defined Power Management mode.

Doze Mode

After the selected period of system inactivity, the CPU clock runs at slower speed while all other devices still operate at full speed.

Standby Mode

After the selected period of system inactivity, the fixed disk drive and the video shut off while all other devices still operate at full speed.

Suspend Mode

After the selected period of system inactivity, all devices except the CPU shut off.

PM Events

You may disable activity monitoring of some devices and interrupt requests so they do not wake up the system. The default wake-up event is keyboard activity.

When *Enabled*, any of the system peripheral devices or IRQs listed below wakes up the system.

```
DMA Request                    IRQ6 (Floppy Disk)
VGA Activity                   IRQ7 (LPT 1)
FDD (3F5)                      IRQ9 (IRQ2 Redir)
LPT (3BC, 378, 278)            IRQ10 (Reserved)
COM Port (3F8, 3E8)            IRQ11 (Reserved)
COM Port (2F8, 2E8)            IRQ12 (PS/2 Mouse)
IRQ3 (COM 2 )                  IRQ13 (Coprocessor)
IRQ4 (COM 1)                   IRQ14 (Hard Disk)
IRQ5 (LPT 2)                   IRQ15 (Reserved)
```

Password Setting

When you select this function, a message appears at the center of the screen:

ENTER PASSWORD:

Type the password, up to eight characters, and press Enter. Typing a password clears any previously entered password from CMOS memory.

Now the message changes:

CONFIRM PASSWORD:

Again, type the password and press Enter.

To abort the process at any time, press Esc.

In the *Security Option* item in the **BIOS Features** Setup screen, select *System* or *Setup*:

```
System    Enter a password each time the system boots and when
          ever you enter Setup.

Setup     Enter a password when ever you enter Setup.
```

NOTE: To clear the password, simply press Enter when asked to enter a password. Then the password function is disabled.

Return to Award Home Page

Return to BIOS Setup Documents

Proprietary Notice and Disclaimer

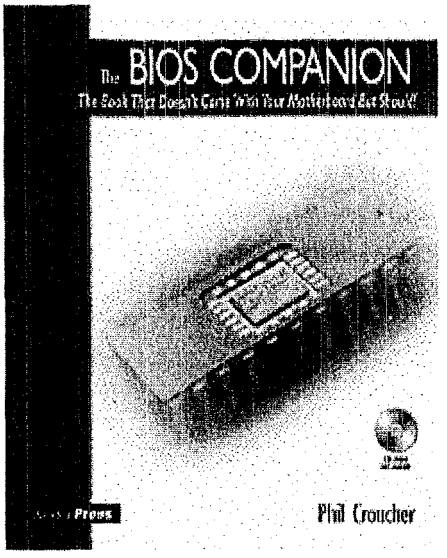
Document Revision 1.3; August 1996. Last updated 17 September 96.

6.2.5 Third-Party BIOS References

The following reference on BIOS software is useful.

The BIOS Companion
by Phil Croucher, Advice Press,
ISBN 1-889671-20-7
480 California Ave, Suite 104, Palo Alto, CA 94306, 1998. $50.

ADVICE*Press*



BIOS Companion

Phil Croucher

May 1999
1-889671-20-7

412 pages, $49.95, Includes CD-ROM

About the Author

How to Order

DESCRIPTION

A complete guide to the inner workings and settings of computer BIOS chips is now available -- The BIOS Companion (ADVICE Press, ISBN 1-889671-20-7 MSRP $49.95).

Configuring a PC for maximum performance is a tricky process, especially if you are unsure of the function of the many different BIOS setup options. This book is the complete guide to hundreds of standard CMOS settings, Advanced Chipset settings and PCI Bus settings found in modern and not-so-modern BIOS chipsets.

This is the only book of its type that is up-to-date and covers both current and past generations of BIOS hardware. The author, Phil Croucher, is a recognized expert in this area and enjoys a tremendous following among PC professionals and hobbyists alike.

Features

- The most comprehensive and up to date book on BIOS settings
- Comprehensive coverage of all BIOS manufacturers and variations.
- Standard and Advanced CMOS settings.
- Complete description of options for maximum performance and stability.
- Complete POST and Beep codes.
- Nasty Noises and POSTmortem Instruction Manual

- Covers Year 2000 issues.
- CD-ROM with several diagnostic BIOS tools and the Hard Disk Companion electronic edition.

Table of Contents

Other Related ADVICE Titles

Motherboard Companion
Hard Disk Companion

ADVICE Home | Product Index | Ordering | Customer Service | About ADVICE Press | Affiliated Companies

7.1 Data_ATT – Computational Block Software

The computational block software was written expressly for controlling the flow of information in this information-barrier-protected measurement system.

7.1.1 Overview Description of Computational Block Software

The DATA_ATT.EXE software monitors the four serial ports, which are connected to the four measurement computers. When measurement results are reported from all four measurement computers, the results are compared to unclassified threshold values. The results of all the attribute measurements and comparisons are transmitted on to the hardware data barrier in the form of PASS and FAIL indicating bits in two 8-bit digital I/O words.

The Computational Block software is written in the C-language using the Microsoft Visual C-Compiler. The obsolete version 1.52 was used because it is the last version, which compiles code for DOS. The visual features of the Microsoft Visual C-Compiler were not utilized for the textual output.

The application program DATA_ATT.EXE makes library calls to the WCSC library to handle communications with the measurement computers. It also makes calls to the standard Microsoft Visual C-Compiler libraries.

7.1.2 Source Code for Computational Block Software

The DATA_ATT.EXE application program was written specifically for this information barrier project. The source code was provided by Los Alamos National Laboratory (LANL).

7.1.2.1 Source Code for the DATA_ATT.H Header File

The DATA_ATT.H file is a header for the application program. This header defines items specific to the attribute measurement system (AMS).

```
/*****************************************************************/
/* File DATA_ATT.H                                               */
/* Header file for the Info Barrier project.                     */
/*                                                               */
/*****************************************************************/

// #defines for the attribute limits
#define    NMC_LIMIT                    500.00
#define    NMC_Alpha_LIMIT              0.50
#define    Pu900_PeakArea_LIMIT         1.00
#define    Pu600_LIMIT                  0.100
#define    Pu300_age_LIMIT              3
#define    Symmetry_Sigma_LIMIT         3
#define    Symmetry_Data_LIMIT          0.15

// #defines to set the DIO port constants
#define    BASE     0x300       // Base I/O address for Emerald board
#define    PORT0    BASE+0      // DIO Port 0 @ 300
#define    PORT1    BASE+1      // DIO Port 1 @ 301
#define    PORT2    BASE+2      // DIO Port 2 @ 302
#define    PORT3    BASE+3      // DIO Port 3 @ 303
#define    PORT4    BASE+4      // DIO Port 4 @ 304
#define    PORT5    BASE+5      // DIO Port 5 @ 305

#define    TRUE     1
#define    FALSE    0

// #define the values for each light
#define    Age_Red                0x04            // DIO Port4 bit 2
#define    Age_Green              0x08            // DIO Port4 bit 3
#define    Symmetry_Red           0x10            // DIO Port4 bit 4
#define    Symmetry_Green         0x20            // DIO Port4 bit 5
#define    Pu_Presence_Red        0x40            // DIO Port4 bit 6
#define    Pu_Presence_Green      0x80            // DIO Port4 bit 7

#define    Pu_Oxide_Red           0x01            // DIO Port5 bit 0
#define    Pu_Oxide_Green         0x02            // DIO Port5 bit 1
#define    Pu_Mass_Red            0x04            // DIO Port5 bit 2
#define    Pu_Mass_Green          0x08            // DIO Port5 bit 3
#define    Pu_Isotope_Red         0x10            // DIO Port5 bit 4
#define    Pu_Isotope_Green       0x20            // DIO Port5 bit 5
#define    System_Error           0x40            // DIO Port5 bit 6
#define    Measurement_Complete   0x80            // DIO Port5 bit 7

// End of header file DATA_ATT.H
```

7.1.2.2 Source Code for the DATA_ATT.C Program File

The following listing of DATA_ATT.C is the commented source code for the primary computational block application program, DATA_ATT.EXE. The DATA_ATT.C source code contains include statements to incorporate the previous header file, DATA_ATT.H, and other standard C-compiler library header files. This source code is compiled using the Microsoft Visual C version 1.52 compiler to produce the DATA_ATT.EXE executable file.

The DATA_ATT.EXE program monitors the serial ports for information from the measurement computers and converts the classified values into unclassified pass/fail outputs for each attribute.

```
//
/*************************************************************/
/* FMTT   Computational Block program                        */
/*                                                           */
/* This program will gather information from four            */
/*  computers: the Pu300/600,                                */
/*             the Pu900,                                    */
/*             the NMC,                                      */
/*         and the symmetry computer.                        */
/*                                                           */
/* Data will be compared to stored limits,                   */
/*  and red and green lights will be illuminated             */
/*  to indicate the pass/fail results.                       */
/*                                                           */
/* By Robert Landry and Connie Buenafe                       */
/*************************************************************/

/* Header Files for compiler library calls                   */
#include <stdio.h>        // Standard IO
#include <stdlib.h>       // Standard Library
#include <conio.h>        // Console IO
#include <signal.h>
#include <ctype.h>
#include <string.h>       // String Functions
#include <time.h>         // Time

/* Header file for WCSC COMM/DRV/LIB                         */
#include <comm.h>         // COM# communications routines

/* File containing #define constants                         */
#include "Data_Att.h"     // Defined limits
//
```

```
/* Function prototypes are listed here.                    */
int clean_exit(void);
void sleep( clock_t wait );
int board_config(void);
int port_config(void);
void start_signal(void);
void clear_lights(void);
void clock_ff(void);
void reset_Pu600_data(void);
void clear_Pu600_buffer(void);
void reset_Pu900_data(void);
void reset_NMC_data(void);
void reset_Symmetry_data(void);
void set_error(void);
void call_cancel(void);
void set_Pu600_Isotope_pass(void);
void set_Pu600_Isotope_fail(void);
void set_Pu300_age_pass(void);
void set_Pu300_age_fail(void);
void set_Pu_Presence_pass(void);
void set_Pu_Presence_fail(void);
void set_Oxide_pass(void);
void set_Oxide_fail(void);
void set_NMC_pass(void);
void set_NMC_fail(void);
void set_Symmetry_pass(void);
void set_Symmetry_fail(void);
void set_measurement_complete(void);
void clear_measurement_complete(void);
void get_serial_data(void);
void get_parallel_data(void);
//
```

```
/* Global variables are listed here.                              */
int  start_data;
unsigned char    datain;     /* Needed for the Emerald I/O board   */
unsigned char    dataout3;
unsigned char    dataout4;
unsigned char    dataout5;

int     Pu600_inc;
int     Pu900_inc;
int     Symmetry_inc;
int     NMC_inc;

int     exitflag;
int     errorflag;
int     port_1=0x0;        // Port number
int     port_2=0x1;        // Port number
int     port_3=0x2;        // Port number
int     port_4=0x3;        // Port number

int     Pu600_cnt;
char    Pu600_buf[2048];
float   Pu600_ratio;
float   Pu600_percent;
char    Pu600_in[2048];

float   Pu300_age;
float   Pu300_percent;
float   Pu300_iteration;
char    *Pu_Presence;

int     NMC_cnt;
char    NMC_buf[2048];
float   NMC_mass;
float   NMC_percent;
float   NMC_Alpha;
float   NMC_error;
float   NMC_scaler1;
float   NMC_scaler2;
char    NMC_in[2048];

int     Pu900_cnt;
char    Pu900_buf[2048];
float   Pu900_PeakArea;
float   Pu900_percent;
char    Pu900_in[2048];

int     Symmetry_cnt;
char    Symmetry_buf[2048];
float   Symmetry_Data;
float   Symmetry_Sigma;
char    Symmetry_in[2048];

float   total_mass;
//
```

```
/*****************************************************************/
/*                        ----- clearscr -----                     */
/*****************************************************************/
/* Description-                                                    */
/*****************************************************************/
static void clearscr()
{
     bio_scrollup(0,7,0,0,24,79);
     bio_setcursorxy(0,0,0);
}

/*****************************************************************/
/*                        ----- ctl_c -----                        */
/*****************************************************************/
/* Description-                                                    */
/*      Captures Control C to cleanup port on exit.                */
/*****************************************************************/
void ctl_c(int val)
{
     signal(SIGINT,SIG_IGN);
     exitflag = 1;
     signal(SIGINT,ctl_c);
}

/*****************************************************************/
/*                        ----- COM 1 -----                        */
/*****************************************************************/
/* Description-                                                    */
/*      Handles interrupts from the NMC serial port.               */
/*****************************************************************/

short FAR COM1_IO(short v, struct port_param __far *p)
{
     NMC_cnt = GetString(port_1, sizeof(NMC_buf), NMC_buf);
     return v;
}

short (FAR * CDECL COM1_interrupt_func)
        (short v, struct port_param __far *p) = COM1_IO;
//
```

```
/*****************************************************************/
/*                      ----- COM 2 -----                         */
/*****************************************************************/
/* Description-                                                  */
/*      Handles interrupts from the Pu300/600 serial port.       */
/*****************************************************************/

short FAR COM2_IO(short v, struct port_param __far *p)
{
     Pu600_cnt = GetString(port_2, sizeof(Pu600_buf), Pu600_buf);
     return v;
}

short (FAR * CDECL COM2_interrupt_func)
        (short v, struct port_param __far *p) = COM2_IO;

/*****************************************************************/
/*                      ----- COM 3 -----                         */
/*****************************************************************/
/* Description-                                                  */
/*      Handles interrupts from the Pu900 serial port.           */
/*****************************************************************/

short FAR COM3_IO(short v, struct port_param __far *p)
{
     Pu900_cnt = GetString(port_3, sizeof(Pu900_buf), Pu900_buf);
     return v;
}

short (FAR * CDECL COM3_interrupt_func)
        (short v, struct port_param __far *p) = COM3_IO;

/*****************************************************************/
/*                      ----- COM 4 -----                         */
/*****************************************************************/
/* Description-                                                  */
/*      Handles interrupts from the Symmetry serial port.        */
/*****************************************************************/

short FAR COM4_IO(short v, struct port_param __far *p)
{
     Symmetry_cnt = GetString(port_4, sizeof(Symmetry_buf), Symmetry_buf);
     return v;
}

short (FAR * CDECL COM4_interrupt_func)
        (short v, struct port_param __far *p) = COM4_IO;
//
```

```
/*****************************************************************/
/*                        ----- clean_exit -----                  */
/*****************************************************************/
/* Description-                                                  */
/*****************************************************************/
int clean_exit(void)
{
     int status=0;

     if ((status = UnInitializePort(port_1)) != RS232ERR_NONE)
        printf("Error #%d closing serial port %d\n",status,
           port_1+1);
     else
        printf("Serial Port %d closed successfully\n",
           port_1+1);

     if ((status = UnInitializePort(port_2)) != RS232ERR_NONE)
        printf("Error #%d closing serial port %d\n",status,
           port_2+1);
     else
        printf("Serial Port %d closed successfully\n",
           port_2+1);

     if ((status = UnInitializePort(port_3)) != RS232ERR_NONE)
        printf("Error #%d closing serial port %d\n",status,
           port_3+1);
     else
        printf("Serial Port %d closed successfully\n",
           port_3+1);

     if ((status = UnInitializePort(port_4)) != RS232ERR_NONE)
        printf("Error #%d closing serial port %d\n",status,
           port_4+1);
     else
        printf("Serial Port %d closed successfully\n",
           port_4+1);

     return status;
}
//
```

```
/*****************************************************************/
/*                        ----- Sleep Timer -----                 */
/*****************************************************************/
/* Description-                                                  */
/*      Introduces a wait of a given microsecond count.          */
/*****************************************************************/

/* Pauses for a specified number of microseconds.                */
void sleep( clock_t wait )
{
    clock_t goal;

    goal = wait + clock();
    while( goal > clock() )
        ;
}
//
```

```
/*****************************************************************/
/*                        ----- start_signal -----                 */
/*****************************************************************/
/* Description-                                                    */
/*                                                                 */
/* This is the routine that will test for a start                  */
/* signal from one of the test systems, the Pu300/600,             */
/* the Pu900, the Symmetry analyzer, or the NMC.                   */
/*                                                                 */
/* The signal will be received on the                              */
/* Emerald board input port, port 0.                               */
/*                                                                 */
/*****************************************************************/
/*                                                                 */
/* The values set are the global value for start_data.             */
/*      start_data returns 1 for Assay Start received.             */
/*      start_data returns 2 for Background received.              */
/*      start_data returns 3 for Gamma Calibration received.       */
/*      start_data returns 4 for Measure Control received.         */
/*      start_data returns 5 for Cancel Test received.             */
/*                                                                 */
/*****************************************************************/
void start_signal(void)
{
     datain = _inp(PORT0);

     if (datain&001)          /* Signal Received                   */
                              /* Note that data is inverted        */
     {
        start_data=1;         /* 1 = Assay Start                   */
     }
     else if (datain&002)     /* Signal Received                   */
     {
        start_data=2;         /* 2 = Background                    */
     }
     else if (datain&004)     /* Signal Received                   */
     {
        start_data=3;         /* 3 = Gamma Calibration             */
     }
     else if (datain&010)     /* Signal Received                   */
     {
        start_data=4;         /* 4 = Measure Control               */
     }
     else if (datain&020)     /* Signal Received                   */
     {
        start_data=5;         /* 5 = Cancel all tests              */
     }
     return;
}
//
```

```
/*********************************************************************/
/*                        ----- clear_lights -----                    */
/*********************************************************************/
/* Description-                                                      */
/*                                                                   */
/* This routine will set the variable to ones - to clear             */
/* the lights and initialize the variable for data results.          */
/*                                                                   */
/* This is the routine that will clear all the                       */
/* indicator lights on the display panel.                            */
/*                                                                   */
/*********************************************************************/
void clear_lights(void)
{
     dataout4=0xff;
     dataout5=0xff;

     _outp(PORT4,dataout4);
     _outp(PORT5,dataout5);

/* Clock data into the Flip-Flops                                    */
     clock_ff();

     return;
}
//
```

```
/*********************************************************************/
/*                  ----- Clock Flip-Flops -----                      */
/*********************************************************************/
/* Description-                                                      */
/*                                                                   */
/* In this routine, the data will be output to the I/O port          */
/* and the measurement complete data line will be                    */
/* toggled to clock the data from the computational block            */
/* to the information barrier flip flops.                           */
/*                                                                   */
/*********************************************************************/
void clock_ff(void)
{
     _outp(PORT3,0xff);          /* Initialize the ff                */
     sleep(50);                  /* Wait for .05 sec                 */

     _outp(PORT4,dataout4);      /* Output data for lights           */
     sleep(5);                   /* Wait for .005 sec                */
     _outp(PORT5,dataout5);
     sleep(5);                   /* Wait for .005 sec                */

     _outp(PORT3,0xfe);
     sleep(50);                  /* Wait for .05 sec                 */

     _outp(PORT3,0xff);
     sleep(50);                  /* Wait for .05 sec                 */

     return;
}
//
```

```
/*****************************************************************/
/*                 ----- Reset the data fields -----              */
/*****************************************************************/
/* Description-                                                  */
/*                                                               */
/* In these routines, all the data fields used to                */
/* store information received from the Pu600 and                 */
/* the NMC will be reset.                                        */
/*                                                               */
/*****************************************************************/
void reset_Pu600_data(void)
{
     memset(Pu600_buf, 0, 2048);
     memset(Pu600_in, 0, 2048);
     Pu600_cnt = 0;
     Pu600_inc = 0;
     Pu600_ratio = 0;
     Pu600_percent = 0;
     Pu300_age = 0;
     Pu300_percent = 0;
     Pu300_iteration = 0;
     Pu_Presence = "No";
     return;
}

/*****************************************************************/
void clear_Pu600_buffer(void)
{
     memset(Pu600_buf, 0, 2048);
     memset(Pu600_in, 0, 2048);
     Pu600_cnt = 0;
     Pu600_inc = 0;
     return;
}

/*****************************************************************/
void reset_NMC_data(void)
{
     memset(NMC_buf, 0, 2048);
     memset(NMC_in, 0, 2048);
     NMC_cnt = 0;
     NMC_inc = 0;
     NMC_mass = 0;
     NMC_percent = 0;
     return;
}
//
```

```
/*****************************************************************/
void reset_Pu900_data(void)
{
     memset(Pu900_buf, 0, 2048);
     memset(Pu900_in, 0, 2048);
     Pu900_cnt = 0;
     Pu900_inc = 0;
     Pu900_PeakArea = 0;
     Pu900_percent = 0;
     return;
}

/*****************************************************************/
void reset_Symmetry_data(void)
{
     memset(Symmetry_buf, 0, 2048);
     memset(Symmetry_in, 0, 2048);
     Symmetry_cnt = 0;
     Symmetry_inc = 0;
     Symmetry_Data = 0;
     Symmetry_Sigma = 0;
     return;
}
//
```

```
/*****************************************************************/
/*                        ----- set_error -----                   */
/*****************************************************************/
/* Description-                                                  */
/*                                                               */
/* These are the routines that will set the                      */
/* indicator lights on the display panel showing                 */
/* an error has occurred.  All other lights will                 */
/* be cleared.                                                   */
/*                                                               */
/*****************************************************************/
void set_error(void)
{
     clear_lights();

     dataout5=dataout5 & System_Error;  /* Set Error light on    */
     _outp(PORT5,dataout5);
     sleep(5);                          /* Wait for .005 sec     */
     return;
}

/*****************************************************************/
/*                        ----- call_cancel -----                 */
/*****************************************************************/
/* Description-                                                  */
/*                                                               */
/* This routine is called anytime the cancel button              */
/* is pressed.  It resets all the variables, sets                */
/* the measurement complete light, and clocks the                */
/* flip-flops.                                                   */
/*                                                               */
/* This routine is set as a function to consolidate              */
/* the many times this sequence is executed.                     */
/*                                                               */
/*****************************************************************/
void call_cancel(void)
{
     printf("Received the Cancel signal.\n\n");
          /* NOTE - the following may not be needed as the      */
          /*        cancel value (start_data=5) will be        */
          /*        carried through after the break.           */
     reset_Pu600_data();
     reset_Pu900_data();
     reset_NMC_data();
     reset_Symmetry_data();
     set_measurement_complete();
     clock_ff();
}
//
```

```
/*****************************************************************/
/*      ----- Pu600 Isotope Passes or Fails -----                 */
/*****************************************************************/
/* Description-                                                  */
/*                                                               */
/* These are the routines that will set the                      */
/* indicator lights on the display panel showing                 */
/* a pass or a fail for the Isotope test.                        */
/*                                                               */
/*****************************************************************/
void set_Pu600_Isotope_pass(void)
{
     dataout5=dataout5 & Pu_Isotope_Green;
     return;
}

/*****************************************************************/
void set_Pu600_Isotope_fail(void)
{
     dataout5=dataout5 & Pu_Isotope_Red;
     return;
}

/*****************************************************************/
/*         ----- Pu300 Age Passes or Fails -----                 */
/*****************************************************************/
/* Description-                                                  */
/*                                                               */
/* These are the routines that will set the                      */
/* indicator lights on the display panel showing                 */
/* a pass or a fail for the Age test.                            */
/*                                                               */
/*****************************************************************/
void set_Pu300_age_pass(void)
{
     dataout4=dataout4 & Age_Green;
     return;
}

/*****************************************************************/
void set_Pu300_age_fail(void)
{
     dataout4=dataout4 & Age_Red;
     return;
}
//
```

```
/*****************************************************************/
/*         ----- Pu Presence Passes or Fails -----                */
/*****************************************************************/
/* Description-                                                  */
/*                                                               */
/* These are the routines that will set the                      */
/* indicator lights on the display panel showing                 */
/* a pass or a fail for the Pu Presence test.                    */
/*                                                               */
/*****************************************************************/
void set_Pu_Presence_pass(void)
{
     dataout4=dataout4 & Pu_Presence_Green;
     return;
}
/*****************************************************************/
void set_Pu_Presence_fail(void)
{
     dataout4=dataout4 & Pu_Presence_Red;
     return;
}

/*****************************************************************/
/*         ----- Pu Oxide Passes or Fails -----                   */
/*****************************************************************/
/* Description-                                                  */
/*                                                               */
/* These are the routines that will set the                      */
/* indicator lights on the display panel showing                 */
/* a pass or a fail for the Pu Oxide test.                       */
/*                                                               */
/*****************************************************************/
void set_Oxide_pass(void)
{
     dataout5=dataout5 & Pu_Oxide_Green;
     return;
}
/*****************************************************************/
void set_Oxide_fail(void)
{
     dataout5=dataout5 & Pu_Oxide_Red;
     return;
}
//
```

```
/*****************************************************************/
/*                        ----- NMC Passes or Fails -----          */
/*****************************************************************/
/* Description-                                                  */
/*                                                               */
/* These are the routines that will set the                      */
/* indicator lights on the display panel showing                 */
/* a pass or fail for the NMC test.                              */
/*                                                               */
/*****************************************************************/
void set_NMC_pass(void)
{
     dataout5=dataout5 & Pu_Mass_Green;
     return;
}
/*****************************************************************/
void set_NMC_fail(void)
{
     dataout5=dataout5 & Pu_Mass_Red;
     return;
}

/*****************************************************************/
/*         ----- Symmetry Passes or Fails -----                  */
/*****************************************************************/
/* Description-                                                  */
/*                                                               */
/* These are the routines that will set the                      */
/* indicator lights on the display panel showing                 */
/* a pass or a fail for the Summetry test.                       */
/*                                                               */
/*****************************************************************/
void set_Symmetry_pass(void)
{
     dataout4=dataout4 & Symmetry_Green;
     return;
}
/*****************************************************************/
void set_Symmetry_fail(void)
{
     dataout4=dataout4 & Symmetry_Red;
     return;
}
//
```

```
/*****************************************************************/
/*                ----- measurement complete -----                */
/*****************************************************************/
/* Description-                                                   */
/*                                                                */
/* These are the routines that will set or clear the              */
/* indicator light on the display panel showing                   */
/* the test process has completed and data has been               */
/* received.                                                      */
/*                                                                */
/*****************************************************************/
void set_measurement_complete(void)
{
     dataout5=dataout5 & Measurement_Complete;
     _outp(PORT5,dataout5);
     sleep(5);                  /* Wait for .005 sec             */
     return;
}
/*****************************************************************/
void clear_measurement_complete(void)
{
     dataout5=dataout5 | Measurement_Complete;
     _outp(PORT5,dataout5);
     sleep(5);                  /* Wait for .005 sec             */
     return;
}
//
```

```
/*****************************************************************/
/*****************************************************************/
/*                        *****   Main Program   *****             */
/*****************************************************************/
/*****************************************************************/
void main(int argc,char *argv[])
{
     int   status;
          /* Initialize the Serial Ports on the main board.      */
     int   ch;
     int   subport, baud, parity, len, stopbit, protocol;
     unsigned short    address_2, irq_2;
     unsigned short    address_3, irq_3;
     unsigned short    address_4, irq_4;
     unsigned short    address, irq, cardtype, cardseg;
     unsigned short    inlen, outlen, flag;

     /*************************************************************/
     /* Default serial port #1 parameters                         */
     /*************************************************************/
     subport = 0;
     address = 0x100;              // Port address 100 hex
     irq = 4;                      // Interrupt Request Level #4
     cardtype = CARD_NORMAL;
     cardseg = 0x0;
     inlen = 2048;                 // Input buffer length
     outlen = 1024;                // Output buffer length
     flag = 0x0;
     baud = BAUD9600;              // Baud rate
     parity = PARITY_NONE;         // No parity used
     len = LENGTH_8;               // 8-bit character length used
     stopbit = STOPBIT_1;          // 1 stop bit used
     protocol = PROT_NONNON;

     /*************************************************************/
     /* Default serial port 2 parameters                          */
     /*************************************************************/
     address_2 = 0x108;            // Port address 108 hex
     irq_2 = 3;                    // Interrupt Request Level #3

     /*************************************************************/
     /* Default serial port 3 parameters                          */
     /*************************************************************/
     address_3 = 0x110;            // Port address 110 hex
     irq_3 = 11;                   // Interrupt Request Level #11

     /*************************************************************/
     /* Default serial port 4 parameters                          */
     /*************************************************************/
     address_4 = 0x118;            // Port address 118 hex
     irq_4 = 10;                   // Interrupt Request Level #10
//
```

```
/*************************************************************/
printf("Barrier %s\n",commdrv_version);

/*************************************************************/
/* Trap control C to cleanup port on exit.                   */
/*************************************************************/
signal(SIGINT,ctl_c);

/*************************************************************/
/* Initialize serial port 1 for NMC                          */
/*************************************************************/
status = InitializePort(port_1, subport, address, irq, cardtype,
      cardseg, inlen, outlen, flag);

/* Set port characteristics                                  */
if (status == RS232ERR_NONE)
   status = SetPortCharacteristics(port_1, baud, parity, len,
            stopbit, protocol);
/* Set timer resolution to 1/1000 sec                        */
if (status == RS232ERR_NONE)
   status = CdrvSetTimerResolution(port_1, 1);
/* Set timeout value to 1/10 sec                             */
if (status == RS232ERR_NONE)
   status = SetTimeout(port_1, 100);
/* Set to interrupt when receive char = '\r'                 */
if (status == RS232ERR_NONE)
   status = ser_rs232_misc_func(port_1, SETRCVCHR, '\r');
/* Set interrupt function                                    */
if (status == RS232ERR_NONE)
   status = ser_rs232_set_intfunc(port_1, &COM1_interrupt_func,
            INTFUNC_RCVCHR);
/* Check for errors                                          */
if (status == RS232ERR_NONE)
   printf("Serial Port %d initialized successfully\n", port_1+1);
else {
   printf("Error #%d setting serial port %d\n", status, port_1+1);
   ch = getch();
   exit(1);
}
//
```

```
/*************************************************************/
/* Initialize serial port 2 for Pu300/600                    */
/*************************************************************/
status = InitializePort(port_2, subport, address_2, irq_2,
         cardtype, cardseg, inlen, outlen, flag);
/* Set port characteristics                                  */
if (status == RS232ERR_NONE)
   status = SetPortCharacteristics(port_2, baud, parity, len,
            stopbit, protocol);
/* Set timer resolution to 1/1000 sec                        */
if (status == RS232ERR_NONE)
   status = CdrvSetTimerResolution(port_2, 1);
/* Set timeout value to 1/10 sec                             */
if (status == RS232ERR_NONE)
   status = SetTimeout(port_2, 100);
/* Set to interrupt when receive char = '\r'                 */
if (status == RS232ERR_NONE)
   status = ser_rs232_misc_func(port_2, SETRCVCHR, '\r');
/* Set interrupt function                                    */
if (status == RS232ERR_NONE)
   status = ser_rs232_set_intfunc(port_2, &COM2_interrupt_func,
            INTFUNC_RCVCHR);

/* Check for errors                                          */
if (status == RS232ERR_NONE)
   printf("Serial Port %d initialized successfully\n", port_2+1);
else {
   printf("Error #%d setting serial port %d\n", status, port_2+1);
   ch = getch();
   exit(1);
}
//
```

```
/*************************************************************/
/* Initialize serial port 3 for Pu900                        */
/*************************************************************/
status = InitializePort(port_3, subport, address_3, irq_3,
         cardtype, cardseg, inlen, outlen, flag);
/* Set port characteristics                                  */
if (status == RS232ERR_NONE)
   status = SetPortCharacteristics(port_3, baud, parity, len,
            stopbit, protocol);
/* Set timer resolution to 1/1000 sec                        */
if (status == RS232ERR_NONE)
   status = CdrvSetTimerResolution(port_3, 1);
/* Set timeout value to 1/10 sec                             */
if (status == RS232ERR_NONE)
   status = SetTimeout(port_3, 100);
/* Set to interrupt when receive char = '\r'                 */
if (status == RS232ERR_NONE)
   status = ser_rs232_misc_func(port_3, SETRCVCHR, '\r');
/* Set interrupt function                                    */
if (status == RS232ERR_NONE)
   status = ser_rs232_set_intfunc(port_3, &COM3_interrupt_func,
            INTFUNC_RCVCHR);
/* Check for errors                                          */
if (status == RS232ERR_NONE)
   printf("Serial Port %d initialized successfully\n", port_3+1);
else {
   printf("Error #%d setting serial port %d\n", status, port_3+1);
   ch = getch();
   exit(1);
}
//
```

```
/*************************************************************/
/* Initialize serial port 4 for the Symmetry analyzer        */
/*************************************************************/
status = InitializePort(port_4, subport, address_4, irq_4,
   cardtype, cardseg, inlen, outlen, flag);
/* Set port characteristics                                  */
if (status == RS232ERR_NONE)
   status = SetPortCharacteristics(port_4, baud, parity, len,
            stopbit, protocol);
/* Set timer resolution to 1/1000 sec                        */
if (status == RS232ERR_NONE)
   status = CdrvSetTimerResolution(port_4, 1);
/* Set timeout value to 1/10 sec                             */
if (status == RS232ERR_NONE)
   status = SetTimeout(port_4, 100);
/* Set to interrupt when receive char = '\r'                 */
if (status == RS232ERR_NONE)
   status = ser_rs232_misc_func(port_4, SETRCVCHR, '\r');
/* Set interrupt function                                    */
if (status == RS232ERR_NONE)
   status = ser_rs232_set_intfunc(port_4, &COM4_interrupt_func,
            INTFUNC_RCVCHR);
/* Check for errors                                          */
if (status == RS232ERR_NONE)
   printf("Serial Port %d initialized successfully\n", port_4+1);
else {
   printf("Error #%d setting serial port %d\n", status, port_4+1);
   ch = getch();
   exit(1);
}

/*************************************************************/
/* Set the exit flag for a successful COM port start         */
/*************************************************************/

exitflag = 0;
//
```

```
/*************************************************************/
/* Start of main program here.                               */
/*************************************************************/

clear_lights();

reset_Pu600_data();
reset_Pu900_data();
reset_NMC_data();
reset_Symmetry_data();

/* Loop forever                                               */
while(1)
{
   if (exitflag)
   {
      /* Clean up for normal exit                             */
      clean_exit();
      exit(0);
   }

   /* Test for the "Start" signal                             */
   printf("Waiting for input signal.....\n\n");
   start_data=0;
   while (start_data==0)
   {
      start_signal();
   }

   /* Clear all data output lights                            */
   clear_lights();

   /* Reset all data variables                                */
   reset_Pu600_data();
   reset_Pu900_data();
   reset_NMC_data();
   reset_Symmetry_data();

   errorflag = FALSE;
//
```

```
switch (start_data)     /* start_data is set in start_signal() */
{
                                    // *********************************
                                    // ** Case 1 = Assay All sensors **
                                    // *********************************
   case 1:
      /* Assay test on all analyzers starts here.        */
      printf("Received Assay Start signal.\n\n");

      while ((Pu600_inc<3) || (Pu900_inc==0) ||
            (Symmetry_inc==0) || (NMC_inc==0))
      {
         clear_lights();

         /*  Test for cancel switch press                */
         start_signal();
         if (start_data == 5)
         {
            call_cancel();
            break;
         }
//
```

```
                /*************************************************/
                /*  Test for reception of the Pu600 signal       */
                /*************************************************/
                if (Pu600_cnt)
                {
                   printf("%d characters received from Pu600-->%s\n\n",
                           Pu600_cnt, Pu600_buf);

                   Pu600_cnt = 0;
                   Pu600_inc++;

                   /* Test which data string was received from Pu600 */
                   status=sscanf(Pu600_buf,"%s",Pu600_in);
                                 // ************************
                                 // * Process Pu600 Signal *
                                 // ************************
                   if (strcmp(Pu600_in,"PU600:")==0)
                   {
                      status=sscanf(Pu600_buf,"%*s %f, %f",
                            &Pu600_ratio,&Pu600_percent);
                      if (status == 0)
                      {
                         sscanf(Pu600_buf,"%*s %s", Pu600_in);
                         if ((strcmp(Pu600_in,"Physics")==0) ||
                            (strcmp(Pu600_in,"Acquisition")==0))
                         {
                            printf("Pu600 reports: %s\n\n",Pu600_buf);
                            errorflag = TRUE;
                            Pu600_inc = 3;
//                          break;
                         }
                         else
                         {
                            printf("Unknown Error in serial data from Pu600
                                  analyzer\n\n");
                            errorflag = TRUE;
                            Pu600_inc = 3;
//                          break;
                         }
                      }
                      else             /* status != 0      */
                      {
                         /* Print the data as stored in the variables */
                         printf("Pu600 ratio is     %lf  ",Pu600_ratio);
                         printf(" percent is   %lf\n\n",Pu600_percent);
                      }
                   }
//
```

```
                                    // ************************
                                    // * Process Pu300 Signal *
                                    // ************************
                        else if (strcmp(Pu600_in,"PU300:")==0)
                        {
                           status=sscanf(Pu600_buf,"%*s %f, %f, %f",
                                    &Pu300_age,&Pu300_percent,
                              &Pu300_iteration);
                           if (status == 0)
                           {
                              sscanf(Pu600_buf,"%*s %s", Pu600_in);
                              if ((strcmp(Pu600_in,"Physics")==0) ||
                                 (strcmp(Pu600_in,"Acquisition")==0))
                              {
                                 printf("Pu300 reports: %s\n\n",Pu600_buf);
                                 Pu600_inc = 3;
                                 errorflag = TRUE;
//                               break;
                              }
                              else
                              {
                                 printf("Unknown Error in serial data from Pu300
                                       analyzer\n\n");
                                 Pu600_inc = 3;
                                 errorflag = TRUE;
//                               break;
                              }
                           }
                           else /* status != 0 */
                           {
                              /* Print the data as stored in the variables */
                              printf("Pu300 age is    %lf  ",Pu300_age);
                              printf(" percent is   %lf  ",Pu300_percent);
                              printf(" iteration is %lf\n\n",Pu300_iteration);
                           }
                        }
//
```

```
                                    // ****************************
                                    // * Process PuPresence Signal *
                                    // ****************************
                        else if (strcmp(Pu600_in,"Presence:")==0)
                        {
                           status=sscanf(Pu600_buf,"%*s %s",Pu_Presence);
                           if (strstr(Pu600_buf, "Error") != NULL)
                           {
                              sscanf(Pu600_buf,"%*s %s", Pu600_in);
                              if ((strcmp(Pu600_in,"Physics")==0))
                              {
                                 printf("Pu Presence analyzer reports:
                                       %s\n\n",Pu600_buf);
                                 Pu600_inc = 3;
                                 errorflag = TRUE;
//                               break;
                              }
                              else
                              {
                                 printf("Unknown Error in serial data from Pu
                                       Presence analyzer\n\n");
                                 Pu600_inc = 3;
                                 errorflag = TRUE;
//                               break;
                              }
                           }
                           else     /* status != 0              */
                           {
                              /* Print the data as stored in the variables */
                              printf("Pu Presence is     %s\n\n",Pu_Presence);
                           }
                        }
                     }
//
```

```
                /*************************************************/
                /*  Test for reception of the Pu900 signal       */
                /*************************************************/
                if (Pu900_cnt)
                {
                   printf("%d characters received from Pu900-->%s\n\n",
                      Pu900_cnt, Pu900_buf);

                   Pu900_cnt = 0;
                   Pu900_inc++;

                   status=sscanf(Pu900_buf,"%*s %f, %f",
                               &Pu900_PeakArea,&Pu900_percent);
                   if (status == 0)
                   {
                      sscanf(Pu900_buf,"%*s %s", Pu900_in);
                      if ((strcmp(Pu900_in,"Physics")==0) ||
                         (strcmp(Pu900_in,"Acquisition")==0))
                      {
                         printf("Pu900 reports: %s\n\n",Pu900_buf);
                         errorflag = TRUE;
//                       break;
                      }
                      else
                      {
                         printf("Unknown Error in serial data from Pu900
                                 analyzer\n\n");
                         errorflag = TRUE;
//                       break;
                      }
                   }
                   else /* status != 0 */
                   {
                      /* Print the data as stored in the variables */
                      printf("Pu900 area is     %lf  ",Pu900_PeakArea);
                      printf(" percent is   %lf\n\n",Pu900_percent);
                   }
                }
//
```

```
                /*************************************************/
                /*  Test for receprion of the NMC signal         */
                /*************************************************/
                if (NMC_cnt)
                {
                   printf("%d characters received from NMC-->%s\n\n",
                      NMC_cnt, NMC_buf);

                   NMC_cnt = 0;
                   NMC_inc++;

                   status=sscanf(NMC_buf,"%*s %f, %f, %*s %f, %f",
                        &NMC_Alpha,&NMC_error,&NMC_mass,&NMC_percent);
                   if (status == 0)
                   {
                      if ((strcmp(NMC_buf,"Shift register error")==0) ||
                            (strcmp(NMC_buf,"Illegal switch state")==0))
                      {
                         printf("NMC reports: %s\n\n",NMC_buf);
                         errorflag = TRUE;
//                       break;
                      }
                      else
                      {
                         printf("Unknown Error in serial data from NMC
                                analyzer\n\n");
                         errorflag = TRUE;
//                       break;
                      }
                   }
                   else     /* status != 0                  */
                   {
                      /* Print the data as stored in the variables */
                      printf("NMC Alpha is    %lf\n",NMC_Alpha);
                      printf("NMC mass is     %lf\n",NMC_mass);
                      printf("NMC percent is   %lf\n\n",NMC_percent);
                   }
                }
//
```

```
                /*************************************************/
                /*  Test for reception of the Symmetry signal    */
                /*************************************************/
                if (Symmetry_cnt)
                {
                   printf("%d characters received from Symmetry-->%s\n\n",
                      Symmetry_cnt, Symmetry_buf);

                   Symmetry_cnt = 0;
                   Symmetry_inc++;

                   status = sscanf(Symmetry_buf,"%*s %f, %f",
                              &Symmetry_Data, &Symmetry_Sigma);
                   if (status == 0)
                   {
                      sscanf(Symmetry_buf,"%*s %s", Symmetry_in);
                      if ((strcmp(Symmetry_in,"Physics")==0) ||
                         (strcmp(Symmetry_in,"Acquisition")==0))
                      {
                         printf("Symmetry analyzer reports:
                                  %s\n\n",Symmetry_buf);
                         errorflag = TRUE;
//                       break;
                      }
                      else
                      {
                         printf("Unknown Error in serial data from Symmetry
                                  analyzer\n\n");
                         errorflag = TRUE;
//                       break;
                      }
                   }
                   else     /* status != 0 */
                   {
                      /* Print the data as stored in the variables */
                      printf("Symmetry difference is  %lf\n",Symmetry_Data);
                      printf("Symmetry Sigma is
                                  %lf\n\n\n",Symmetry_Sigma);
                   }
                }
             }
//
```

```
/*****************************************************************/
/*****************************************************************/
/*      Start comparing received values to limits.               */
/*****************************************************************/
/*****************************************************************/
            if (!errorflag && (start_data != 5))
            {
               /* Test for good Isotopic ratio                   */
               if (Pu600_ratio==0)
               {
                  printf("\nFailure! Pu600 returned an error.\n\n");
                  errorflag = TRUE;
                  break;
               }
               else if (Pu600_ratio < Pu600_LIMIT)
               {
                  set_Pu600_Isotope_pass();
                  printf("***** Passes the Pu600 Isotopic test. *****\n");
               }
               else
               {
                  set_Pu600_Isotope_fail();
                  printf("***** Fails the Pu600 Isotopic test. *****\n");
               }

               /* Test for good Age                              */
               if (Pu300_age > Pu300_age_LIMIT)
               {
                  set_Pu300_age_pass();
                  printf("***** Passes the Pu300 Age test.  *****\n");
               }
               else
               {
                  set_Pu300_age_fail();
                  printf("***** Fails the Pu300 Age test.  *****\n");
               }

               /*  Test for Pu Presence                          */
               if (strcmp(Pu_Presence,"Yes")==0)
               {
                  set_Pu_Presence_pass();
                  printf("***** Passes the Pu Presence test.  *****\n");
               }
               else if (strcmp(Pu_Presence,"No")==0)
               {
                  set_Pu_Presence_fail();
                  printf("***** Fails the Pu Presence test.  *****\n");
               }
               else
               {
                  printf("*****  Pu300 Physics Error  *****\n");
                  errorflag = TRUE;
               }
```

```
        /* Test for presence of Oxide in Pu                          */
        /* This test uses data from the NMC and the Pu900            */
        if ((Pu900_PeakArea > Pu900_PeakArea_LIMIT) &&
            (NMC_Alpha > NMC_Alpha_LIMIT))
        {
           set_Oxide_fail();
           printf("***** Fails the Oxide test *****\n");
        }
        else
        {
           set_Oxide_pass();
           printf("***** Passes the Oxide test *****\n");
        }

        /* Test for good Total Mass                                  */
        total_mass = NMC_mass * (1 + (1/Pu600_ratio));
        printf("Calculated total mass = %lf\n",total_mass);
        if (total_mass > NMC_LIMIT)
        {
           set_NMC_pass();
           printf("***** Passes the Multiplicity test. *****\n");
        }
        else
        {
           set_NMC_fail();
           printf("***** Fails the Multiplicity test. *****\n");
        }

        /*    Test for Symmetry                                       */
        if ((Symmetry_Data > Symmetry_Data_LIMIT) &&
           (Symmetry_Sigma > Symmetry_Sigma_LIMIT))
        {
           set_Symmetry_fail();
           printf("***** Fails the Symmetry test.  *****\n");
        }
        else
        {
           set_Symmetry_pass();
           printf("***** Passes the Symmetry test.  *****\n");
        }
     }
     /*  Clock information through the flip flops                    */
     if (errorflag)
        set_error();
     set_measurement_complete();
     clock_ff();
     break;
                        /* End of Case 1  */
//
```

```
                        // ************************************
                        // ** Case 2 = Background All sensors **
                        // ************************************
case 2:
   printf("Received Background start signal.\n");
   reset_Pu600_data();
   reset_Pu900_data();
   reset_NMC_data();
   reset_Symmetry_data();
                              // ********************************
                              // * Process Symmetry bkg Signal *
                              // ********************************
   /* Start of Symmetry background test          */
   while (Symmetry_cnt == 0) /* loop until SYM data is available */
   {
      start_signal();
      if (start_data == 5)
      {
         call_cancel();
         break;
      }
   }
   if (start_data != 5)
   {
      printf("%d characters received from Symmetry-->%s\n",
               Symmetry_cnt, Symmetry_buf);

      if (strcmp(Symmetry_buf,"Symmetry: Background Error")==0)
      {
         printf("Fails the Symmetry Background test.\n\n\n");
         errorflag = TRUE;
      }
   }
   /* End of Symmetry background test               */

//
```

```
                                    // ****************************
                                    // * Process Pu600 bkg Signal *
                                    // ****************************
/* Start of Pu300/600 background test                          */
/*   First loop is for Pu600                                   */
while (Pu600_cnt == 0) /* Loop until Pu600 data is available */
{
   start_signal();
   if (start_data == 5)
   {
      call_cancel();
      break;
   }
}
if (start_data != 5)
{
   printf("%d characters received from Pu600-->%s\n",
            Pu600_cnt, Pu600_buf);

   if ((strcmp(Pu600_buf,"PU600: Background Error")==0) ||
      (strcmp(Pu600_buf,"PU300: Background Error")==0))
   {
      printf("Fails the Pu300/600 Background test.\n\n\n");
      errorflag = TRUE;
   }
}
Pu600_cnt = 0;                  /* Reset Pu600 count to wait for   */
                                /* Pu300 information               */

//
```

```
                                    // **************************
                                    // * Process MNC bkg Signal *
                                    // **************************
/* Start of NMC background test                                    */
while (NMC_cnt == 0)      /* loop until NMC data is available */
{
   start_signal();
   if (start_data == 5)
   {
      call_cancel();
      break;
   }
}
if (start_data != 5)
{
   printf("%d characters received from NMC-->%s\n",NMC_cnt,
         NMC_buf);

   if (strcmp(NMC_buf,"FAIL")==0)
   {
      printf("Fails the NMC Background test.\n\n\n");
      errorflag = TRUE;
   }
}
/* End of NMC background test                              */
//
```

```
                                    // ****************************
                                    // * Process Pu900 bkg Signal *
                                    // ****************************
/* Start of Pu900 background test                          */
while (Pu900_cnt == 0) /* loop until Pu900 data is available */
{
   start_signal();
   if (start_data == 5)
   {
      call_cancel();
      break;
   }
}
if (start_data != 5)
{
   printf("%d characters received from Pu900-->%s\n",
            Pu900_cnt, Pu900_buf);

   if (strcmp(Pu900_buf,"PU900: Background Error")==0)
   {
      printf("Fails the Pu900 Background test.\n\n\n");
      errorflag = TRUE;
   }
}
/* End of Pu900 background test                            */
//
```

```
                                    // ****************************
                                    // * Process Pu300 bkg Signal *
                                    // ****************************
/*   Second loop is for Pu300                                    */
while (Pu600_cnt == 0)  /* loop until Pu300 data is available */
{
   start_signal();
   if (start_data == 5)
   {
      call_cancel();
      break;
   }
}
if (start_data != 5)
{
   printf("%d characters received from Pu300-->%s\n\n",
            Pu600_cnt, Pu600_buf);

   if ((strcmp(Pu600_buf,"PU600: Background Error")==0) ||
      (strcmp(Pu600_buf,"PU300: Background Error")==0))
   {
      printf("Fails the Pu300/600 Background test.\n\n\n");
      errorflag = TRUE;
   }
}
/* End of Pu300/600 background test                          */

                                    // ******************************
                                    // * Process Background Summary *
                                    // ******************************
if (errorflag)
   set_error();
set_measurement_complete();
clock_ff();
break;
//
```

```
                        // ************************************
                        // ** Case 3 = Gamma Calibration      **
                        // ************************************
case 3:
   printf("Received Gamma Calibration start signal.\n");
   reset_Pu600_data();
   reset_Pu900_data();
   reset_NMC_data();
   reset_Symmetry_data();

   /* Start of Pu300/600 Gamma Calibration test  */
   while (Pu600_cnt == 0)    /* loop until data is available */
   {
      start_signal();
      if (start_data == 5)
      {
         call_cancel();
         break;
      }
   }
   if (start_data != 5)
   {
      printf("%d characters received from Pu600-->%s\n",
               Pu600_cnt, Pu600_buf);

      if ((strcmp(Pu600_buf,"PU300: Calibrate Error")==0) ||
         (strcmp(Pu600_buf,"PU600: Calibrate Error")==0))
      {
         printf("Fails the Pu300/600 Gamma Calibration
                        test.\n\n\n");
         errorflag = TRUE;
      }
   }
   Pu600_cnt = 0;  /*  Reset count to wait for the Pu300 response
                        */
   /* End of Pu300/600 Gamma Calibration test    */

   /* Start of Pu900 Gamma Calibration test       */
   while (Pu900_cnt == 0)    /* loop until data is available */
   {
      start_signal();
      if (start_data == 5)
      {
         call_cancel();
         break;
      }
   }
   if (start_data != 5)
   {
      printf("%d characters received from Pu900-->%s\n",
               Pu900_cnt, Pu900_buf);

      if (strcmp(Pu900_buf,"PU900: Calibrate Error")==0)
      {
```

```
            printf("Fails the Pu900 Gamma Calibration test.\n\n\n");
            errorflag = TRUE;
        }
    }
    /* End of Pu900 Gamma Calibration test          */

    /* Start of Pu300/600 Gamma Calibration test  */
    /*   This second loop is for the Pu300 to respond */
    while (Pu600_cnt == 0)              /* loop until data is avail-
                        able */
    {
        start_signal();
        if (start_data == 5)
        {
            call_cancel();
            break;
        }
    }
    if (start_data != 5)
    {
        printf("%d characters received from Pu300-->%s\n",
                Pu600_cnt, Pu600_buf);

        if ((strcmp(Pu600_buf,"PU300: Calibrate Error")==0) ||
            (strcmp(Pu600_buf,"PU600: Calibrate Error")==0))
        {
            printf("Fails the Pu300/600 Gamma Calibration
                        test.\n\n\n");
            errorflag = TRUE;
        }
    }
    /* End of Pu300/600 Gamma Calibration test     */

    if (errorflag)
        set_error();
    set_measurement_complete();
    clock_ff();
    break;

//
```

```
                        // ************************************
                        // ** Case 4 = NMC Calibration        **
                        // ************************************
case 4:
   printf("Received Measure Control start signal.\n");
   reset_Pu600_data();
   reset_Pu900_data();
   reset_NMC_data();
   reset_Symmetry_data();

   /* Start of NMC Measure Control (Bias) test       */
   while (NMC_cnt == 0)      /* loop until data is available */
   {
      start_signal();
      if (start_data == 5)
      {
         call_cancel();
         break;
      }
   }
   if (start_data != 5)
   {
      printf("%d characters received from NMC-->%s\n",
               NMC_cnt, NMC_buf);

      if (strcmp(NMC_buf,"FAIL")==0)
      {
         printf("Fails the NMC Measure Control (Bias)
                        test.\n\n\n");
         errorflag = TRUE;
      }
   }
   /* End of NMC Measure Control (Bias) test      */

   if (errorflag)
      set_error();
   set_measurement_complete();
   clock_ff();
   break;
//
```

```
                                    // ************************************
                                    // ** Case 5 = Cancel                 **
                                    // ************************************
            case 5:
               printf("Received Cancel signal.\n");
               reset_Pu600_data();
               reset_Pu900_data();
               reset_NMC_data();
               reset_Symmetry_data();
               set_measurement_complete();
               clock_ff();
               break;
                                    // ************************************
                                    // ** Case ? = Error                  **
                                    // ************************************
            default:
               printf("Error in case statement.  ");
               printf("Start_data received is %d\n",start_data);
               set_error();
               clean_exit();
               exit(1);
               break;
         }
      }
      clean_exit();
}                       /* End of void MAIN  */
```

7.1.3 Interrupt-Driven Code

The standard interrupt driven code of ROM-DOS is used with additions for the four serial ports contained on the Emerald-MM-DIO card.

The interrupts for the four serial ports are altered in DATA_ATT.EXE using the COMM-DRV function, **ser_rs232_set_intfunction(port#, ds:di, ax)**. Where **port#** is the port number, **ds:di** is the address of the function call (segment:offset), and **ax** is the interrupt to process mask. See page 362 of the COMM-DRV/LIB manual. The DATA_ATT.EXE program supplies **&COM1_interrupt_func** as the pointer to actual interrupt function and **INTFUNC_RCVCHR** as the mask.

In DATA_ATT.C, the **COM1_interrupt_func** is defined in relation to a structure of type **port_param__far** and **COM1_IO**, which is also defined in relation to a structure of type **port_param__far** and a **GetString** function. The **GetString** function is a high-level COMM-DRV function prototype defined in COMM.H. The source code for **GetString** is presumably in one the included source code files. The structure type **port_param__far** is defined in COMM.H.

7.1.4 I/O Interface to COM

The following table shows the hardware I/O parameters for the COM port interfaces.

Table Showing the Utilization of the Serial (COM) Ports

IRQ	DMA	PORT hex	CARD	FUNCTION
4		100 – 07	DIO – Serial-1	→ DIO COM1 = NMC
3		108 – 10F	DIO – Serial-2	→ DIO COM2 = PU300/PU600
11		110 – 117	DIO – Serial-3	→ COM3 = PU900
10		118 – 11F	DIO – Serial-4	→ COM4 = Symmetry
[3]		2F8 – 2FF	3SXi – Serial-2	3SXi Serial Port 2 = COM2 [IRQ disabled]
[4]		3F8 - 2FF	3SXi – Serial-1	3SXi Serial Port 1 = COM1 [IRQ disabled]

The software INPUT interface to the serial ports is through the interrupt-driven code discussed above (in section 7.1.3).

7.1.5 I/O Interface to Digital I/O Bits

The following table shows the hardware I/O parameters for the DIO port interfaces to the operator switches, which determine the current mode or state of the system. The following table indicates the path of the unclassified input data bits

- from the operator switches
- to the input connector pins on the back panel
- to the DIO card connector pins
- to the DIO ports and bits
- to the Start_Data parameter value used by the software to routine data through the DATA_ATT.EXE program.

Table of Input Control Lines to the Computational Block from Operator Switches

Mode	**Input Pin**	**DIO Card Pin**	**DIO Port**	**DIO Bit**	**Start_Data Value**	**Explanation**
	1					NC
Assay (Verify)	2	J6-47	0	0	1	
Background	3	J6-45	0	1	2	
Gamma Calibrate	4	J6-43	0	2	3	
Measure Control	5	J6-41	0	3	4	
Cancel	6	J6-39	0	4	5	
	7					NC
	8					NC
	9					+5 volts from Power Supply
	10					NC
	11					NC
	12					NC
	13					NC
	14	J6-50				Ground from DIO card
	15					Ground from Power Supply
		J6-49				+5 volts from DIO card cut & unused
		J6-48				Signal grounds cut & unused

The software INPUT interface to the digital I/O ports is relatively straight forward and is contained in the source code for DATA_ATT.C (provided in section 7.1.2.2).

The following table indicates the path of the unclassified output data bits determined and set by the DATA_ATT program

- from the DIO ports and bits
- to the DIO card connector pins
- to the output connector on the side panel.

There is one additional bit (DIO port3, bit0), which is used to set the data into the data barrier flip-flops.

Table of Digital Output Values from Emerald-MM-DIO Card

DIO Port	**DIO Bit**	**DIO Card Pin**	**Output Pin**	**Explanation**
3	0	J5-47	10	Initializing the Data Barrier Flip-Flops
4	2	J5-27	22	Age Red
4	3	J5-25	21	Age Green
4	4	J5-23	20	Symmetry Red
4	5	J5-21	19	Symmetry Green
4	6	J5-19	18	Pu Presence Red
4	7	J5-17	17	Pu Presence Green
5	0	J5-15	9	Pu Oxide Red
5	1	J5-13	8	Pu Oxide Green
5	2	J5-11	7	Pu Mass Red
5	3	J5-9	6	Pu Mass Green
5	4	J5-7	5	Pu Isotope Red
5	5	J5-5	4	Pu Isotope Green
5	6	J5-3	3	System Error
5	7	J5-1	2	Measurement Complete
		J5-49	14	+5V
		J5-50	25	Ground
		J5-others		Cut & Unused
			1	NC
			11	NC
			12	NC
			13	NC
			15	NC
			16	NC
			23	NC
			24	NC

The software OUTPUT interface to the digital I/O ports is relatively straight forward and is contained in the source code for DATA_ATT.C (provided in section 7.1.2.2).

7.1.6 Threshold Values for the Calculations

The computational block software contains attribute threshold values as defined parameters in an included header file, DATA_ATT.H. This approach allows the threshold values to be readily changed at one location in all the AMS software in response to future negotiations. The following table summarizes these attribute threshold values.

Table of Attribute Threshold Values for the Attribute Measurements

Threshold Value	**Subsystem**	**Explanation**
500 grams	NMC	Minimum total weight of plutonium in canister
0.50	NMC	Maximum ratio of alpha-n neutrons to spontaneous fission neutrons to avoid being considered plutonium-oxide
1.0 cts/second	Pu900	Maximum 871-keV peak amplitude to avoid being considered plutonium-oxide
0.10	Pu600	Maximum ^{240}Pu:^{239}Pu ratio to be accepted as weapons-grade plutonium
3 years	Pu300	Minimum age of plutonium to be accepted as from a weapon
0.15	Symmetry	Maximum fractional deviation from the mean at any angle accepted as from an axially symmetric object
3-sigma	Symmetry	Minimum statistical significance of fractional deviation from the mean required to reject as axially symmetric object

7.2 WCSC, COMM-DRV/LIB Version 17 – Communication Library

This communication-software library contains the drivers and interrupt functions used to handle the I/O between the DATA_ATT.EXE application program and the four serial COM ports on the extra I/O board.

This software package is available from Willies Computer Software Company (WCSC) (6215 Longflower Lane, Kingwood, TX 77345 – phone 281-360-4232) for $189.95.

7.2.1 Description of COMM-DRV/LIB

The vendor's description of the COMM-DRV/LIB software package follows as downloaded from the WCSC Internet site at:

http://www.wcscnet.com/CdrvLBro.htm.

The level of documentation provided by the vendor is consistent with that necessary for a completely transparent information-barrier-protected measurement system. The distribution disks include complete source code.

Serial I/O **WCSC** The answer to all of your serial communication needs Serial I/O

COMM-DRV/Lib

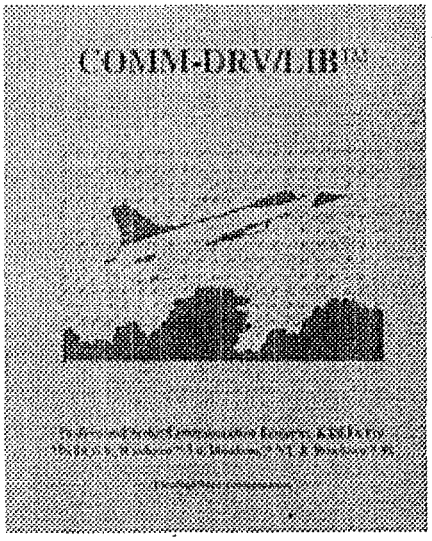
ORDER ONLINE NOW!!

NEW PCI Multiport Cards

COMM-DRV/LIB

Price $189.95

Add to Your SHOPPING CART | View Your SHOPPING CART | Go To CHECKOUT NOW



COMM-DRV/LIB is simply the best and fastest serial communications libraries and DLLs on th market. It supports MS-DOS, Windows 3.x, Windows 95, Windows 98, Windows 2000, and Windows NT. It was designed by real develope for real developers and programmers.

COMM-DRV/LIB Includes

- A set of serial communication libraries tha C/C++, QuickBasic, and Assembly Langu
- A set of high level Hayes compatible mod information communication libraries that li C/C++, QuickBasic, and Assembly Langu
- A set of state driven Xmodem, Ymodem, a transfer communication libraries that link QuickBasic, and Assembly Language app
- A set of serial communication DLLs, high l serial control and information communicat Ymodem, & Zmodem DLLs that may be d application that is capable of calling the ACCESS, or Visual Basic).
- An extensive set of examples for using th Included are example C/C++, Visual Basi Basic, and other projects.

Major Features

- Support ALL compilers and tools that can call 16 bit or 32 bit DLLs.
- Built-in hardware and software handshaking for flow control (DTR/DSR, RTS/CTS, XON/XOFF).
- One API to learn. Same for Windows 3.x, Windows 95, Windows 98, Wind 2000, Windows NT, & MS-DOS.
- Any number of ports may be active at the same time.
- Adjustable communication buffers of any size.
- Remap baud rates logically. Modify baud rate divisor for non-standard bau rates.
- Supports all non-intelligent and several well known intelligent multiport cards(Digiboard, Arnet, Boca Research, GTEK, etc.)..
- Only library that really allows you to combine different multiport cards in o PC.
- Extensive statistics on every port(bytes lost, sent, received, errors, etc.).
- Asynchronously call user's C or assembly functions on any serial communication or timed event (receive, transmit, modem, or status chang interrupt event, buffer count events, specific character reception, error eve and much more).
- Support for all memory models compatible with Microsoft and Borland C/C compilers, and many other compilers.
- Automatic UART type detection; 16550 support (including FIFO).
- Nondestructive reading of data (read ahead).
- Support baud rates in excess of 115,200 baud, with the ability to use toda high-powered communications devices.
- State-driven file transfer libraries allow Xmodem, Ymodem, and Zmodem fi transfers on multiple ports at the same time.
- Completely port re-entrant, allowing it to be time-sliced.
- Send data & modify modem signals from within user interrupt(callback) functions. Essential in writing multidrop communication applications or applications requiring fast response to the reception of special packets or characters.
- Supports Network INT14H modem pools, any INT14H server, or INT14H driver.
- Supports WIN32 and WIN16 applications.
- On line help.

What The Experts Say

"Serial communication is at the core of what we do, and COMM-DRV has helpe rapidly develop new applications, and easily add multiple-port features. We've b very impressed with the flexibility of COMM-DRV, and the technical support has been outstanding."

Lee Perryman,Deputy Director and head of technology development, Associate Press Broadcast Services, Washington, DC

"COMM-DRV from WCSC is a source/object serial communications library for D and Windows that lives up to its hype... its feature set puts it right at the top... In terms of sheer versatility, COMM-DRV is nonparalleled."

Tom Campbell, PC Techniques Review

Companies Using COMM-DRV/LIB

COMM-DRV/LIB is used in telecommunications, transaction processing, several major bulletin board systems, satellite communications, and other simple and complex applications by many companies large and small.

A few companies using COMM-DRV/LIB include Ford Motor Company, General Motors, IBM, NASA, Jet Propulsion Laboratory, Lockheed, MCI, AT&T, Rockwell ALCOA, AMOCO, Associated Press, Chevron, Boeing Aerospace, Central Point Software, Cellular One, Citicorp, DEC, Duns & Bradstreet, Federal Express, General Electric, Howard Johnson, Intel, L.A. Cellular, MetroMedia Paging, Metrocellular, National Radio and Astronomy, NEC, Nissan, Panasonic, RJ Reynolds, Southwestern Bell, Texas Instrument, Goodyear, Lawrence Berkeley Laboratory, and the US Postal Service.

COMM-DRV/LIB API(Partial List)

High Level Functions

- **BytesInReceiveBuffer()** Returns the number of bytes in the receive buffer
- **BytesInTransmitBuffer()** Returns the number of bytes in the transmit buff
- **CdrvCheckTime()** Determine if time expired form a previous call to CdrvSetTime().
- **CdrvCrc16()** Returns the 16bit CRC of a packet.
- **CdrvCrc32()** Returns the 32bit CRC of a packet.
- **CdrvDelay()** Delay specified time.
- **CdrvGetPcb()** Returns a pointer to the port's PCB.
- **CdrvSetTime()** Sets a timer to a specified delay. Expiration is tested by CdrvCheckTime().
- **CdrvSetTimeoutFunction()** Sets the address of a function that gets calle when Delay() or CdrvCheckTime() is called.
- **CdrvSetTimerResolution()** Sets the timer resolution used by CdrvSetTim and Delay().
- **DtrOff()** Turns DTR off.
- **DtrOn()** Turns DTR on.

- **FlushReceiveBuffer()** Discards the contents of the receive buffer.
- **FlushTransmitBuffer()** Discards the contents of the transmit buffer.
- **GetByte()** Gets a byte from the receive buffer.
- **GetPaceTime()** Get the current inter-character pace time.
- **GetPacket()** Gets a packet from receive buffer.
- **GetString()** Gets a carriage return, line feed, or null terminated string from receive buffer.
- **GetTimeout()** Get the current timeout value.
- **InitializePort()** Initializes the serial port(Allocate buffers, cardtype, etc.).
- **IsBreak()** Returns true if a break signal was detected.
- **IsCarrierDetect()** Returns true if carrier detected.
- **IsCts()** Returns true if CTS signal high.
- **IsDsr()** Returns true if DSR signal high.
- **IsFramingError()** Returns true if a framing error occurred.
- **IsInputOverrun()** Returns true if the COMM-DRV receive buffer was overr
- **IsOverrunError()** Returns true if the UART receive register was overrun.
- **IsParityError()** Returns true if a parity error occurred.
- **IsPortAvailable()** Determine if a particular port is in use.
- **IsReceiveBufferEmpty()** Returns true if receive buffer is empty.
- **IsRing()** Returns true if ring detected.
- **IsTransmitBufferEmpty()** Returns true if transmit buffer is empty.
- **PeekChar()** Returns the next character from receive buffer non-destructiv
- **PutByte()** Queues a byte for transmission.
- **PutPacket()** Queues a packet for transmission.
- **PutString()** Outputs a null terminated string.
- **ReceiveBufferSize()** Returns the receive buffer size.
- **RtsOff()** Turns RTS off.
- **RtsOn()** Turns RTS on.
- **SendBreak()** Sends a break signal.
- **SetBaud()** Sets new baud rate.
- **SetFlowControlCharacters()** Sets characters used for flow control.
- **SetFlowControlThreshold()** Sets high and low receive buffer thresholds.
- **SetPaceTime()** Sets the current inter-character pace time.
- **SetPortCharacteristics()** Set line control parameters(baudrate, length, parity,etc.).
- **SetTimeout()** Sets the current transmit/receive timeouts.
- **SpaceInReceiveBuffer()** Returns space unused in receive buffer.
- **SpaceInTransmitBuffer()** Returns space unused in transmit buffer.
- **TransmitBufferSize()** Returns transmit buffer size.
- **UnInitializePort()** Undo the effects of InitializePort() (unhook vectors. rele memory, etc.).
- **WaitFor()** Outputs a string and waits for a matching response.
- **WaitForPeek()** Non-destructive WaitFor().
- WaitForPeekTable() Outputs a string and non-destructively waits for a matching response to a table of strings.
- **WaitForTable()** Destructive WaitForPeekTable().

Modem Functions

- **Dial()** Dials using the modem with the Hayes command set.
- **ModemAnswerMode()** Puts modem in answer mode.
- **ModemAttention()** Puts modem in command state.
- **ModemConnect()** Returns true if modem connection attained.
- **ModemGetCarrierSpeed()** Returns carrier speed.
- **ModemGetConnectSpeed()** Returns connect speed.
- **ModemHangup()** Hangup modem connection.
- **ModemInit()** Set modem initialization string.
- **ModemModifyString()** Modify modem string.

File Transfer Functions

- **cdrvxfer_files()** Transmit or receive file(s) with specified protocol to completion.
- **cdrvxfer_getfiles()** Receive file(s) with specified protocol. Must be called several times till transfer complete.
- **cdrvxfer_sendfiles()** Transmit file(s) with specified protocol. Must be call several times till transfer complete.
- **cdrvxfer_sfiles()** Transmit or receive file(s) with specified protocol to completion(additional features).
- **FileTransferDialog()** Enables an automatic dialog to be displayed on som the file transfer functions.
- **SetXferParameters()** Set file transfer parameters.
- **TransferFiles()** Transfer files.

Low Level Functions

- **ser_rs232_block()** Set timeouts on character reception & trasmission.
- **ser_rs232_cleanup()** Uninstalls a port.
- **ser_rs232_dtr_off()** Turns the DTR signal off.
- **ser_rs232_dtr_on()** Turns the DTR signal on.
- **ser_rs232_flush()** Selectively fushes input and output buffers.
- **ser_rs232_get_sdata()** Gets pointer to **COMM-DRV** system data area.
- **ser_rs232_getbyte()** Reads a byte from input buffer.
- **ser_rs232_getpacket()** Reads a packet from input buffer.
- **ser_rs232_getport()** Gets port information.
- **ser_rs232_getstatus()** Gets port modem and line status.
- **ser_rs232_maxport()** Returns the highest addressable port number.
- **ser_rs232_misc_func()** **COMM-DRV** multiplex function. Used to setup us callback functions, initialize certain variables, etc.
- **ser_rs232_putbyte()** Queues a byte for transmission.
- **ser_rs232_putpacket()** Queues a packet for transmission.
- **ser_rs232_putregister()** Writes to specified 8250 type register.
- **ser_rs232_rts_off()** Turns RTS off.
- **ser_rs232_rts_on()** Turns RTS on.
- **ser_rs232_set_intfunc()** Setup user interrupt functions(user callback).
- **ser_rs232_setbauddiv()** Sets baudrate divisor for 8250 style UARTs.
- **ser_rs232_setup()** Installs or modify a serial port.
- **ser_rs232_viewpacket()** Non-destructively reads a packet from input buff

[Home] [Up]

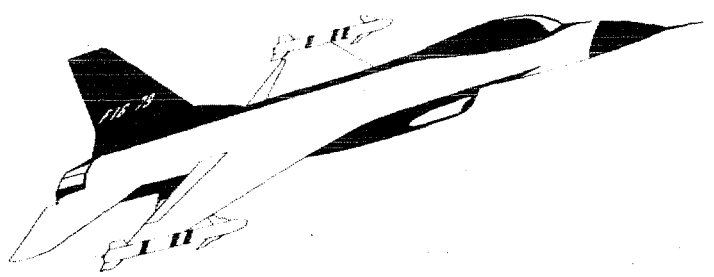
Send mail to webmaster@wcscnet.com with questions or comments about this web site.

Last modified: January 14, 2000

7.2.2 COMM-DRV/LIB User Manual

The COMM-DRV/LIB User Manual is included in the $190 software package or it can be purchased separately for $35. The following is a copy of this manual.

COMM-DRV/LIB™



Professional Serial Communication Libraries & DLLs For MS-DOS®, Windows™ 3.x, Windows™ NT, & Windows™ 95

Ultra High Speed Communications

Willies Computer Software Co.
6215 Longflower Lane
Kingwood, TX 77345

Telephone: (281)360-4232
Tech Support: (281)360-3187
Fax: (281)360-3231

Email:

Support: support@wcscnet.com
Sales: sales@wcscnet.com
WorldWideWeb: http://www.wcscnet.com/

COMM-DRV/LIB™

Professional Asynchronous Communications Library For
Windows™ 3.x, Windows 95, Windows NT, and MSDOS®

USERS & TECHNICAL MANUAL

LIMITED WARRANTY

THE PROGRAM IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU (NOT WILLIES' COMPUTER SOFTWARE COMPANY (WCSC) OR AN AUTHORIZED DEALER) ASSUME THE ENTIRE COST OF ALL NECESSARY SERVICING, REPAIR, OR CORRECTION. IN NO EVENT WILL WCSC BE LIABLE TO YOU FOR ANY DAMAGES , INCLUDING ANY LOST PROFITS, LOST SAVINGS, OR OTHER INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE SUCH PROGRAM EVEN IF WCSC OR AN AUTHORIZED WCSC PERSONAL COMPUTER DEALER HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES, OR FOR ANY CLAIM BY ANY OTHER PARTY.

Willies Computer Software Company(WCSC)
6215 Longflower Lane
Kingwood, Texas 77345
FAX Telephone #: (281)360-3231
Business Telephone #: (281)360-4232
Technical Support #: (281)360-3187
WorldWideWeb: http://www.wcscnet.com/
Internet(Sales): sales@wcscnet.com
Internet(Support): support@wcscnet.com
FTP: ftp.wcscnet.com

Rev. 2/March 1998

This Page Intentionally Left Blank

COMM-DRV/LIB™ Professional Serial Communication Library
Willies Computer Software Company

Table Of Contents

Table Of Figures

Introduction

Please note that this manual will use the terms "**COMM-DRV/LIB**" and "**COMM-DRV**" interchangeably. Where distinctions are necessary to differentiate this product from our other family of **COMM-DRV** serial communications products, we will make it obvious. Likewise, "***Windows***" will be used to refer to **Windows 3.x, Windows 95**, and **Windows NT**. Where distinctions are necessary, we will specify.

This introduction should be read in its entirety in order that one may get an overview of the vast capabilities and scope of **COMM-DRV/LIB**. At first, the capabilities of this product may seem overwhelming. Remember however, that this product can be learned incrementally. If you need to get something up and running quickly , be sure to scan the sections listed below in boldface in their entirety. Following this you should have an idea of the components of **COMM-DRV/LIB** that you need to use. Of course our technical support staff will be available to make your transition into the **COMM-DRV/LIB** realm easy and pleasurable.

These are the sections to be read in their entirety.

INTRODUCTION
- **Technical Specification**
- **Product Support**
- **Product Licensing**
- **Installing The Software**

Getting Started
Basic Definitions & Conventions
- **Serial Communication/A Basic Tutorial**
- **Port address, Port number, & Device explanation**
- **RS232 cabling**

Note that you should also print a copy of the file READMECL.TXT to view any changes made to the product since the printing of the manual. READMECL.TXT is an ASCII file. A Microsoft Word for Windows version of READMECL.TXT is also distributed under the filename READMECL.DOC. A Windows Write version is provided under the name READMECL.WRI.

COMM-DRV/LIB is a robust and reliable set of serial communication libraries and dynamic link libraries(DLLs) for MS-DOS and Windows. Programming with **COMM-DRV/LIB** under Windows is identical to programming with **COMM-DRV/LIB** under MS-DOS. It is a fully tested product with thousands of hours of continuous, bug free operation. It is currently used as the serial I/O backbone for several transaction processing, SCADA, billing, banking, terminal emulation,

bulletin board, fax, satellite, factory floor, submarine, space, multi-user, and several other applications. It was designed to be extremely easy to use. **COMM-DRV/LIB** may be used in applications ranging from the very simple to the very complex. It will not interfere with other serial drivers, DLLs, or TSRs that follow standard Windows and MS-DOS conventions.

Technical Specifications

COMM-DRV/LIB is a very technically advanced product. **It supports MSDOS, 16 bit(WIN16), and 32 bit(WIN32®) applications.** It may be statically linked or dynamically linked with several languages and development tools. Specifically, it may be used with Microsoft QuickBasic 4.x, Microsoft Visual Basic For DOS, Microsoft Visual Basic For Windows, Microsoft Professional Basic 7.x, Microsoft Access for Windows, Microsoft C/C++ compilers, Borland C/C++ compilers, and any language, database tool, or language that can make calls to the Windows API. **COMM-DRV/LIB** API is identical for both 16 bits and 32 bits.

COMM-DRV/LIB supports the default **COM1-COM4** serial ports on the PCs and compatibles, in addition to virtually all non-intelligent multiport cards based on the 8250/16450/16550/16650 family of UARTs. Intelligent cards from Arnet Corporation, Digiboard, GTEK, and Boca Research are supported directly(without the need of manufacturer distributed drivers). Virtually any serial device, intelligent or non-intelligent, with its own INT14H driver, or Windows driver is supported as well.

COMM-DRV/LIB allows multiport cards from several different manufacturers to exist and function concurrently in one machine. This allows the user to mix and match devices based on cost, need, or simply to make use of devices already acquired. **COMM-DRV/LIB** was successfully tested with a multiport card from each supported manufacturer in a specially designed box. All ports from all devices were activated concurrently.

COMM-DRV/LIB allows the sharing of individual IRQs on the PC. Several serial ports can be connected to the same IRQ. IRQ00 through IRQ15 are supported. Each port supports adjustable communication buffers of up to 128k bytes with 16 bit programs or just under 2 Gigabytes with 32 bit programming. DTR/RTS/DSR/CTS hardware protocol is supported on both reception and transmission as well as XON/XOFF software protocol.

Thresholds (high water and low water marks) for activating the respective flow control are adjustable. **COMM-DRV/LIB** operates at speeds in excess of 460K baud on cards that allow it.

COMM-DRV/LIB will run perfectly under multitaskers like Desqview, Omniview, Taskview, Windows 3.x, Windows 95, Windows NT, and UNIX under VPIX. It can also function as a redirected asynchronous client on a network (i.e. a modem pool, etc.).

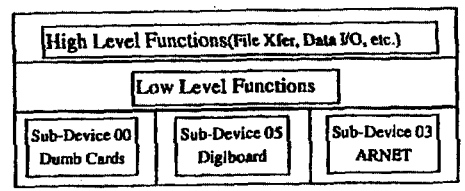


Figure 1

COMM-DRV/LIB is a well designed and layered library. The programmer need not worry about the following concepts, however, it will be detailed here for completeness. **COMM-DRV/LIB** has three discernible layers.

The lowest layer is known as the device specific layer. The files that comprise this layer are called sub-device drivers. Sub-devices are device specific files that contain some very basic communication primitives specific to a particular type of serial card. **COMM-DRV/LIB** has a sub-device for each type of multiport card that it supports. The interface, structure and functions in these sub-devices are very well defined and allow for very fast development for any non-standard card not currently supported by simply creating this sub-device file.

The middle layer of **COMM-DRV/LIB** is the only software that communicates with the lowest layer(device specific layer). This layer has a very small set of functions that application programmers can use to perform all basic serial communications tasks. This is the abstracting layer of **COMM-DRV/LIB**. In fact, the intent of this layer is to leverage all the work done in the highest layer that includes file transfer functions, data I/O functions and more. Whenever we move **COMM-DRV/LIB** to a new environment(e.g.,, OS/2, VMS) this layer will simply be replaced. This means we can adapt to any environment rapidly.

The top layer of **COMM-DRV/LIB** is the layer most application programmers will use. It has a myriad of esoteric functions for serial I/O. It includes the file transfer functions, data input/output functions, data stream processing functions, modem functions, and much more. These functions make calls to the middle layer and to other functions in the same layer.

Even though we are layered as shown above, the interfaces are very efficient. The actual sub-device format is explained in a later section. Note these sub-devices are generally very small (800 to 3000 bytes). Moreover most dumb cards are able to use one specific sub-device. In general each different type of smart card will use a different sub-device. Sub-devices can be active concurrently (e.g., a Digiboard COMxi, PC/Xe, PC/Xi, PC/Xm, ARNET Smartport Plus, AST, card may be in one machine concurrently).

Product Support

Technical support for **COMM-DRV** is provided for all registered users. We will not provide technical support for calls made on our 800 number. **Support via telephone is provided from 1:00pm to 5:00pm Central Standard Time.** Callbacks generally occur between 8:00am and 12:00 noon Central Standard Time. Support via email and fax is provided 24 hours per day. **Please help us keep technical support efficient and inexpensive by using email(preferably Internet mail to support@wcscnet.com).**

Please note that email and fax are the preferable avenues for technical support as we have found that in writing down a problem, the developer organizes his/her thought more precisely and resolution is much quicker. **We respond to email to "support@wcscnet.com" throughout the entire day and sometimes into the night.** Any reported bugs that are reproducible will be fixed promptly. Fixes may be downloaded from our BBS or FTP(ftp.wcscnet.com) *server free of charge.*

Following are the different avenues through which support is provided.

1. Email via **Internet** at **support@wcscnet.com**.
2. Fax at (281)360-3231.
3. Telephone support at (281)360-3187.

When reporting a problem or requesting assistance please have the answers to the following questions. We will not begin to address the problem until you have them. It is also preferable that you have immediate access to your computer and serial cards if applicable. You must also have access to an editor so that we may modify the necessary files.

1) What is your serial number?
2) Where did you purchase COMM-DRV?
3) What type of CPU are you using and who is the manufacturer (Pentium, 80486,80386,8088,80186, & Intel, AMD, Cyrix, etc.)?
4) What type of serial card(s) are you using, manufacturer, UART types?
5) Which cards are you using in your PC (FAX card, Voice mail card, etc.)? We must know what IRQ, if any, they are using and also what I/O ports if applicable.
6) What type of Bus does your PC have (AT Bus(ISA), EISA Bus, Microchannel Bus(MCA)?
7) What is in your config.sys and autoexec.bat files?
8) What is in your win.ini and system.ini files?

Product Licensing

1. The **COMM-DRV/LIB** object files can be linked directly into any MS-DOS or Windows application and sold as desired. No royalties or further licensing is required as long as the **COMM-DRV/LIB** objects are bound to the application.

2. Source code, object code, or static libraries distributed with this package may not be re-distributed or resold.

3. The DLLs **CDRVDLL.DLL, CDRVXF.DLL, COMMSCRW.DLL, CDRVHF.DLL, CDRVDL32.DLL, CDRVXF32.DLL, CDRVHF32.DLL, COMMSC32.DLL,** and ***.BIN** may be re-distributed with the user application without the requirements of licensing or royalties.

4. **COMM-DRV/LIB** is sold one copy per computer, unless a site license is purchased.

Installing The Software

Installing **COMM-DRV/LIB** on your hard disk is very simple. You can install it from either the Windows Program Manager or from the DOS command line.

Installing From Windows 95 or Windows NT 4.0.

- Click on **Start** on the Taskbar.
- Click on **Run** in the menu.
- Enter **a:\setup** if the **COMM-DRV/LIB** diskette is in the A: floppy drive or enter **b:\setup** if the diskette is in the B: floppy drive.
- Follow instructions displayed by the installation program.

Installing From Windows NT 3.x or Windows 3.x.

- Click on **Files** in the Program Manager.
- Click on **Run** in the menu.
- Enter **a:\setup** if the **COMM-DRV/LIB** diskette is in the A: floppy drive or enter **b:\setup** if the diskette is in the B: floppy drive.
- Follow instructions displayed by the installation program.

Installing From The DOS Command Line

- Contact WCSC technical support to get a special zipped version of the installation.

Getting Started

The following sections detail how to use **COMM-DRV/LIB** with different compilers and in different environments. The basic sequence of **COMM-DRV/LIB** function (subroutine) calls in a communication application would be similar to the following. If your compiler/tool is not mentioned review the file **READMECL.DOC** or call our technical support department.

1. Initialize the port (**InitializePort()** or **ser_rs232_setup()**). This call is neccssary to associate a physical serial port to a number that will be subsequently used by all other library functions to identify the port. It attaches interrupt vectors, allocate buffers, etc.
2. Set the port's line characteristics like baud rate, parity, etc.(**SetPortCharacteristics()** or **ser_rs232_setup()**).
3. Send and receive data(**GetPacket()**, **ser_rs232_getpacket()**, **PutPacket()**, **ser_rs232_putpacket()**, **GetByte()**, **PutByte()**, **TransferFiles()**, etc.).
4. Shutdown communication routine/hardware(**UnInitializePort()** or **ser_rs232_cleanup()**). Removed any allocated interrupts, buffers, memory, etc.

The above sequence is followed by any language or tool using **COMM-DRV/LIB**. Most of the C/C++ example sources are found in the directory **examples** in the **COMM-DRV/LIB** installation directory. Most batch files (*.bat) are found in the sub-directory **bin** in the **COMM-DRV/LIB** installation directory.

In order to give a working example of the different serial communication functions, a terminal application with file transfer is provided for most of the examples. Note that any application that uses any **COMM-DRV/LIB** DLL or support files(*.BIN) must place these DLLs and support files either in the Windows System directory(e.g., **C:\WINDOWS\SYSTEM**) or in the directory from which the application is launched.

Additionally, a simple application may be reviewed in **Appendix D**. This should illustrate the ease of use of **COMM-DRV/LIB**.

C/C++ MS-DOS Applications(Microsoft C/C++ Compiler)

To create an MS-DOS application with the Microsoft C/C++ compilers, compile your application, then link with the libraries **commdrms.lib**, and **libsms.lib** for the small and compact memory models. Link with the libraries **commdrml.lib**, and **libsml.lib** if you are using the medium, large, or huge memory models. The include file **comm.h** must be included in every module that makes calls to the **COMM-DRV/LIB** libraries.

It is a good idea to start development by first compiling and linking **example4.c**. A batch file named **mexample.bat** is provided to compile this example and many other examples. A Microsoft Visual Workbench project for this example is found in subdirectory **prjmsex4** under the **COMM-DRV/LIB** root installation directory..

The **mexample.bat** batch file contains some environment variables that must be edited to reflect your environment(location of compilers, etc.). **mexample.bat** contains instructions that should be followed before running it. Please edit the file with your editor of choice (e.g., DOS edit.exe, etc.).

mexample.bat places all the compiled objects and executables in the current directory. You should create a separate directory to build your examples. To compile and link the example type

install_path\bin\mexample example4.exe ***model install_path***

where

install_path	is the root installation directory of **COMM-DRV/LIB**.
model	is the memory model to use (large or small).

Example:

c:\commdrv\bin\mexample example4.exe large c:\commdrv

C/C++ MS-DOS Applications(Borland C/C++ Compilers)

To create an MS-DOS application with the Borland C/C++ compilers, compile your application, then link with the libraries **commdrbs.lib**, and **libsbs.lib** for the small and compact memory models. Link with the libraries **commdrbl.lib**, and **libsbl.lib** if you are using the medium, large, or huge memory models. The include file **comm.h** must be included in every module that makes calls to the **COMM-DRV/LIB** libraries.

It is a good idea to start development by first compiling and linking example4.c. A batch file named **bexample.bat** is provided to compile this example and many other examples. A Borland project for this example is found in subdirectory **prjboex4** under the **COMM-DRV/LIB** root installation directory.

The **bexample.bat** batch file contains some environment variables that must be edited to reflect your environment(location of compilers, etc.). **bexample.bat** contains instructions that should be followed before running it. Please edit the file with your editor of choice (e.g., DOS edit.exe, etc.).

bexample.bat places all the compiled objects and executables in the current directory. You should create a separate directory to build your examples. To compile and link the example type

install_path\bin\bexample example4.exe ***model install_path***

where

install_path	is the root installation directory of **COMM-DRV/LIB**.
model	is the memory model to use (large or small).

Example:

c:\commdrv\bin\bexample example4.exe large c:\commdrv

C/C++ Windows Applications(Microsoft C/C++ Compilers)

A Windows 3.x or Windows 95 16 bit application can either be linked with the **COMM-DRV/LIB** static libraries or with the DLLs. **Using the DLLs is the preferred method.**

To create a Microsoft C/C++ compiled Windows application in which the libraries are a part of the DLLs provided with this package, compile your application, then link with the libraries **cdrvdll.lib, cdrvxf.lib, cdrvhf.lib,** and **libswms.lib** for the small and compact memory models. Link with the libraries **cdrvdll.lib, cdrvxf.lib, cdrvhf.lib,** and **libswml.lib** if you are using the medium, large, or huge memory models. The include file **comm.h** must be included in every module that makes calls to the **COMM-DRV/LIB** libraries. The C/C++ definition **MSWIN** and **MSWINDLL** must be defined ahead of the **comm.h** inclusion as follows.

```
#define MSWIN
#define MSWINDLL
#include <comm.h>
```

MSWIN and **MSWINDLL** may instead be declared from the command line or from the workbench as well.

It is a good idea to start development by first compiling and linking **example8.c**. A Microsoft Visual Workbench project for this example is found in subdirectory **prjmsex8** under the **COMM-DRV/LIB** root installation directory.

To compile and link the example type

install_path\bin\mexample example8.exe ***model install_path***

where

install_path	is the root installation directory of **COMM-DRV/LIB**.
model	is the memory model to use (large or small).

Example:

c:\commdrv\bin\mexample example8.exe large c:\commdrv

To create a Microsoft C/C++ compiled Windows application in which the libraries are an actual part of the executable, compile your application, then link with the libraries **commdwms.lib,** and **libswms.lib** for the small and compact memory models. Link with the libraries **commdwml.lib,** and **libswml.lib** if you are using the medium, large, or huge memory models. The include file **comm.h** must be included in every module that makes calls to the **COMM-DRV/LIB** libraries. The C/C++ definition **MSWIN** must be defined ahead of the **comm.h** inclusion as follows.

```
#define MSWIN
#include <comm.h>
```

MSWIN may instead be declared from the command line or from the workbench as well.

It is a good idea to start development by first compiling and linking **example9.c**. A batch file named **mexample.bat** is provided to compile this example and many other examples. A Microsoft Visual Workbench project for this example is found in subdirectory **prjmsex9** under the **COMM-DRV/LIB** root installation directory.

The **mexample.bat** batch file contains some environment variables that must be edited to reflect your environment(location of compilers, etc.). **mexample.bat** contains instructions that should be followed before running it. Please edit the file with your editor of choice (e.g., DOS edit.exe, etc.).

mexample.bat places all the compiled objects and executables in the current directory. You should create a separate directory to build your examples. To compile and link the example type

install_path\bin\mexample example9.exe ***model install_path***

where

install_path	is the root installation directory of **COMM-DRV/LIB**.
model	is the memory model to use (large or small).

Example:

c:\commdrv\bin\mexample example9.exe large c:\commdrv

C/C++ Windows Applications(Borland C/C++ Compilers)

A Windows 3.x or a Windows 95 16 bit application can either be linked with the **COMM-DRV/LIB** static libraries or with the DLLs. Using the DLLs is the preferred method.

To create a Borland C/C++ compiled Windows application in which the libraries are a part of the DLLs provided with this package, compile your application, then link with the libraries **cdrvdll.lib**, **cdrvxf.lib**, **cdrvhf.lib**, and **libswbs.lib** for the small and compact memory models. Link with the libraries **cdrvdll.lib**, **cdrvxf.lib**, **cdrvhf.lib**, and **libswbl.lib** if you are using the medium, large, or huge memory models. The include file **comm.h** must be included in every module that makes calls to the **COMM-DRV/LIB** libraries. The C/C++ definition **MSWIN** and **MSWINDLL** must be defined ahead of the **comm.h** inclusion as follows.

```
#define  MSWIN
#define  MSWINDLL
#include <comm.h>
```

It is a good idea to start development by first compiling and linking **example8.c**. A Borland IDE project for this example is found in subdirectory **prjboex8** under the **COMM-DRV/LIB** root installation directory.

To compile and link the example type

install_path\bin\bexample example8.exe ***model install_path***

where

install_path	is the root installation directory of **COMM-DRV/LIB**.
model	is the memory model to use (large or small).

Example:

c:\commdrv\bin\bexample example8.exe large c:\commdrv

To create a Borland C/C++ compiled Windows application in which the libraries are an actual part of the executable, compile your application, then link with the libraries **commdwbs.lib**, and **libswbs.lib** for the small and compact memory models. Link with the libraries **commdwbl.lib**, and **libswbl.lib** if you are using the medium, large, or huge memory models. The include file **comm.h** must be included in every module that makes calls to the **COMM-DRV/LIB** libraries. The C/C++ definition **MSWIN** must be defined ahead of the **comm.h** inclusion as follows.

```
#define  MSWIN
#include <comm.h>
```

MSWIN may instead be declared from the command line or from the Borland IDE as well.

It is a good idea to start development by first compiling and linking **example9.c**. A batch file named **bexample.bat** is provided to compile this example and many other examples. A Borland IDE project for this example is found in subdirectory **prjboex9** under the **COMM-DRV/LIB** root installation directory.

The **bexample.bat** batch file contains some environment variables that must be edited to reflect your environment(location of compilers, etc.). **mexample.bat** contains instructions that should be followed before running it. Please edit the file with your editor of choice (e.g., DOS edit.exe, etc.).

bexample.bat places all the compiled objects and executables in the current directory. You should create a separate directory to build your examples. To compile and link the example type

install_path\bin\mexample example9.exe ***model install_path***

where

install_path	is the root installation directory of **COMM-DRV/LIB**.
model	is the memory model to use (large or small).

Example:

c:\commdrv\bin\bexample example9.exe large c:\commdrv

C/C++ Windows 95 & Windows NT Applications

A Windows 95 and Windows NT C/C++ applications should link to the 32 bit DLLs provided with **COMM-DRV/LIB**.

The include file **comm.h** must be included in every module that makes calls to the **COMM-DRV/LIB** DLLs. The C/C++ definition **MSWIN, MSWIN32, and MSWINDLL,** must be defined ahead of the **comm.h** inclusion as follows.

```
#define  MSWIN
#define  MSWIN32
#define  MSWINDLL
#include <comm.h>
```

The application program should be linked with the import libraries **cdrvdl32.lib, cdrvhf32.lib, cdrvxf32.lib, commsc32.lib**. These import libraries correspond to the DLLs that with the same respective base names.

Note that the above mentioned definition could have been defined either on the compile command line or from the workbench.

It is a good idea to start development by first compiling and linking **example8.c**. A Microsoft Visual Workbench project for this example is found in subdirectory **prjmsw32** under the **COMM-DRV/LIB** root installation directory. A Borland IDE project example is in **prjbow32**. A Watcom project example is in **prjwaw32**. A Symantec project example is in **prjsyw32**.

Microsoft Visual Basic For MS-DOS Applications

Interfacing **COMM-DRV/LIB** to a Microsoft Visual Basic For MS-DOS (VBDOS) application is very simple. The libraries are actually linked into the **VBDOS** applications. All the files related to **VBDOS** application development are in subdirectory **vbdos** found in the **COMM-DRV/LIB** installation directory.

The file **vbdocdrv.lib** is the library with the serial communication functions that is linked to the application when compiled to an .exe program. The file **vbdocdrv.qlb** is the quick library that should be loaded by the **VBDOS** environment when developing the application.

The file **vbdocdr2.lib** is a library that contains all the **COMM-DRV/Lib** serial communication routines without the Microsoft C runtime library. If you are having duplicate definitions when linking several third party libraries, you should use this library for linking the basic program into an executable.

A **VBDOS** example project is provided in subdirectory **vbdos** named **vbdoexam.mak**. This example provides a very complete example of using several of the **COMM-DRV/LIB** API functions.

It is important to note that all **COMM-DRV/LIB** functions that require a string as output require that a NULL character be added to the end of the string as the following source clip demonstrates.

```
'Set Modem Init String
a$="ATS0=0&C1&D2"+CHR$(0)

'Output the string
stat = PutString(port,SSEGADD(A$))
```

Additionally, all strings should be pre-initialized before calling any function that retrieves a string or a packet of information. If this is not done the **COMM-DRV/LIB** will certainly write over random memory. The following illustrates an example of pre-initializing string variables.

```
'Allocate a 50 byte string.
a$ = SPACE$(50)

'Get a string from the serial port
stat = GetString(port,50,SSEGADD(a$))
```

Some **COMM-DRV/LIB** functions attempt to allocate memory when called. A **VBDOS** application generally allocates all of memory when it starts. In order to release some of this memory to allow the **COMM-DRV/LIB** API to allocate memory, the **SETMEM() VBDOS** function must be used.

All the functions detailed in the major section of the manual, ***Using The C Library Interface***, and its sub-sections, may be used by the Visual Basic application. The equivalent declaration to all the functions and definitions detailed in the above mentioned sections are declared in the **VBDOS** include file called **vbdocomm.bas**.

As a matter of completeness we will map the C data types to the Visual Basic data types. This may be useful to the few who are interested in getting more technical with Visual Basic than required. ***var*** is the declared variable.

C Declaration	VisualBasic Declaration
char var	DIM var as STRING * 1
unsigned char var	DIM var as STRING * 1
signed char var	DIM var as STRING * 1
int var	DIM var as INTEGER
unsigned int var	DIM var as INTEGER
signed int var	DIM var as INTEGER
short var	DIM var as INTEGER
unsigned short var	DIM var as INTEGER
signed short var	DIM var as INTEGER
long var	DIM var as LONG
unsigned long var	DIM var as LONG
signed long var	DIM var as LONG

Microsoft Visual Basic For Windows Applications

Interfacing **COMM-DRV/LIB** to Visual Basic For Windows is very simple. The most simple and preferred method of interfacing Visual Basic to **COMM-DRV/LIB** however, is via the **COMM-DRV** DLLs.

The best and fastest way to get up to speed with using **COMM-DRV/LIB** with Visual Basic is to start from the example project. In fact, you should be able to cut portions out of the example project and paste it directly into your application. Do make an attempt however to understand the core of the example project. The example Visual Basic project is found in the subdirectory **vbwin** under the **COMM-DRV/LIB** root installation directory. The make file used by the Visual Basic environment is called **vbwiexam.mak** for 16 bit applications and **vb32exam.mak** for 32 bit applications.

Before running the developed application or the example, the necessary DLLs must be installed correctly by copying the DLLs **cdrvdll.dll**, **cdrvhf.dll**, **commscrw.dll**, **cdrvxf.dll**, and ***.bin** for 16 bit applications, or **cdrvdl32.dll**, **cdrvhf32.dll**, **cdrvxf32.dll**, and **commsc32.dll** for 32 bit applications, to the Windows system directory (e.g., ***C:\WINDOWS\SYSTEM***). Alternatively, DLLs may be placed in the directory that will be the default directory when the compiled application is run.

It is important to note that all **COMM-DRV/LIB** functions that require a string as output require that a NULL character be added to the end of the string as the following source clip demonstrates.

```
'Set Modem Init String
a$="ATS0=0&C1&D2"+CHR$(0)

'Output the string
stat = PutString(port,A$)
```

Additionally, all strings should be pre-initialized before calling any function that retrieves a string or a packet of information. If this is not done the **COMM-DRV/LIB** will certainly write over random memory. The following illustrates an example of pre-initializing string variables.

```
'Allocate a 100 byte string.
a$ = SPACE$(100)

'Get  a string from the serial port
stat = GetString(port,100,a$)
```

All the functions detailed in the major section of the manual, ***Using The C Library Interface***, and its sub-sections, may be used by the Visual Basic application. The description for the functions applies to any language that can make DLL calls (e.g., ACCESS, Visual Basic, Magic, etc.). The equivalent declaration to all the functions and definitions detailed in the above mentioned sections are declared in the Visual Basic include file called **vbwicomm.bas** for 16 bit applications, or **vb32comm.bas** for 32 bit applications.

As a matter of completeness we will map the C data types to the Visual Basic data types. This may be useful to the few who are interested in getting more technical with Visual Basic than required. ***var*** is the declared variable.

Mapping for 16 bit applications

C Declaration	VisualBasic Declaration
char var	DIM var as STRING * 1
unsigned char var	DIM var as STRING * 1
signed char var	DIM var as STRING * 1
int var	DIM var as INTEGER
unsigned int var	DIM var as INTEGER
signed int var	DIM var as INTEGER
short var	DIM var as INTEGER
unsigned short var	DIM var as INTEGER
signed short var	DIM var as INTEGER
long var	DIM var as LONG
unsigned long var	DIM var as LONG
signed long var	DIM var as LONG

Mapping for 32 bit applications

C Declaration	VisualBasic Declaration
char var	DIM var as STRING * 1
unsigned char var	DIM var as STRING * 1
signed char var	DIM var as STRING * 1
int var	DIM var as LONG
unsigned int var	DIM var as LONG
signed int var	DIM var as LONG
short var	DIM var as INTEGER
unsigned short var	DIM var as INTEGER
signed short var	DIM var as INTEGER
long var	DIM var as LONG
unsigned long var	DIM var as LONG
signed long var	DIM var as LONG

Microsoft Access For Windows Applications.

Interfacing **COMM-DRV** to Microsoft Access For Windows is very simple. The most simple and preferred method of interfacing Microsoft Access to **COMM-DRV/LIB** is via the COMM-DRV/LIB DLLs.

The best and fastest way to get up to speed with using **COMM-DRV/LIB** with Microsoft Access is to start from the example database provided in the subdirectory **access** found in the **COMM-DRV/LIB** installation directory. In fact, you should be able to cut and paste portions out of the example project and paste it directly into your application. The Microsoft Access database is called **acceedrv.mdb** for 16 bit applications, and **ac32cdrv.mdb** for 32 bit applications.

Before running the developed application or the example, the necessary DLLs must be installed correctly by copying the DLLs **cdrvdll.dll, cdrvhf.dll, commscrw.dll, cdrvxf.dll,** and ***.bin** for 16 bit applications, or **cdrvdl32.dll, cdrvhf32.dll, cdrvxf32.dll,** and **commsc32.dll** for 32 bit applications, to the Windows system directory (e.g., *C:\WINDOWS\SYSTEM*). Alternatively, DLLs may be placed in the directory that will be the default directory when the compiled application is run.

It is important to note that all **COMM-DRV/LIB** functions that require a string as output require that a NULL character be added to the end of the string as the following source clip demonstrates.

```
'Set Modem Init String
a$="ATS0=0&C1&D2"+CHR$(0)

'Output the string
stat = PutString(port,A$)
```

Additionally, all strings should be pre-initialized before calling any function that retrieves a string or a packet of information. If this is not done the **COMM-DRV/LIB** *will certainly write over* random memory. The following illustrates an example of pre-initializing string variables.


```
'Allocate a 50 byte string.
a$ = SPACE$(50)

'Get a string from the serial port
stat = GetString(port,50,a$)
```

All the functions detailed in the major section of the manual, ***Using The C Library Interface***, and its sub-sections, may be used by the Microsoft Access application. The description for the functions apply to any language that can make DLL calls (e.g., Microsoft Access, Visual Basic, Magic, etc.).

The equivalent declaration to all the functions and definitions detailed in the above mentioned sections are declared in Microsoft Access form in the module portion of the example database. As a matter of completeness we will map the C data types to the Microsoft Access data types. This may be useful to the few who are interested in getting more technical with Microsoft Access than required. ***var*** is the declared variable.

Mapping for 16 bit applications

C Declaration	Microsoft Access Declaration
char var	DIM var as STRING * 1
unsigned char var	DIM var as STRING * 1
signed char var	DIM var as STRING * 1
int var	DIM var as INTEGER
unsigned int var	DIM var as INTEGER
signed int var	DIM var as INTEGER
short var	DIM var as INTEGER
unsigned short var	DIM var as INTEGER
signed short var	DIM var as INTEGER
long var	DIM var as LONG
unsigned long var	DIM var as LONG
signed long var	DIM var as LONG

Mapping for 32 bit applications

C Declaration	Microsoft Access Declaration
char var	DIM var as STRING * 1
unsigned char var	DIM var as STRING * 1
signed char var	DIM var as STRING * 1
int var	DIM var as LONG
unsigned int var	DIM var as LONG
signed int var	DIM var as LONG
short var	DIM var as INTEGER
unsigned short var	DIM var as INTEGER
signed short var	DIM var as INTEGER
long var	DIM var as LONG
unsigned long var	DIM var as LONG
signed long var	DIM var as LONG

Microsoft QuickBasic 4.x Applications

QuickBasic 4.x (QB45) has limited interface capabilities to objects from other languages. As such it can only use a subset of the COMM-DRV/LIB functions. This subset corresponds to the COMM-DRV/LIB low level functions.

The file qb45cdrv.lib is the library with the serial communication functions that is linked to the application when compiled to an .exe program. The file qb45cdrv.qlb is the quick library that should be loaded by the QB45 environment when developing the application.

All strings should be pre-initialized before calling any function that retrieves a packet of information. If this is not done the COMM-DRV/LIB will certainly write over random memory. The following illustrates an example of pre-initializing string variables.

```
'Allocate a 100 byte string.
a$ = SPACE$(100)

'Get  a string from the serial port
stat = ser.rs232.getpacket(port,100,SADD(a$)+VARSEG(a$))
```

Some COMM-DRV/LIB functions attempt to allocate memory when called. A QB45 application generally allocates all of memory when it starts. In order to release some of this memory to allow the COMM-DRV/LIB API to allocate memory, the SETMEM() QB45 function must be used.

All the low level functions detailed in the major section of the manual, ***Using The C Library Interface***, and its sub-sections, may be used by the Visual Basic application. The equivalent declaration to all the functions and definitions detailed in the above mentioned sections are declared in the QB45 include file called qb45comm.bas.

Care must be taken when QB45 strings are passed to COMM-DRV/LIB . In general when calling any COMM-DRV/LIB function, the string address must be calculated as shown in the example below.

```
stat = ser.rs232.putbyte(port, SADD(ch$) + VARSEG(ch$) * 65536)
```

As a matter of compleleness we will map the C data types to the Visual Basic data types. This may be useful to the few who are interested in getting more technical with **QB45** than required.

C Declaration	QuickBasic Declaration
char var	DIM var as STRING * 1
unsigned char var	DIM var as STRING * 1
signed char var	DIM var as STRING * 1
int var	DIM var as INTEGER
unsigned int var	DIM var as INTEGER
signed int var	DIM var as INTEGER
short var	DIM var as INTEGER
unsigned short var	DIM var as INTEGER
signed short var	DIM var as INTEGER
long var	DIM var as LONG
unsigned long var	DIM var as LONG
signed long var	DIM var as LONG

Microsoft Professional Basic 7.x Applications

Interfacing **COMM-DRV/LIB** to a Microsoft Professional Basic 7.x (**BC7**) application is very simple. The libraries are actually linked into the **BC7** applications. All the files related to **BC7** application development are in subdirectory **bc7** found in the **COMM-DRV/LIB** installation directory.

The file **bc7cdrv.lib** is the library with the serial communication functions that is linked to the application when compiled to an .exe program. The file **bc7cdrv.qlb** is the quick library that should be loaded by the **BC7** environment when developing the application.

The file **bc7cdrv2.lib** is a library that contains all the **COMM-DRV/Lib** serial communication routines without the Microsoft C runtime library. If you are having duplicate definitions when linking several third party libraries, you should use this library for linking the basic program into an executable.

There are several example programs in the **bc7** of the **COMM-DRV/LIB** installation directory showing how to use **COMM-DRV/LIB** with **BC7**. Among them are the files **bc7exam.bas** and **bc7exm2.bas**.

All the functions detailed in the major section of the manual, ***Using The C Library Interface***, and its sub-sections, may be used by the BC7 application. The equivalent declaration to all the functions and definitions detailed in the above mentioned sections are declared in the **BC7** include file called **bc7comm.bas**.

Care must be taken when **BC7** strings are passed to **COMM-DRV/LIB** . In general when calling any **COMM-DRV/LIB** function, the string address must be calculated as shown in the example below.

```
stat = ser.rs232.putbyte(port, SADD(ch$) + SSEG(ch$) * 65536)

or

stat = ser.rs232.putbyte(port, SEGSADD(ch$))
```

It is important to note that all **COMM-DRV/LIB** functions that require a string as output require that a NULL character be added to the end of the string as the following source clip demonstrates.

```
'Set Modem Init String
a$="ATS0=0&C1&D2"+CHR$(0)

'Output the string
stat = PutString(port,SSEGADD(A$))
```

Additionally, all strings should be pre-initialized before calling any function that retrieves a string or a packet of information. If this is not done the **COMM-DRV/LIB** will certainly write over random memory. The following illustrates an example of pre-initializing string variables.

```
'Allocate a 50 byte string.
a$ = SPACE$(50)

'Get  a string from the serial port
stat = GetString(port,50,SSEGADD(a$))
```

Some **COMM-DRV/LIB** functions attempt to allocate memory when called. A **BC7** application generally allocates all of memory when it starts. In order to release some of this memory to allow the **COMM-DRV/LIB** API to allocate memory, the **SETMEM() BC7** function must be used.

As a matter of completeness we will map the C data types to the **BC7** data types. This may be useful to the few who are interested in getting more technical with Visual Basic than required.

C Declaration	Professional Basic For Declaration
char var	DIM var as STRING * 1
unsigned char var	DIM var as STRING * 1
signed char var	DIM var as STRING * 1
int var	DIM var as INTEGER
unsigned int var	DIM var as INTEGER
signed int var	DIM var as INTEGER
short var	DIM var as INTEGER
unsigned short var	DIM var as INTEGER
signed short var	DIM var as INTEGER

long var	DIM var as LONG
unsigned long var	DIM var as LONG
signed long var	DIM var as LONG

Compilers and Tools Not Listed Specifically

COMM-DRV can be used with all compilers and tools that allow its applications to be linked with Windows 3.x, Windows NT, or Windows 95 DLLs (e.g., **Paradox, Delphi™, Foxpro,** etc.). Review the readme file (**readmecl.doc**) for information regarding tools and compilers not listed in this manual.

Using COMM-DRV/Lib with COMM-DRV/VxD

COMM-DRV/Lib can be integrated with WCSC's ultra high speed serial communication VxD. The ultra high speed VxD is sold separately under the name **COMM-DRV/VxD**. This VxD allows **COMM-DRV/Lib** to receive serial data in excess of 115.2k baud with the 8250/16450/16550/16650 family of UARTs on several ports concurrently under **Windows 3.x** and **Windows 95.**

Using **COMM-DRV/Lib** with **COMM-DRV/VxD** requires no real change in the application programmer's code. The only difference is that when specifying the cardtype, the constant **CARD_WCSCVXD** instead of **CARD_NORMAL** or any other.

Example1:

```
// The first thing to be done is to initialize the port
if ((stat = InitializePort(port,subport,COMADR,COMIRQ,
CARD_WCSCVXD,0,2048,1024,0)) != RS232ERR_NONE)
      {
      printf("Error #%d initializing serial port\n",stat);
      exit(1);
      }
```

Example2:

```
pcb.ser_rs232_base   = 0x3f8;
pcb.irq              = 4;
pcb.baud             = BAUD9600;
                     :
pcb.cardtype      = CARD_WCSCVXD;
if ((stat = ser_rs232_setup(port,&pcb)) != RS232ERR_NONE)
        {
        printf("Error # %d\n");

        /* Take remedial action */
        }
```

Using COMM-DRV/Lib with COMM-DRV/Dos

COMM-DRV/Lib can be integrated with WCSC's state of the art serial communication TSR. The TSR is sold separately as a part of the product **COMM-DRV/Dos**. When the TSR is used, **COMM-DRV/LIB** gets all its serial communication services from the TSR by dynamically linking to it at runtime.

Dynamically linking to the TSR is useful for multiport DOS applications that are multitasked by multitaskers like Desqview or Windows DOS boxes. Since multiport cards generally share one interrupt, it is necessary that there be only one software object controlling the entire multiport card's behavior. The **COMM-DRV/Dos** TSR serves that purpose. In general, the TSR should be loaded before Windows or Desqview is started. The applications then runs within a Desqview window or a Windows DOS box.

One great advantage of using this procedure is that while you application is running in one window(task), you can monitor statistics on that same port from another window. In fact, **COMM-DRV/Dos** includes a monitor that can allow you to snoop at byte counts, data rates, error rates, and much more while the application is running.

There is very little difference in telling your application to use the TSR as opposed to its own built in routines. The procedure follows.

Example:

```
#define  COMMDRV_DRIVER
#include <comm.h>

main()
{
int      stat;

// Dynamically link to the TSR
if ((stat = ser_rs232_init()) != RS232ERR_NONE)
        {
        printf("Error #%d dynamically linking to TSR\n",stat);

// Program as usual from this point on
                :
                :
}
```


When creating the MS-DOS application with the Microsoft C/C++ compilers, compile your application, then link with the libraries **commdms.lib**, and **libsms.lib** for the small and compact memory models. Link with the libraries **commdml.lib**, and **libsml.lib** if you are using the medium, large, or huge memory models. The include file **comm.h** must be included in every module that makes calls to the **COMM-DRV/LIB** libraries.

When creating the MS-DOS application with the Borland C/C++ compilers, compile your application, then link with the libraries **commdbs.lib**, and **libsbs.lib** for the small and compact memory models. Link with the libraries **commdbl.lib**, and **libsbl.lib** if you are using the medium, large, or huge memory models. The include file **comm.h** must be included in every module that makes calls to the **COMM-DRV/LIB** libraries.

Rebuilding The Libraries

COMM-DRV/LIB is distributed with batch files to rebuild the libraries. There are two batch files. **makemlib.bat** should be used to rebuild the Microsoft libraries. **makeblib.bat** should be used to rebuild the Borland libraries.

makemlib.bat and **makeblib.bat** places all the compiled objects and executables in the current directory. You should create a separate directory to build your libraries.

To build any Microsoft library type

install_path\bin\makemlib ***library_name install_path***

where

library_name is the name of the library to rebuild.
install_path is the root installation directory of **COMM-DRV/LIB**.

Example:

c:\commdrv\bin\makemlib commdrml.lib c:\commdrv

To build any Borland library type

install_path\bin\makeblib ***library_name install_path***

where

library_name is the name of the library to rebuild.
install_path is the root installation directory of **COMM-DRV/LIB**.

Example:

c:\commdrv\bin\makeblib commdrbl.lib c:\commdrv

Basic Definitions And Conventions

At this point it is important that we go over some basic definitions that we will be using throughout this manual. This section will give you a basic overview of serial communications. It is not a replacement for any good book on serial communication. It is however all that should be necessary to use the **COMM-DRV** package.

Serial Communication/A Basic Tutorial

Serial communication is basically the conversion of a 5 to 8 bit byte data to a stream of digital data and back. In effect, a byte may transmitted as its binary equivalent over a single wire or received over a single wire. The advantage to this method of transmission is that less wires are required to transmit data which means this method of communication is much cheaper and simpler than transmitting the data one full byte at one time. It is the responsibility of an electronic device called the Universal Asynchronous Receiver Transmitter(**UART**) to take a 5 to 8 bit byte and convert it to a serial data stream and back.

At any given time the signal on a serial line(wire) is either at a positive voltage (logical 1), also known as the ***MARK*** state, or at a negative voltage (logical 0), also known as the **SPACE** state. Before the first bit of any byte is sent, the signal drops to a logical 0 to indicate the start of a stream of data. This logical zero at the start of the stream is called the ***START BIT***. Following this ***START BIT*** the signal fluctuates between logical 0 and logical 1 depending on the character being transmitted. At the end of the transmission of the character, the signal returns to the ***MARK*** state. The **MARK** state immediately after the transmission of a stream of data is called a ***STOP BIT***.

Figure 2 gives a graphical example of the above.

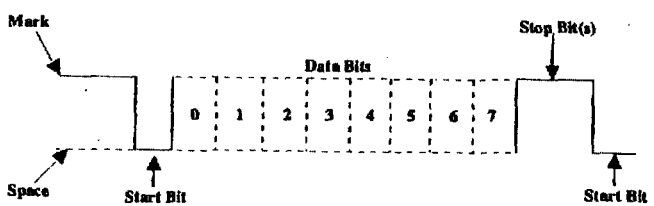


Figure 2

At this point you may want to know what is the duration of a bit. In other words, how long does the signal stay in a particular state to define a bit. The answer is simple. It is dependent on the baud rate. The baud rate is the number of times the signal can switch states in one second. Therefore, if the line is operating at 9600 baud, the line can switch states 9,600 times per second. This means each bit has the duration of 1/9600 of a second or about 104.17 µsec.

When transmitting a character there are other characteristics other than the baud rate that must be known or that must be setup. These characteristics define the entire interpretation of the data stream.

The first characteristic is the length of the byte that will be transmitted. This length in general can be anywhere from 5 to 8 bits.

The second characteristic is parity. The parity characteristic can be even, odd, mark, space, or none. If even parity, then the last data bit transmitted will be a logical 1 if the data transmitted had an odd amount of 1 bits. If odd parity, then the last data bit transmitted will be a logical 1 if the data transmitted had an even amount of 1 bits. If mark parity, then the last transmitted data bit will always be a logical 1. If space parity, then the last transmitted data bit will always be a logical 0. If no parity then there is no parity bit transmitted.

The third characteristic is the amount of stopbits. This value in general is 1, 1½, or 2.

Assuming we wanted to send the letter 'A' over the serial port. The binary representation of the letter 'A' is 01000001. Remembering that bits are transmitted from least significant bit(LSB) to most significant bit(MSB), the bit stream transmitted would be as follows for the line characteristics 8 bits, no parity, 1 stop bit, 9600 baud.

LSB (1 1 0 0 0 0 0 1 0 0) MSB

The above represents

(Start Bit) (Data Bits) (Stop Bit)

To calculate the actual byte transfer rate simply divide the baud rate by the number of bits that must be transferred for each byte of data. In the case of the above example, each character requires 10 bits to be transmitted for each character. As such, at 9600 baud, up to 960 bytes can be transferred in one second.

The above discussion was concerned with the "electrical/logical" characteristics of the data stream. We will expand the discussion to line protocol.

Serial communication can be half duplex or full duplex. Full duplex communication means that a device can receive and transmit data at the same time. Half duplex means that the device cannot send and receive at the same time. It can do them both, but not at the same time. Half duplex communication is all but outdated except for a very small focused set of applications.

Half duplex communication needs at a minimum two wires, signal ground and the data line. Full duplex communication needs at a minimum three wires, signal ground, transmit data line, and receive data line. The RS232 specification governs the physical and electrical characteristics of serial communications. This specification defines several additional signals that are asserted(set to logical 1) for information and control beyond the data signals and signal ground.

These signals are the Carrier Detect Signal(**CD**), asserted by modems to signal a successful connection to another modem, Ring Indicator(**RI**), asserted by modems to signal the phone ringing, Data Set Ready(**DSR**), asserted by modems to show their presence, Clear To Send(**CTS**), asserted by modems if they can receive data, Data Terminal Ready(**DTR**), asserted by terminals to show their presence, Request To Send(**RTS**), asserted by terminals if they can receive data. The section ***RS232 Cabling*** describes these signals and how they are connected.

The above paragraph eluded to hardware flow control. Hardware flow control is a method that two connected devices use to tell each other electronically when to send or when not to send data. A modem in general drops (logical 0) its **CTS** line when it can no longer receive characters. It re-asserts it when it can receive again. A terminal does the same thing instead with the **RTS** signal. Another method of hardware flow control in practice is to perform the same procedure in the previous paragraph except that the **DSR** and **DTR** signals are used for the handshake.

Note that hardware flow control requires the use of additional wires. The benefit to this however is crisp and reliable flow control. Another method of flow control used is known as software flow control. This method requires a simple 3 wire serial communication link, transmit data, receive data, and signal ground. If using this method, when a device can no longer receive, it will transmit a character that the two devices agreed on. This character is known as the **XOFF** character. This character is generally a hexadecimal 13. When a device can receive again it transmits an **XON** character that both devices agreed to. This character is generally a hexadecimal 11.

COMM-DRV has the ability to perform all the above mentioned flow control protocols internally, and without user intervention.

There is a special line condition that should be mentioned here. Whenever the transmit signal remains in the **SPACE** state for more than the time it takes to transmit a character, it is called a **BREAK**. This is usually used to get the attention of another device.

RS232 Cabling

RS232 cabling is one of the simplest parts of serial communications that is made difficult by the preponderance of poorly made adapters like null modems. Following are a few rules of thumb. This discussion will simply deal with the DB-25 standard connectors and cables.

Serial devices are shipped in two basic configuration. These are DCE (data communications equipment) or DTE (data terminal equipment). Devices like terminals are considered DTEs, while devices like modems are considered DCEs. When connecting devices using standard straight through cables (a straight through cable is a cable where pin 1 on one side of the cable is directly connected to pin one on the other side) it is assumed that one device is a DCE device while the other is a DTE. If both devices are DTEs or both are DCEs, then a null modem is needed in between the two cables. The null modems simply remap the pins.

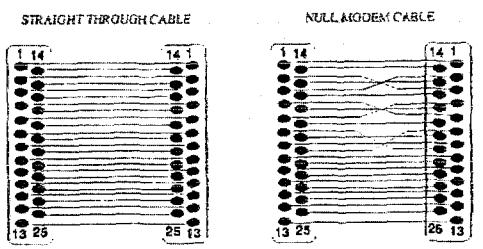


Figure 3

Most serial ports on PCs or multiport cards are shipped assuming they will be connected to a modem. **Figure 3** shows the configuration for the prevalent two types of cables.

The relevant pin name and numbers are:

DB25 Pinouts

Pin	Signal Symbol	Signal Name
1	NC	Chassis Ground
2	TX	Transmit Data
3	RX	Receive Data
4	RTS	Request To Send
5	CTS	Clear To Send
6	DSR	Data Set Ready
7	GND	Signal Ground
8	CD	Carrier Detect
20	DTR	Data Terminal Ready
22	RI	Ring Indicator

DB9 Pinouts

Pin	Signal Symbol	Signal Name
1	CD	Carrier Detect
2	RX	Receive Data
3	TX	Transmit Data
4	DTR	Data Terminal Ready
5	GND	Signal Ground
6	DSR	Data Set Ready
7	RTS	Request To Send
8	CTS	Clear To Send
9	RI	Ring Indicator

Port address, Port number, & Device explanation

Note that when we refer to ports, in general we are referring simply to some number that will refer to a specific UART. We use that number to keep track of a myriad of information about some UART(physical address, IRQ, buffers, etc.). When the **COMM-DRV** TSR(**COMM-DRV/DOS**) is loaded, this number is used to talk to a serial port that was explicitly mapped to the port number. Likewise, when **COMM-DRV** is linked into the application, the port number is used to map to, and identify a serial port.

The physical address of the UART(serial port) is the hardware address that that UART is residing at on the PC Bus. These are numbers that can range between 000 and 3ff hex on ATs and higher on EISA and MCA machines.. In the case of a smart multiport card, this is the address of the port used to access the smart multiport card.

The COM device is yet another level of abstraction. The COM device in general have names like COM1, COM2, COM#, DOG, CAT, etc. These are true MS-DOS character device drivers. Remember a character device driver could be considered a set of subroutines that the operating system knows about that make any piece of computer hardware perform input and output like any standard file. **COMM-DRV/DOS** can create devices by installing the **COMMDRV.DRV** device driver.

An IRQ is the interrupt request line that a serial port(UART) uses to tell the CPU that it needs to be serviced. That is, a character has come in, a character was transmitted, or some UART state changed. **COMM-DRV** uses the IRQ to acquire data, statistics, and to transmit data.

Note that by convention IBM PCs and compatibles map COM1 to port 0 on IRQ 4 at physical address 3f8. This again is only by convention. **COMM-DRV** allows you to remap any port, COM device, IRQ, or physical port address. Therefore one could remap COM1 to port 5 on IRQ 7 at physical address 2a0. Alternatively it could map device COMMDEV1 to port 10 on IRQ 14 at physical address 280.

Note that when the **COMM-DRV** objects are linked into the application, there are no character devices to address.

Using The C Library Interface

The following sections detail the **COMM-DRV/LIB** API. The following API descriptions are "C" centric, however their functional specification is relevant to all languages supported by **COMM-DRV/LIB**.

High Level Functions

These functions are the high level **COMM-DRV/LIB** functions. Programming via the use of the high level API is the preferred method of using **COMM-DRV/LIB**.

BytesInReceiveBuffer()	Returns the number of bytes in the receive buffer.
BytesInTransmitBuffer()	Returns the number of bytes in the transmit buffer.
CdrvCheckTime()	Determine if time expired form a previous call to CdrvSetTime().
CdrvCrc16()	Returns the 16bit CRC of a packet.
CdrvCrc32()	Returns the 32bit CRC of a packet.
CdrvDelay()	Delay specified time.
CdrvGetPcb()	Returns a pointer to the port's PCB.
CdrvReturnStringAddress()	Returns the address of the passed string. This function is used in languages like Visual Basic to get addresses of strings since they are not ready available.
CdrvSetTime()	Sets a timer to a specified delay. Expiration is tested by CdrvCheckTime().
CdrvSetTimeoutFunction()	Sets the address of a function that gets called when Delay() or CdrvCheckTime() is called.
CdrvSetTimerResolution()	Sets the timer resolution used by CdrvSetTime() and Delay().
DtrOff()	Turns DTR off.
DtrOn()	Turns DTR on.
FlushReceiveBuffer()	Discards the contents of the receive buffer.
FlushTransmitBuffer()	Discards the contents of the transmit buffer.
GetByte()	Gets a byte from the receive buffer.
GetNumber()	Converts a received ASCII number from the serial port into a binary value.
GetPaceTime()	Get the current inter-character pace time.
GetPacket()	Gets a packet from receive buffer.
GetString()	Gets a carriage return, line feed, or null terminated string from receive buffer.
GetTimeout()	*Get the current time-out value.*
InitializePort()	Initializes the serial port(Allocate buffers, cardtype, etc.).
IsAllDataOut()	Returns true if all data has left the UART/Serial device.
IsBreak()	Returns true if a break signal was detected.
IsCarrierDetect()	Returns true if carrier detected.
IsCts()	Returns true if CTS signal high.
IsDsr()	Returns true if DSR signal high.
IsFramingError()	Returns true if a framing error occurred.
IsInputOverrun()	Returns true if the COMM-DRV receive buffer was overrun.

IsOverrunError()	Returns true if the UART receive register was overrun.
IsParityError()	Returns true if a parity error occurred.
IsPortAvailable()	Determine if a particular port is in use.
IsReceiveBufferEmpty()	Returns true if receive buffer is empty.
IsRing()	Returns true if ring detected.
IsTransmitBufferEmpty()	Returns true if transmit buffer is empty.
PeekChar()	Returns the next character from receive buffer non-destructively.
PutByte()	Queues a byte for transmission.
PutPacket()	Queues a packet for transmission.
PutString()	Outputs a null terminated string.
ReceiveBufferSize()	Returns the receive buffer size.
RtsOff()	Turns RTS off.
RtsOn()	Turns RTS on.
SendBreak()	Sends a break signal.
SetBaud()	Sets new baud rate.
SetDataStreamFunction()	This function causes functions like WaitFor() to call the specified function for all the data that it reads from the serial port.
SetFlowControlCharacters()	Sets characters used for flow control.
SetFlowControlThreshold()	Sets high and low receive buffer thresholds.
SetPaceTime()	Sets the current inter-character pace time.
SetPortCharacteristics()	Set line control parameters(baudrate, length, parity, etc.).
SetSpecialBehavior()	This functions changes the behavior of other functions.
SetTimeout()	Sets the current transmit/receive time-outs.
SpaceInReceiveBuffer()	Returns space unused in receive buffer.
SpaceInTransmitBuffer()	Returns space unused in transmit buffer.
TransmitBufferSize()	Returns transmit buffer size.
UnInitializePort()	Undo the effects of InitializePort() (unhook vectors. release memory, etc.).
WaitFor()	Outputs a string and waits for a matching response.
WaitForFixed()	Same as WaitFor() except that it does not depend on NULL terminated string, but fixed size character array of data.
WaitForPeek()	Non-destructive WaitFor().
WaitForPeekFixed()	Same as WaitForPeek() except that it does not depend on NULL terminated string, but fixed size character arrays of data.
WaitForPeekTable()	Outputs a string and non-destructively waits for a matching response to a table of strings.
WaitForPeekTableFixed()	Same as WaitForPeekTable() except that it does not depend on NULL terminated string, but fixed size character array of data.
WaitForTable()	Destructive WaitForPeekTable().

WaitForTableFixed()	Same as WaitForTable() except that it does not depend on NULL terminated string, but fixed size character array of data.

BytesInReceiveBuffer

Description-

Returns the number of bytes in the receive buffer.

Syntax-

count = BytesInReceiveBuffer(port);

On Entry-

int port;

Port previously opened with **InitializePort()**.

On Exit-

int count;

Number of bytes in the receive buffer.

else

-1 if port is not active.

See Also-

IsReceiveBufferEmpty()
ReceiveBufferSize()
SpaceInReceiveBuffer()

Example-

```
#include <comm.h>

int             count;
int             port=0;

if ((count = BytesInReceiveBuffer(port)) == -1)
        {
        printf("Port was not initialized\n");

        /* Take remedial action */
        }
else
        printf("Number of byte =>%d\n",count);
```

BytesInTransmitBuffer

Description-

Returns the number of bytes in the transmit buffer.

Syntax-

count = BytesInTransmitBuffer(port);

On Entry-

int port;

Port previously opened with **InitializePort()**.

On Exit-

int count;

Number of bytes in the transmit buffer.

else

-1 if port does not exist.

See Also-

IsTransmitBufferEmpty()
SpaceInTransmitBuffer()
TransmitBufferSize()

Example-

```
#include <comm.h>

int             count;
int             port=0;

if ((count = BytesInTransmitBuffer(port)) == -1)
        {
        printf("Port was not initialized\n");

        /* Take remedial action */
        }
else
        printf("Number of bytes =>%d\n",count);
```

CdrvCheckTime

Description-

This function returns whether a time period set by a previous call to **CdrvSetTime()** has expired. Additionally, if the **CdrvSetTimeoutFunction()** was used to install a time-out function, then that time-out function will be called each time this routine is called. That time-out function can control the status returned by this function.

Syntax-

stat = CdrvCheckTime(timerblk);

On Entry-

unsigned int *timerblk;

Timer controlled block that was previously initialized by **CdrvSetTime()**.

On Exit-

int stat;

If the **CdrvSetTimeoutFunction()** was not called to install a user time-out function then the return values are as follows.

0	If timer expired.
!=0	If timer not expired.

If the **CdrvSetTimeoutFunction()** was called to install a user function, this function returns the return value of the user function immediately.

See Also-

CdrvDelay()
CdrvSetTime()
CdrvSetTimeoutFunction()
CdrvSetTimerResolution()

Example-

```
#include <comm.h>

int     stat;
int     port=0;
int     time_interval=18;
int     timerblk[10];

if ((stat = CdrvSetTime(port,time_interval,timerblk)) == -1)
        {
        printf("Error setting timer\n");

        /* Take remedial action */
        }

while(CdrvCheckTime(timerblk) != 0)
        {
        /* Do desired work for "timer not expired yet" */

        }
```

CdrvCrc16

Description-

This function returns a 16 bit CRC for the passed packet.

Syntax-

crc=CdrvCrc16(count, buffer);

On Entry-

int count;

Number of bytes in the passed buffer.

char *buffer;

Pointer to the buffer with the data to CRCed.

On Exit-

unsigned short crc;

16 bit CRC of the passed packet.

See Also-

CdrvCrc32()

Example-

```
#include <comm.h>

int             port=0;
int             count;
unsigned short  crc;
char            buf[256];

if ((count = GetPacket(port,sizeof(buf),buf)) == -1)
        {
        printf("Port not initialized error\n");

        /* Take remedial action */
        }

crc=CdrvCrc16(count, buffer);

printf("16 Bit CRC =>%d\n",crc);
```

CdrvCrc32

Description-

This function returns a 32 bit CRC for the passed packet.

Syntax-

crc=CdrvCrc32(count, buffer);

On Entry-

int count;

Number of bytes in the passed buffer.

char *buffer;

Pointer to the buffer with the data to CRCed.

On Exit-

unsigned long crc;

32 bit CRC of the passed packet.

See Also-

CdrvCrc16()

Example-

```
#include <comm.h>

int             count;
int             port=0;
unsigned short  crc;
char            buf[256];

if ((count = GetPacket(port,sizeof(buf),buf)) == -1)
        {
        printf("Port not initialized error\n");

        /* Take remedial action */
        }

crc=CdrvCrc32(count, buffer);

printf("32 Bit CRC =>%d\n",crc);
```

CdrvDelay

Description-

This function delays for the specified amount of time. Additionally, if the **CdrvSetTimeoutFunction()** was used to install a time-out function, then that time-out function will be called each time this routine is called. That time-out function can control the status returned by this function.

Syntax-

stat = CdrvDelay(port,delay_time);

On Entry-

int port;

Port previously opened with **InitializePort()**. If this value is -1 then this function will not attempt to validate the port or will not call any user installed function.

int delay_time;

Number of tics to delay. Note that the tic resolution is dependent on the setting by the function **CdrvSetTimerResolution()**.

On Exit-

int stat;

If the **CdrvSetTimeoutFunction()** was not called to install a user time-out function then the return values are as follows.

0	If successful.
!=0	If port was not initialized.

If the **CdrvSetTimeoutFunction()** was called to install a user function, and the return value from the user function is **0**, this function returns immediately.

See Also-

CdrvCheckTime()
CdrvSetTime()
CdrvSetTimerResolution()
CdrvSetTimeoutFunction()

Example-

```
#include <comm.h>

int     ret;
int     port=0;
int     timeout=10;

if ((ret = CdrvDelay(port,timeout)) == -1)
        {
        printf("Port was not initialized \n");

        /* Take remedial action */
        }
```

CdrvGetPcb

Description-

This function gets a pointer to the port control block for the specified port. This function is provided for completeness and is generally not used by application programmers.

Syntax-

pcbptr = CdrvGetPcb(port);

On Entry-

int port;

Port previously opened with **InitializePort()**.

On Exit-

struct port_param *pcbptr;

Pointer to the port control block for the specified port. The elements of this structure that would be of interest is detailed in **Appendix C**.

See Also-

Example-

```
#include <comm.h>

int                 port=0;
struct port_param   *pcbptr;
#define             NULLPTR (struct port_param *)0;

if ((pcbptr = CdrvGetPcb(port)) == NULLPTR)
        {
        printf("Port was not initialized \n");

        /* Take remedial action */
        }
```

CdrvReturnStringAddress

Description-

This function is used to return the address of the passed string. It is necessary since certain languages do not allow access to their actual location.

Syntax-

address = CdrvReturnStringAddress(str);

On Entry-

char *str;

Address of string to pass.

On Exit-

unsigned long address;

Argument description.

See Also-

Example-

'Visual Basic Example

address& = CdrvReturnStringAddress(a$)

CdrvSetTime

Description-

This function sets a timer to a specified interval. The timer is then queried with the **CdrvCheckTime()** function to determine if the timer interval has expired.

Syntax-

stat = CdrvSetTime(port,time_interval,timerblk);

On Entry-

int port;

Port previously opened with **InitializePort()**. If this value is -1 then this function will not attempt to validate the port.

int time_interval;

Length of time interval. Note that the tic resolution is dependent on the setting by the function **CdrvSetTimerResolution()**.

unsigned int timerblk[10];

An array of integers that is initialized by this function and is subsequently used when calling the **CdrvCheckTime()** function.

On Exit-

int stat;

-1 If port was not initialized.

See Also-

CdrvCheckTime()
CdrvDelay()
CdrvSetTimeoutFunction()
CdrvSetTimerResolution()

Example-

```
#include <comm.h>

int     stat;
int     port=0;
int     time_interval=18;
int     timerblk[10];

if ((stat = CdrvSetTime(port,time_interval,timerblk)) == -1)
        {
        printf("Error setting timer\n");

        /* Take remedial action */
        }

while(CdrvCheckTime(timerblk) != 0)
        {
        /* Do desired work for "timer not expired yet" */

        }
```


CdrvSetTimeoutFunction

Description-

This function instructs the **CdrvDelay()** and **CdrvCheckTime()** functions to call the specified user function when awaiting the time-out to occur. The function is called at least once when either of the above functions is called. Additionally, all functions that are dependent on the time-out set by **SetTimeout()** will cause the user function set by this function to get called.

Syntax-

stat = CdrvSetTimeoutFunction(port,Func);

On Entry-

int port;

Port previously opened with **InitializePort()**.

int (*Func)(int port);

Pointer to a user function that is called while awaiting a time-out set by **CdrvDelay()** or a time-out being checked by **CdrvCheckTime()**. The user function is called at least once when either of the above functions is called. It will be called repeatedly for the duration of the time interval set with **CdrvDelay()**. It will be called each time **CdrvCheckTime()** is called to determine if a time-out has occurred. The user function should return 0 to abort a delay in progress(**CdrvDelay()**), to cause **CdrvCheckTime()** to indicate that the timer has expired, or to cause any function dependent on the time-out set by **SetTimeout()** function to be aborted.

On Exit-

int stat;

-1 If port was not initialized.

See Also-

CdrvCheckTime()
CdrvDelay()
CdrvSetTime()
CdrvSetTimerResolution()
SetTimeout()

Example-

```
#include <comm.h>

int     stat;
int     port=0;
int     time_interval=18;
int     count;
char    buf[256];


if ((stat = CdrvSetTimeoutFunction(port,TimeoutFunction) == -1)
        {
        printf("Error setting timeout function\n");

        /* Take remedial action */
        }

if ((stat = SetTimeout(port,time_interval)) == -1)
        {
        printf("Error setting timeout\n");

        /* Take remedial action */
        }

/* Get packet will wait for up to "time_interval" tics to return */
/* or when the user hits the escape key or button */
count = GetPacket(port,sizeof(buf),buf);

/*******************************************************/

int TimeoutFunction(int port)
{
if (user hit escape cause an abort)
        return(0);
else
        return(1);
}
```

CdrvSetTimerResolution

Description-

This function sets the resolution of the timer tic used by the **CdrvSetTime()** and **CdrvDelay()** functions.

Syntax-

stat = CdrvSetTimerResolution(port,resolution);

On Entry-

int port;

Port previously opened with **InitializePort()**. If this value is -1 then this function will not attempt to validate the port.

int resolution;

0 *1/18* second per tic.
1 1/1000 second per tic(1 millisecond per tic).

On Exit-

int stat;

-1 If port was not initialized.

See Also-

CdrvCheckTime()
CdrvDelay()
CdrvSetTime()
CdrvSetTimeoutFunction()
SetTimeout()

Example-

```
#include <comm.h>

int     stat;
int     port=0;
int     resolution=0;
int     time_interval=18;
int     timerblk[10];

if ((ret = CdrvSetTimerResolution(port,resolution)) == -1)
        {
        printf("Error setting timer resolution\n");

        /* Take remedial action */
        }

if ((ret = CdrvSetTime(port,time_interval,timerblk)) == -1)
        {
        printf("Error setting timer\n");

        /* Take remedial action */
        }

while(CdrvCheckTime(timerblk) != 0)
        {
        /* Do desired work for "timer not expired yet" */

        }
```

DtrOff

Description-

Turns the DTR signal off.

Syntax-

stat = DtrOff(port);

On Entry-

int port;

Port previously opened with **InitializePort()**.

On Exit-

int stat;

-1 If port was not initialized.

See Also-

DtrOn()
RtsOff()
RtsOn()

Example-

```
#include <comm.h>

int             port=0;
int             stat;

if ((stat = DtrOff(port)) == -1)
        {
        printf("Port not initialized\n");

        /* Take remedial action */
        }
```

DtrOn

Description-

Turns the DTR signal on.

Syntax-

stat = DtrOn(port);

On Entry-

int port;

Port previously opened with **InitializePort()**.

On Exit-

int stat;

-1 If port was not initialized.

See Also-

DtrOff()
RtsOff()
RtsOn()

Example-

```
#include <comm.h>

int             port=0;
int             stat;

if ((stat = DtrOn(port)) == -1)
        {
        printf("Port not initialized\n");

        /* Take remedial action */
        }
```

FlushReceiveBuffer

Description-

Discards the contents of the receive buffer.

Syntax-

stat = FlushReceiveBuffer(port);

On Entry-

int port;

Port previously opened with **InitialIzePort()**.

On Exit-

int stat;

-1 If port was not initialized.

See Also-

IsReceiveBufferEmpty()
ReceiveBufferSize()
SpaceInReceiveBuffer()

Example-

```
#include <comm.h>

int             port=0;
int             stat;

if ((stat = FlushReceiveBuffer(port)) == -1)
        {
        printf("Port not initialized\n");

        /* Take remedial action */
        }
```

FlushTransmitBuffer

Description-

Discards the contents of the Transmit buffer.

Syntax-

stat = FlushTransmitBuffer(port);

On Entry-

int port;

Port previously opened with **InitializePort()**.

On Exit-

int stat;

-1 If port was not initialized.

See Also-

IsTransmitBufferEmpty()
TransmitBufferSize()
SpaceInTransmitBuffer()

Example-

```
#include <comm.h>

int             port=0;
int             stat;

if ((stat = FlushTransmitBuffer(port)) == -1)
        {
        printf("Port not initialized\n");

        /* Take remedial action */
        }
```

GetByte

Description-

Gets a byte from the receive buffer. This function will wait for up to the amount of time set by **SetTimeout()** before returning failure if a byte is not present.

Syntax-

byte = GetByte(port);

On Entry-

int port;

Port previously opened with **InitializePort()**.

On Exit-

int byte;

The returned byte.

else

-1 If port not initialized or no byte to be read.

See Also-

GetPacket()
GetString()

Example-

```
#include <comm.h>

int     data;
int     i;
int     port=0;
char    buf[256];


for (i=o;i<sizeof(buf);i++)
        {
        if ((data=GetByte(port)) == -1)
                {
                printf("No more bytes available\n");

                /* End of serial data collection */
                break;
                }
        else
                buf[i] = (char)data;
        }
```

GetNumber

Description-

This function gets a decimal number in ASCII digits from the serial port. It starts collecting the number from the first decimal digit in the stream until the first non-decimal digit or till the buffer size passed. (e.g., if the stream is "abcdefgh12345xyz678" the string "12345" is returned, "abcdefgh" and "x" are discarded).

Syntax-

stat = GetNumber(port,count,str);

On Entry-

int port;

Port previously opened with **InitializePort()**.

int count;

One less than the size of the buffer to store returned string.

char *str;

Pointer to buffer to place data.

On Exit-

int stat;

0 if no error.

See Also-

GetByte()
GetPacket()
GetString()

Example-

```
#include <comm.h>

char    str[12];
int     port=0;
int     count=sizeof(str) - 1;

// Get the number
if (GetNumber(port,count,str) != 0)
        {
        // Error reading number from the serial port

        }
```

GetPaceTime

Description-

Get the current inter-character pace time. The time retrieved is in timer tics. The timer tic resolution is set with the **CdrvSetTimerResolution()** function.

Syntax-

pacetime = GetPaceTime(port);

On Entry-

int port;

Port previously opened with **InitializePort()**.

On Exit-

unsigned int pacetime;

The current pacetime in timer tics.

else

0xffff If port was not initialized.

See Also-

CdrvSetTimerResolution()
SetPaceTime()

Example-

```
#include <comm.h>

unsigned int    oldpacetime;
unsigned int    newpacetime=1;
int             port=0;


/* Save old pace time */
if ((oldpacetime=GetPaceTime(port)) == -1)
        {
        printf("Port not initialized\n");

        /* Take remedial action */
        }

/* Set a new pace time */
SetPaceTime(port,newpacetime);

/* Restore the old pace time */
SetPaceTime(port,oldpacetime);
```

ModemWaitForCall

Description-

This functions waits for a call, answers the phone, and returns when two modems have successfully connected.

Syntax-

stat = ModemWaitForCall(Port,RingCount,Mode,Fnc);

On Entry-

int port;

Port previously opened with **InitIalIzePort()**.

int RingCount;

Number of rings before answering the phone.

int *Mode*;

0 Puts modem in auto answer mode to answer on specified ring count. Determines connect based on receiving the string **"CONNECT"** or the string specified as being the connect string.

1 The modem is set to non auto answer mode. The stream is scanned for the string **"RING"** or the string specified as being the ring string. Determines connect based on receiving the string **"CONNECT"** or the string specified as being the connect string.

int (*fnc)(int port);

Pointer to a function that will be called periodically to allow the application programmer to terminate this function before connect or other failure. The function pointed to by this pointer must return "0" to indicate that this function should not be aborted. Any other return will cause this function to abort and return said return value. The function pointed to by this function should not return a negative value as these are reserved for internal use. If the value of this function pointer is NULL, then no callback function is called and this function will only return on an error or when a connection occurs.

On Exit-

```
int             stat;
```

0	If the modem is connected to another modem.
-1	If a hardware error or modem error has occurred.
Positive Value	Return value if function aborted by application.

See Also-

ModemConnect()

Example-

```
#include  <comm.h>

int       fnc(int port)
{
if (UserHitEscapeKey() == TRUE)
          return(1);
return(0);
}

main()
{
int       Port=0;
int       RingCount=1;
int       Mode=0;
int       stat;

if ((stat = ModemWaitForCall(Port,RingCount,Mode,fnc)) < 0)
          printf("Modem or hardware error has occurred\n");
else
if (stat > 0)
          printf("Application programmer aborted function\n");
else
          printf("Modems are connected\n");
          :
          :
}
```


ModemWaitForRing

Description-

This function waits for the modems "RING" string. This command assumes the modem is in the command state. The modem remains in the command state at the end of this function.

Syntax-

stat = ModemWaitForRing(port,RingCount,Timeout);

On Entry-

```
int             port;
```

Port previously opened with **InitializePort()**.

```
int             RingCount;
```

Number of rings to wait for before return.

```
int             Timeout;
```

Number tics to delay before returning if a "RING" is not detected.

On Exit-

```
int             stat;
```

0	If successful.
else	
-1	If time-out.

See Also-

Dial()
ModemAnswerMode()
ModemAttention()
ModemCommandState()
ModemConnect()
ModemGetCarrierSpeed()
ModemGetConnectSpeed()
ModemHangup()
ModemModifyString()

Example-

```
#include <comm.h>

int             port=0;
int             stat;

//Turn the speaker volume on modem to medium
if ((stat = ModemWaitForRing(port,1,180)) == -1)
        {
        printf("Timeout waiting for a ring\n");

        /* Take remedial action */
        }
```


File Transfer Functions

The following are the file transfer functions.

cdrvxfer_files()	Transmit or receive file(s) with specified protocol to completion.
cdrvxfer_gclose()	This function is used to close any opened files if one of the state driven file transfer calls is stopped abruptly.
cdrvxfer_getfiles()	Receive file(s) with specified protocol. Must be called several times till transfer complete.
cdrvxfer_sendfiles()	Transmit file(s) with specified protocol. Must be called several times till transfer complete.
cdrvxfer_sfiles()	Transmit or receive file(s) with specified protocol to completion(additional features).
CdrvXferCreateDialog()	Creates a file transfer dialog box.
CdrvXferDestroyDialog()	Destroys a file transfer dialog box.
CdrvXferUpdateDialog()	Updates information in file transfer dialog box.
FileTransferDialog()	Enables an automatic dialog to be displayed on some of the file transfer functions.
SetXferParameters()	Set file transfer parameters.
TransferFiles()	Transfer files.

cdrvxfer_files

Description-

Transmit or receive file(s) with specified protocol to completion. This function is used for transmitting and receiving files. It does not return to the caller until the file transfer completes. It does however call the function that the caller passed when the function was called in order to allow the caller to get a progress report on the file transfer. This function **will not** stop other Windows applications from running or prevent messages for the current application from being processed.

Syntax-

stat = cdrvxfer_files(port,direction,protocol,fspec,fnc);

On Entry-

int port;

Port previously opened with **InitializePort()**.

int direction;

0	To send a file.
1	To receive a file.

int protocol;

0	XMODEM
1	XMODEM 1K
2	YMODEM
3	YMODEM Batch
4	YMODEM G
5	YMODEM G Batch
6	ZMODEM
7	XMODEM(Checksum)

char *fspec;

Filename including path to file to transfer. Note that if receiving a file, the file is placed in the directory pertaining to the path portion of the filespec.

int (far *fnc)(int chgflg,char *fname,long bytcnt,int curblk, int errcnt, unsigned rate);

Pointer to routine that is called by this function anytime one of the passed parameters changes. This function may abort the transfer by returning a non-zero value. If this pointer is NULL, then no display function is called.

int chgflg;

Non zero if calling due to a new file being transferred.

char *fname;

Filename being processed.

long bytcnt;

Current file byte count.

int curblk;

Current block being worked on.

int errcnt;

Current error count for file.

unsigned rate;

Current transfer rate in bytes per second.

On Exit-

int stat;

CDRVXFER_ERR_OK	If the file transfer was successful.

CDRVXFER_ERR_SERIAL	A serial I/O error has occurred. This could be caused by a broken serial communication cable, a bad UART or some other serial error.
CDRVXFER_ERR_OS	An MSDOS error occurred. This may be caused by an error opening, writing, reading, or closing a file.
CDRVXFER_ERR_PROTOCOL	A protocol error. This happens when an incompatibility between the file transfer protocol between the two transferring machines occur.
CDRVXFER_ERR_BUSY	File transfer is still in progress.
CDRVXFER_ERR_CANCEL	The remote issued a cancel.
CDRVXFER_ERR_INVPORT	An invalid serial port was selected.
CDRVXFER_ERR_FNAME	An invalid filename was given.
CDRVXFER_ERR_MEMORY	A memory allocation error has occurred. This error may be caused due to a lack of memory. Most compilers causes an executable to allocate all memory on startup. You may want to use the **exehdr** program or some other compiler utility to adjust the maximum amount of memory the executable may allocate.

See Also-

cdrvxfer_getfiles()
cdrvxfer_sendfiles()
cdrvxfer_sfiles()
TransferFiles()

Examples-


```
#include  <comm.h>

main()
{
int             port=0;
int             direction=1;
int             protocol=6;
char            *fspec="e:\\tmp\\*.*";
int             stat;
int             (far *fnc)(int chgflg,char *fname,long bytcnt,int curblk, int errcnt, unsigned rate) =
                DisplayFunction;

stat = cdrvxfer_files(port,direction,protocol,fspec,fnc);
return;
}


//User specified callback function. This function is
//called with file transfer progress information.

DisplayFunction(int chgflg,char *fname,long bytcnt,int curblk, int errcnt, unsigned rate)
{
char   fnam[12+1];

fstrcpy(fnam,fname);  //Filename being transferred

if (fchg != 0)                  //Display on next line only
    printf("\n");       //if moving a new file.

//Display the current transfer statistics
printf("\rNAM=>%-12.12s BLK=>% 5u BYT=>% 7lu ERR=>% 4u R=% 5u Byt/Sec",
fnam,curblk,bytcnt,errcnt,rate);
return(0);
}
```

cdrvxfer_gclose

Description-

This function is used to close opened files and resources when one of the state driven file transfer functions is abruptly ended by the application programmer. Specifically this function should be called if the user stops calling **cdrvxfer_getfiles()**, or **cdrvxfer_sendfiles()**.

Syntax-

cdrvxfer_gclose(ftpcb);

On Entry-

struct cdrvxfer_ftpcb *p;

Pointer to a file transfer port control block. It is assumed that the structure that is being pointed to was initialized as specified in the **cdrvxfer_getfiles()** or **cdrvxfer_sendfiles()** functions.

On Exit-

None

See Also-

cdrvxfer_getfiles()
cdrvxfer_sendfiles()

Example-

```
#include   <comm.h>

main()
{
int                                 port=0;
int                                 protocol=6;
char                                *fspec="e:\\tmp\\*.*";
int                                 stat;
static char                         buf[1024];
static struct cdrvxfer_ftpcb        ftpcb;

//Initialize the file transfer structure input parameters
fstrcpy(ftpcb.fspec,fspec);
ftpcb.port                  = port;
ftpcb.buf                   = buf;
ftpcb.bufsiz                = sizeof(buf);
ftpcb.flag                  = CDRVXFER_FLAG_RECOVER;
ftpcb.rcvtimeout            = 90;
ftpcb.xmttimeout            = 90;
ftpcb.retrycnt              = 3;
ftpcb.state                 = CDRVXFER_RSTATE_INIT;

while ((stat = cdrvxfer_getfiles(protocol,&ftpcb) == CDRVXFER_ERR_BUSY)
                {
                if (ftpcb.flag & CDRVXFER_FLAG_FILEDONE)
                        {
                        printf("\rFilename=>%s ByteCnt=>%ld\n",
                        ftpcb.fspec,ftpcb.numbytes);

                        ftpcb.flag &= (~CDRVXFER_FLAG_FILEDONE);
                        }
                else
                        printf("\rFilename=>%s ByteCnt=>%ld",
                        ftpcb.fspec,ftpcb.numbytes);

                if (UserHitEscapeKey() == TRUE)
                        {
                        // Cleanup if abruptly aborting to an external request
                        cdrvxfer_gclose(&ftpcb);
                        break;
                        }
                }
return;
}
```

cdrvxfer_getfiles

Description-

This function gets a particular file(s) based on the selected file transfer protocol. Note that the file transfer engine is a *state machine*. As such, this routine must be called repeatedly until a return value other than **CDRVXFER_ERR_BUSY** (protocol engine busy so file transfer is still in progress) is returned. This format allows an application to run several file transfers concurrently by calling this function repeatedly with different port numbers.

Syntax-

stat =cdrvxfer_getfiles(protocol,ftpcb);

On Entry-

int protocol;

0	XMODEM
1	XMODEM 1K
2	YMODEM
3	YMODEM Batch
4	YMODEM G
5	YMODEM G Batch
6	ZMODEM
7	XMODEM(Checksum)

struct cdrvxfer_ftpcb *ftpcb;

Pointer to the structure with the information to provide and to receive for the file transfer. Description of pertinent elements follows.

ftpcb->fspec[]

Null terminated filename to use if the filename not given by the particular file transfer protocol. (May contain wildcards for protocols that allow it). If a path is specified then the file(s) received will be placed in the subdirectory specified by the path portion of the **fspec** given.

ftpcb->port

Port file transfer taking place on.

ftpcb->state

Must be set to **CDRVXFER_RSTATE_INIT** before the first call to **cdrvxfer_getfiles()**.

ftpcb->buf

Pointer to the buffer that the file transfer engine will use. This buffer must be at least **1K**.

ftpcb->bufsiz

Size of buffer pointed to by **ftpcb->buf**.

ftpcb->flag

Set to one of the following values if receiving a file.

CDRVXFER_FLAG_RECOVER

If a file with the name of the file being received exists, and the file is shorter than the file being received, then restart the file transfer from where the last transfer left off.

CDRVXFER_FLAG_OVERWRITE

If a file with the name of the file being received exists, overwrite it.

CDRVXFER_FLAG_RENAME

If a file with the name of the file being received exists, change the name of the file being received.

ftpcb->retrycnt

Maximum number of times to retry sending the same failed packet.

ftpcb->rcvtimeout

Number of 1/18th of a second tics to wait for incoming bytes. This value is generally 180(10 seconds).

ftpcb->xmttimeout

Number of 1/18th of a second tics to wait for space to become available in transmission buffer. This value is generally 180(10 seconds).

On Exit-

int stat;

CDRVXFER_ERR_OK	If the file transfer was successful.
CDRVXFER_ERR_SERIAL	A serial I/O error has occurred. This could be caused by a broken serial communication cable, a bad UART or some other serial error.
CDRVXFER_ERR_OS	An MSDOS error occurred. This may be caused by an error opening, writing, reading, or closing a file.
CDRVXFER_ERR_PROTOCOL	A protocol error. This happens when an incompatibility between the file transfer protocol between the two transferring machines occur.
CDRVXFER_ERR_BUSY	If file transfer is still in progress. Note this routine should be constantly called until this code is not returned. The structure elements below are kept updated in between calls to **cdrvxfer_getfiles()**.
CDRVXFER_ERR_CANCEL	The remote issued a cancel.
CDRVXFER_ERR_INVPORT	An invalid serial port was selected.
CDRVXFER_ERR_FNAME	An invalid filename was given.

CDRVXFER_ERR_MEMORY	A memory allocation error has occurred. This error may be caused due to a lack of memory. Most compilers causes an executable to allocate all memory on startup. You may want to use the **exehdr** program or some other compiler utility to adjust the maximum amount of memory the executable may allocate.
ftpcb->blksiz	Size of the block being transferred.
ftpcb->curblk	Current block being transferred.
ftpcb->currenterr	Error count for the file currently being transferred.
ftpcb->error	Current error code(usually **CDRVXFER_ERR_BUSY** during file transfer). This is the same value returned by this function.
ftpcb->exterror	An extended error code.
ftpcb->fhandle	Operating system file handle of file currently being transferred or -1 if file is closed.
ftpcb->fspec	Fully qualified pathname and filename of the file being transferred.
ftpcb->numbytes	Number of bytes transferred in the current file.
ftpcb->offsetbytes	This is the offset in the file where the current file transfer began. The difference between ftpcb->numbytes and this value gives the actual number of bytes transferred in the current file this session. This value is greater than zero for the ZMODEM transfer when crash recovery is in effect.
ftpcb->totalerr	Total of all errors for the current session.

ftpcb->flag	The bit **CDRVXFER_FLAG_FILEDONE** is set when the file transfer on a particular file is complete. This bit should be reset after it is tested. This bit may be used to determine when some statistics need updating(filename, counts, etc.).
ftpcb->f.size	Size of file that is being received. Note that some file transfer programs do not transmit the file size. In said case this value will be zero.
ftpcb->f.date	Date of the file being received(DOS file date format). Note that some file transfer programs do not transmit the file date. In said case this value will be zero.
ftpcb->f.time	Time of the file being received(DOS file time format). Note that some file transfer programs do not transmit the file time. In said case this value will be zero.

See Also-

cdrvxfer_files()
cdrvxfer_sendfiles()
cdrvxfer_sfiles()
FileTransferDialog()
TransferFiles()

Examples-

```
#include   <comm.h>

main()
{
int                                 port=0;
int                                 protocol=6;
char                                *fspec="e:\\tmp\\*.*";
int                                 stat;
static char                         buf[1024];
static struct cdrvxfer_ftpcb        ftpcb;

//Initialize the file transfer structure input parameters
fstrcpy(ftpcb.fspec,fspec);
ftpcb.port                  = port;
ftpcb.buf                   = buf;
ftpcb.bufsiz                = sizeof(buf);
ftpcb.flag                  = CDRVXFER_FLAG_RECOVER;
ftpcb.rcvtimeout            = 90;
ftpcb.xmttimeout            = 90;
ftpcb.retryent              = 3;
ftpcb.state                 = CDRVXFER_RSTATE_INIT;

while ((stat = cdrvxfer_getfiles(protocol,&ftpcb) == CDRVXFER_ERR_BUSY)
                    {
                    if (ftpcb.flag & CDRVXFER_FLAG_FILEDONE)
                                {
                                printf("\rFilename=>%s ByteCnt=>%ld\n",
                                ftpcb.fspec,ftpcb.numbytes);

                                ftpcb.flag &= (~CDRVXFER_FLAG_FILEDONE);
                                }
                    else
                                printf("\rFilename=>%s ByteCnt=>%ld",
                                ftpcb.fspec,ftpcb.numbytes);
                    }
return;
}
```

cdrvxfer_sendfiles

Description-

This function sends a particular file(s) based on the selected file transfer protocol. Note that the file transfer engine is a state machine. As such, this routine must be called repeatedly until a return value other than **CDRVXFER_ERR_BUSY** (protocol engine busy so file transfer is still in progress) is returned. This format allows an application to run several file transfers concurrently by calling this function repeatedly with different port numbers.

Syntax-

stat =cdrvxfer_sendfiles(protocol,ftpcb);

On Entry-

int protocol;

0	XMODEM
1	XMODEM 1K
2	YMODEM
3	YMODEM Batch
4	YMODEM G
5	YMODEM G Batch
6	ZMODEM
7	XMODEM(Checksum)

struct cdrvxfer_ftpcb *ftpcb;

Pointer to the structure with the information to provide and to receive for the file transfer. Description of pertinent elements follows.

ftpcb->fspec[]

Null terminated filename to use if the filename not given by the particular file transfer protocol. (May contain wildcards for protocols that allow it).

ftpcb->port

Port file transfer taking place on.

ftpcb->state

Must be set to **CDRVXFER_SSTATE_INIT** before the first call to **cdrvxfer_sendfiles()**.

ftpcb->buf

Pointer to the buffer that the file transfer engine will use. This buffer must be at least 1K.

ftpcb->bufsiz

Size of buffer pointed to by **ftpcb->buf**.

ftpcb->flag

Set to 0.

ftpcb->retrycnt

Maximum number of times to retry sending the same failed packet.

ftpcb->rcvtimeout

Number of 1/18th of a second tics to wait for incoming bytes. This value is generally 180(10 seconds).

ftpcb->xmttimeout

Number of 1/18th of a second tics to wait for space to become available in transmission buffer. This value is generally 180(10 seconds).

On Exit-

int stat;

CDRVXFER_ERR_OK	If the file transfer was successful.

CDRVXFER_ERR_SERIAL	A serial I/O error has occurred. This could be caused by a broken serial communication cable, a bad UART or some other serial error.
CDRVXFER_ERR_OS	An MSDOS error occurred. This may be caused by an error opening, writing, reading, or closing a file.
CDRVXFER_ERR_PROTOCOL	A protocol error. This happens when an incompatibility between the file transfer protocol between the two transferring machines occur.
CDRVXFER_ERR_BUSY	If file transfer is still in progress. Note this routine should be constantly called until this code is not returned. The structure elements below are kept updated in between calls to **cdrvxfer_sendfiles()**.
CDRVXFER_ERR_CANCEL	The remote issued a cancel.
CDRVXFER_ERR_INVPORT	An invalid serial port was selected.
CDRVXFER_ERR_FNAME	An invalid filename was given.
CDRVXFER_ERR_MEMORY	A memory allocation error has occurred. This error may be caused due to a lack of memory. Most compilers causes an executable to allocate all memory on startup. You may want to use the **exehdr** program or some other compiler utility to adjust the maximum amount of memory the executable may allocate.
ftpcb->blksiz	Size of the block being transferred.
ftpcb->curblk	Current block being transferred.
ftpcb->currenterr	Error count for the file currently being transferred.

ftpcb->error	Current error code(usually **CDRVXFER_ERR_BUSY** during file transfer). This is the same value returned by this function.
ftpcb->exterror	An extended error code.
ftpcb->fhandle	Operating system file handle of file currently being transferred or -1 if file is closed.
ftpcb->fspec	Fully qualified pathname and filename of the file being transferred.
ftpcb->numbytes	Number of bytes transferred in the current file.
ftpcb->offsetbytes	This is the offset in the file where the current file transfer began. The difference between ftpcb->numbytes and this value gives the actual number of bytes transferred in the current file this session. This value is greater than zero for the ZMODEM transfer when crash recovery is in effect.
ftpcb->totalerr	Total of all errors for the current session.
ftpcb->flag	The bit **CDRVXFER_FLAG_FILEDONE** is set when the file transfer on a particular file is complete. This bit should be reset after it is tested. This bit may be used to determine when some statistics need updating(filename, counts, etc.).
ftpcb->f.fbuf.dos_size	Size of file that is being transmitted.
ftpcb->f.fbuf.dos_wr_date	Date of the file being transmitted(DOS file date format).
ftpcb->f.fbuf.dos_wr_time	Time of the file being transmitted(DOS file time format).

See Also-

cdrvxfer_getfiles()
cdrvxfer_files()
cdrvxfer_sfiles()
TransferFiles()

Examples-

```
#include   <comm.h>

main()
{
int                                port=0;
int                                protocol=6;
char                               *fspec="e:\\tmp\\*.*";
int                                stat;
static char                        buf[1024];
static struct cdrvxfer_ftpcb       ftpcb;

//Initialize the file transfer structure input parameters
fstrcpy(ftpcb.fspec,fspec);
ftpcb.port                 = port;
ftpcb.buf                  = buf;
ftpcb.bufsiz               = sizeof(buf);
ftpcb.flag                 = 0;
ftpcb.rcvtimeout           = 90;
ftpcb.xmttimeout           = 90;
ftpcb.retrycnt             = 3;
ftpcb.state                = CDRVXFER_SSTATE_INIT;

while ((stat = cdrvxfer_sendfiles(protocol,&ftpcb) == CDRVXFER_ERR_BUSY)
                {
                if (ftpcb.flag & CDRVXFER_FLAG_FILEDONE)
                        {
                        printf("\rFilename=>%s ByteCnt=>%ld\n",
                        ftpcb.fspec,ftpcb.numbytes);

                        ftpcb.flag &= (~CDRVXFER_FLAG_FILEDONE);
                        }
                else
                        printf("\rFilename=>%s ByteCnt=>%ld",
                        ftpcb.fspec,ftpcb.numbytes);
                }
return;
}
```


cdrvxfer_sfiles

Description-

Transmit or receive file(s) with specified protocol to completion. This function is used for transmitting and receiving files. It does not return to the caller until the file transfer completes. It does however call the function that the caller passed when the function was called in order to allow the caller to get a progress report on the file transfer. This function is similar to the **cdrvxfer_files()** function. This function differs in that it additionally passes a pointer to the file transfer port control block (a structure with a wealth of information about the active file transfer) to the user specified callback function.

Syntax-

stat = cdrvxfer_sfiles(port,direction,protocol,fspec,fnc);

On Entry-

int port;

Port previously opened with **InitializePort()**.

int direction;

0	To send a file.
1	To receive a file.

int protocol;

0	XMODEM
1	XMODEM 1K
2	YMODEM
3	YMODEM Batch
4	YMODEM G
5	YMODEM G Batch
6	ZMODEM
7	XMODEM(Checksum)

char fspec;

Fully qualified pathname and filename of the file being transferred

int (far *fnc)(int chgflg,char *fname,long bytcnt,int curblk, int errcnt, unsigned rate, struct cdrvxfer_ftpcb far *ftpcb);

Pointer to routine that is called by this function anytime one of the passed parameters changes. This function may abort the transfer by returning a non-zero value. If this pointer is NULL, then no display function is called.

int chgflg;

Non zero if calling for a new filename.

char *fname;

Filename being processed.

long bytcnt;

Current file bytecount.

int curblk;

Current block being worked on.

int errcnt;

Current error count for file.

unsigned rate;

Current transfer rate in bytes per second.

struct cdrvxfer_ftpcb *ftpcb;

Pointer to the file transfer control structure. Between calls to this function, this structure will contain updated information as follows. These values may be displayed or processed by the calling application.

ftpcb->blksiz Size of the block being transferred.

ftpcb->curblk	Current block being transferred.
ftpcb->currenterr	Error count for the file currently being transferred.
ftpcb->error	Current error code(usually **CDRVXFER_ERR_BUSY** during file transfer). This is the same value returned by this function.
ftpcb->exterror	An extended error code.
ftpcb->fhandle	Operating system file handle of file currently being transferred.
ftpcb->fspec	Fully qualified pathname and filename of the file being transferred.
ftpcb->numbytes	Number of bytes transferred in the current file.
ftpcb->offsetbytes	This is the offset in the file where the current file transfer began. The difference between ftpcb->numbytes and this value gives the actual number of bytes transferred in the current file this session. This value is greater than zero for the ZMODEM transfer when crash recovery is in effect.
ftpcb->totalerr	Total of all errors for the current session.
ftpcb->f.size	Size of file that is being received. Note that some file transfer programs do not transmit the file size. In said case this value will be zero.
ftpcb->f.date	Date of the file being received(DOS file date format). Note that some file transfer programs do not transmit the file date. In said case this value will be zero.

ftpcb->f.time	Time of the file being received(DOS file time format). Note that some file transfer programs do not transmit the file time. In said case this value will be zero.
ftpcb->f.fbuf.dos_size	Size of file that is being transmitted.
ftpcb->f.fbuf.dos_wr_date	Date of the file being transmitted(DOS file date format).
ftpcb->f.fbuf.dos_wr_time	Time of the file being transmitted(DOS file time format).

On Exit-

int stat;

CDRVXFER_ERR_OK	If the file transfer was successful.
CDRVXFER_ERR_SERIAL	A serial I/O error has occurred. This could be caused by a broken serial communication cable, a bad UART or some other serial error.
CDRVXFER_ERR_OS	An MSDOS error occurred. This may be caused by an error opening, writing, reading, or closing a file.
CDRVXFER_ERR_PROTOCOL	A protocol error. This happens when an incompatibility between the file transfer protocol between the two transferring machines occur.
CDRVXFER_ERR_BUSY	File transfer is still in progress.
CDRVXFER_ERR_CANCEL	The remote issued a cancel.
CDRVXFER_ERR_INVPORT	An invalid serial port was selected.
CDRVXFER_ERR_FNAME	An invalid filename was given.

CDRVXFER_ERR_MEMORY	A memory allocation error has occurred. This error may be caused due to a lack of memory. Most compilers causes an executable to allocate all memory on startup. You may want to use the exehdr program or some other compiler utility to adjust the maximum amount of memory the executable may allocate.

See Also-

cdrvxfer_files()
cdrvxfer_getfiles()
cdrvxfer_sendfiles()
TransferFiles()

Examples-

```
#include   <comm.h>

main()
{
int             port=0;
int             direction=1;
int             protocol=6;
char            *fspec="e:\\tmp\\*.*";
int             stat;
int             (far *fnc)(int chgflg,char *fname,long bytcnt,int curblk, int errcnt, unsigned rate,struct
                cdrvxfer_ftpcb *ftpcb) = DisplayFunction;

stat = cdrvxfer_sfiles(port,direction,protocol,fspec,fnc);
return;
}


//User specified callback function. This function is
//called with file transfer progress information.

DisplayFunction(int chgflg,char *fname,long bytcnt,int curblk, int errcnt, unsigned rate,struct cdrvxfer_ftpcb
*ftpcb)
{
char   fnam[12+1];

fstrcpy(fnam,fname); //Filename being transferred

if (fchg != 0)                  //Display on next line only
    printf("\n");       //if moving a new file.
```

```
//Display the current transfer statistics
printf("\rNAM=>%-12.12s BLK=>%5u BYT=>%7lu ERR=>%4u R=%5u Byt/Sec",
fnam,curblk,bytcnt,errcnt,rate);
return(0);
}
```


CdrvXferCreateDialog

Description-

This functions creates a dialog box that displays the file transfer statistics. It is used in conjunction with the **CdrvXferUpdateDialog()** and **CdrvXferDestroyDialog()** functions.

Syntax-

stat = CdrvXferCreateDialog(ftpcb);

On Entry-

struct cdrvxfer_ftpcb *ftpcb;

Pointer to a file transfer port control block to be used with a file transfer. This function is usually used when one of the state driven file transfer functions (**cdrvxfer_getfiles()**, **cdrvxfer_getfiles()**, **TransferFiles()**).

On Exit-

int stat;

0 if no error. Otherwise error opening a dialog box that may be caused by low memory or low resource conditions.

See Also-

cdrvxfer_getfiles()
cdrvxfer_sendfiles()
CdrvXferDestroyDialog()
CdrvXferUpdateDialog()
TransferFiles()

Example-

```
#include <comm.h>
struct cdrvxfer_ftpcb  p;
int                    stat;
int                    port=0;
int                    direction=0;
int                    protocol=6;
char                   *fspec="\\tmp\\*.*";

strcpy(p.fspec,"");

/* Initialize the transfer */
if ((stat = TransferFiles(0,port,direction,protocol,fspec,&p)) != 0)
        return(-1);

CdrvXferCreateDialog(&p);

/* Do the file transfer */
while((stat = TransferFiles(1,port,direction,protocol,fspec,&p)) != CDRVXFER_ERR_BUSY)
        CdrvXferUpdateDialog(&p);

/* Close the file transfer */
TransferFiles(2,port,direction,protocol,fspec,&p);

/* Remove File Transfer Dialog */
CdrvXferDestroyDialog(&p);
```


CdrvXferDestroyDialog

Description-

This functions destroys a dialog box that displayed the file transfer statistics for a state driven file transfer. It is used in conjunction with the **CdrvXferCreateDialog()** and **CdrvXferUpdateDialog()** functions.

Syntax-

stat = CdrvXferDestroyDialog(ftpcb);

On Entry-

struct cdrvxfer_ftpcb *ftpcb;

Pointer to a file transfer port control block to be used with a file transfer. This function is usually used when one of the state driven file transfer functions (**cdrvxfer_getfiles(), cdrvxfer_getfiles(), TransferFiles()**).

On Exit-

int stat;

0 if no error.

See Also-

cdrvxfer_getfiles()
cdrvxfer_sendfiles()
CdrvXferCreateDialog()
CdrvXferUpdateDialog()
TransferFiles()

Example-

```
#include   <comm.h>

main()
{
int                                 port=0;
int                                 protocol=6;
char                                *fspec="e:\\tmp\\*.*";
int                                 stat;
static char                         buf[1024];
static struct cdrvxfer_ftpcb        ftpcb;

//initialize the file transfer structure input parameters
fstrcpy(ftpcb.fspec,fspec);
ftpcb.port                  = port;
ftpcb.buf                   = buf;
ftpcb.bufsiz                = sizeof(buf);
ftpcb.flag                  = 0;
ftpcb.rcvtimeout            = 90;
ftpcb.xmttimeout            = 90;
ftpcb.retrycnt              = 3;
ftpcb.state                 = CDRVXFER_SSTATE_INIT;

CdrvXferCreateDialog(&ftpcb);
while ((stat = cdrvxfer_sendfiles(protocol,&ftpcb) == CDRVXFER_ERR_BUSY)
                    {
                    if (ftpcb.flag & CDRVXFER_FLAG_FILEDONE)
                                {
                                printf("\rFilename=>%s ByteCnt=>%ld\n",
                                ftpcb.fspec,ftpcb.numbytes);

                                ftpcb.flag &= (~CDRVXFER_FLAG_FILEDONE);
                                }
                    else
                                printf("\rFilename=>%s ByteCnt=>%ld",
                                ftpcb.fspec,ftpcb.numbytes);
                    CdrvXferUpdateDialog(&ftpcb);
                    }
CdrvXferDestroyDialog(&ftpcb);
return;
}
```


CdrvXferUpdateDialog

Description-

This functions updates a dialog box that displays the file transfer statistics for a state driven file transfer. It is used in conjunction with the **CdrvXferCreateDialog()** and **CdrvXferDestroyDialog()** functions.

Syntax-

stat = CdrvXferUpdateDialog(ftpcb);

On Entry-

struct cdrvxfer_ftpcb *ftpcb;

*Pointer to a file transfer port control block to be used with a file transfer. This function is usually used when one of the state driven file transfer functions (**cdrvxfer_getfiles(), cdrvxfer_getfiles(), TransferFiles()**).*

On Exit-

int stat;

0 if no error.

See Also-

cdrvxfer_getfiles()
cdrvxfer_sendfiles()
CdrvXferCreateDialog()
CdrvXferUpdateDialog()
TransferFiles()

Example-

```
#include   <comm.h>

main()
{
int                                 port=0;
int                                 protocol=6;
char                                *fspec="e:\\tmp\\*.*";
int                                 stat;
static char                         buf[1024];
static struct cdrvxfer_ftpcb        ftpcb;

//Initialize the file transfer structure input parameters
fstrcpy(ftpcb.fspec,fspec);
ftpcb.port                  = port;
ftpcb.buf                   = buf;
ftpcb.bufsiz                = sizeof(buf);
ftpcb.flag                  = 0;
ftpcb.rcvtimeout            = 90;
ftpcb.xmittimeout           = 90;
ftpcb.retrycnt              = 3;
ftpcb.state                 = CDRVXFER_SSTATE_INIT;

CdrvXferCreateDialog(&ftpcb);
while ((stat = cdrvxfer_sendfiles(protocol,&ftpcb) == CDRVXFER_ERR_BUSY)
                    {
                    if (ftpcb.flag & CDRVXFER_FLAG_FILEDONE)
                              {
                              printf("\rFilename=>%s ByteCnt=>%ld\n",
                              ftpcb.fspec,ftpcb.numbytes);

                              ftpcb.flag &= (~CDRVXFER_FLAG_FILEDONE);
                              }
                    else
                              printf("\rFilename=>%s ByteCnt=>%ld",
                              ftpcb.fspec,ftpcb.numbytes);
                    CdrvXferUpdateDialog(&ftpcb);
                    }
CdrvXferDestroyDialog(&ftpcb);
return;
}
```


FileTransferDialog

Description-

This function enables or disables the display of a dialog box with file transfer progress for the cdrvxfer_files(), cdrvxfer_sfiles(), and TransferFiles() functions.

Syntax-

stat = FileTransferDialog(port,mode);

On Entry-

int	port;	
	Port previously opened with **InitializePort()**.	
int	mode;	
	0	Disable the file transfer dialog.
	1	Enable the file transfer dialog.

On Exit-

int	stat;	
	-1	If port was not initialized.

See Also-

cdrvxfer_files()
cdrvxfer_sfiles()
TransferFiles()

Examples-

```
#include   <comm.h>

main()
{
int                stat;
int                mode=1;


if ((stat = FileTransferDialog(port,mode)) == -1)
        {
        printf("Port not initialized\n");

        /* Take remedial action */
        }
}
```


SetXferParameters

Description-

Sets file transfer custom parameters like retries, time-outs, etc.

Syntax-

stat = SetXferParameters(command,value);

On Entry-

unsigned int command;

This variable is set to one of the following symbols. Note that the ***value*** variable contains the value or symbols as explained next to the ***command*** symbol.

unsigned long value;

This value depends on ***command*** above and its possible content are described above as well.

Command = CDRVXFER_RCVFILEOPTIONS

Sets one of the receive file options. The options are described below.

value = CDRVXFER_PARM_RECOVER

Used cause a restart of a file transfer that crashed or was previously discontinued from the crash point.

value = CDRVXFER_PARM_OVERWRITE

Used to cause the overwrite of a file if it already exists.

value = CDRVXFER_PARM_RENAME

Used to cause the incoming file to be renamed if a duplicate file was present.

value = CDRVXFER_PARM_SKIPSAME

Used to cause the incoming file to be skipped if a duplicate file was present and have the same date/time and size.

command = CDRVXFER_RCVRETRYCNT

Sets the retry count on failed packets while receiving files.

value = retry count

command = CDRVXFER_RCVITIMER

Sets the number of 1/18th seconds tics to wait for characters while receiving files.

value=tics

command = CDRVXFER_RCVOTIMER

Sets the number of 1/18th seconds to wait for outgoing characters to be transmitted before assuming an error while receiving files.

value=tics

command = CDRVXFER_XMTRETRYCNT

Sets the retry count on failed packets while sending files.

value=retry count

command = CDRVXFER_XMTITIMER

Sets the number of 1/18th seconds tics to wait for characters while sending files.

value=tics

command = CDRVXFER_XMTOTIMER

Sets the number of 1/18th seconds to wait for outgoing characters to be transmitted before assuming an error while sending files.

value=tics

On Exit-

int stat;

CDRVXFER_ERR_NONE If the file transfer was successful.

See Also-

cdrvxfer_files()
cdrvxfer_sfiles()
TransferFiles()

Examples-

```
#include <comm.h>

main()
{

/* Force the file transfer engine to overwrite a file if it already exist*/
SetXferParameters(CDRVXFER_RCVFILEOPTIONS, CDRVXFER_PARM_OVERWRITE);
}
```

TransferFiles

Description-

Transmit or receive file(s) with specified protocol. This function is used for transmitting and receiving files. For any file transfer session it must be called with three different commands with the same arguments. The first is to initialize some file transfer parameters. The second command is to actually perform the file transfer. The user should continue to call this function with the second command until it returns a status other than **CDRVXFER_ERR_BUSY**. Before the user exits the file transfer session, the third command should be sent to release any resources that were allocated. If the user needs to prematurely abort the transfer, the function with the third command should be called and the session exited.

Syntax-

stat = TransferFiles(cmd,port,direction,protocol,fspec,ftpcb);

On Entry-

int port;

Port over which transfer will occur.

int cmd;

0 Initialize transfer. This function should be called once with this command for the start of every file transfer session.

1 Perform transfer. This function should be called with this command continuously until it ***no longer*** returns the status **CDRVXFER_ERR_BUSY**. To abort the transfer prematurely, simply stop calling this function with this command.

2 Cleanup when transfer complete or aborted. This function ***must*** be called at the end of a file transfer function.

int direction;

0 To send a file.
1 To receive a file.

int protocol;

0 XMODEM
1 XMODEM 1K
2 YMODEM
3 YMODEM Batch
4 YMODEM G
5 YMODEM G Batch
6 ZMODEM
7 XMODEM(Checksum)

char *fspec;

Filename. If the filename has a path and files are being received, the files will reside in the directory in the path.

struct cdrvxfer_ftpcb *ftpcb;

Pointer to the file transfer control structure. Between calls to this function, this structure will contain updated information as follows. These values may be displayed or processed by the calling application.

On Exit-

int stat;

CDRVXFER_ERR_OK	If the file transfer was successful.
CDRVXFER_ERR_SERIAL	A serial I/O error has occurred. This could be caused by a broken serial communication cable, a bad UART or some other serial error.
CDRVXFER_ERR_OS	An MSDOS error occurred. This may be caused by an error opening, writing, reading, or closing a file.

CDRVXFER_ERR_PROTOCOL	A protocol error. This happens when an incompatibility between the file transfer protocol between the two transferring machines occur.
CDRVXFER_ERR_BUSY	File transfer is still in progress.
CDRVXFER_ERR_CANCEL	The remote issued a cancel.
CDRVXFER_ERR_INVPORT	An invalid serial port was selected.
CDRVXFER_ERR_FNAME	An invalid filename was given.
CDRVXFER_ERR_MEMORY	A memory allocation error has occurred. This error may be caused due to a lack of memory. Most compilers causes an executable to allocate all memory on startup. You may want to use the **exehdr** program or some other compiler utility to adjust the maximum amount of memory the executable may allocate.
ftpcb->blksiz	Size of the block being transferred.
ftpcb->curblk	Current block being transferred.
ftpcb->currenterr	Error count for the file currently being transferred.
ftpcb->error	Current error code(usually **CDRVXFER_ERR_BUSY** during file transfer). This is the same value returned by this function.
ftpcb->exterror	An extended error code.
ftpcb->fhandle	Operating system file handle of file currently being transferred or -1 if file is closed.
ftpcb->fspec	Fully qualified pathname and filename of the file being transferred.

ftpcb->numbytes	Number of bytes transferred in the current file.
ftpcb->offsetbytes	This is the offset in the file where the current file transfer began. The difference between ftpcb->numbytes and this value gives the actual number of bytes transferred in the current file this session. This value is greater than zero for the ZMODEM transfer when crash recovery is in effect.
ftpcb->totalerr	Total of all errors for the current session.
ftpcb->flag	The bit **CDRVXFER_FLAG_FILEDONE** is set when the file transfer on a particular file is complete. This bit should be reset after it is tested. This bit may be used to determine when some statistics need updating(filename, counts, etc.).
ftpcb->f.size	Size of file that is being received. Note that some file transfer programs do not transmit the file size. In said case this value will be zero.
ftpcb->f.date	Date of the file being received(DOS file date format). Note that some file transfer programs do not transmit the file date. In said case this value will be zero.
ftpcb->f.time	Time of the file being received(DOS file time format). Note that some file transfer programs do not transmit the file time. In said case this value will be zero.
ftpcb->f.fbuf.dos_size	Size of file that is being transmitted.
ftpcb->f.fbuf.dos_wr_date	Date of the file being transmitted(DOS file date format).
ftpcb->f.fbuf.dos_wr_time	Time of the file being transmitted(DOS file time format).

See Also-

cdrvxfer_files()
cdrvxfer_getfiles()
cdrvxfer_sendfiles()
cdrvxfer_sfiles()

Examples-

```
#include   <comm.h>

main()
{
int                                     direction=1;
int                                     port=0;
int                                     protocol=6;
char                                    *fspec="e:\\tmp\\*.*";
int                                     stat;
static char                             buf[1024];
static struct cdrvxfer_ftpcb            ftpcb;

/* Initialize the transfer */
if ((stat = TransferFiles(0,port,direction,protocol,fspec,&ftpcb))
== CDRVXFER_ERR_OK)
        //Transfer loop
        while(TransferFiles(1,port,direction,protocol,fspec,    &ftpcb)
        == CDRVXFER_ERR_BUSY)
                {
                if (ftpcb.flag & CDRVXFER_FLAG_FILEDONE)
                        {
                        printf("\rFilename=>%s ByteCnt=>%ld\n",
                        ftpcb.fspec,ftpcb.numbytes);

                        ftpcb.flag &= (~CDRVXFER_FLAG_FILEDONE);
                        }
                else
                        printf("\rFilename=>%s ByteCnt=>%ld",
                        ftpcb.fspec,ftpcb.numbytes);
                }

/* Close the file transfer */
TransferFiles(2,port,direction,protocol,fspec,&ftpcb);
}
```


Low Level Functions

These low level **COMM-DRV** functions are the core functions of the product. The higher level functions are derived from these. Note that these functions should not be called from within the **COMM-DRV** interrupt functions(user callbacks). They may however be called from interrupt handlers as long as the calls aren't nested.

ser_rs232_block()	Set time-outs on character reception & transmission.
ser_rs232_cleanup()	Un-installs a port.
ser_rs232_dtr_off()	Turns the DTR signal off.
ser_rs232_dtr_on()	Turns the DTR signal on.
ser_rs232_flush()	Selectively flushes input and output buffers.
ser_rs232_get_sdata()	Gets pointer to **COMM-DRV** system data area.
ser_rs232_getbyte()	Reads a byte from input buffer.
ser_rs232_getpacket()	Reads a packet from input buffer.
ser_rs232_getport()	Gets port information.
ser_rs232_getstatus()	Gets port modem and line status.
ser_rs232_maxport()	Returns the highest addressable port number.
ser_rs232_misc_func()	**COMM-DRV** multiplex function. Used to setup user callback functions, initialize certain variables, etc.
ser_rs232_putbyte()	Queues a byte for transmission.
ser_rs232_putpacket()	Queues a packet for transmission.
ser_rs232_putregister()	Writes to specified 8250 type register.
ser_rs232_rts_off()	Turns RTS off.
ser_rs232_rts_on()	Turns RTS on.
ser_rs232_set_intfunc()	Setup user interrupt functions(user callback).
ser_rs232_setbauddiv()	Sets baudrate divisor for 8250 style UARTs.
ser_rs232_setup()	Installs or modify a serial port.
ser_rs232_viewpacket()	Non-destructively reads a packet from input buffer.

ser_rs232_block

Description-

This routine sets the blocking time on both the inputting and outputting of data.

Syntax-

stat = ser_rs232_block(port,inblock,outblock)

On Entry-

int port;

A port previously opened with the **ser_rs232_setup()** function.

int inblock;

Number of 1/18th sec increments to wait for input. The **ser_rs232_getpacket(), ser_rs232_viewpacket(), and ser_rs232_getbyte()** functions will hang for up to the amount before returning with an error.

int outblock;

Number of 1/18th sec increments to wait for the output buffer to get space for data. The **ser_rs232_putpacket() and ser_rs232_putbyte()** functions will hang for up to the amount before returning with an error.

On Exit-

int stat;

Error codes described under **errors** in **APPENDIX C.**

See Also-

ser_rs232_setup()

Examples-

```
#include <comm.h>

int             stat;
int             port=0;
int             inblock=4;
int             outblock=4;

if ((stat = ser_rs232_block(port,inblock,outblock)) != RS232ERR_NONE)
        {
        printf("Error # %d\n");

        /* Take remedial action */
        }
```

ser_rs232_cleanup

Description-

This function restore vectors and places the serial port in its original state.

Note: This function must be called on all ports opened with the **ser_rs232_setup()** function before the executable exits. Otherwise stray interrupts will occur and lockup the computer.

Syntax-

stat = ser_rs232_cleanup(port)

On Entry-

int port;

A port previously opened with the **ser_rs232_setup()** function.

On Exit-

int stat;

Error codes described under **errors** in **APPENDIX C.**

See Also-

ser_rs232_setup()

Examples-

```
#include <comm.h>

int             stat;
int             port=0;

if ((stat = ser_rs232_cleanup(port)) != RS232ERR_NONE)
        {
        printf("Error # %d\n");

        /* Take remedial action */
        }
```

ser_rs232_dtr_off

Description-

Turns the DTR signal off.

Syntax-

stat = ser_rs232_dtr_off(port);

On Entry-

int port;

A port previously opened with the **ser_rs232_setup()** function.

On Exit-

int stat;

Error codes described under **errors** in **APPENDIX C**.

See Also-

ser_rs232_dtr_on()

Examples-

```
#include <comm.h>

int     stat;
int     port=0;

if ((stat = ser_rs232_dtr_off(port)) != RS232ERR_NONE)
        {
        printf("Error # %d\n");

        /* Take remedial action */
        }
```

ser_rs232_dtr_on

Description-

Turns the DTR signal on.

Syntax-

stat = ser_rs232_dtr_on(port);

On Entry-

int port;

A port previously opened with the **ser_rs232_setup()** function.

On Exit-

int stat;

Error codes described under **errors** in **APPENDIX C**.

See Also-

ser_rs232_dtr_off()

Examples-

```
#include <comm.h>

int             stat;
int             port=0;

if ((stat = ser_rs232_dtr_on(port)) != RS232ERR_NONE)
        {
        printf("Error # %d\n");

        /* Take remedial action */
        }
```

ser_rs232_flush

Description-

This routine selectively flushes either the input, output, or both the input and output communication buffers.

Syntax-

stat = ser_rs232_flush(port,arg)

On Entry-

int port;

A port previously opened with the **ser_rs232_setup()** function.

int arg;

0 to flush input buffer.
1 to flush output buffer.
2 to flush input/output buffers.

On Exit-

int stat;

Error codes described under **errors** in **APPENDIX C**.

See Also-

Examples-

```
#include <comm.h>

int             stat;
int             port=0;

if ((stat = ser_rs232_flush(port,0)) != RS232ERR_NONE)
        {
        printf("Error # %d\n");

        /* Take remedial action */
        }
```

ser_rs232_getbyte

Description-

This routine gets a byte from the circular input communication buffer. If a time-out was set with the **ser_rs232_block()** or the **ser_rs232_setup()** call, then this call will hang for up to the time-out value before returning with an error if no data was present.

Syntax-

stat = ser_rs232_getbyte(port,ch)

On Entry-

int port;

A port previously opened with the **ser_rs232_setup()** function.

char *ch;

Pointer to a byte in which to place the received character.

On Exit-

int stat;

Error codes described under **errors** in **APPENDIX C**.

See Also-

ser_rs232_getpacket()

Examples-

```
#include <comm.h>

int     stat;
int     port=0;
char    ch[1];

if ((stat = ser_rs232_getbyte(port,ch)) != RS232ERR_NONE)
        {
        printf("Error # %d\n");

        /* Take remedial action */
        }
```

ser_rs232_getpacket

Description-

This function gets a fixed length packet from the input communication buffer. *Note that if the size of the input communication buffer is not at least the size of the packet, then this call will always fail.* If a time-out was set with the **ser_rs232_block()** or the **ser_rs232_setup()** call, then this call will hang for up to the time-out value before returning with an error if enough data was not present.

Syntax-

stat = ser_rs232_getpacket(port,count,ch)

On Entry-

int port;

A port previously opened with the **ser_rs232_setup()** function.

int count;

Number of bytes to read.

char *ch;

Pointer to the buffer to place the packet.

On Exit-

int stat;

Error codes described under **errors** in **APPENDIX C**.

See Also-

ser_rs232_getbyte()
ser_rs232_viewpacket ()

Examples-

```
#include <comm.h>

int     stat;
int     port=0;
char    ch[10];

if ((stat = ser_rs232_getpacket(port,sizeof(ch),ch)) != RS232ERR_NONE)
        {
        printf("Error # %d\n");

        /* Take remedial action */
        }
```

ser_rs232_getport

Description-

This routine gets a copy of the particular port's port control block(**pcb**). The port control block is defined in detail in **APPENDIX B** & **APPENDIX C**.

Syntax-

stat = ser_rs232_getport(port,pcb)

On Entry-

int port;

A port previously opened with the **ser_rs232_setup()** function or a port not initialized. If the port was not previously opened, then the returned PCB is a default template. This function should be called before the first use of the function **ser_rs232_setup()** to pre-initialize the PCB template.

struct port_param *pcb;

Pointer to the structure that will contain the copy of the port control block. The different items in the structure above are defined in **APPENDIX B** and **APPENDIX C**.

On Exit-

int stat;

Error codes described under **errors** in **APPENDIX C**.

See Also-

ser_rs232_setup()

Examples-

```
#include <comm.h>

int                 stat;
int                 port=0;
struct port_param   pcb;

if ((stat = ser_rs232_getport(port,&pcb)) != RS232ERR_NONE)
        {
        printf("Error # %d\n");

        /* Take remedial action */
        }
```

ser_rs232_getregister

Description-

This routine gets the particular register from the UART. It is assumed that it is a UART compatible with the 8250 register set. Because **COMM-DRV** work with multiport cards that may not contain an 8250 or compatible UART, it is possible that all the registers will not be implemented. Sub-devices in general will always return a valid 8250 style MSR and LSR register.

Syntax-

stat = ser_rs232_getregister(port,offset,value)

On Entry-

int port;

A port previously opened with the **ser_rs232_setup()** function.

int offset;

Offset from base of the physical UART address. These offsets are define symbolically in **comm.h**. They are

BAUD_LOW	Low baud rate divisor register.
BAUD_HIGH	High baud rate divisor register.
DATABUF	Input/output data register.
IER	Interrupt enable register.
IIR	Interrupt identification register.
FCR	FIFO control register.
LCR	Line control register.
MCR	Modem control register.
MSR	Modem status register.
SCR.	Scratch register.

int *value;

Pointer to the byte where register content is placed.

On Exit-

int stat;

Error codes described under **errors** in **APPENDIX C.**

See Also-

ser_rs232_putregister()

Examples-

```
#include <comm.h>

int     stat;
int     port=0;
int     value;

if ((stat = ser_rs232_getregister(port,MCR,&value)) != RS232ERR_NONE)
        {
        printf("Error # %d\n");

        /* Take remedial action */
        }
```

ser_rs232_getstatus

Description-

This routine retrieves the line status and modem status for a particular port. This function may be used to determine if Carrier Detect, CTS, DSR, Ring Indicator, Break, or Line Errors have occurred.

Syntax-

stat = ser_rs232_getstatus(port,status)

On Entry-

int port;

A port previously opened with the **ser_rs232_setup()** function.

char *status;

Pointer to a two byte array to store the **MSR** and **LSR**.

On Exit-

char *status;

status[0] modem status register.

If anding **stat[0]** with **MSR_CTS** is true, the **CTS** signal present.
If anding **stat[0]** with **MSR_DSR** is true, the **DSR** signal present.
If anding **stat[0]** with **MSR_RI** is true, the **Ring Indicator** signal present.
If anding **stat[0]** with **MSR_DCD** is true, the **Carrier Detect** signal present.

status[1] line status register.

If anding **stat[1]** with **LSR_DATA** is true, the UART has input data.
If anding **stat[1]** with **LSR_OVERRUN** is true, an overrun has occurred.
If anding **stat[1]** with **LSR_PARITY** is true, a parity error has occurred.
If anding **stat[1]** with **LSR_FRAMING** is true, a framing error has occurred.
If anding **stat[1]** with **LSR_BREAK** is true, a break condition is in effect.
If anding **stat[1]** with **LSR_THRE** is true, transmit holding register is empty.
If anding **stat[1]** with **LSR_TSRE** is true, transmit shift register is empty.

int stat;

Error codes described under errors in **APPENDIX C**.

See Also-

ser_rs232_getregister()

Examples-

```
#include <comm.h>

int     stat;
int     port=0;
char    status[2];

if ((stat = ser_rs232_getstatus(port,status)) != RS232ERR_NONE)
        {
        printf("Error # %d\n");

        /* Take remedial action */
        }
```

ser_rs232_maxport

Description-

This routine returns the highest port number supported by this package.

Syntax-

numport = ser_rs232_maxport()

On Entry-

None.

On Exit-

int numport;

Highest port number that may be used.

See Also-

Examples-

```
#include <comm.h>

int                     numport;
int                     port=0;

if ((numport=ser_rs232_maxport()) != RS232ERR_NONE)
        {
        printf("Error # %d\n");

        /* Take remedial action */
        }
```


ser_rs232_misc_func

Description-

This routine is the **COMM-DRV** catch all function. The use of this call is shown in **example8.c**, **exintfnc.c**, **example7.c**, and others. It is used to set parameters for asynchronous callback functions, setup timed callback functions, send data immediately ahead of the output buffer queue, and much more.

Syntax-

stat = ser_rs232_misc_func(port,cmd,arg);

On Entry-

int port;

A port previously opened with the **ser_rs232_setup()** function.

unsigned short cmd;

Sub-command for this function. See notes below.

unsigned long arg;

Value specific to the particular command selected.

Notes-

All commands beginning with the prefix ***WAIT*** create a timed asynchronous event. A timed asynchronous event calls the user function on two conditions. The first is if the timed event had occurred. In this case the user interrupt routine is called with the lowest bit of the **val** parameter to the user interrupt function set to 1. The second is if the timed event did not occur and the time has expired.. In this case the lowest bit of the **val** parameter to the user interrupt function is set to 0.

The return value from the user determines if the event will be recycled. If the return from the user interrupt function is 0 then the event is freed, otherwise it is recycled with the returned value being the new event time-out. The user interrupt function must have been previously installed with the **ser_rs232_set_intfunc()** function with the **INTFUNC_OTHER** bit set. See the description

WAITCDON This command creates a timed asynchronous event callback that will wait up to the specified tics in **arg** for the carrier detect to turn on. The user interrupt function is called with the high byte in **val** set to INTFUNC_VWAITCDON.

WAITCDOFF This command creates a timed asynchronous event that will wait up to the specified tics in **arg** for the carrier detect to turn off. The user interrupt function is called with the high byte in **val** set to INTFUNC_VWAITCDOFF.

WAITCTSON This command creates a timed asynchronous event that will wait up to the specified tics in **arg** for the clear to send signal to turn on. The user interrupt function is called with the high byte in **val** set to INTFUNC_VWAITCTSON.

WAITCTSOFF This command creates a timed asynchronous event that will wait up to the specified tics in **arg** for the clear to send signal to turn off. The user interrupt function is called with the high byte in **val** set to INTFUNC_VWAITCTSOFF.

WAITDSRON This command creates a timed asynchronous event that will wait up to the specified tics in **arg** for the data set ready signal to turn on. The user interrupt function is called with the high byte in **val** set to INTFUNC_VWAITDSRON.

WAITDSROFF This command creates a timed asynchronous event that will wait up to the specified tics in **arg** for the data set ready signal to turn off. The user interrupt function is called with the high byte in **val** set to INTFUNC_VWAITDSROFF.

WAITRCOUNT This command creates a timed asynchronous event that will wait up to the specified tics in **arg** for the receive buffer to be equal to or greater than the value set with the **ser_rs232_misc_func() SETRCOUNT** command. The user interrupt function is called with the high byte in **val** set to INTFUNC_VWAITRCOUNT.

WAITRION This command creates a timed asynchronous event that will wait up to the specified tics in **arg** for the ring indicator signal to turn on. The user interrupt function is called with the high byte in **val** set to INTFUNC_VWAITRION.

WAITRIOFF This command creates a timed asynchronous event that will wait up to the specified tics in **arg** for the ring indicator signal to turn off. The user interrupt function is called with the high byte in **val** set to INTFUNC_VWAITRIOFF.

WAITXEMPTY This command creates a timed asynchronous event that will wait up to the specified tics in **arg** for the transmit buffer to empty. The user interrupt function is called with the high byte in **val** set to INTFUNC_VWAITXEMPTY.

WAITXNOTEMPTY This command creates a timed asynchronous event that will wait up to the specified tics in **arg** for the transmit buffer to be not empty. The user interrupt function is called with the high byte in **val** set to INTFUNC_VWAITNOTXEMPTY.

WAITREMPTY This command creates a timed asynchronous event that will wait up to the specified tics in **arg** for the receive buffer to empty. The user interrupt function is called with the high byte in **val** set to INTFUNC_VWAITREMPTY.

WAITRNOTEMPTY This command creates a timed asynchronous event that will wait up to the specified tics in **arg** for the receive buffer to be not empty. The user interrupt function is called with the high byte in **val** set to INTFUNC_VWAITRNOTEMPTY.

WAITTIME This command creates a timed asynchronous event that will wait up to the specified tics in **arg** before issuing a callback to the user interrupt function. The user interrupt function is called with the high byte in **val** set to INTFUNC_VWAITTIME.

WAITXCOUNT This command creates a timed asynchronous event that will wait up to the specified tics in **arg** for the transmit buffer to be equal to or less than the value set with the **ser_rs232_misc_func() SETXCOUNT** command. The user interrupt function is called with the high byte in **val** set to INTFUNC_VWAITXCOUNT.

SENDNOW This command causes data to be transmitted ahead of any data buffered in the transmit buffer. **arg** is the far pointer to data structured as follows

```
struct
{
unsigned    count;
char        data[count];
};
```

Count cannot be larger than **SENDNOWBUFSIZE**. This mechanism that is used to transmit the data is referred to as the **"sendnow"** mechanism, transmission, or method throughout the manual. Note that there can be only one outstanding **"sendnow"** in effect per port at anytime. As such if another thread of the application issued a **"sendnow"** transmission, and it is not complete when this function is called, this function will fail. Whenever a **SENDNOW** data transmission terminates, the asynchronous user callback (if it was installed with the ***ser_rs232_set_intfunc()*** function) will be called with the high byte of the **val** parameter set to **INTFUNC_VSENDNOWDONE**.

SETRCVCHR This command sets the character that will be used as a receive character comparator (**arg**). It is used by the asynchronous callback function INTFUNC_RCVCHR that is set with the **ser_rs232_set_intfunc()** function.

SETXMTCHR This command sets the character that will be used as a transmit character comparator (**arg**). It is used by the asynchronous callback function INTFUNC_XMTCHR that is set with the **ser_rs232_set_intfunc()** function.

SETIBFDCNT This command sets the count that will be used to generate an asynchronous callback to the user interrupt function. The callback occurs when the input buffer decrements to the count in **arg**. It is used by the asynchronous callback function INTFUNC_IBFDCNT that is set with the **ser_rs232_set_intfunc()** function.

SETIBFICNT This command sets the count that will be used to generate an asynchronous callback to the user interrupt function. The callback occurs when the input buffer increments to the count in **arg**. It is used by the asynchronous callback function INTFUNC_IBFICNT that is set with the **ser_rs232_set_intfunc()** function.

SETOBFDCNT This command sets the count that will be used to generate an asynchronous callback to the user interrupt function. The callback occurs when the output buffer decrements to the count in **arg**. It is used by the asynchronous callback function INTFUNC_OBFDCNT that is set with the **ser_rs232_set_intfunc()** function.

SETOBFICNT This command sets the count that will be used to generate an asynchronous callback to the user interrupt function. The callback occurs when the output buffer increments to the count in **arg**. It is used by the asynchronous callback function INTFUNC_OBFICNT that is set with the **ser_rs232_set_intfunc()** function.

SETRCOUNT This command sets the receive buffer count used by the **WAITRCOUNT** timed event. **arg** specifies the count.

SETXCOUNT This command sets the receive buffer count used by the **WAITXCOUNT** timed event. **arg** specifies the count.

On Exit-

int stat;

Error codes described under *errors* in APPENDIX C.

See Also-

ser_rs232_set_intfunc()

Examples-

```
#include <comm.h>

int     stat;
int     port=0;

if ((stat = ser_rs232_misc_func(port,WAITRNOTEMPTY,18))!=RS232ERR_NONE)
        {
        printf("Error # %d\n");

        /* Take remedial action */
        }
```

ser_rs232_putbyte

Description-

This routine puts a byte into the circular output communication buffer and is queued for transmission. If a time-out was set with the **ser_rs232_block()** or the **ser_rs232_setup()** call, then this call will hang for up to the time-out value before returning with an error if no space in output buffer was available.

Syntax-

stat = ser_rs232_putbyte(port,ch)

On Entry-

int port;

A port previously opened with the **ser_rs232_setup()** function.

char *ch;

Pointer to buffer with character to output.

On Exit-

int stat;

Error codes described under **errors** in **APPENDIX C.**

See Also-

ser_rs232_putpacket()

Examples-

```
#include <comm.h>

int             stat;
int             port=0;
static char     ch[]='a';

if ((stat = ser_rs232_putbyte(port,ch)) != RS232ERR_NONE)
        {
        printf("Error # %d\n");

        /* Take remedial action */
        }
```

ser_rs232_putpacket

Description-

This routine puts a fixed size packet into the output communication circular buffer. Note that if the output buffer is not at least the size of the packet the call will always fail. If a time-out was set with the **ser_rs232_block()** or the **ser_rs232_setup()** call, then this call will hang for up to the time-out value before returning with an error if no space in output buffer was available.

Syntax-

stat = ser_rs232_putpacket(port,count,ch)

On Entry-

int port;

A port previously opened with the **ser_rs232_setup()** function.

int count;

Number of bytes to read (Size of packet).

char *ch;

Pointer to buffer to place packet..

On Exit-

int stat;

Error codes described under **errors** *in* **APPENDIX C**.

See Also-

ser_rs232_putbyte()

Examples-

```
#include <comm.h>

int             stat;
int             port=0;
static char     ch[]="PASSWORD: ;

if ((stat = ser_rs232_putpacket(port,sizeof(ch),ch)) != RS232ERR_NONE)
        {
        printf("Error # %d\n");

        /* Take remedial action */
        }
```

ser_rs232_putregister

Description-

Writes a byte to a particular UART's register. Because **COMM-DRV** also works with non-standard serial port, this call may not be supported in those instances.

Syntax-

stat = ser_rs232_putregister(port,offset,value);

On Entry-

int port;

A port previously opened with the **ser_rs232_setup()** function.

int offset;

Offset from base of the physical UART address. These offsets are define symbolically in **comm.h**. They are

BAUD_LOW	Low baud rate divisor register.
BAUD_HIGH	High baud rate divisor register.
DATABUF	Input/output data register.
IER	Interrupt enable register.
IIR	Interrupt identification register.
FCR	FIFO control register.
LCR	Line control register.
MCR	Modem control register.
MSR	Modem status register.
SCR.	Scratch register.

int value;

Value to put in register.

On Exit-

int stat;

Error codes described under errors in **APPENDIX C**.

See Also-

ser_rs232_getregister()

Examples-

```
#include <comm.h>

int     stat;
int     port=0;
int     value=0x3e;

if ((stat = ser_rs232_putregister(port,MCR,value)) != RS232ERR_NONE)
        {
        printf("Error # %d\n");

        /* Take remedial action */
        }
```

ser_rs232_setbauddiv

Description-

This routine sets the baud rate divisor. This routine is a device specific function and should be used with care.

Syntax-

stat = ser_rs232_setbauddiv(cardtype,baudidx,bauddiv);

int cardtype;

Sub device to be modified. Values defined in **Appendix B**.

int baudidx;

Baudrate index. This corresponds to the baud rate whose divisor will be replaced (e.g., BAUD9600).

int bauddiv;

If cardtype=CARD_NORMAL or CARD_WCSCVXD, then the value to use as the new baud rate divisor is

$$\text{BaudrateDivisor} = \frac{1{,}843{,}200}{(\text{DesiredBaudrate X } 16)}$$

If cardtype=CARD_WINAPI, then the value to use as the new baud rate divisor is simply the baudrate(e.g., for baud=4650 then bauddiv=4650). Other cardtypes are not supported.

On Exit-

int stat;

Error codes described under **errors** in **APPENDIX C**.

See Also-

Examples-

```
#include <comm.h>


int             stat;
int             port=0;
int             cardtype = CARD_NORMAL;
int             baudidx = BAUD300;
int             bauddiv=2;

if ((stat = ser_rs232_setbauddiv(cardtype,baudidx,bauddiv)) != RS232ERR_NONE)
        {
        printf("Error # %d\n");

        /* Take remedial action */
        }
```

ser_rs232_rts_off

Description-

Turns the RTS signal off.

Syntax-

stat = ser_rs232_rts_off(port);

On Entry-

int port;

A port previously opened with the **ser_rs232_setup()** function.

On Exit-

int stat;

Error codes described under **errors** in **APPENDIX C**.

See Also-

ser_rs232_rts_on()

Examples-

```
#include <comm.h>

int             stat;
int             port=0;

if ((stat = ser_rs232_rts_off(port)) != RS232ERR_NONE)
        {
        printf("Error # %d\n");

        /* Take remedial action */
        }
```

ser_rs232_rts_on

Description-

Turns the RTS signal on.

Syntax-

stat = ser_rs232_rts_on(port);

On Entry-

int port;

A port previously opened with the **ser_rs232_setup()** function.

On Exit-

int stat;

Error codes described under **errors** in **APPENDIX C.**

See Also-

ser_rs232_rts_off()

Examples-

```
#include <comm.h>

int             stat;
int             port=0;

if ((stat = ser_rs232_rts_on(port)) != RS232ERR_NONE)
        {
        printf("Error # %d\n");

        /* Take remedial action */
        }
```

ser_rs232_set_intfunc

Description-

Installs or de-installs user interrupt function. The example **exintfnc.c** and several Windows examples illustrates the usage of this routine. The routine that is installed is called either during a timer interrupt or a serial communication interrupt. As such the user interrupt function must be reentrant.

Syntax-

stat = ser_rs232_set_intfunc(port,func,imask);

On Entry-

int port;

A port previously opened with the **ser_rs232_setup()** function.

short ((far * (far *func))(short val);

Variable with pointer to the user interrupt function. This was defined this way so that with one argument we are able to get both the function and its data segment.

short imask;

The different masks in imask tells what asynchronous events to monitor. The special mask **INTFUNC_X** allows the setting of an additional set of masks. Following are the events. If imask==0 then all interrupts to user interrupt functions are disabled The different masks may be orred together to create asynchronous callbacks for several events.

Masks valid only if INTFUNC_X is not orred to *imask*.

INTFUNC_MODEM Causes a callback to the user interrupt function on every modem status change. The user interrupt function is called with the high byte of the **val** parameter set to **INTFUNC_VMODEM**. The low byte of **val** contains the modem status register.

INTFUNC_STAT Causes a callback to the user interrupt function on every line status change. The user interrupt function is called with the high byte of the **val** parameter set to **INTFUNC_VSTAT**. The low byte of **val** contains the line status register.

INTFUNC_RCV Causes a callback to the user interrupt function on reception of every character. The user interrupt function is called with the high byte of the **val** parameter set to **INTFUNC_VRCV**. The low byte of **val** contains the received character. If the user interrupt function returns -1 then the character is not buffered, otherwise the user return is taken as the actual character to buffer.

INTFUNC_XMIT Causes a callback to the user interrupt function before transmission of every character. If the user interrupt function returns a -1 then the character is not transmitted, otherwise the user return is taken as the actual character to transmit. The user interrupt function is called with the high byte of the **val** parameter set to **INTFUNC_VXMIT**. The low byte of **val** contains the character to be transmitted.

INTFUNC_IBFDCNT Causes a callback to the user interrupt function when the input buffer decrements to the value that was previously specified with **ser_rs232_misc_func()** function, sub-command **SETIBFDCNT**.

INTFUNC_IBFICNT Causes a callback to the user interrupt function when the input buffer increments to the value that was previously specified with **ser_rs232_misc_func()** function, sub-command **SETIBFICNT**.

INTFUNC_OBFDCNT Causes a callback to the user interrupt function when the output buffer decrements to the value that was previously specified with **ser_rs232_misc_func()** function, sub-command **SETOBFDCNT**.

INTFUNC_OBFICNT Causes a callback to the user interrupt function when the output buffer increments to the value that was previously specified with **ser_rs232_misc_func()** *function*, sub-command **SETOBFICNT**.

INTFUNC_OTHER This bit must be set if their will be timed event callbacks that are setup with the ***ser_rs232_misc_func()***.

INTFUNC_RCVCHR Causes a callback to the user interrupt function on reception of the character set by the ser_rs232_misc_func() function's sub-command **SETRCVCHR**. The user interrupt function is called with the high byte of the **val** parameter set to **INTFUNC_VRCVCHR**. The user interrupt function should return the new character that will cause the next interrupt function of this type.

INTFUNC_XMTCHR Causes a callback to the user interrupt function on transmission of the character set by the **ser_rs232_misc_func()** function's sub-command **SETXMTCHR**. The user interrupt function is called with the high byte of the **val** parameter set to **INTFUNC_VXMTCHR**. The user interrupt function should return the new character that will cause the next interrupt function of this type.

Masks valid only if INTFUNC_X is orred to *imask*.

INTFUNC_X_CTS Causes a callback to the user interrupt function on every change of the state of the **CTS** input. The user interrupt function is called with the high byte of the **val** parameter set to **INTFUNC_VX_CTS**. Note that the low byte of **val** contains the modem status register. As such, the actual state may be determined by testing the low byte of **val** with **MSR_CTS**. This mask is only valid if **INTFUNC_X** is also orred.

INTFUNC_X_CD Causes a callback to the user interrupt function on every change of the state of the **CD**(carrier detect) input. The user interrupt function is called with the high byte of the **val** parameter set to **INTFUNC_VX_CD**. Note that the low byte of val contains the modem status register. As such, the actual state may be determined by testing the low byte of **val** with **MSR_DCD**. This mask is only valid if **INTFUNC_X** is also orred.

INTFUNC_X_DSR Causes a callback to the user interrupt function on every change of the state of the **DSR** input. The user interrupt function is called with the high byte of the val parameter set to **INTFUNC_VX_DSR**. Note that the low byte of **val** contains the modem status register. As such, the actual state may be determined by testing the low byte of **val** with **MSR_DSR**. This mask is only valid if **INTFUNC_X** is also orred.

INTFUNC_X_RING Causes a callback to the user interrupt function on every change of the state of the **RI**(ring indicator) input. The user interrupt function is called with the high byte of the **val** parameter set to **INTFUNC_VX_RING**. Note that the low byte of **val** contains the modem status register. As such, the actual state may be determined by testing the low byte of **val** with **MSR_RI**. This mask is only valid if **INTFUNC_X** is also orred.

INTFUNC_X_BREAK Causes a callback to the user interrupt function on a break detect. The user interrupt function is called with the high byte of the **val** parameter set to **INTFUNC_VX_BREAK**. Note that the low byte of **val** contains the line status register. This mask is only valid if **INTFUNC_X** is also orred.

INTFUNC_X_PARER Causes a callback to the user interrupt function on a parity error detect. The user interrupt function is called with the high byte of the **val** parameter set to **INTFUNC_VX_PARER**. Note that the low byte of **val** contains the line status register. This mask is only valid if **INTFUNC_X** is also orred.

INTFUNC_X_FRAME Causes a callback to the user interrupt function on a frame error detect. The user interrupt function is called with the high byte of the **val** parameter set to **INTFUNC_VX_FRAME**. Note that the low byte of **val** contains the line status register. This mask is only valid if **INTFUNC_X** is also orred.

INTFUNC_X_OVRUN Causes a callback to the user interrupt function on an overrun error detect. The user interrupt function is called with the high byte of the **val** parameter set to **INTFUNC_VX_OVRUN**. Note that the low byte of **val** contains the line status register. This mask is only valid if **INTFUNC_X** is also orred.

On Exit-

```
int     stat;
```

Error codes described under **errors** in **APPENDIX C**.

See Also-

ser_rs232_misc_func()

Examples-

```
#include <comm.h>


int             stat;
int             port=0;
int             value=0;

//Function called at interrupt time
short FAR intfunc(short v,struct port_param far *p)
        {
        switch(v>>8)
        {
        /* Modem status change */
        case    INTFUNC_VMODEM:
        value++;
        break;

        return(v);
        }

short (FAR * CDECL func)(short v,struct port_param FAR *p)=intfunc;

/*****/

main()
{
if      ((stat      =       ser_rs232_set_intfunc(port,&func,INTFUNC_MODEM))      !=
RS232ERR_NONE)
        {
        printf("Error # %d\n");

        /* Take remedial action */
        }
```

ser_rs232_setup

Description-

This routine initializes the communication port, setting vectors and buffers as necessary. When the libraries are linked into the application or when the DLL **CDRVDLL.DLL** is used buffers and vectors are setup the very first time **ser_rs232_setup()** is called. Thereafter the **ser_rs232_setup()** call is only used to modify parameters like the line characteristics(baud rate, parity, stopbits, time-outs, etc.). Before issuing the setup call, the **ser_rs232_getport()** call should be made to pre-initialize the port control block that will be used to setup the port.

Syntax-

stat = ser_rs232_setup(port,pcb)

On Entry-

int port;

Port number to open.

struct port_param *pcb;

Pointer to an initialized port control block. See **APPENDIX B** to determine the values necessary for this structures initialization.

On Exit-

int stat;

Error codes described under errors in **APPENDIX C**.

See Examples-

ser_rs232_getport()

Examples-

```
#include<comm.h>

#define     INBUF  1024
#define     OUTBUF 512
#define     COM1IRQ  0x4
#define     COM1ADR  0x3f8
static char buf[INBUF+1+OUTBUF+1];


if ((stat = ser_rs232_getport(port,&pcb)) != RS232ERR_NONE)
        {
        printf("Error # %d\n");


        /* Take remedial action */
        }

pcb.ser_rs232_base  = COM1ADR;
pcb.irq             = COM1IRQ;
pcb.baud            = BAUD9600;
pcb.parity          = PAR_NONE;
pcb.lngth           = LENGTH_8;
pcb.stopbit         = STOPBIT_1;
pcb.brk             = BREAK_OFF;
pcb.protocol        = PROT_NONNON;
pcb.block[0]        = 0;
pcb.block[1]        = 4;
pcb.inbuf_low       = INBUF/4;
pcb.inbuf_high      = INBUF/2;
pcb.buffer_seg      = (unsigned short)(((long)((void far *)buf))>>16);
pcb.buffer_off      = (unsigned short)((long)(void far *)buf);
pcb.inbuf_len       = INBUF;
pcb.outbuf_len      = OUTBUF;
pcb.cardtype        = CARD_NORMAL;
pcb.aux_addr1       = 0;
pcb.cardseg         = 0x0;
```

```
if ((stat = ser_rs232_setup(port,&pcb)) != RS232ERR_NONE)
        {
        printf("Error # %d\n");

        /* Take remedial action */
        }
```


ser_rs232_viewpacket

Description-

This routine gets a fixed length packet from the input communication buffer. Note that if the size of the input communication buffer is not at least the size of the packet then this call will always fail. The data isn't actually removed from the buffer. It is a non-destructive read.

Syntax-

stat = ser_rs232_viewpacket(port,count,ch)

On Entry-

int port;

A port previously opened with the **ser_rs232_setup()** function.

int count;

Number of bytes to read.

char *ch;

Pointer to the buffer to place the packet.

On Exit-

int stat;

Error codes described under **errors** in **APPENDIX C.**

See Also-

ser_rs232_getbyte()
ser_rs232_getpacket ()

Examples-

```
#include <comm.h>


int             stat;
int             port=0;
char            ch[16];


if ((stat = ser_rs232_viewpacket(port,sizeof(ch),ch)) != RS232ERR_NONE)
        {
        printf("Error # %d\n");


        /* Take remedial action */
        }
```


User Callback Functions

The user callback functions may be called from within a user callback routine. Other **COMM-DRV/LIB** functions must not be called from within user callbacks.

ser_rs232_ifnc_dtr_off()	Turns the DTR signal off.
ser_rs232_ifnc_dtr_on()	Turns the DTR signal on.
ser_rs232_ifnc_getpacket()	Inputs a packet.
ser_rs232_ifnc_putpacket()	Output a packet.
ser_rs232_ifnc_rts_off()	Turns RTS off.
ser_rs232_ifnc_rts_on()	Turns RTS on.

ser_rs232_ifnc_dtr_off

Description-

Turns the DTR signal off.

Syntax-

stat = ser_rs232_ifnc_dtr_off(p);

On Entry-

struct port_param *p;

Pointer to the PCB for the particular port. This is the second parameter passed to the user interrupt function.

On Exit-

int stat;

Error codes described under **errors** in **APPENDIX C**.

See Also-

ser_rs232_ifnc_dtr_on()

Examples-

```
#include<comm.h>

int                 stat;
int                 port=0;
char                ch[10];
struct port_param   pcb;

if ((stat = ser_rs232_ifnc_dtr_off(&pcb)) != RS232ERR_NONE)
        {
        printf("Error # %d\n");

        /* Take remedial action */
        }
```

ser_rs232_ifnc_dtr_on

Description-

Turns the DTR signal on.

Syntax-

stat = ser_rs232_ifnc_dtr_on(p);

On Entry-

struct port_param *p;

Pointer to the PCB for the particular port. This is the second parameter passed to the user interrupt function.

On Exit-

int stat;

Error codes described under **errors** in **APPENDIX C**.

See Also-

ser_rs232_ifnc_dtr_off()

Examples-

```
#include <comm.h>

int             stat;
int             port=0;
char            ch[10];
struct port_param   pcb;

if ((stat = ser_rs232_ifnc_dtr_on(&pcb)) != RS232ERR_NONE)
        {
        printf("Error # %d\n");

        /* Take remedial action */
        }
```

ser_rs232_ifnc_getpacket

Description-

This functions is called from a user interrupt function. The function takes data from the **COMM-DRV** receive buffer and stuffs it at the location specified by the passed pointer. If the requested number of bytes is not present, then the function fails.

Syntax-

stat = ser_rs232_ifnc_getpacket(p,buf,cnt);

On Entry-

struct port_param *p;

Pointer to the PCB for the particular port. This is the second parameter passed to the user interrupt function.

char *buf;

Pointer to the buffer for inputted data.

int count;

Number of characters to input.

On Exit-

int stat;

Error codes described under **errors** in **APPENDIX C.**

See Also-

ser_rs232_ifnc_putpacket()

Examples-

```
#include <comm.h>

int             stat;
int             port=0;
char            ch[10];
struct port_param pcb;

if ((stat = ser_rs232_ifnc_getpacket(&pcb,ch,sizeof(ch))) != RS232ERR_NONE)

        {
        printf("Error # %d\n");

        /* Take remedial action */
        }
```

ser_rs232_ifnc_putpacket

Description-

Output a packet. This functions is called from a user interrupt function. The function takes the passed pointer to the data and returns immediately. The mechanism that is used to transmit the data is referred to as the **"sendnow"** mechanism, transmission, or method throughout the manual. Note that there can be only one outstanding **"sendnow"** in effect per port at anytime. As such if another thread of the application issued a **"sendnow"** transmission, and it is not complete when this function is called, this function will fail. Whenever a **"sendnow"** data transmission terminates, the asynchronous user callback will be called with the high byte of the val parameter set to **INTFUNC_VSENDNOWDONE**.

Syntax-

stat = ser_rs232_ifnc_putpacket(p,buf,cnt);

On Entry-

struct port_param *p;

Pointer to the PCB for the particular port. This is the second parameter passed to the user interrupt function.

char *buf;

Pointer to the buffer with data to output. Note that this data must remain available since when this function returns the data transmission would most likely not be completed as yet.

int count;

Number of characters to output.

On Exit-

int stat;

Error codes described under **errors** in **APPENDIX C**.

See Also-

ser_rs232_ifnc_getpacket()

Examples-

```
#include<comm.h>


int             stat;
int             port=0;
char            ch[10];
struct port_param   pcb;


if ((stat = ser_rs232_ifnc_putpacket(&pcb,ch,sizeof(ch))) != RS232ERR_NONE)

        {
        printf("Error # %d\n");


        /* Take remedial action */
        }
```

ser_rs232_ifnc_rts_off

Description-

Turns the RTS signal off.

Syntax-

stat = ser_rs232_ifnc_rts_off(p);

On Entry-

struct port_param *p;

Pointer to the PCB for the particular port. This is the second parameter passed to the user interrupt function.

On Exit-

int stat;

Error codes described under errors in **APPENDIX C**.

See Also-

ser_rs232_ifnc_rts_on()

Examples-

```
#include <comm.h>


int             stat;
int             port=0;
char            ch[10];
struct port_param   pcb;


if ((stat = ser_rs232_ifnc_rts_off(&pcb)) != RS232ERR_NONE)
        {
        printf("Error # %d\n");


        /* Take remedial action */
        }
```

ser_rs232_ifnc_rts_on

Description-

Turns the RTS signal on.

Syntax-

stat = ser_rs232_ifnc_rts_on(p);

On Entry-

struct port_param *p;

Pointer to the PCB for the particular port. This is the second parameter passed to the user interrupt function.

On Exit-

int stat;

Error codes described under errors in **APPENDIX C**.

See Also-

ser_rs232_ifnc_rts_off()

Examples-

```
#include <comm.h>


int                 stat;
int                 port=0;
char                ch[10];
struct port_param   pcb;


if ((stat = ser_rs232_ifnc_rts_on(&pcb)) != RS232ERR_NONE)
        {
        printf("Error # %d\n");

        /* Take remedial action */
        }
```

Using The Assembly Language Interface

Assembly language programmers may call any of the serial communication routines without the C overhead. These routines will be embedded in the application code. These routines are simply called as per the conventions shown below.

Applications should have the following externals declared

```
extrn   ser_rs232_block:near
extrn   ser_rs232_cleanup:near
extrn   ser_rs232_dtr_off:near
extrn   ser_rs232_dtr_on:near
extrn   ser_rs232_flush:near
extrn   ser_rs232_getbyte:near
extrn   ser_rs232_getpacket:near
extrn   ser_rs232_getport:near
extrn   ser_rs232_getregister:near
extrn   ser_rs232_getstatus:near
extrn   ser_rs232_putbyte:near
extrn   ser_rs232_putpacket:near
extrn   ser_rs232_putregister:near
extrn   ser_rs232_max_port:near
extrn   ser_rs232_rts_off:near
extrn   ser_rs232_rts_on:near
extrn   ser_rs232_set_intfunc:near
extrn   ser_rs232_setbauddiv:near
extrn   ser_rs232_setup:near
extrn   ser_rs232_viewpacket:near
;or
```



```
extrn   ser_rs232_block:far
extrn   ser_rs232_cleanup:far
extrn   ser_rs232_dtr_off:far
extrn   ser_rs232_dtr_on:far
extrn   ser_rs232_flush:far
extrn   ser_rs232_getbyte:far
extrn   ser_rs232_getpacket:far
extrn   ser_rs232_getport:far
extrn   ser_rs232_getregister:far
extrn   ser_rs232_getstatus:far
extrn   ser_rs232_putbyte:far
extrn   ser_rs232_putpacket:far
extrn   ser_rs232_putregister:far
extrn   ser_rs232_max_port:far
extrn   ser_rs232_rts_off:far
extrn   ser_rs232_rts_on:far
extrn   ser_rs232_set_intfunc:far
extrn   ser_rs232_setbauddiv:far
extrn   ser_rs232_setup:far
extrn   ser_rs232_viewpacket:far
```

ser_rs232_block

Description-

Sets the number of 1/18th seconds to wait for outputting data or receiving data.

On Entry-

si port #(0-n).
ai Number of 1/18th seconds to wait on input.
ah Number of 1/18th seconds to wait on output.
If both al = 255 and ah = 255 then
bx Number of 1/18th seconds to wait on input
cx Number of 1/18th seconds to wait on output

On Exit-

Carry is set if error.

All registers preserved.

ser_rs232_cleanup

Description-

Restores interrupt vector if was not changed, reset 8259, resets 8250, and cleanup necessary port information.

On Entry-

si port #(0-n).

On Exit-

Carry is set if error occurred.

All registers preserved.

ser_rs232_dtr_off

Description-

Turns DTR off.

On Entry-

si port #(0-n).

On Exit-

Carry set if an error occurred.

All registers preserved.

ser_rs232_dtr_on

Description-

Turns DTR on.

On Entry-

si - port #(0-n).

On Exit-

if stc then error.

All registers preserved.

ser_rs232_flush

Description-

Flushes either the input or output communication buffer.

On Entry-

si	port #(0-n).
ax	0 to flush the input buffer. 1 to flush the output buffer. 2 to flush both buffers.

On Exit-

if stc then error.
All registers preserved.

ser_rs232_getbyte

Description-

Get a byte from circular input buffer.

On Entry-

si	port #(0-n).

On Exit-

al	Extracted byte
ah	Line status register

If carry is set then an error occurred.

All registers preserved.

ser_rs232_getpacket

Description-

Get a packet from the circular input buffer. Note that if the input buffer is not at least the size of the packet the call will always fail.

On Entry-

si	port #(0-n).
cx	Byte count of packet.
es:di	Segment:offset of packet.

On Exit-

ah	Line status register

If carry set then an error occurred.

All registers preserved.

ser_rs232_getport

Description-

Get the byte counts, error codes, and status flag for the particular port. The results are either returned in registers or placed into a user provided structure (**struc port_param**) defined in **COMM.INC**.

On Entry-

si	port #(0-n).	
al	0	Then return select metrics in registers.
	1	ds:di points to buffer defined with the port_param structure.
	2	ds:di points to buffer defined with the port_param structure.
	3	Then return select metrics in registers.

On Exit-

if al=0 on input

ax	error code for last occurring error.
bx	# of characters in output buffer.
cx	# of characters in input buffer.
dx	state flag

if al=1 on input

Data copied to memory storage that was passed in ds:si. The data was defined by the structure **port_param** in **COMM.INC** or **COMM.H**. Values as shown in **Appendix D**.

if al = 2 on input

Same as (al=1) but only returns the first 20 words of the **port_param** structure.

if al = 3 on input

ax	# of characters in input buffer.
bx	Size of input buffer.
cx	# of characters in output buffer.
dx	Size of output buffers.

If carry set the an error occurred.

All registers preserved.

ser_rs232_getregister

Description-

Get RS232 register.

On Entry-

si	port #(0-n).
al	register offset from port base

On Exit-

ah	value read from port.

If carry set the an error occurred.

All registers preserved.

ser_rs232_getstatus

Description-

Get the status as per INT14.

On Entry-

si port #(0-n).

On Exit-

ah line status register
al modem status register

If carry set the an error occurred.

All registers preserved.

ser_rs232_max_port

Description-

Get maximum number of ports allowed by this library.

On Entry-

None.

On Exit-

ax maximum number of ports.

If carry set the an error occurred.

All registers preserved.

ser_rs232_putbyte

Description-

Put a byte in circular output buffer.

On Entry-

si	port #(0-n).
al	Byte to transmit.

On Exit-

ah	Line status register

If carry set the an error occurred.

All registers preserved.

ser_rs232_putpacket

Description-

Put a packet in circular output buffer. Note that if the output buffer is not at least the size of the packet the call will always fail.

On Entry-

si	port #(0-n).
cx	Byte count of packet.
es:di	Segment:offset of packet.

On Exit-

ah	Line status register

If carry set the an error occurred.

All registers preserved.

ser_rs232_putregister

Description-

Put RS232 register.

On Entry-

si	port #(0-n).
al	register offset from port base
ah	value to put

On Exit-

If carry set the an error occurred.

All registers preserved.

ser_rs232_rts_off

Description-

Turns RTS off.

On Entry-

si port #(0-n).

On Exit-

If carry set the an error occurred.

All registers preserved.

ser_rs232_rts_on

Description-

Turns RTS on.

On Entry-

si port #(0-n).

On Exit-

If carry set the an error occurred.

All registers preserved.

ser_rs232_set_intfunc

Description-

Sets the address of a user function that will be interrupted on interrupts.

On Entry-

si	port #(0-n).
ds:di	Address of variable with address of function to call (segment:offset). Variable must be in the data segment of function.
ax	Interrupts to process mask.

The values in ax are identical to those described in the section ***USING THE C LANGUAGE INTERFACE for*** function ***ser_rs232_set_intfunc()'s imask*** value.

On Exit-

If carry set error occurred.

All registers preserved.

ser_rs232_setbauddiv

Description-

Sets the baud rate divisor to a new value..

On Entry-

cx	New baud rate divisor.
dx	Index to the appropriate baud.

On Exit-

If carry set then error.

All registers preserved.

ser_rs232_setup

Description-

Perform actual serial port and interrupt handler initialization.

On Entry-

si	port # (0-n)
ds:di	Points to memory storage defined by the structure port_param in COMM.INC or COMM.H. See APPENDIX B for values to be used for initialization.

On Exit-

ah	line status register
al	modem status register

If carry set then error.

All registers preserved.

ser_rs232_viewpacket

Description-

Get a packet from the circular input buffer. Note that if the input buffer is not at least the size of the packet the call will always fail. This is a non-destructive call.

On Entry-

si	port #(0-n).
cx	Byte count of packet.
es:di	Segment:offset of packet.

On Exit-

ah Line status register

If carry set, an error occurred.

All registers preserved.

Appendix A (Tips)

Problem:

It seems that data is not being transmitted.

Solution:

Check your cabling. You may have selected hardware protocol and not assert CTS and DSR. Try running with no hardware or software protocol(protocol = PROT_NONNON). If problem persist, verify that the serial card is on your selected IRQ.

Problem:

When linking the libraries directly into the application, there are several undefined symbols.

Solution:

Make sure you are linking with the appropriate libraries. Read the section **Getting *Started*** in detail to decide which libraries to use.

Problem:

After exiting my application in Windows or MS-DOS, my system hangs.

Solution:

This problem typically occurs if the **COMM-DRV** routines are actually linked into the application or if the **CDRVDLL.DLL** DLL is used. *It is necessary to issue* the **UnInitializePort()** or **ser_rs232_cleanup()** call on every port that was opened with the **InitializePort()** or **ser_rs232_setup()** before the application exits.

Problem:

I only need one port in my application. How do I reduce the overhead that is imposed by the **COMM-DRV** libraries for supporting 32 ports and several different types of communication boards?.

Solution:

Modify the files **comm.h** and **comm.inc** to reflect the amount of resources needed. If your application only needs one port you could make the following changes to **comm.inc**, **comm.h**. Note that a batch file is distributed with **COMM-DRV/LIB** to rebuild the libraries. Simply run **makemlib.bat** and **makeblib.bat** to make the Microsoft or Borland **COMM-DRV** libraries respectively.

MAX_PORT	7	Maximum port number to use.
MAX_PORTS	1	Max number of concurrently active ports.
MAX_DRIVERS	1	Only using generic serial ports.
MAX_IRQ	12	Maximum IRQ number used.
MAX_IRQS	1	Maximum concurrently active IRQs.

Appendix B (Initialization Structure)

All calls that use the **port_param** structure for initialization should stuff the individual structure elements. Note that if the port was already initialized then only the baud, parity, lngth, stopbit, brk, protocol, and block fields can be changed. In general it is a good practice to first read in the **port_param** structure for the particular port (using the ser_rs232_getport call) and modifying only the elements that should be changed.

Following are the elements and the values they can take on. Note that the symbolic values can be found in **comm.h** or **comm.inc**.

ser_rs232_base

Base address of the 8250(or compatible chip) port. If a smart card is being used then it would be the address of the smart cards O/O port.

irq

0	IRQ 0
1	IRQ 1
2	IRQ 2
3	IRQ 3
4	IRQ 4
5	IRQ 5
6	IRQ 6
7	IRQ 7
8	IRQ 8
9	IRQ 9
10	IRQ 10
11	IRQ 11
12	IRQ 12
13	IRQ 13
14	IRQ 14
15	IRQ 15

baud

BAUD110	110 baud
BAUD150	150 baud
BAUD300	300 baud
BAUD600	600 baud
BAUD1200	1200 baud
BAUD2400	2400 baud
BAUD4800	4800 baud
BAUD9600	9600 baud
BAUD14400	14400 baud
BAUD19200	19200 baud
BAUD28800	28800 baud
BAUD38400	38400 baud
BAUD57600	57600 baud
BAUD115200	115200 baud

parity

PAR_NONE	none
PAR_ODD	odd
PAR_EVEN	even
PAR_SODD	sticky odd(MARK Parity)
PAR_SEVEN	sticky even(SPACE Parity)

brk

BREAK_OFF	Off
BREAK_ON	On

lngth

LENGTH_5	5 bits
LENGTH_6	6 bits
LENGTH_7	7 bits
LENGTH_8	8 bits

stopbit

STOPBIT_1	1 stop bit
STOPBIT_2	2 stop bits

block[0]

0-255	Input block(low byte). Number of 1/18th of a second intervals to wait for a character.

hblock[0]

0-255	Input block(high byte). Number of 1/18th of a second intervals*256 to wait for a character.

block[1]

0-255	Output block(low byte). Number of 1/18th of a second intervals to wait for output buffer to have available space.

hblock[1]

0-255	Output block(high byte). Number of 1/18th of a second intervals*256 to wait for output buffer to have available space.

protocol

PROT_RTSRTS	Local CTS/RTS handshake-Remote CTS/RTS handshake.
PROT_RTSXON	Local CTS/RTS handshake-Remote XON/XOFF handshake.
PROT_RTSDTR	Local CTS/RTS handshake-Remote DSR/DTR handshake.
PROT_RTSNON	Local CTS/RTS handshake-Remote no handshake.
PROT_NONNON	Local no protocol-Remote no protocol.
PROT_NONXON	Local no protocol-XON/XOFF handshake.
PROT_XONNON	Local XOFF/XON protocol-Remote no protocol.
PROT_XONXON	Local XON/XOFF protocol-Remote XON/XOFF protocol.
PROT_DTRNON	Local DSR/DTR protocol-Remote no protocol.
PROT_DTRRTS	Local DSR/DTR protocol-Remote CTS/RTS protocol.
PROT_DTRDTR	Local DSR/DTR protocol-Remote DSR/DTR protocol.
PROT_DTRXON	Local DSR/DTR protocol-Remote DSR/DTR protocol.
PROT_NONRTS	Local no protocol-Remote CTS/RTS protocol.
PROT_NONDTR	Local no protocol-Remote DSR/DTR protocol.
PROT_XONRTS	Local XOFF/XON protocol-Remote CTS/RTS protocol.
PROT_XONDTR	Local XOFF/XON protocol-Remote DSR/DTR protocol.

The first three bytes specifies the protocol the local computer is using, while the last three bytes specifies the protocol the remote device is expected to be using. PROT_RTS???, PROT_DTR???, and PROT_XON??? means that when the local computer's input buffer is filling up, the local computer will de-assert RTS, DTR, or send an XOFF respectively to tell the remote device to stop sending. The local machine will re-assert RTS, DTR, or send an XON to make remote device restart transmission. PROT_???RTS, PROT_???DTR, and PROT_???XON means that when the remote device's input buffer is filling up, the remote device is expected to de-assert RTS, DTR, or send an XOFF respectively to tell the local computer to stop sending. The remote device is expected to re-assert RTS, DTR, or send an XON to make the local computer restart transmission.(Default PROT_RTSRTS).

inbuf_low

1-n Buffer threshold at which an XON, an RTS high, or a DTR high is sent if XON/XOFF or hardware protocol is enabled.

inbuf_high

2-n Buffer threshold at which an XOFF, an RTS low, or a DTR low is sent if XON/XOFF or hardware protocol is enabled.

buffer_seg

The segment for both input and output buffers.

buffer_off

Offset to the start of the communications I/O buffers. If the **DIFFXBUF** bit was set in **flag2** structure element, then this address indicates the offset for the input buffer. Otherwise it is the offset to the input buffer and the output buffer is located **inbuf_len+1** bytes away from this offset.

obufseg

Segment of the output communication I/O buffers. If the **DIFFXBUF** bit was not set in the **flag2** structure element, then this element is not used and the output buffer segment is given by **buffer_seg**.

obufoff

Offset to the start of the communications I/O buffers. If the **DIFFXBUF** bit was not set in the **flag2** structure element, then this element is not used and the output buffer offset is given by **buffer_off+inbuf_len+1**.

inbuf_len

Input communication buffer length. Note that the actual allocated buffer must be 1 byte longer than this value.

outbuf_len

Output communication buffer length. Note that the actual allocated buffer must be 1 byte longer than this value.

Note that **buffer_seg:buffer_off** points to a buffer of length **inbuf_len+outbuf_len+2** if the **DIFFXBUF** bit in **flag2** is not set. Otherwise it points to a buffer of length **inbuf_len+1**.

aux_addr1

This value should be 0 unless a multiport card is being used or unless **cardtype = CARD_WINAPI**. If a multiport card is being used(and cardtype is not CARD_WINAPI), then the first serial port on the multiport card is 0. The second port is 1, and so on. If **cardtype = CARD_WINAPI**, then this value is used to specify which **COM** port is being used(0 for **COM1**, 1 for **COM2**, 2 for **COM3**, and so on).

cardtype

This corresponds to the type of serial card. This number tells **COMM-DRV** what device to use for the particular port.

CARD_NORMAL	Standard PC COM1, COM2, COM3, and COM4. Also used for most dumb cards on the market.
CARD_INTELH	Intel HUB 6 multiport non-intelligent card.
CARD_DIGCXI	Digiboard COMXi multiport cards.
CARD_CYCLOMY	Cyclades ISA & PCI Yx Cards.
CARD_BOCA1610	BOCA 1610
CARD_DIGPCX	Digiboard PCX
CARD_GTEK	GTEK
CARD_INT14	*3rd party INT14H driver.*
CARD_WCSCVXD	WCSC high speed VxD for 8250 family. Used within standard PC COM1, COM2, COM3, and COM4. Also used for most dumb cards on the market. This setting requires that the **COMM-DRV/VxD** product be installed.
CARD_WINAPI	Use Windows 3.x, Windows 95, or Window NT's built in driver. Note that when this type is selected, **aux_addr1** must be used to specify the **COM** port (e.g., **COM1 => pcb.aux_addr1 = 0, COM2 => pcb.aux_addr1** = 1, and so on).

cardseg

Segment address of the shared memory used by smart multiport cards. Additionally it has special meanings for dumb cards used with the **NORMAL** *sub-device(commdv00.asm).* For the **BOCA Dumb Multiport Port Cards**, the value should be **ffff**. For the **WCSC AST Compatible Four Port Multiport Card(PCCOM4 or LCS 8880)** the value should be **1bf** or **2bf** depending on whether the first port on the card is 1a0 or 2a0 respectively. For the **AST Four Port Multiport Cards** the value should be **f1bf** or **f2bf** depending on whether the first port on the card is 1a0 or 2a0 respectively. For the **Cyclades' CyclomY ISA** cards it is the segment address of the card(the CD1400). For the **Cyclades' CyclomY PCI** it is the slot number in which the card is located.

flag2

This offset must be initialized to the flags that further modify the behavior of **COMMDRV**.

CHAIN_INT	If this bit is set, then if the interrupt was not generated by this port, then the interrupt is chained to the interrupt handler installed before **COMM-DRV.**
DIFFXBUF	If this bit is set, then **COMM-DRV** will give separate segments for both transmit and receive buffers. In this mode **buffer_seg** and **buffer_off** must be initialized to the segment:offset of the receive buffer. **obufseg** and **obufoff** must be initialized to the segment:offset of the transmit buffer.
DISABLE_MODEMSIG	If this bit is set, then the DTR & RTS signals are disabled when the **ser_rs232_setup()** function is issued.
IGNORE_16550	If this bit is set, then the 16550 UART is not enabled if detected.
LEAVE_DTR	If this bit is set, then the DTR signal will be left at the state it is at when the **ser_rs232_cleanup()** function is issued.
LEAVE_RTS	If this bit is set, then the RTS signal will be left at the state it is at when the **ser_rs232_cleanup()** function is issued.
NOCHANGE_MODEMSIG	If this bit is set, then when the **ser_rs232_setup()** function is made the DTR/RTS signals are left in their pre-installed state.
STOREALL	If this bit is set, then for every byte that is received, there are four bytes written to the receive buffer. These bytes are the data, line status register, modem status register, and port the data is from in this order.
STORESOME	If this bit is set when ***ser_rs232_setup()*** is called, then **COMM-DRV** will store the data followed by the port number.
LOCKBAUD	If this bit is set when ***ser_rs232_setup()*** is called, baud rate commands will only have an effect when the port is first opened.

POLLONSTATUS	If this bit is set when *ser_rs232_setup()* is called, COMM-DRV will poll the specific hardware anytime the **ser_rs232_getstatus()** function is called.
MULTIDROP	Use the multidrop protocol when outputting data. When data is outputted with **PutPacket()**, the **RTS** signal is dropped after all data has left the UART. This commonly used with multidrop configurations with RS485 devices. This is only functional when **cardtype = CARD_WCSCVXD.**
NINEBITPROTOCOL	When data is outputted with the **PutPacket()** function, it forces the first byte transmitted to have the parity bit set. All characters sent thereafter are sent normally. This is commonly referred to as the nine bit protocol. This is only functional when **cardtype = CARD_WCSCVXD.**
CUSTOMFLAG1	If this bit is set and the Arnet Smartport is being used, then an initial reset of the Arnet device is ignored.
CUSTOMFLAG2	If this bit is set and the Digiboard PC/Xe is being used, then the board will operate in 64K mode.

pport

Port reception characteristics can be inherited. In other words, several ports can share the same receive buffers. To enable this each port is said to have a parent port. If the port has no parent port then **pport** must be set to the same port number (COMM-DRV initializes the structure this way) otherwise it must be set to its parent. If **pport** is set to some port other than the current port then all data received by this port will go to the parent port. The **STOREALL** flag in flag2 can be used to also buffer the port number so the originating port can be determined.

char_xoff

The character that will be interpreted as an XOFF character from the remote.

char_xon

The character that will be interpreted as an XON character from the remote.

char_xxoff

The character that will be transmitted as an XOFF character to the remote.

char_xxon

The character that will be transmitted as an XON character to the remote.

Appendix C (Returned Structure)

Following is a description of the values returned in the port_param structure when a **ser_rs232_getport** or similar function call is made. Note that the values that were stuffed in those elements described in **APPENDIX B** remain unchanged and as such will not be further discussed here.

error

RS232ERR_NONE	0	No error.
RS232ERR_BUFFER	1	Buffer not set or an attempt to change the buffer while the port is active was made.
RS232ERR_ACTIVE	2	Port not active.
RS232ERR_XMTBUF	3	Transmit buffer full.
RS232ERR_RCVBUF	4	Receive buffer empty.
RS232ERR_SYNTAX	5	Syntax error.
RS232ERR_BUFSIZ	6	Invalid buffer size.
RS232ERR_PORT	7	Invalid port.
RS232ERR_HANDLR	8	Handler changed.
RS232ERR_BAUD	9	Invalid baud rate.
RS232ERR_PARITY	10	Invalid parity.
RS232ERR_LNGTH	11	Invalid data length.
RS232ERR_STOPBIT	12	Invalid # stopbits.
RS232ERR_PROTOCOL	13	Invalid protocol.
RS232ERR_IRQCHANGED	14	IRQ changed.
RS232ERR_PORTCHANGED	15	Port changed.
RS232ERR_THRESHOLD	16	Invalid threshold.
RS232ERR_INVIRQ	17	Invalid IRQ.
RS232ERR_INTOFF	18	Interrupts not enabled.
RS232ERR_BREAK	19	Invalid break syntax.
RS232ERR_FATAL	20	Fatal error.
RS232ERR_DSR	21	CTS error.
RS232ERR_INVADR	22	Invalid RS232 address. This usually means that a serial chip is not at the specified address.
RS232ERR_ENVIRON	23	Environment variable not set.
RS232ERR_IOCTL	24	Error issuing IOCTL call .
RS232ERR_ATEXIT	25	Error issuing atexit cleanup .
RS232ERR_DEVINIT	26	Error mapping for direct calls.
RS232ERR_DOSOPEN	27	Error opening Device .
RS232ERR_MALLOC	28	Error allocating memory .

RS232ERR_EXTMICRO	29	Error on external micro card.
RS232ERR_CARDCHANGED	30	Card changed error .
RS232ERR_CARDTYPE	31	Card type error .
RS232ERR_NOSUPPORT	32	Not supported .
RS232ERR_CMDBUFFULL	33	Card command buffer full.
RS232ERR_PPORT	34	Parent port error.
RS232ERR_NODEVICE	35	No subdevice for this port.
RS232ERR_UNKNOWN	36	Unknown error
RS232ERR_BUSY	37	External card busy
RS232ERR_NOTIMER	38	No more timers available
RS232ERR_INT14VEC	39	INT14 vector was changed
RS232ERR_INT08VEC	40	INT08 vector was changed
RS232ERR_DPMI	41	DPMI error
RS232ERR_WINBUF	42	TSR buffer too small or non-existent
RS232ERR_NOASYNCRES	43	No asynchronous resources left.
RS232ERR_NOTIMERRES	44	No timer resources left
RS232ERR_NOOTHERES	45	No "other" timer resources left
RS232ERR_FILEIO	46	File I/O error.
RS232ERR_HMEMG64	47	Hardware memory exceeded 64K.
RS232ERR_MAPVXD	48	VxD not loaded.

flag

ACTIVE_FLG	Set=Port is active.
XMTOFF_STATE	Is set when XOFF received, DSR reset, or CTS reset. Is reset when XON received, DSR set, or CTS set.
RCVOFF_STATE	Is set when XOFF is set, DTR reset, or RTS reset. Is reset when XON sent, DTR set, or RTS set.
INTFUNC_MODEM	Set=**COMM-DRV** will call user interrupt function on MSR interrupt.
INTFUNC_STAT	Set=**COMM-DRV** will call user interrupt function on LSR interrupt.
INTFUNC_RCV	Set=**COMM-DRV** will call user interrupt function on RCV interrupt.
INTFUNC_XMIT	Set=**COMM-DRV** will call user interrupt function on XMT interrupt.
INTFUNC_RCVPKT	Set=**COMM-DRV** will call user interrupt function on framed packet reception.
INTFUNC_RCVCHR	Set=**COMM-DRV** will call user interrupt function on reception of particular character.
INTFUNC_IBFDCNT	Set=**COMM-DRV** will call user interrupt function when input buffer decrements to particular count.
INTFUNC_IBFICNT	Set=**COMM-DRV** will call user interrupt function when input buffer increments to particular count.
INTFUNC_OBFDCNT	Set=**COMM-DRV** will call user interrupt function when output buffer decrements to particular count.

INTFUNC_OBFICNT	Set=**COMM-DRV** will call user interrupt function when output buffer increments to particular count.
INTFUNC_XMTCHR	Set=**COMM-DRV** will call user interrupt function when a particular character is about to be transmitted.
INTFUNC_OTHER	Set=**COMM-DRV** will call user interrupt function when a timed asynchronous event or other event occurs.

inbuf_count

Number of bytes currently in the input communication buffer.

outbuf_count

Number of bytes currently in the output communication buffer.

lost_ichar

Number of characters that were lost on input. Note that this value increases both when the input buffer is full and a new byte comes in, or when the input buffer is flushed.

lost_ochar

Number of bytes that did not make it into the output buffer when any of the write character or packet routines are called. Note this value also increases when the output buffer is flushed.

total_ichar

Total number of bytes that was inputted. This is the sum of lost characters plus the read characters plus those in the input buffer.

total_ochar

Total number of bytes that were outputted are attempted. This is the sum of the bytes in the output buffer, plus lost outputted bytes, plus bytes actually outputted.

msr_reg

Current Modem Status Register.

lsr_reg

Current Line Status Register.

auxpcb

This the a pointer to the Auxiliary Port Control Block(**struct aux_pparam**). See the **comm.h** file for the description of the contents of the structure.

irqp

This is the pointer to the Interrupt Control Block(**struct irq_param**) for this port. See the **comm.h** file for the description of the contents of the structure.

opcb

This is the **pointer** to the actual Port Control Block(**struct port_param**) for this port. Remember that when the **ser_rs232_getport()** call is issued only a copy of the port is received. This address gives one the ability to query items directly from the actual structure without issuing slower **ser_rs232_getport()** calls.

The above elements are read only. Do not modify under any circumstance.

Appendix D (Example Application)

```
/*************************************************************/
/*                  ------ Example.c -----                   */
/*                                                           */
/*Description-                                               */
/*    Example program that illustrates the usage of the      */
/*    COMM-DRV/Lib libraries.                                */
/*************************************************************/


// If windows.h was included assume compilation for
// Windows(QuickWin or similar
#ifdef    _INC_WINDOWS
#define   MSWIN
#define   MSWINDLL
#endif

#include  <comm.h>

#ifdef    MSWIN
#define   COMADR      0x0
#define   COMIRQ      0x0
#define   CARDTYPE    CARD_WINAPI
#define   SUBPORT     1                 //0=COM1, 1=COM2, etc
#else
#define   COMADR      0x3f8
#define   COMIRQ      0x4
#define   CARDTYP     CARD_NORMAL
#define   SUBPORT     0
#endif

main()
{
int       stat;
int       port=0;
char      ch[80];
```



```
// The first thing to be done is to initialize the port
if ((stat = InitializePort(port,SUBPORT,COMADR,COMIRQ,
CARDTYPE,0,2048,1024,0)) != RS232ERR_NONE)
      {
      printf("Error #%d initializing serial port\n",stat);
      exit(1);
      }

// Now set the port characteristics
if ((stat = SetPortCharacteristics(port,BAUD9600,PAR_EVEN,
LENGTH_8,STOPBIT_1,PROT_NONNON)) != RS232ERR_NONE)
      {
      printf("Error #%d setting characteristics\n",stat);
      exit(1);
      }

// At this point data may be sent and received from
// the serial port.

// Output a string
PutString(port,"This is a test\r\n");

// Get a string
if (GetString(port,sizeof(ch),ch) > 0)
      printf("Returned string=>%s\n",ch);

// Close the serial port.
UnInitializePort(port);

exit(0);
}
```

Index

COMM-DRV/Dos™

Universal Serial Communication Driver For MS-DOS

COMM-DRV/DOS is a robust and reliable serial communication TSR, MS-DOS device driver, and supporting utilities for MS-DOS®, Windows®, and Desqview®. It will not interfere with other serial drivers, DLLs, or TSRs that follow standard Windows and MS-DOS conventions.

COMM-DRV/DOS has several interfaces. It is the FOSSIL DRIVER of choice for many BBS operators. In fact it is licensed to several BBS developers. It also provides the most common file transfer protocols and programs that allow sending and receiving files given a filename, filename with filepath, or filepath with wild cards.

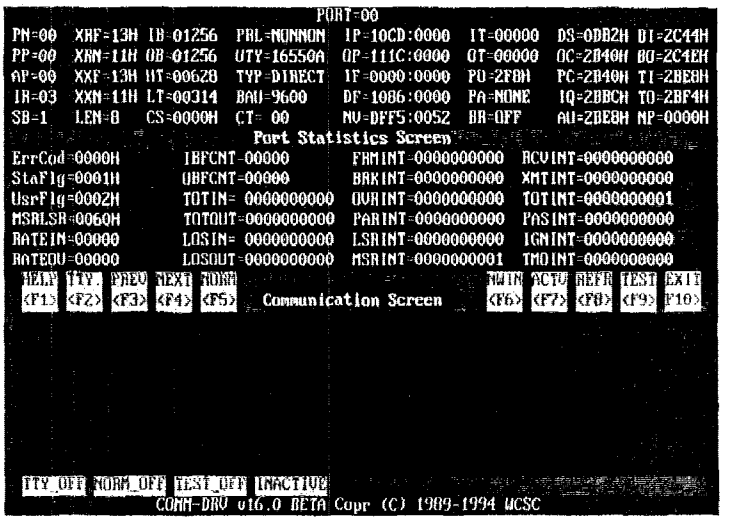
COMM-DRV/DOS is compatible with packages like PROCOMM Plus Network Version, PROCOMM®/Windows, PCBoard®, PC Anywhere, and all communication packages using the standard INT14H. It will run perfectly under multitaskers like Desqview, Omniview, Taskview, and Windows.

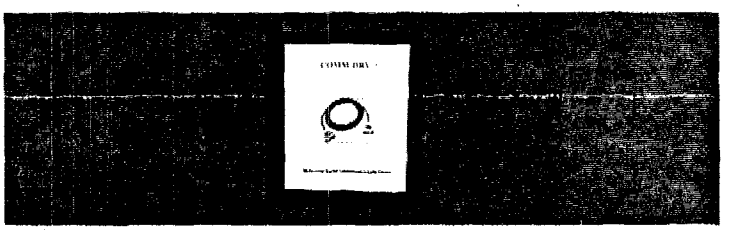
COMM-DRV/DOS is comprised of several independent and interdependent components. These components may be used separately or combined, depending on the application. The major components comprising this product are as follows.

- A true MS-DOS device driver that allows addressing the serial port as a standard file. This driver allows the user to send and receive data from the serial port exactly as if he/her were reading and writing from a standard disk file. This interface is compatible with any database package that can read and write a standard DOS file. Moreover, this interface is compatible with both MS-DOS and Windows. One could even transfer a file between two computers as simply as typing "copy file.txt COM1" on the sending computer and typing "copy COM1 file.txt" on the receiving computer without getting the dreaded "Abort, Fail, Retry" message.
- A dynamically configured communication TSR. This one TSR gives the application programmer several APIs that may be used concurrently to perform serial I/O. This TSR contains the FOSSIL interface, the standard INT14H(BIOS) interface with extensions, and the Direct Address Mapping(DAM) Interface, our version of dynamic linking under MS-DOS. In addition it supports the INT21H(MS-DOS) interface with large set of IOCTL calls. In general one or more of these interfaces are used by off-the-shelf applications to allow them to use non-standard multiport and other serial communications cards.
- A serial communication monitor that aids in trouble shooting serial communication problems.
- A TSR that redirects the incoming serial data to the keyboard buffer, allowing applications to get keyboard data from the serial port transparently.
- A custom file transfer utility that allows a computer to transfer files to and from a remote computer without remote user intervention.
- A spawnable XMODEM/YMODEM/ZMODEM file transfer program.
- A standalone dumb terminal emulator that can display communication on several ports concurrently.
- A very user friendly installation program that builds the necessary configuration files for the user's choice of multiport cards. The install program also has extensive context sensitive help.

Supported Hardware

- PC COM1-COM4
- Arnet(Smartport, Smartport Plus, Ports Plus)
- Digiboard(COM/Xi, PC/Xi, PC/Xe)
- Boca Research(BB1610, BB8002)
- GTEK(PCSS-8FA, PCSS-8FX)
- All non-intelligent multiport cards
- Any card with an INT14H driver.





Technical Specifications

- Supports baud rates up to 115,200 baud. This baud rate is sustainable on several ports concurrently with the 8250/16450/16550 family of UARTs.
- Support standard and non-standard baud rates to 115,200 baud
- Peek ahead virtually any number of bytes.
- Supports transparent XON/XOFF & hardware (RTS/CRS, DTR/DSR) flow control.
- Supports virtually any number of serial ports running at the same time. Interrupts, buffering, and flow control handled in 32 bit mode of the CPU in Ring 0.
- Supports COM1-COM4 and virtually any manufacturer's 8250/16450/16550 based multiport cards. Different vendor's multiport cards can be in the same machine and running at the same time.
- Both Windows applications and MS-DOS applications running in DOS boxes have access to the high speed VxD at the same time.
- Transmit & receive buffers are adjustable up to 128K per port.
- Autodetects 16550 and uses it in high speed mode.
- Allows sharing IRQs on supporting hardware.

WCSC, Willies Computer Software Co.
6215 Longflower Lane, Kingwood, TX 77345
Tel: (281)360-4232 **Fax:** (281)360-3231
Internet: **http://www.wcscnet.com**

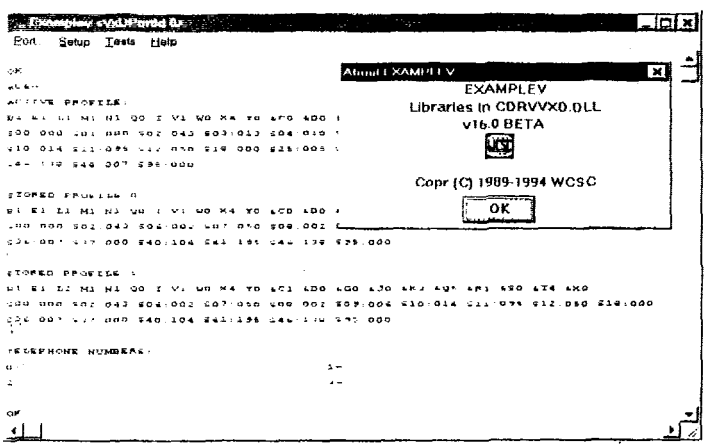
COMM-DRV/VxD™

Ultra High Speed VxD for 8250/16450/16550/16650 UARTs under Windows 3.x & Windows 95.

COMM-DRV/VxD is the state of the art serial communications Virtual Device Driver (VxD) for Windows and MS-DOS applications running under Windows. COMM-DRV/VxD extracts the last ounce of performance from the aging 8250, & 16450 UARTS as well as lightning fast performance from the 16550 & 16650 family of UARTs. If your application requires ultra fast serial communications, COMM-DRV/VxD is the tool of choice for most serious developers.

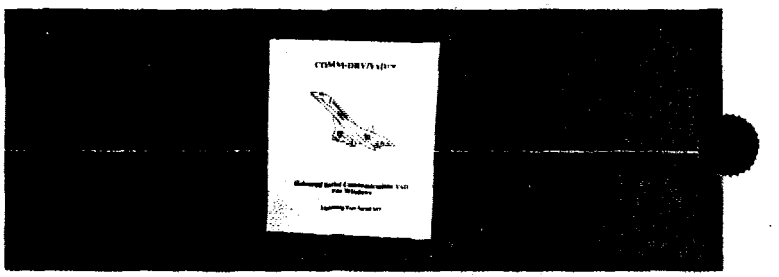
COMM-DRV/VxD was designed with interoperability in mind. It uses the Windows standard DLL calling conventions. As such, **COMM-DRV/VxD** may be integrated with any Windows tool, application, or language that can call the Windows API (application programming interface). **COMM-DRV/VxD** may be used with Visual C/C++, Borland C/C++, Watcom C/C++, Visual Basic, Access, Excel, Paradox, and a myriad of other tools and applications.

COMM-DRV/VxD is extremely easy to use. The API is simple, concise, and intuitive. Ample examples are provided. **COMM-DRV/VxD** may be integrated with all other WCSC communication products.



Application Programming Interface(API)

WCSCVxDBytesInRcvBuf Gets number of bytes in receive buffer.
WCSCVxDBytesInXmtBuf Gets number of bytes in transmit buffer.
WCSCVxDCleanup Closes an opened port.
WCSCVxDDtrOff Turn DTR off.
WCSCVxDDtrOn Turn DTR on.
WCSCVxDFlush Purges communication buffers.
WCSCVxDGetPacket Get a packet of data.
WCSCVxDGetPort Get port metrics.
WCSCVxDGetRegister Read an 8250/16450/16550 register value.
WCSCVxDInit Map dynamically to the VxD.
WCSCVxDInitializePort Open a port.
WCSCVxDPutPacket Output a packet of data.
WCSCVxDPutRegister Write to an 8250/16450/16550 register.
WCSCVxDRtsOff Turn RTS off.
WCSCVxDRtsOn Turn RTS on.
WCSCVxDSetBaudDiv Set new baud rate divisor.
WCSCVxDSetPortCharacteristics Set baud rates, etc.
WCSCVxDSetup Extended port open.
WCSCVxDSizeOfRcvBuf Get size of receive buffer.
WCSCVxdSizeOfXmtBuf Get size of transmit buffer.
WCSCVxDSpaceInRcvBuf Get available space in receive buffer.
WCSCVxdSpaceInXmtBuf Get available space in transmit buffer.
WCSCVxDViewPacket Non-destructively reads a packet of data.



Technical Specifications

- Supports baud rates up to 115,200 baud. This baud rate is sustainable on several ports concurrently with the 8250/16450/16550 family of UARTs.
- Support standard and non-standard baud rates to 115,200 baud
- Peek ahead virtually any number of bytes.
- Supports transparent XON/XOFF & hardware (RTS/CRS, DTR/DSR) flow control.
- Supports virtually any number of serial ports running at the same time.
- Interrupts, buffering, and flow control handled in 32 bit mode of the CPU in Ring 0.
- Supports COM1-COM4 and virtually any manufacturer's 8250/16450/16550 based multiport cards. Different vendor's multiport cards can be in the same machine and running at the same time.
- Both Windows applications and MS-DOS applications running in DOS boxes have access to the high speed VxD at the same time.
- Transmit & receive buffers are adjustable up to 128K per port.
- Autodetects 16550 and uses it in high speed mode.
- Allows sharing IRQs on supporting hardware.

WCSC (Willies Computer Software Co)
6215 Longflower Lane, Kingwood, TX 77345
Tel: (281)360-4232 Fax: (281)360-3231
Internet: http://www.wcsc.net

7.2.3 Source Code for COMM.H

The COMM.H file is the header file providing defines and links to the COMM-DRV/LIB software.

The following items, which were explicitly referenced in the DATA_ATT.C file, have been extracted from the COMM.H file. These items (in this greatly reduced subset) are COMM-DRV functions or defines. This list of explicitly referenced items may not be complete. In addition, COMM.H may contain items necessary for the COMM-DRV functions, which the are indirectly referenced through the function's separate source code. However, this list serves to illustrate the portion of the software package that was used.

Include file

```
COMM.H                          Header file for COMM-DRV/LIB
```

Function prototypes found in COMM.H

```
bio_scrollup                    Entry into BIOVIDEO.ASM
bio_setcursorxy                 Entry into BIOVIDEO.ASM
GetString                       High level COMM-DRV function prototype
UnInitializePort                High level COMM-DRV function prototype
InitializePort                  High level COMM-DRV function prototype
SetPortCharacteristics          High level COMM-DRV function prototype
CdrvSetTimerResolution          High level COMM-DRV function prototype
SetTimeout                      High level COMM-DRV function prototype
ser_rs232_misc_func             High level COMM-DRV function prototype
ser_rs232_set_intfunc           High level COMM-DRV function prototype
```

Defines found in COMM.H

```
RS232ERR_NONE                   COMM-DRV define = 0  // RS232 no error
CARD_NORMAL                     COMM-DRV define = 0  // Normal cards
BAUD9600                        COMM-DRV define = 7  // 9600 baud
PARITY_NONE                     COMM-DRV define = 0  // No parity
LENGTH_8                        COMM-DRV define = 3  // 8 bits
STOPBIT_1                       COMM-DRV define = 0  // 1 Stop bit
PROT_NONNON                     COMM-DRV define = 4  // Protocol Index
                                  // Local no protocol - Remote no protocol
INTFUNC_RCVCHR                  COMM-DRV define = 0x0020
                                  // Set = Exec receive interrupt function
CDECL                           COMM-DRV Definition = cdecl
COM1_interrupt_func             Port Control Block structure (PCB)
COM2_interrupt_func             Port Control Block structure (PCB)
COM3_interrupt_func             Port Control Block structure (PCB)
COM4_interrupt_func             Port Control Block structure (PCB)
COM1_IO                         Port Control Block structure (PCB)
COM2_IO                         Port Control Block structure (PCB)
COM3_IO                         Port Control Block structure (PCB)
COM4_IO                         Port Control Block structure (PCB)
```

The following is a listing of the COMM.H file, which will serve as an example of the source code documentation included in the COMM-DRV/LIB package.

```
/*****************************************************************************/
/*                                ----- comm.h -----                          */
/*                                                                           */
/*              Willies' Computer Software Company                           */
/*              Copyright (C) 1989-1992 Egberto Willies, 1989                */
/*              All rights reserved.                                         */
/*                                                                           */
/*Author                                                                     */
/*      Egberto Willies 06-FEB-1989                                          */
/*                                                                           */
/*Description-                                                               */
/*      Include for communicatons package.                                   */
/*****************************************************************************/
/* $Header: o:/projects/commdrv/work/inc/rcs/comm.h 1.9 1996/12/14 18:31:01
ewillies Exp ewillies $ */
/*****************************************************************************/
#ifndef FILE_COMM_H
#define FILE_COMM_H             1
/*****************************************************************************/

/*****************************************************************************/
/*===== Compatibility Equates                                                */
/*****************************************************************************/
#ifndef MSNTDRV
#ifdef MSWIN
#ifndef _INC_WINDOWS
#include <windows.h>
#endif
#endif
#else
#define HANDLE  unsigned int
#define DWORD   unsigned long
#define WINAPI
#define HWND    unsigned int
#define WPARAM  unsigned int
#define LPARAM  unsigned long
#endif

#ifdef __cplusplus
extern "C" {            /* Assume C declarations for C++ */
#endif  /* __cplusplus */

#ifdef  MSWIN32
#ifndef CDECL
#define CDECL   cdecl
#endif

#ifndef FAR
#define FAR
#endif

#ifndef FARC
#define FARC    CDECL
#endif
```

```
#ifndef FARPAS
#define FARPAS  WINAPI
#endif

#ifndef CDRVUINT
#define CDRVUINT        unsigned int
#endif

#ifndef CDRVIFNC
#define CDRVIFNC        unsigned long
#endif

#define FP_SEG(s)       ((CDRVUINT)s)
#define FP_OFF(s)       ((CDRVUINT)s)

#else

#ifndef CDECL
#define CDECL   cdecl
#endif

#ifndef FAR
#define FAR     far
#endif

#ifndef FARC
#define FARC    FAR CDECL
#endif

#ifndef FARPAS
#define FARPAS  FAR PASCAL
#endif

#ifndef CDRVUINT
#define CDRVUINT        unsigned short
#endif

#ifndef CDRVIFNC
#define CDRVIFNC        short
#endif

#endif

#ifndef DOSDEF_H
#include <dosdef.h>
#endif

/*************************************************************************/
/*======Version information                                              */
/*************************************************************************/
#define COMMDRV_COMPANY         "Willies Computer Software Co.(WCSC)"
#define COMMDRV_VERSION         "COMM-DRV "
#define COMMDRV_VERSION1        "v17.0a "
```

```
#define COMMDRV_VERSION2        "Copr (C) 1989-1999 WCSC"
#define COMMDRV_VERSION3        17,0,0,0
#define COMMDRV_VERSION_MAJOR   17
#define COMMDRV_VERSION_MINOR   0

/*************************************************************************/
/*=======General Symbols                                                 */
/*************************************************************************/
#define CDRVPATHLEN             64      /* COMMDRV path array length */

#define CDRV_ON         1
#define CDRV_OFF        0
#define CDRV_TRUE       1
#define CDRV_FALSE      0

/*************************************************************************/
/*=======These values may be changed to reduce the amount of space used  */
/*       for port/irq/other resources. If any of these items change,     */
/*       then the corresponding item in COMM.INC must be changed. The    */
/*       libraries must then be reassembled and recompiled.              */
/*************************************************************************/
#define MAX_IRQ         16      /*Maximum IRQ #*/
#define MAX_IRQS        17      /*Max # of IRQs active concurrently*/
#define MAX_PORT        31      /*Maximum port #*/
#define MAX_PORTS       32      /*Max # of ports active concurrently*/
#define MAX_DRIVERS     10      /*Max # of sub-device drivers*/
#define MAX_TIMERS      32      /*Max # of Timer resources*/
#define MAX_ASYNCS      32      /*Max # of async callback resources*/

/*************************************************************************/
/*======Argument limits                                                  */
/*************************************************************************/
#define MAX_BAUD        21      /*Max baud*/
#define MAX_BREAK       1       /*Maximum break*/
#define MAX_LENGTH      3       /*Max char length*/
#define MAX_PARITY      4       /*Max parity*/
#define MAX_PROTOCOL    15      /*Maximum protocol*/
#define MAX_STOPBIT     1       /*Maximum stop bit*/

/*************************************************************************/
/*=====Defaults                                                          */
/*************************************************************************/
#define DEF_AUXADDR     0               /*Default auxiliary address*/
#define DEF_BASE        0x3f8           /*Default port*/
#define DEF_BAUD        BAUD9600        /*Default baud rate (9600 baud)*/
#define DEF_BLOCK       0x0             /*Default xmit/recv block time*/
#define DEF_BREAK       BREAK_OFF       /*Default break(OFF)*/
#define DEF_CARD        CARD_NORMAL     /*Default card type */
#define DEF_CSEG        0               /*Default card segment*/
#define DEF_INLEN       1024            /*Input buffer length*/
#define DEF_IRQ         4               /*Default IRQ*/
#define DEF_ITIMEOUT    0               /*Default input timeout*/
#define DEF_LENGTH      LENGTH_8        /*Default word length(8 bits)*/
#define DEF_OTIMEOUT    0               /*Default output timeout*/
```

```
#define DEF_OUTLEN      256             /*Output buffer length*/
#define DEF_PARITY      PAR_NONE        /*Default parity setting (NONE)*/
#define DEF_PORT        0               /*Default port number*/
#define DEF_PORTTYPE    0               /*Default port type(Normal)*/
#define DEF_PROTOCOL    PROT_NONNON     /*Default protocol*/
#define DEF_STOPBIT     STOPBIT_1       /*Default stopbit (1 stop bit)*/
#define DEF_THHIGH      1               /*Default High Threshold*/
#define DEF_THLOW       0               /*Default low threshold*/

/*************************************************************************/
/*=====Flow control characters                                           */
/*************************************************************************/
#define XOFF            0x13    /*ASCII code for XOFF */
#define XON             0x11    /*ASCII code for XON  */

/*************************************************************************/
/*=====INT14 Bits and flags                                              */
/*************************************************************************/
#define SER_INT14_ERROR_BIT     0x80    /* Error bit */
#define SER_INT14_ACTIVE        0x1     /* Port can be initted via INT14 */
#define SER_INT14_FOSSIL        0x2     /* Fossil driver flag */
#define SER_INT14_INTERRUPT     0x14    /* Interrupt */

/*************************************************************************/
/*=====8250 Asynchronous adapter offsets and bits                        */
/*************************************************************************/
#define BAUD_LOW        0       /*Offset to low byte of baud divisor*/
#define DATABUF         0       /*Offset to rcv/xmt buffer*/

#define BAUD_HIGH       1       /*Offset to high byte of baud divisor*/
#define IER             1       /*Interrupt Enable Register*/
#define IER_RDA         0x001   /*Receive Data Available int bit*/
#define IER_THRE        0x002   /*Transmitter Hold Reg. Empty int bit*/
#define IER_RLS         0x004   /*Receive Line Status int bit*/
#define IER_MS          0x008   /*Modem Status int bit*/

#define IIR             2       /*Interrupt Identification Register*/
#define IIR_RLS         0x6     /**equal* if Receiver Line Stat int*/
#define IIR_RDA         0x4     /**equal* to if character ready*/
#define IIR_THRE        0x2     /**equal* to if TX buffer empty*/
#define IIR_PEND        0x1     /* zero if any interrupt pending*/
#define IIR_MS          0x0     /**equal* to if Modem Status int*/
#define IIR_FIFOL       0x40    /*FIFO ACTIVE*/
#define IIR_FIFOH       0x80    /*FIFO ACTIVE*/

#define FCR             2       /*FIFO Control Register*/
#define FCR_ENAB        0x01    /*FIFO Enable*/
#define FCR_RRESET      0x02    /*Receiver FIFO Reset*/
#define FCR_XRESET      0x04    /*Xmitter FIFO Reset*/
#define FCR_DMASEL      0x08    /*DMA Select*/
#define FCR_RTRGLSB     0x40    /*Receiver Trigger LSB*/
#define FCR_RTRGMSB     0x80    /*Receiver Trigger MSB*/

#define LCR             3       /*Line Control Register*/
```

```
#define LCR_WLS0        0       /*Word Length Select Bit 0*/
#define LCR_WLS1        0x1     /*Word Length Select Bit 1*/
#define LCR_STOPBITS1   0x4     /*Number of stop bits*/
#define LCR_PARITYEN    0x8     /*Enab Parity(see SPARITY & EPARITY)*/
#define LCR_EPARITY     0x10    /*Even Parity Bit*/
#define LCR_SPARITY     0x20    /*Stick Parity*/
#define LCR_BREAK       0x40    /*set if break desired*/
#define LCR_DLAB        0x80    /*Divisor Latch Access Bit*/

#define MCR             4       /*Modem Control Register*/
#define MCR_DTR         0x1     /*Data Terminal Ready*/
#define MCR_RTS         0x2     /*Request To Send*/
#define MCR_OUT1        0x4     /*Output 1 (unused)*/
#define MCR_OUT2        0x8     /*Output 2 (External Int Enable)*/
#define MCR_LOOP        0x10    /*Loopback enable*/

#define LSR             5       /*Line Status Register*/
#define LSR_DATA        0x1     /*Data Ready bit*/
#define LSR_OVERRUN     0x2     /*Overrun error bit*/
#define LSR_PARITY      0x4     /*Parity error bit*/
#define LSR_FRAMING     0x8     /*Framing error bit*/
#define LSR_BREAK       0x10    /*Break Detect*/
#define LSR_THRE        0x20    /*Transmit Holding Register Empty*/
#define LSR_TSRE        0x40    /*Transmit Shift Register Empty*/

#define MSR             6       /*Modem Status Register*/
#define MSR_DEL_CTS     0x1     /*Delta Clear To Send*/
#define MSR_DEL_DSR     0x2     /*Delta Data Set Ready*/
#define MSR_EDGE_RI     0x4     /*Trailing Edge of Ring Indicator*/
#define MSR_DEL_SIGD    0x8     /*Delta Receive Line Signal Detect*/
#define MSR_CTS         0x10    /*Clear To Send*/
#define MSR_DSR         0x20    /*Data Set Ready*/
#define MSR_RI          0x40    /*Ring Indicator - during entire ring*/
#define MSR_DCD         0x80    /*Data Carrier Detect - on line*/

#define SCR             7       /*Scratch Register*/

/*************************************************************************/
/*=====8259 registers Addresses                                          */
/*************************************************************************/
#define PORT_8259_REG1  0x20    /*8259 reg1*/
#define PORT_8259_REG2  0x21    /*8259 reg2*/
#define EOI_8259        0x20    /*End of int instruction to 8259*/

/*************************************************************************/
/*=====Baud Rate Indeces                                                 */
/*************************************************************************/
#define BAUD110         0       /* 110 baud */
#define BAUD150         1       /* 150 baud */
#define BAUD300         2       /* 300 baud */
#define BAUD600         3       /* 600 baud */
#define BAUD1200        4       /* 1200 baud */
#define BAUD2400        5       /* 2400 baud */
#define BAUD4800        6       /* 4800 baud */
```

```
#define BAUD9600        7       /* 9600 baud */
#define BAUD19200       8       /* 19200 baud */
#define BAUD38400       9       /* 38400 baud */
#define BAUD57600       10      /* 57600 baud */
#define BAUD115200      11      /* 115.2 kbaud */
#define BAUDUSER00      12      /* User baud rate */
#define BAUD14400       12      /* 14400 baud */
#define BAUDUSER01      13      /* User baud */
#define BAUD28800       13      /* 28800 baud */
#define BAUDUSER02      14      /* User baud rate */
#define BAUDUSER03      15      /* User baud rate */
#define BAUDUSER04      16      /* User baud rate */
#define BAUDUSER05      17      /* User baud rate */
#define BAUDUSER06      18      /* User baud rate */
#define BAUDUSER07      19      /* User baud rate */
#define BAUDUSER08      20      /* User baud rate */
#define BAUDUSER09      21      /* User baud rate */

/*************************************************************************/
/*=====Parity Indeces                                                    */
/*************************************************************************/
#define PAR_NONE        0       /* No parity */
#define PAR_ODD         1       /* ODD parity */
#define PAR_EVEN        2       /* Even parity */
#define PAR_SODD        3       /* Sticky ODD parity */
#define PAR_SEVEN       4       /* Sticky Even parity */

#ifndef MSWIN32
#define PARITY_NONE     0       /* No parity */
#define PARITY_ODD      1       /* ODD parity */
#define PARITY_EVEN     2       /* Even parity */
#define PARITY_SODD     3       /* Sticky ODD parity */
#define PARITY_SEVEN    4       /* Sticky Even parity */
#endif

/*************************************************************************/
/*=====Length Indeces                                                    */
/*************************************************************************/
#define LENGTH_5        0       /* 5 bits */
#define LENGTH_6        1       /* 6 bits */
#define LENGTH_7        2       /* 7 bits */
#define LENGTH_8        3       /* 8 bits */

/*************************************************************************/
/*=====Stopbit Indeces                                                   */
/*************************************************************************/
#define STOPBIT_1       0       /* 1 Stop bit */
#define STOPBIT_2       1       /* 2 Stop bit */

/*************************************************************************/
/*=====Break Indeces                                                     */
/*************************************************************************/
#define BREAK_OFF       0       /* Break off */
#define BREAK_ON        1       /* Break on */
```

```
/*************************************************************************/
/*=====Protocol Indeces                                                  */
/*************************************************************************/
#define PROT_RTSRTS      0       /*Local CTS/RTS-Remote CTS/RTS */
#define PROT_RTSXON      1       /*Local CTS/RTS-Remote XON/XOFF*/
#define PROT_RTSDTR      2       /*Local CTS/RTS-Remote DSR/DTR*/
#define PROT_RTSNON      3       /*Local CTS/RTS-Remote no prot*/
#define PROT_NONNON      4       /*Local no prot-Remote no prot*/
#define PROT_NONXON      5       /*Local no  protocol-Remote XOFF/XON*/
#define PROT_XONNON      6       /*Local XOFF/XON-Remote no prot*/
#define PROT_XONXON      7       /*Local XON/XOFF-Remote XON/XOFF*/
#define PROT_DTRNON      8       /*Local  DSR/DTR-Remote no protocol*/
#define PROT_DTRRTS      9       /*Local  DSR/DTR-Remote CTS/RTS*/
#define PROT_DTRDTR      10      /*Local  DSR/DTR-Remote DSR/DTR */
#define PROT_DTRXON      11      /*Local  DSR/DTR-Remote DSR/DTR*/
#define PROT_NONRTS      12      /*Local no  prot-Remote CTS/RTS prot*/
#define PROT_NONDTR      13      /*Local no  prot-Remote DSR/DTR prot*/
#define PROT_XONRTS      14      /*Local XOFF/XON- Remote CTS/RTS*/
#define PROT_XONDTR      15      /*Local XOFF/XON-Remote DSR/DTR*/

/*************************************************************************/
/*=====Cardtype Indeces                                                  */
/*************************************************************************/
#define CARD_NORMAL      0       /* Normal cards */
#define CARD_INTELH      1       /* Intel HUB card(No Longer supported) */
#define CARD_CYCLOMY     1       /* Cyclades ISA & PCI Yx Cards */
#define CARD_DIGCXI      2       /* Digiboard COMXI */
#define CARD_ARNETSPLUS  3       /* Arnet Smartplus */
#define CARD_BOCA1610    4       /* BOCA 1610  */
#define CARD_DIGPCX      5       /* Digiboard PCX  */
#define CARD_GTEK        6       /* GTEK board  */
#define CARD_INT14       7       /* 3rd party INT14H board  */
#define CARD_WCSCVXD     8       /* WCSC high speed VxD for 8250 family */
#define CARD_WINAPI      9       /* Uses Windows API */

/*************************************************************************/
/*=====Flag bit settings(System Flag(pcb.flag))                          */
/*************************************************************************/
#define ACTIVE_FLG       0x0001  /* Set = port is active. */
#define XMTOFF_STATE     0x0002  /* Set = Remote host in XOFF state. */
#define RCVOFF_STATE .   0x0004  /* Set = Local host in XOFF state. */
#define INTFUNC_MODEM    0x0008  /* Set = Exec modem intrpt func */
#define INTFUNC_STAT     0x0010  /* Set = Exec status intrpt func */
#define INTFUNC_RCV      0x0020  /* Set = Exec receive intrpt func */
#define INTFUNC_XMIT     0x0040  /* Set = Exec xmit interpt func */
#define INTFUNC_AVAIL    0x0080  /* Set =  */
#define INTFUNC_RCVCHR   0x0100  /* Set = Exec specific char rcv intrpt func */
#define INTFUNC_IBFDCNT  0x0200  /* Set = Exec inbuf dec to val intrpt func */
#define INTFUNC_IBFICNT  0x0400  /* Set = Exec inbuf inc to val intrpt func */
#define INTFUNC_OBFDCNT  0x0800  /* Set = Exec outbuf dec to val intrpt func */
#define INTFUNC_OBFICNT  0x1000  /* Set = Exec outbuf inc to val intrpt func */
#define INTFUNC_XMTCHR   0x2000  /* Set = Exec if about to xmit chr intrpt fnc
*/
```

```
#define INTFUNC_X        0x4000  /* Set = Use extended masks  */
#define INTFUNC_OTHER    0x8000  /* Set = Exec by other conditions */
#define INTFUNC_X_BREAK  0x0001  /* Set = Exec on Break intrpt func */
#define INTFUNC_X_FRAME  0x0002  /* Set = Exec on Frame intrpt func */
#define INTFUNC_X_OVRUN  0x0004  /* Set = Exec on Parity intrpt func */
#define INTFUNC_X_PARER  0x0008  /* Set = Exec on Frame intrpt func */
#define INTFUNC_X_CTS    0x0010  /* Set = Exec on CTS intrpt func */
#define INTFUNC_X_DSR    0x0020  /* Set = Exec on DSR intrpt func */
#define INTFUNC_X_CD     0x0040  /* Set = Exec on CD intrpt func */
#define INTFUNC_X_RING   0x0080  /* Set = Exec on RING intrpt func */

/*************************************************************************/
/*=====Flag2 bit settings(User flag(pcb.flag2))                          */
/*************************************************************************/
#define STOREALL                 0x0001  /*Set = Store LSR/MSR/PORT/DATA*/
#define DIFFXBUF                 0x0002  /*Set = Different seg for xbuf */
#define CHAIN_INT                0x0004  /*Set = Chain interrupts */
#define LEAVE_DTR                0x0008  /*Set = DTR unchanged on cleanup */
#define LEAVE_RTS                0x0010  /*Set = RTS unchanged on cleanup */
#define IGNORE_16550             0x0020  /*Set = Do not enable 16550 */
#define DISABLE_MODEMSIG         0x0040  /*Set = Disable RTS/DTR on setup */
#define NOCHANGE_MODEMSIG        0x0080  /*Set = On setup leave RTS/DTR as is
*/
#define STORESOME                0x0100  /*Set = Store PORT/DATA only*/
#define LOCKBAUD                 0x0200  /*Set = Lock baud at initial setup */
#define POLLONSTATUS             0x0400  /*Set = call uoprime&uiprime */
#define PARENTPORT               0x0800  /*Set = VxD will use pport for parent
*/
#define CUSTOMFLAG1              0x1000  /*Custom usage explained elsewhere */
#define CUSTOMFLAG2              0x2000  /*Custom usage explained elsewhere */
#define TIMESTAMPBYTES           0x2000  /*Timestamp bytes in stream  */
#define CUSTOMFLAG3              0x4000  /*Custom usage explained elsewhere */
#define NINEBITPROTOCOL          0x4000  /*Use Nine Bit protocol*/
#define CUSTOMFLAG4              0x8000  /*Custom usage explained elsewhere */
#define MULTIDROP                0x8000  /*Use multidrop protocol*/

/*************************************************************************/
/*=====Values returned to intfunc() handler                              */
/*************************************************************************/
#define INTFUNC_VMODEM           0x0000  /* Modem intrpt */
#define INTFUNC_VSTAT            0x0001  /* Status intrpt */
#define INTFUNC_VRCV             0x0002  /* Receive intrpt */
#define INTFUNC_VXMIT            0x0003  /* Xmit interpt */
#define INTFUNC_VAVAIL           0x0004  /*  */
#define INTFUNC_VRCVCHR          0x0005  /* Specific char rcv intrpt */
#define INTFUNC_VIBFDCNT         0x0006  /* Inbuf dec to val intrpt */
#define INTFUNC_VIBFICNT         0x0007  /* Inbuf inc to val intrpt */
#define INTFUNC_VOBFDCNT         0x0008  /* Outbuf dec to val intrpt */
#define INTFUNC_VOBFICNT         0x0009  /* Outbuf inc to val intrpt */
#define INTFUNC_VXMTCHR          0x000a  /* XMIT specific char int */
#define INTFUNC_VWAITCDON        0x000b  /* Waitfor CD on int */
#define INTFUNC_VWAITCDOFF       0x000c  /* Waitfor CD off int */
#define INTFUNC_VWAITCTSON       0x000d  /* Waitfor CTS on int */
#define INTFUNC_VWAITCTSOFF      0x000e  /* Waitfor CTS off int */
```

```
#define INTFUNC_VWAITDSRON      0x000f  /* Waitfor DSR on int */
#define INTFUNC_VWAITDSROFF     0x0010  /* Waitfor DSR off int */
#define INTFUNC_VWAITRION       0x0011  /* Waitfor RI on int */
#define INTFUNC_VWAITRIOFF      0x0012  /* Waitfor RI off int */
#define INTFUNC_VWAITXEMPTY     0x0013  /* Waitfor xmit empty int */
#define INTFUNC_VWAITXNOTEMPTY  0x0014  /* Waitfor xmit not empty int */
#define INTFUNC_VWAITREMPTY     0x0015  /* Waitfor receive empty int */
#define INTFUNC_VWAITRNOTEMPTY  0x0016  /* Waitfor receive not empty int */
#define INTFUNC_VWAITTIME       0x0017  /* Waitfor time int */
#define INTFUNC_VX_BREAK        0x0018  /* Break intrpt */
#define INTFUNC_VX_FRAME        0x0019  /* Frame intrpt */
#define INTFUNC_VX_OVRUN        0x001a  /* Overrun intrpt */
#define INTFUNC_VX_PARER        0x001b  /* Parity intrpt */
#define INTFUNC_VX_CTS          0x001c  /* CTS intrpt */
#define INTFUNC_VX_DSR          0x001d  /* DSR intrpt */
#define INTFUNC_VX_CD           0x001e  /* CD intrpt */
#define INTFUNC_VX_RING         0x001f  /* RING intrpt */
#define INTFUNC_VWAITRCOUNT     0x0020  /* RCV Count intrpt */
#define INTFUNC_VWAITXCOUNT     0x0021  /* XMT Count intrpt */
#define INTFUNC_VSENDNOWDONE    0x0022  /* Send now done */

/*************************************************************************/
/*=====ser_rs232_misc_func() commands                                     */
/*************************************************************************/
#define WAITCDON        0x0000  /* Waitfor CD on */
#define WAITCDOFF       0x0001  /* Waitfor CD off */
#define WAITCTSON       0x0002  /* Waitfor RTS on */
#define WAITCTSOFF      0x0003  /* Waitfor RTS off */
#define WAITDSRON       0x0004  /* Waitfor DSR on  */
#define WAITDSROFF      0x0005  /* Waitfor DSR off */
#define WAITRION        0x0006  /* Waitfor RI on */
#define WAITRIOFF       0x0007  /* Waitfor RI off */
#define WAITXEMPTY      0x0008  /* Waitfor xmit buf empty */
#define WAITXNOTEMPTY   0x0009  /* Waitfor xmit buf not empty */
#define WAITREMPTY      0x000a  /* Waitfor receive buf empty */
#define WAITRNOTEMPTY   0x000b  /* Waitfor receive buf not empty */
#define SETRCVCHR       0x000c  /* Set receive character comparator */
#define SETXMTCHR       0x000d  /* Set transmit character comparator */
#define SETIBFDCNT      0x000e  /* Set input buffer down count threshold */
#define SETIBFICNT      0x000f  /* Set input buffer up count threshold */
#define SETOBFDCNT      0x0010  /* Set output buffer down count threshold */
#define SETOBFICNT      0x0011  /* Set output buffer up count threshold */
#define WAITTIME        0x0012  /* Waitfor time */
#define WAITRCOUNT      0x0013  /* Waitfor at least RCV buffer count */
#define WAITXCOUNT      0x0014  /* Waitfor at least XMT buffer count */
#define SETRCOUNT       0x0015  /* Waitfor at least RCV buffer count */
#define SETXCOUNT       0x0016  /* Waitfor at least XMT buffer count */
#define SENDNOW         0x0017  /* Send a string ahead of buffered data */

/*************************************************************************/
/*=====IRQ Flag bit settings(icb.irq_flag)                                */
/*************************************************************************/
#define IRQ_ACTIVE      0x0004          /* IRQ active */
#define IRQ_CHAIN_INT   0x0008          /* Chain interrupt */
```

```
/*****************************************************************************/
/*=====Return error codes                                                    */
/*****************************************************************************/
#define RS232ERR_NONE           0       /*RS232 no error*/
#define RS232ERR_BUFFER         1       /*RS232 buffer not set or buf
changed*/
#define RS232ERR_ACTIVE         2       /*RS232 port not active*/
#define RS232ERR_XMTBUF         3       /*RS232 xmit buffer full*/
#define RS232ERR_RCVBUF         4       /*RS232 recv buffer empty*/
#define RS232ERR_SYNTAX         5       /*RS232 port syntax  error*/
#define RS232ERR_BUFSIZ         6       /*RS232 invalid buffer size*/
#define RS232ERR_PORT           7       /*RS232 invalid port*/
#define RS232ERR_HANDLR         8       /*RS232 handler changed*/
#define RS232ERR_BAUD           9       /*RS232 invalid baud rate*/
#define RS232ERR_PARITY         10      /*RS232 invalid parity*/
#define RS232ERR_LNGTH          11      /*RS232 invalid data length*/
#define RS232ERR_STOPBIT        12      /*RS232 invalid # stopbits*/
#define RS232ERR_PROTOCOL       13      /*RS232 invalid protocol*/
#define RS232ERR_IRQCHANGED     14      /*RS232 IRQ changed*/
#define RS232ERR_PORTCHANGED    15      /*RS232 port changed*/
#define RS232ERR_THRESHOLD      16      /*RS232 invalid threshold*/
#define RS232ERR_INVIRQ         17      /*RS232 invalid IRQ*/
#define RS232ERR_INTOFF         18      /*RS232 interrupts not enabled*/
#define RS232ERR_BREAK          19      /*RS232 invalid break syntax*/
#define RS232ERR_FATAL          20      /*RS232 fatal error*/
#define RS232ERR_DSR            21      /*RS232 CTS error*/
#define RS232ERR_INVADR         22      /*RS232 Invalid RS232 address*/
#define RS232ERR_ENVIRON        23      /*Environment variable not set */
#define RS232ERR_IOCTL          24      /*Error issuing IOCTL call */
#define RS232ERR_ATEXIT         25      /*Error issuing atexit cleanup */
#define RS232ERR_DEVINIT        26      /*Error mapping device for dir calls*/
#define RS232ERR_DOSOPEN        27      /*Error opening Device */
#define RS232ERR_MALLOC         28      /*Error allocating memory */
#define RS232ERR_EXTMICRO       29      /*Error on external micro*/
#define RS232ERR_CARDCHANGED    30      /*Card changed error */
#define RS232ERR_CARDTYPE       31      /*Card type error */
#define RS232ERR_NOSUPPORT      32      /*Not supported */
#define RS232ERR_CMDBUFFULL     33      /*Card command buffer full*/
#define RS232ERR_PPORT          34      /*Invalid parent PCB*/
#define RS232ERR_NODEVICE       35      /*No device for this port*/
#define RS232ERR_UNKNOWN        36      /*Unknow error*/
#define RS232ERR_BUSY           37      /*Busy*/
#define RS232ERR_NOTIMER        38      /*No more timers available*/
#define RS232ERR_INT14VEC       39      /*INT 14H vector changed*/
#define RS232ERR_INT1CVEC       40      /*Timer vector changed*/
#define RS232ERR_DPMI           41      /*DPMI error*/
#define RS232ERR_WINBUF         42      /*No windows buffer or too small*/
#define RS232ERR_NOASYNCRES     43      /*No asynchronous resources left */
#define RS232ERR_NOTIMERRES     44      /*No timer resources left */
#define RS232ERR_NOOTHERES      45      /*Out of other resources */
#define RS232ERR_FILEIO         46      /*File I/O error */
#define RS232ERR_HMEMG64        47      /*Hardware memory exceeded 64K */
#define RS232ERR_MAPVXD         48      /*VXD/Kernel Driver not loaded */
```

```
#define RS232ERR_THREAD         49      /*Could not start a thread */
#define RS232ERR_MAPVDD         50      /*NT VDD CDRVVDD.DLL not loaded */

/*************************************************************************/
/*======Offsets to callbacks from sub-device functions to COMM-DRV      */
/*************************************************************************/
#define CALLBACK_MSR            0       /* Called on every MSR change int. */
#define CALLBACK_XMT            4       /* Called on every XMIT interrupt. */
#define CALLBACK_RCV            8       /* Called on every RCV interrupt. */
#define CALLBACK_LSR            12      /* Called on every LSR interrupt. */
#define CALLBACK_SETTIM         16      /* Called to set a timer. */
#define CALLBACK_RSTTIM         20      /* Called to reset a timer. */
#define CALLBACK_GETPATH        24      /* Called to reset a timer. */
#define CALLBACK_GETDATA        28      /* Called to get data pointer. */

/*************************************************************************/
/*======Offsets to sub-device functions                                  */
/*************************************************************************/
#define CALLDEV_CALLBACK        0       /* IDX for adr of ptr to c'back funcs
*/
#define CALLDEV_UINTHDR         4       /* IDX for sub-dev interrupt handler
*/
#define CALLDEV_UDTROFF         8       /* IDX for sub-dev dtr_off function */
#define CALLDEV_UDTRON          12      /* IDX for sub-dev dtr_on function */
#define CALLDEV_UGETREG         16      /* IDX for sub-dev  get uart reg func
*/
#define CALLDEV_UIPRIME         20      /* IDX for sub-dev input primer */
#define CALLDEV_UOPRIME         24      /* IDX for sub-dev output primer */
#define CALLDEV_UPUTREG         28      /* IDX for sub-dev put uart reg func
*/
#define CALLDEV_URESET          32      /* IDX for sub-dev reset/cleanup */
#define CALLDEV_URTSOFF         36      /* IDX for sub-dev rts_off */
#define CALLDEV_URTSON          40      /* IDX for sub-dev rts_on */
#define CALLDEV_USETUP          44      /* IDX for sub-dev init/setup */
#define CALLDEV_UXMIT           48      /* IDX for sub-dev xmit trigger */
#define CALLDEV_UKXMIT          52      /* IDX for sub-dev kill xmitter */
#define CALLDEV_UIXMIT          56      /* IDX for sub-dev immediate xmit */
#define CALLDEV_UBAUDIV         60      /* IDX for sub-dev baud rate divisor
*/
#define CALLDEV_INTENTER        64      /* IDX for sub-dev enter isr */
#define CALLDEV_INTEXIT         68      /* IDX for sub-dev exit isr */
#define CALLDEV_MPX             72      /* IDX for multiplex functions */

#define COMMDRV_SUBDEV_FLAG     96      /* Offset to subdevice flags */

/*************************************************************************/
/*======Auxiliary flag bits(auxpcb.aux_flag)                             */
/*************************************************************************/
#define AUXFLAG_BRKERR          0x00000001L     /* Set=Break error occurred */
#define AUXFLAG_FRAMERR         0x00000002L     /* Set=Framing error occurred
*/
#define AUXFLAG_IOVRERR         0x00000004L     /* Set=Input overrun occurred
*/
#define AUXFLAG_OVRERR          0x00000008L     /* Set=UART overrun occurred
*/
```

```
#define AUXFLAG_PARERR          0x00000010L     /* Set=Parity error occurred
*/
#define AUXFLAG_RING            0x00000020L     /* Set=Ring detected on UART
*/
#define AUXFLAG_XFERDLG         0x00000040L     /* Set=Show File Xfer Dialog
*/
#define AUXFLAG_XFERCAN         0x00000080L     /* Set=Cancel File Xfer */
#define AUXFLAG_NOMSGLOOP       0x00000100L     /* Set=No to 2ndary Msg Loop
*/
#define AUXFLAG_NOTIMER         0x00000200L     /* Set=No to timer routine */
#define AUXFLAG_SWITCHONDELAY   0x00000400L     /* Set=Force context on del */
#define AUXFLAG_9BITTOGGLE1     0x00000800L     /*  Nine bit toggle*/
#define AUXFLAG_9BITTOGGLE2     0x00001000L     /*  Nine bit toggle*/

/*************************************************************************/
/*======Async Event Resource                                             */
/*************************************************************************/
struct async_res
{
#ifndef MSWIN32
CDRVUINT                async_ptr;      /* Pointer to next async resource */
CDRVUINT                async_tim;      /* Timer */
CDRVUINT                async_val;      /* Value to pass to intfunc */
CDRVUINT                async_pcb;      /* Port PCB */
CDRVUINT                async_fnc;      /* Testing sub-function */
#else
struct async_res        *async_ptr;     /* Pointer to next async resource */
unsigned long           async_tim;      /* Timer */
short                   async_val;      /* Value to pass to intfunc */
struct port_param       *async_pcb;     /* Port PCB */
int                     async_fnc;      /* Testing sub-function index */
#endif
};

/*************************************************************************/
/*======Timed Event Resource                                             */
/*************************************************************************/
struct timer_res
{
CDRVUINT        timer_ptr;      /* Pointer to next timer resource */
CDRVUINT        timer_tim;      /* Timer */
void FAR        *timer_fnc;     /* user function */
CDRVUINT        timer_rtm;      /* Timer reload */
};

/*************************************************************************/
/*======Modem Information structure.                                     */
/*************************************************************************/
struct modem_param
{
int     Pacetime;       /* Modem pacetime(def=>1)                        */
int     PlusPacetime;   /* +++ Pacetime(def=>2)                          */
int     Waitresponse;   /* Timeout to wait for modem(def=>1*18)          */
int     Waitconnect;    /* Timeout for connect(def=>30*18)               */
```

```
int     Waitecho;       /* Timeout to wait for echo(def=>1*18)          */
int     Waitcommand;    /* Timeout to wait for command(def=>1*18)       */
int     Waitdtr;        /* DTR flash time(def=>2*18)                    */
int     RingInterval;   /* Interval between rings(def=>10*18)           */
char    CmdPref[4];     /* AT command prefix (def=>"AT")                */
char    Init[80];       /* Initialization string (def=>"V1Q0")          */
char    ToneDial[4];    /* Tone Dial (def=>"DT")                        */
char    PulseDial[4];   /* Pulse Dial(def=>"DP")                        */
char    Reset[4];       /* Reset modem string (def=>"Z")                */
char    Escape[12];     /* Escape to command mode(def=>"+++")           */
char    Hangup[4];      /* Hangup string (def=>"H")                     */
char    CmdSufx[4];     /* AT command suffix (def=>"\r")                */
char    Ok[10];         /* AT command acknowledge (def=>"OK")           */
char    Error[10];      /* AT command failure (def=>"ERROR")            */
char    Connect[15];    /* AT command connect (def=>"CONNECT")          */
char    NoCarrier[15];  /* AT command carrier (def=>"NO CARRIER")       */
char    Busy[10];       /* AT command connect (def=>"BUSY")             */
char    NoTone[15];     /* AT command connect (def=>"NO DIALTONE")      */
char    Verbose[10];    /* AT Verbose mode (def=>"V1Q0")                */
char    Answer[10];     /* AT Verbose mode (def=>"S0=")                 */
char    Carrier[15];    /* AT command connect (def=>"CARRIER")          */
char    Ring[10];       /* AT Ringing (def=>"RING")                     */
char    AnswerCmd[4];   /* AT Answering (def=>"A")                      */
char    Online[4];      /* AT Go Online (def=>"O")                      */
char    Offhook[4];     /* AT Go Offhook (def=>"H1")                    */
char    SpeakerOff[4];  /* AT Speaker Off(def=>"M0")                    */
char    SpeakerOn[4];   /* AT Speaker On(def=>"M1")                     */
char    VolumeLow[4];   /* AT Volume Low(def=>"L1")                     */
char    VolumeMedium[4];/* AT Volume Medium(def=>"L2")                  */
char    VolumeHigh[4];  /* AT Volume High(def=>"L3"                     */
};

/*****************************************************************************/
/*======Auxiliary port control block                                      */
/*****************************************************************************/
struct aux_pparam
{
#define SENDNOWBUFSIZE  10      /* Size of send now buffer */
unsigned long   aux_itimer[3];  /* Input timer block */
unsigned long   aux_otimer[3];  /* Output timer */
unsigned long   aux_rcvint;     /* Receiver interrupts */
unsigned long   aux_timint;     /* Timeout Receiver interrupts */
unsigned long   aux_xmtint;     /* Transmit interrupts */
unsigned long   aux_lsrint;     /* LSR interrupts */
unsigned long   aux_msrint;     /* MSR interrupts */
unsigned long   aux_ovrint;     /* Overrun error */
unsigned long   aux_parint;     /* Parity error */
unsigned long   aux_frmint;     /* Framing error */
unsigned long   aux_brkint;     /* Break error */
unsigned long   aux_rcvxon;     /*XONs received*/
unsigned long   aux_rcvxoff;    /*XOFFs received*/
unsigned long   aux_xmtxon;     /*XONs xmitted*/
unsigned long   aux_xmtxoff;    /*XOFFs xmitted*/
unsigned long   aux_ctson;      /*CTS off to on transitions*/
```

```
unsigned long   aux_dsron;      /*DSR off to on transitions*/
unsigned long   aux_rtson;      /*RTS off to on transitions*/
unsigned long   aux_dtron;      /*DTR off to on transitions*/
unsigned int    aux_iblk[5];    /* Input buffer control block */
unsigned int    aux_oblk[5];    /* Output buffer control block */
void FAR        *aux_wintfunc;  /* func */
CDRVUINT        aux_wintfuncds; /* Windows function data segment */
unsigned char   aux_spare01[2]; /* */
CDRVUINT        aux_ibfdcnt;    /* Input buffer dec flag point(intfunc) */
CDRVUINT        aux_ibficnt;    /* Input buffer inc flag point(intfunc) */
CDRVUINT        aux_obfdcnt;    /* Output buffer dec flag point(intfunc) */
CDRVUINT        aux_obficnt;    /* Output buffer inc flag point(intfunc) */
unsigned char   aux_rcvchr;     /* Receive char flag point(intfunc) */
unsigned char   aux_xmtchr;     /* Transmit char flag point(intfunc) */
unsigned char   aux_cmask;      /* Character mask */
unsigned char   aux_spare2[1];  /* Spares */
CDRVUINT        aux_rcount;     /* RCV Buffer Count Event thresh */
CDRVUINT        aux_xcount;     /* XMT Buffer Count Event thresh */
int             aux_wport;      /* Windows driver port for this PCB*/
CDRVUINT                aux_sendnowcnt;         /* Send now count */
unsigned char FAR       *aux_sendnowptr;        /* Pointer to data */
char            aux_sendnowbuf[SENDNOWBUFSIZE]; /* Send now buffer size */
long            aux_sendnowtot; /* #of sendnow characters sent. */
long            aux_icharflush; /* #of input characters flushed */
long            aux_ocharflush; /* #of output characters flushed */
long            aux_charnotbuf; /* #of chars disposed by "RCV" intfunc */
long            aux_charnotsent;/* #of chars not sent because of"XMT"intfunc*/
CDRVUINT        aux_user[5];            /* For user purposes */
long            aux_wintfunclost;       /* Calls not made to callback */
CDRVIFNC (FARC *(FAR *aux_callback))(short c,struct port_param FAR *p);
int             aux_pacetime;   /* Character pacing time interval */
int             aux_ftimeout;   /* Function timeout */
int (FARC *aux_tfunc)(int port);        /* Fnc set w/CdrvSetTimeoutFunction()
*/
int             aux_timerres;           /* Timer resolution */
long            aux_flag;               /* Flag used by high level functions
*/
struct modem_param FAR  *aux_modem;     /* Modem Information */
long            aux_carrierspeed;       /* Modem carrier speed */
long            aux_connectspeed;       /* Modem connect speed */
void (FARC *aux_dsfnc)(int p,int c, char FAR *h); /* Data stream fnc  */
int             aux_delimcnt;           /* # of characters in delimiter list
*/
char FAR        *aux_delimptr;          /* Pointer to delimiter list */
};

/*************************************************************************/
/*======Interrupt Control Block structure (ICB)                          */
/*************************************************************************/
struct irq_param
{
CDRVUINT        irq_hdr;        /* Offset to IRQ handler */
CDRVUINT        irq_lnk;        /* Head of all PCB belonging to this IRQ */
void    (FARC *irq_oldvec)();   /* Old IRQ vector for this IRQ */
```

```
CDRVUINT        irq_num;        /* IRQ number */
CDRVUINT        irq_flag;       /* Flags/Subdevices may use upper 8 bits */
unsigned long   irq_pass;       /* Total interupts passed to next handler */
unsigned long   irq_int;        /* Interrupt instances processed */
unsigned long   irq_ign;        /* Interrupts ignored */
unsigned long   irq_oldvec2[2]; /* Old IRQ vector for this IRQ */
unsigned long   irq_spare[3];   /* Spares */
};

/*************************************************************************/
/*======Port Control Block structure (PCB)                               */
/*************************************************************************/
struct port_param
{
CDRVUINT        ser_rs232_base; /*Base address of port*/
unsigned short  irq;            /*Interrupt number*/
CDRVUINT        baud;           /*baud rate index*/
unsigned short  parity;         /*parity index*/
unsigned short  lngth;          /*length index*/
unsigned short  stopbit;        /*stopbit index*/
unsigned short  brk;            /*Break*/
unsigned short  protocol;       /*protocol index*/
unsigned char   block[2];       /*# of 1/18.2 secs count*/
CDRVUINT        inbuf_low;      /*Low threshold of recv buffer*/
CDRVUINT        inbuf_high;     /*High threshold of xmit buffer*/
CDRVUINT        buffer_seg;     /*Segment to start of buffer*/
CDRVUINT        buffer_off;     /*Offset to start of buffer*/
CDRVUINT        inbuf_len;      /*Input buffer length*/
CDRVUINT        outbuf_len;     /*Output buffer length*/
unsigned char   aux_addr1;      /*Auxiliary Address*/
unsigned char   sflag;          /*System flag*/
CDRVUINT        cardseg;        /*Memory card segment*/
unsigned short  cardtype;       /*Card Type*/
CDRVIFNC (FAR *(FAR *intfunc))(short c,struct port_param FAR *p);
CDRVUINT        error;          /* Error flag*/
CDRVUINT        flag;           /* general purpose flag*/
CDRVUINT        inbuf_count;    /* # bytes in input buf*/
CDRVUINT        outbuf_count;   /* # of bytes in ouput*/
unsigned long   lost_ichar;     /* Lost inputted bytes*/
unsigned long   lost_ochar;     /* Lost outputted bytes*/
unsigned long   total_ichar;    /* Total attempted bytes in*/
unsigned long   total_ochar;    /* Total attempted bytes out*/
void FAR *   FAR *devfuncs;     /* Pointer to table of pointer to dev funcs
*/
unsigned char   msr_reg;        /* Current modem status register*/
unsigned char   lsr_reg;        /* Current line status register*/
CDRVUINT        ddata;          /* Data that sub-device driver may use */
CDRVUINT        inbuf_start;    /* Input  buffer control block*/
CDRVUINT        outbuf_start;   /* Output buffer control block*/
struct irq_param FAR *irqp;     /* Pointer to irq parameter block */
CDRVUINT        port;           /* Port number */
CDRVUINT        flag2;          /* Secondary flag */
CDRVUINT        obufoff;        /* Output buf off if DIFFXBUF set */
CDRVUINT        obufseg;        /* Output buf seg if DIFFXBUF set */
```

```
CDRVUINT        pport;          /* Parent port */
CDRVUINT        ppcb;           /* Parent pcb */
CDRVUINT        timer_iblk;     /*Timer block.*/
CDRVUINT        timer_oblk;     /*Timer block.*/
unsigned char   hblock[2];      /* Hi byte of input/output timeouts */
unsigned char   char_xon;       /* XON character */
unsigned char   char_xoff;      /* XOFF character */
unsigned char   char_xxon;      /* XON xmit character */
unsigned char   char_xxoff;     /* XOFF xmit character */
struct aux_pparam FAR *auxpcb;  /* Pointer to auxiliary PCB*/
unsigned int    prot_local;     /* Local protocol */
unsigned int    prot_remote;    /* Remote protocol */
struct port_param FAR *opcb;    /* Pointer to this PCB*/
CDRVUINT        dflag;          /* Subdevice specific flag */
unsigned char   o_baud_reg[2];  /* Old baud rate*/
unsigned char   o_lcr_reg;      /* Old line control register*/
unsigned char   o_mcr_reg;      /* Old modem control register*/
unsigned char   o_ier_reg;      /* Old interrupt enable register*/
unsigned char   tuart;          /* Type of UART*/
CDRVUINT        next_pcb;       /* Pointer to next */
};

/*****************************************************************************/
/*=====System Data                                                           */
/*****************************************************************************/
struct ser_rs232_sdata
{
CDRVUINT        dfunc;          /* Offset to table of driver functions  */
unsigned short  drvmn;          /* Maximum number of drivers            */
unsigned short  drvna;          /* Number of active drivers             */
unsigned short  icbmn;          /* Highest IRQ allowed                  */
unsigned short  icbmx;          /* Max num of concurrently active IRQs  */
CDRVUINT        icbs;           /* Offset pointer to pool of icbs       */
CDRVUINT        icbss;          /* Offset to pool of icbs               */
unsigned short  icbna;          /* Number of ICBs Active                */
unsigned short  pcbmn;          /* Highest port number allowed          */
unsigned short  pcbmx;          /* Max num of concurrently active ports */
CDRVUINT        pcbs;           /* Offset pointer to pool of pcbs       */
CDRVUINT        pcbss;          /* Offset to pool of pcbs               */
unsigned short  pcbna;          /* Number of PCBs Active                */
unsigned short  timmx;          /* Maximum Number of timers             */
CDRVUINT        timss;          /* Start of timers                      */
CDRVUINT        timact;         /* Start of timers active               */
CDRVUINT  timfre;               /* Start of free timers                 */
unsigned short  asyncmx;        /* Maximum Number of async resources    */
CDRVUINT        asyncss;        /* Start of async resources             */
CDRVUINT        asyncact;       /* Start of async active                */
CDRVUINT  asyncfre;     /* Start of free asyncs                 */
void FAR        *wbuf;          /* Pointer Windows shared buffer        */
CDRVUINT        wbufsz;         /* Windows buffer size                  */
CDRVUINT        wbuffg;         /* Windows shared buffer flag           */
void FAR        *wintfunc;      /* DPMI function to call Windows func   */
CDRVUINT        irqign;         /* Ints ignored during COMM-DRV init    */
void FAR        *old_thandler;  /* Old timer handler                    */
```

```
void FAR        *callback[10];  /* Array of pointers to callbacks       */
unsigned short  FAR *dev_head;  /* Start of MS-DOS/COMMDRV device chain */
unsigned short  dev_count;      /* Number of MSDOS/COMMDRV devices      */
char    cdrvpath[CDRVPATHLEN];  /* Path to COMMDRV                      */
unsigned char   baudremap[22];  /* Baud rate remapping                  */
struct port_param FAR * (FARC *getpcbfptr)(int port);   /* Get PCB fnc  */
int             multitasker;    /* 0=NONE 1=MTASK 2=Desqview 3=WindowsE */
int             devpolltime;    /* Device poll time                     */
void (FARC *begincritical)(void);       /* Begin critical section       */
void (FARC *endcritical)(void);         /* End critical section         */
void (FARC *ibegincritical)(void);      /* Begin int critical section   */
void (FARC *iendcritical)(void);        /* End int critical section     */
long            critdenied;             /* Critical section denied      */
long            icritdenied;            /* Crit sectione denied from int*/
};

/*****************************************************************************/
/* Structure of Ring Buffer Control Block                                    */
/*****************************************************************************/
struct RingCtl
{
int                     RingTop;
int                     RingBottom;
int                     RingInput;
int                     RingOutput;
int                     Size;
int                     Cnt;
unsigned char FAR       *Buf;
};

/*****************************************************************************/
/*=====IOCTL Commands for COMM-DRV/Dos                                        */
/*****************************************************************************/
#define SER_IOCTL_OUT_SET       0       /* Set communication parameters */
#define SER_IOCTL_OUT_PUTREG    1       /* Change register */
#define SER_IOCTL_OUT_BLOCK     2       /* Set blocking */
#define SER_IOCTL_OUT_FLUSH     3       /* Flush buffers */
#define SER_IOCTL_OUT_DTR_ON    4       /* Turn DTR on */
#define SER_IOCTL_OUT_DTR_OFF   5       /* Turn DTR off */
#define SER_IOCTL_OUT_RTS_ON    6       /* Turn RTS on */
#define SER_IOCTL_OUT_RTS_OFF   7       /* Turn RTS off */
#define SER_IOCTL_OUT_INTFUNC   8       /* User Interrupt Function */
#define SER_IOCTL_OUT_HBLOCK    9       /* Set blocking 16bit */
#define SER_IOCTL_OUT_ASET      10      /* Set advance comm parameters */
#define SER_IOCTL_OUT_ORAW      11      /* Set raw mode */

/*****************************************************************************/
/*=====Structure for Initializing COM port through IOCTL                     */
/*****************************************************************************/
union ser_ioctl_out
{
struct ser_ioctl_out_sets
     {
     unsigned char   type;          /*Function type*/
```

```
        unsigned char   baud;           /*baud rate index*/
        unsigned char   parity;         /*parity index*/
        unsigned char   lngth;          /*length index*/
        unsigned char   stopbit;        /*stopbit index*/
        unsigned char   brk;            /*Break*/
        unsigned char   protocol;       /*protocol index*/
        unsigned char   block[2];       /*# of 1/18.2 secs count*/
        } set;
struct ser_ioctl_out_putregs
        {
        unsigned char   type;           /*Function type*/
        unsigned char   reg;            /*Register to change*/
        unsigned char   value;          /*Value to stuf */
        } putreg;
struct ser_ioctl_out_blks
        {
        unsigned char   type;           /*Function type*/
        unsigned char   block[2];       /*Blocking counts*/
        } blk;
struct ser_ioctl_out_flshs
        {
        unsigned char   type;           /*Function type*/
        unsigned char   flush_buf;      /*Buffer to flush */
        } flsh;
struct ser_ioctl_out_dtron
        {
        unsigned char   type;           /*Function type*/
        } dtr_on;
struct ser_ioctl_out_dtroff
        {
        unsigned char   type;           /*Function type*/
        } dtr_off;
struct ser_ioctl_out_rtson
        {
        unsigned char   type;           /*Function type*/
        } rts_on;
struct ser_ioctl_out_rtsoff
        {
        unsigned char   type;           /*Function type*/
        } rts_off;
struct ser_ioctl_out_intfunc
        {
        unsigned char   type;           /*Function type*/
        unsigned char   filler;
        CDRVIFNC (FAR *(FAR *func))(short val,struct port_param FAR *p);
        short   imask;
        } intfunc;
struct ser_ioctl_out_hblks
        {
        unsigned char   type;           /*Function type*/
        unsigned char   filler;
        unsigned int    block[2];       /*Blocking counts*/
        } hblk;
struct ser_ioctl_out_asets
```

```
     {
     unsigned char   type;          /*Function type*/
     unsigned char   subtype;       /* =0 */
     unsigned char   baud;          /*baud rate index*/
     unsigned char   parity;        /*parity index*/
     unsigned char   lngth;         /*length index*/
     unsigned char   stopbit;       /*stopbit index*/
     unsigned char   brk;           /*Break*/
     unsigned char   protocol;      /*protocol index*/
     unsigned short  block[2];      /*# of 1/18.2 secs count*/
     unsigned char   xon;           /* XON */
     unsigned char   xoff;          /* XOFF */
     unsigned char   xxon;          /* XON xmit */
     unsigned char   xxoff;         /* XOFF xmit */
     unsigned short  spare[19];     /* Spares */
     } aset;
struct ser_ioctl_out_oraw
     {
     unsigned char   type;          /*Function type*/
     unsigned char   flag;          /* Flag */
     } oraw;
};

/*************************************************************************/
/*=====Structure for reading status of COM port through IOCTL           */
/*************************************************************************/
union ser_ioctl_in
{
struct ser_rs232_routines      FAR *ser_rs232; /*Ptr to routine ptrs*/
struct port_param              port;           /* Port param block */
};

/*************************************************************************/
/*=====Offsets to functions in direct address mapping table of ptrs     */
/*************************************************************************/
#define CLEANUP         0
#define GETBYTE         4
#define GETPORT         8
#define GETREGISTER     12
#define GETSTATUS       16
#define PUTBYTE         20
#define PUTREGISTER     24
#define MAXPORT         28
#define SETUP           32
#define PUTPACKET       36
#define GETPACKET       40
#define FLUSH           44
#define BLCK            48
#define DTR_ON          52
#define DTR_OFF         56
#define RTS_ON          60
#define RTS_OFF         64
#define SET_INTFUNC     68
#define GET_SDATA       72
```

```
#define MISC_FUNC       76
#define SETBAUDDIV      80
#define VIEWPACKET      84

/*************************************************************************/
/*=====Structure of ptrs to serial routines in COMMTSR                   */
/*************************************************************************/
struct ser_rs232_routines
{
int (FAR *cleanup)(int port);
int (FAR *getbyte)(int port,unsigned char FAR *ch);
int (FAR *getport)(int port,struct port_param FAR *pcb);
int (FAR *getregister)(int port,int offset,int FAR *value);
int (FAR *getstatus)(int port,unsigned char FAR *stat);
int (FAR *putbyte)(int port,unsigned char FAR *ch);
int (FAR *putregister)(int port,int offset,int value);
int (FAR *maxport)(void);
int (FAR *setup)(int port,struct port_param FAR *pcb);
int (FAR *putpacket)(int port,int count,unsigned char FAR *ch);
int (FAR *getpacket)(int port,int count,unsigned char FAR *ch);
int (FAR *flush)(int port,int arg);
int (FAR *block)(int port,int_input,int output);
int (FAR *dtr_on)(int port);
int (FAR *dtr_off)(int port);
int (FAR *rts_on)(int port);
int (FAR *rts_off)(int port);
int (FAR *set_intfunc)(int port,
CDRVIFNC (FAR *(FAR *func))(short val,struct port_param FAR *p),
short imask);
struct ser_rs232_sdata FAR * (FAR *get_sdata)(void);
int (FAR *misc_func)(int port,unsigned short cmd,unsigned long val);
int (FAR *setbauddiv)(int cardtype,int baudidx,unsigned bauddiv);
int (FAR *viewpacket)(int port,int count,unsigned char FAR *ch);
};

/*************************************************************************/
/*=====INT14 PCB                                                         */
/*************************************************************************/
struct ser_int14_pcb
{
int     ser_int14_flag;         /* Flags */
};

/*************************************************************************/
/*======Int14 handler information record.                                */
/*************************************************************************/
struct ser_int14_info
{
char                            hdrname[9+1];   /* Name of handler */
struct ser_rs232_routines FAR   *ser_rs232;     /*Ptr to routine ptrs*/
long                            oldvec;         /* Old INT14H vector */
};

/*************************************************************************/
```

```
/*=====Control block used by serlprim.c.                                    */
/*************************************************************************/
struct commdrv_ctl
{
unsigned int    flag;                           /* Flag */
#define COMMDRV_PORT_OPEN       0x0001          /* Port was opened */
int             h;                              /* Handle */
unsigned char   *buf;                           /* Pointer to buf for data */
char            *dev;                           /* Pointer to device name */
};

/*************************************************************************/
/*=====Communication Device header structure(COMM-DRV/Dos)                */
/*************************************************************************/
struct  dev_head
{
unsigned FAR    *next_dev;              /* Pointer to next device */
unsigned        attribute;              /* Type of device */
unsigned        strategy;               /* Pointer to strategy routine */
unsigned        interrpt;               /* Pointer to interrupt routine */
char            dev_name[8];            /* Device name */
unsigned        com_id;                 /* Port id */
unsigned        rh_seg;                 /* Request Header segment */
unsigned        rh_off;                 /* Request Header offset */
unsigned        device_flag;            /* Local device flag */
unsigned char   ahead_byte;             /* Look ahead byte */
unsigned char   dev_flag;               /*  */
};

/*************************************************************************/
/*=====File transfer definitions                                          */
/*************************************************************************/
/* Definitions */
#define CDRVXFER_SOH    '\001'
#define CDRVXFER_STX    '\002'
#define CDRVXFER_EOT    '\004'
#define CDRVXFER_ACK    '\006'
#define CDRVXFER_NAK    '\025'
#define CDRVXFER_CAN    '\030'
#define CDRVXFER_CC     '\103'
#define CDRVXFER_G      '\107'
#define CDRVXFER_XON    '\021'
#define CDRVXFER_XOFF   '\023'
#define CDRVXFER_DEL    '\177'

/* Zmodem special packet characters */
#define CDRVXFER_ZPAD   '*'             /* 052 Pad character begins frames */
#define CDRVXFER_ZDLE   0x18            /* Ctrl-X Zmodem escape */
#define CDRVXFER_ZDLEE  (0x18^0x40)     /* Escaped ZDLE as transmitted */
#define CDRVXFER_ZBIN   'A'             /* Binary frame indicator */
#define CDRVXFER_ZHEX   'B'             /* HEX frame indicator */
#define CDRVXFER_ZBIN32 'C'             /* Binary frame with 32 bit FCS */

/* ZDLE escape pairs */
```

```
#define CDRVXFER_ZCRCE 'h'      /* CRC next, frame ends, header pkt follows */
#define CDRVXFER_ZCRCG 'i'      /* CRC next, frame continues nonstop */
#define CDRVXFER_ZCRCQ 'j'      /* CRC next, frame continues, ZACK expected */
#define CDRVXFER_ZCRCW 'k'      /* CRC next, ZACK expected, end of frame */
#define CDRVXFER_ZRUB0 'l'      /* Translate to rubout 0177 */
#define CDRVXFER_ZRUB1 'm'      /* Translate to rubout 0377 */

/* Bit Masks for ZRINIT first flag byte(flag0=offset+3,flag1=offset+2,...  */
#define CDRVXFER_CANFDX         01      /* Rx can snd and rcv true FDX */
#define CDRVXFER_CANOVIO        02      /* Rx can rcv data during disk I/O */
#define CDRVXFER_CANBRK         04      /* Rx can snd a break signal */
#define CDRVXFER_CANCRY         010     /* Rx can decrypt */
#define CDRVXFER_CANLZW         020     /* Rx can uncompress */
#define CDRVXFER_CANFC32        040     /* Rx can use 32 bit Frame Check */
#define CDRVXFER_ESCCTL         0100    /* Rx expects ctl chars to be escaped
*/
#define CDRVXFER_ESC8           0200    /* Rx expects 8th bit to be escaped */

/* Byte positions within header array */
#define CDRVXFER_ZF0    3               /* First flags byte */
#define CDRVXFER_ZF1    2
#define CDRVXFER_ZF2    1
#define CDRVXFER_ZF3    0
#define CDRVXFER_ZP0    0               /* Low order 8 bits of position */
#define CDRVXFER_ZP1    1
#define CDRVXFER_ZP2    2
#define CDRVXFER_ZP3    3               /* High order 8 bits of file position
*/

/* Parameters for ZSINIT frame */
#define CDRVXFER_ZATTNLEN 32    /* Max length of attention string */
/* Bit Masks for ZSINIT flags byte ZF0 */
#define CDRVXFER_TESCCTL 0100   /* Trans expects ctl chars to be escaped */
#define CDRVXFER_TESC8   0200   /* Trans expects 8th bit to be escaped */

/* Parameters for ZFILE frame */
/* Conversion options one of these in ZF0 */
#define CDRVXFER_ZCBIN          1       /* Binary xfer - inhibit conversion */
#define CDRVXFER_ZCNL           2       /* Convert NL to local eol convention
*/
#define CDRVXFER_ZCRESUM        3       /* Resume interrupted file transfer */
/* Management include options, one of these ored in ZF1 */
#define CDRVXFER_ZMSKNOLOC      0200    /* Skip file if not present at rx */
/* Management options, one of these ored in ZF1 */
#define CDRVXFER_ZMMASK         037     /* Mask for the choices below */
#define CDRVXFER_ZMNEWL         1       /* Xfer if source newer or longer */
#define CDRVXFER_ZMCRC          2       /* Xfer if diff file CRC or length */
#define CDRVXFER_ZMAPND         3       /* Append contents to existing file */
#define CDRVXFER_ZMCLOB         4       /* Replace existing file */
#define CDRVXFER_ZMNEW          5       /* Transfer if source newer */
/* Number 5 is alive ... */
#define CDRVXFER_ZMDIFF         6       /* Xfer if dates or lengths different
*/
#define CDRVXFER_ZMPROT         7       /* Protect destination file */
```

```
/* Transport options, one of these in ZF2 */
#define CDRVXFER_ZTLZW          1       /* Lempel-Ziv compression */
#define CDRVXFER_ZTCRYPT        2       /* Encryption */
#define CDRVXFER_ZTRLE          3       /* Run Length encoding */
/* Extended options for ZF3, bit encoded */
#define CDRVXFER_ZXSPARS        64      /* Encod for sparse file operations */

/* Parameters for ZCOMMAND frame ZF0 (otherwise 0) */
#define CDRVXFER_ZCACK1         1       /* Acknowledge, then do command */

/* Zmodem frame types */
#define CDRVXFER_ZRQINIT        0       /* Request receive init.*/
#define CDRVXFER_ZRINIT         1       /* Receive init */
#define CDRVXFER_ZSINIT         2       /* Send init sequence (optional) */
#define CDRVXFER_ZACK           3       /* ACK to above */
#define CDRVXFER_ZFILE          4       /* File name from sender */
#define CDRVXFER_ZSKIP          5       /* To sender: skip this file */
#define CDRVXFER_ZNAK           6       /* Last packet was garbled */
#define CDRVXFER_ZABORT         7       /* Abort batch transfers */
#define CDRVXFER_ZFIN           8       /* Finish session */
#define CDRVXFER_ZRPOS          9       /* Resume data trans at this position
*/
#define CDRVXFER_ZDATA          10      /* Data packet(s) follow */
#define CDRVXFER_ZEOF           11      /* End of file */
#define CDRVXFER_ZFERR          12      /* Fatal Read or Write error Detected
*/
#define CDRVXFER_ZCRC           13      /* Request for file CRC and response
*/
#define CDRVXFER_ZCHALLENGE     14      /* Receiver's Challenge */
#define CDRVXFER_ZCOMPL         15      /* Request is complete */
#define CDRVXFER_ZCAN           16      /* Other end canned session w/CAN*5 */
#define CDRVXFER_ZFREECNT       17      /* Req for free bytes on filesystem */
#define CDRVXFER_ZCOMMAND       18      /* Command from sending program */
#define CDRVXFER_ZSTDERR        19      /* Output to stdd error, data follows
*/

/* Lengths */
#define LEN_FSPEC       255

/* States */
#define CDRVXFER_RSTATE_INIT            0
#define CDRVXFER_RSTATE_BEGIN           1
#define CDRVXFER_RSTATE_SENDSYNC        2
#define CDRVXFER_RSTATE_GETCHAR         3
#define CDRVXFER_RSTATE_SENDACK         4
#define CDRVXFER_RSTATE_SENDNAK         5
#define CDRVXFER_RSTATE_ZRINIT          6
#define CDRVXFER_RSTATE_SENDHEADER      7
#define CDRVXFER_RSTATE_WZPKT01         8

/* Sub-States */
#define CDRVXFER_RSUBSTATE_GETFRAME     20
#define CDRVXFER_RSUBSTATE_GETBLKBYT1   21
#define CDRVXFER_RSUBSTATE_GETBLKBYT2   22
```

```
#define CDRVXFER_RSUBSTATE_GETDATA      23
#define CDRVXFER_RSUBSTATE_GETCRC1      24
#define CDRVXFER_RSUBSTATE_GETCRC2      25
#define CDRVXFER_RSUBSTATE_SENDHDR      26
#define CDRVXFER_RSUBSTATE_SENDDATA     27

/* Sending States */
#define CDRVXFER_SSTATE_INIT            40
#define CDRVXFER_SSTATE_BEGIN           41
#define CDRVXFER_SSTATE_RECVSYNC        42
#define CDRVXFER_SSTATE_SENDPACKET      43
#define CDRVXFER_SSTATE_SENDFINFO       44
#define CDRVXFER_SSTATE_SENDHEADER      45
#define CDRVXFER_SSTATE_READBLOCK       46
#define CDRVXFER_SSTATE_REPLY           47
#define CDRVXFER_SSTATE_EOT             48
#define CDRVXFER_SSTATE_ZRQINIT         49
#define CDRVXFER_SSTATE_WZPKT01         50
#define CDRVXFER_SSTATE_STARTSPACKET    51
#define CDRVXFER_SSTATE_SENDBFRAME      52
#define CDRVXFER_SSTATE_WZPKT02         53

/* Substates */
#define CDRVXFER_SSUBSTATE_PUTFRAME     60
#define CDRVXFER_SSUBSTATE_PUTBLKBYT1   61
#define CDRVXFER_SSUBSTATE_PUTBLKBYT2   62
#define CDRVXFER_SSUBSTATE_PACKETSENT   63
#define CDRVXFER_SSUBSTATE_EOTSENT      64

/* Error code */
#define CDRVXFER_ERR_OK         0       /* Complete without errors */
#define CDRVXFER_ERR_NONE       0       /* Complete without errors */
#define CDRVXFER_ERR_SERIAL     1       /* Some serial I/O error */
#define CDRVXFER_ERR_OS         2       /* MSDOS Error */
#define CDRVXFER_ERR_PROTOCOL   3       /* Protocol Error */
#define CDRVXFER_ERR_BUSY       4       /* Protocol engine still busy */
#define CDRVXFER_ERR_CANCEL     5       /* Cancel terminated */
#define CDRVXFER_ERR_INVPORT    6       /* Invalid Port */
#define CDRVXFER_ERR_NOPCB      7       /* No more PCBs */
#define CDRVXFER_ERR_MTASK      8       /* MTASK error */
#define CDRVXFER_ERR_ACTIVE     9       /* XFER on port still active */
#define CDRVXFER_ERR_FNAME      10      /* Invalid filename */
#define CDRVXFER_ERR_KILLED     11      /* Transfer killed */
#define CDRVXFER_ERR_MEMORY     12      /* Memory allocation error */
#define CDRVXFER_ERR_NOCD       13      /* No Carrier Detect */


/* X(Y)MODEM Definitions */
#define CDRVXFER_MAXGRETRY      2
#define CDRVXFER_MAXCRETRY      5
#define CDRVXFER_MAXRETRY       10
#define CDRVXFER_MAXBRETRY      10
#define CDRVXFER_MAXTIMEOUT     180
#define CDRVXFER_MAXPAUSE       90
```

```
/* ZMODEM Definitions */
#define CDRVXFER_MAXORETRY      10      /* Max number of char output retries
*/
#define CDRVXFER_MAXOTIMOUT     180     /* Max time for char output */
#define CDRVXFER_MAXIRETRY      10      /* Max number of char input retries */
#define CDRVXFER_MAXITIMOUT     540     /* Max time for char input */
#define CDRVXFER_MAXZRETRY      10

/*************************************************************************/
/*=====File find structure                                                */
/*************************************************************************/
struct cdrvxfer_fileinfo
{
unsigned                time;
unsigned                date;
long                    size;
char FAR                *fspec;
struct dos_find_struct  fbuf;
};

/*************************************************************************/
/*=====File transfer control structure                                    */
/*************************************************************************/
struct cdrvxfer_ftpcb
{
/* User Provided */
char                    fspec[LEN_FSPEC+1];
int                     port;
int                     state;
unsigned char FAR       *buf;
unsigned int            bufsiz;
int                     retrycnt;
int                     rcvtimeout;
int                     xmttimeout;

/* Xfer Progress Info */
int                     curblk;
int                     blksiz;
long                    numbytes;
long                    offsetbytes;
int                     error;
int                     exterror;
int                     totalerr;
int                     currenterr;

/* System Usage */
int                     fhandle;
int                     saveerror;
int                     deltaerr;
int                     substate;
int                     blkidx;
int                     retry;
int                     crcretry;
```

```
int                     gretry;
int                     bretry;
#define CDRVXFER_FLAG_NOTCRC     0x00000001L
#define CDRVXFER_FLAG_SYNCED     0x00000002L
#define CDRVXFER_FLAG_YMODEM     0x00000004L
#define CDRVXFER_FLAG_YMODEMG    0x00000008L
#define CDRVXFER_FLAG_SFIRST     0x00000010L
#define CDRVXFER_FLAG_SCNVHEX    0x00000020L
#define CDRVXFER_FLAG_SDOCRC     0x00000040L
#define CDRVXFER_FLAG_SCRC32     0x00000080L
#define CDRVXFER_FLAG_SCRC       0x00000100L
#define CDRVXFER_FLAG_RFIRST     0x00000200L
#define CDRVXFER_FLAG_RNHEX      0x00000400L
#define CDRVXFER_FLAG_RNOREAD    0x00000800L
#define CDRVXFER_FLAG_RCRC       0x00001000L
#define CDRVXFER_FLAG_RZDLE      0x00002000L
#define CDRVXFER_FLAG_RESC       0x00004000L
#define CDRVXFER_FLAG_NEXTFILE   0x00008000L
#define CDRVXFER_FLAG_NORHEAD    0x00010000L
#define CDRVXFER_FLAG_SENDING    0x00020000L
#define CDRVXFER_FLAG_SESC       0x00040000L
#define CDRVXFER_FLAG_SHDX       0x00080000L
#define CDRVXFER_FLAG_SCANDO32   0x00100000L
#define CDRVXFER_FLAG_RECOVER    0x00200000L
#define CDRVXFER_FLAG_OVERWRITE  0x00400000L
#define CDRVXFER_FLAG_RENAME     0x00800000L
#define CDRVXFER_FLAG_SKIPSAME   0x01000000L
#define CDRVXFER_FLAG_RCVACK     0x02000000L
#define CDRVXFER_FLAG_FILEDONE   0x80000000L
unsigned long           flag;
unsigned short          itimer_buf[6];
unsigned short          otimer_buf[6];
unsigned short          ftimer_buf[6];
long                    elapsed;
unsigned char           blockn[2];
unsigned char           crc[2];
unsigned short          ccrc;
unsigned int            flag2;
unsigned int            zconv;
unsigned int            zmanag;
unsigned int            ztrans;
unsigned char           zfe;
unsigned char           tzfe;
unsigned long           ocrc32;
unsigned int            xmtcnt;
unsigned int            xmtidx;
unsigned char FAR       *xmtdat;
unsigned char           xmtopkt[2];
unsigned int            xmtocnt;
unsigned int            xmtoidx;
unsigned char           xmtpkt[20];
unsigned int            xmtpcnt;
unsigned int            xmtpidx;
unsigned char FAR       *xmtpdat;
```

```
unsigned int                    xmtncnt;
unsigned int                    xmtnidx;
unsigned char FAR               *xmtndat;
unsigned int                    txmtcnt;
unsigned int                    txmtidx;
unsigned char                   txmtpkt[20];
unsigned long                   icrc32;
unsigned int                    crcstat;
unsigned int                    lasthdr;
unsigned int                    rcvstate;
unsigned int                    rcvsubstate;
unsigned int                    rcvcancel;
unsigned int                    rcvhtype;
unsigned int                    rcvcnt;
unsigned char FAR               *rcvdat;
unsigned int                    rcvidx;
unsigned char                   rcvpkt[20];
unsigned int                    trcvidx;
unsigned int                    xbufcnt;
unsigned int                    xbufidx;
unsigned char                   xbuf[50];
unsigned int                    rbufcnt;
unsigned int                    rbufidx;
unsigned char                   rbuf[50];
struct cdrvxfer_fileinfo        f;
struct port_param               pcb;
};

/*************************************************************************/
/*======Definition for file transfer dialog box(FileTransferDialog).     */
/*************************************************************************/
#define CDRVXFER_NAME           0x651
#define CDRVXFER_BYTECOUNT      0x652
#define CDRVXFER_ERRORCOUNT     0x653
#define CDRVXFER_RATE           0x654
#define CDRVXFER_FSIZ           0x655

/*************************************************************************/
/*======Definitions for the SetXferParameters() function                 */
/*************************************************************************/
#define CDRVXFER_RCVFILEOPTIONS 0
#define CDRVXFER_PARM_RECOVER   CDRVXFER_FLAG_RECOVER
#define CDRVXFER_PARM_OVERWRITE CDRVXFER_FLAG_OVERWRITE
#define CDRVXFER_PARM_RENAME    CDRVXFER_FLAG_RENAME
#define CDRVXFER_PARM_SKIPSAME  CDRVXFER_FLAG_SKIPSAME
#define CDRVXFER_RCVRETRYCNT    1
#define CDRVXFER_RCVITIMER      2
#define CDRVXFER_RCVOTIMER      3
#define CDRVXFER_XMTFILEOPTIONS 4
#define CDRVXFER_XMTRETRYCNT    5
#define CDRVXFER_XMTITIMER      6
#define CDRVXFER_XMTOTIMER      7
#define CDRVXFER_MONITORCD      8
```

```
/*************************************************************************/
/*=====Definitions for the SetSpecialBehavior() function                 */
/*************************************************************************/
#define SB_SetGetStringDelimiter        0       /* Set String Delimiters */
#define SB_DoNotCall2ndMsgLoop          1       /* No Secondary loop */
#define SB_DisableTiming                2       /* Don't use timer */
#define SB_SwitchOnDelay                3       /* Context switch on delay */

/*************************************************************************/
/*======File Custom file transfer definitions                            */
/*************************************************************************/
#define SER2XFER_ERR_NONE       0       /* No error */
#define SER2XFER_ERR_SERIAL     -1      /* Error opening serial port */
#define SER2XFER_ERR_RFOPEN     -2      /* Remote file open error */
#define SER2XFER_ERR_LFOPEN     -3      /* Local file open error */
#define SER2XFER_ERR_RFREAD     -4      /* Remote file read error */
#define SER2XFER_ERR_LFREAD     -5      /* Local file read error */
#define SER2XFER_ERR_RFWRITE    -6      /* Remote file write error */
#define SER2XFER_ERR_LFWRITE    -7      /* Local file write error */
#define SER2XFER_ERR_TIMEOUT    -8      /* Remote timeout */
#define SER2XFER_ERR_CRC        -9      /* CRC error */
#define SER2XFER_ERR_SYNTAX     -10     /* Syntax error */
#define SER2XFER_ERR_CLEANUP    -11     /* Serial port cleanup failed */
#define SER2XFER_ERR_RFCLOSE    -12     /* Remote file close error */
#define SER2XFER_ERR_LFCLOSE    -13     /* Local file close error */
#define SER2XFER_ERR_BUFOVR     -14     /* Buffer overrun */
#define SER2XFER_ERR_NOFILE     -15     /* No more files */
#define SER2XFER_ERR_DRIVER     -16     /* Error initializing driver */
#define SER2XFER_ERR_SEQ        -17     /* Out of sequence error */

#define SER2XFER_RETRY          3
#define SER2XFER_PACKET_LEN     256-sizeof(PacketH)     /* Packet length */

/*************************************************************************/
/*=====Custom Xfer packet structure                                      */
/*************************************************************************/
typedef struct
     {
     unsigned short  crc;           /* CRC of packet starting at "len" */
     unsigned short  len;           /* Length of data portion of packet */
     unsigned short  seq;           /* Packet sequence number */
     unsigned short  date;          /* File date  */
     unsigned short  time;          /* File time  */
#define SER2XFER_OPEN_READ      0       /* Open file for read slave command */
#define SER2XFER_OPEN_WRITE     1       /* Open file for write slave command
*/
#define SER2XFER_CLOSE          2       /* Close file slave command */
#define SER2XFER_READ           3       /* Read file slave command */
#define SER2XFER_WRITE          4       /* Write file slave command */
#define SER2XFER_FINDFIRST      5       /* Find first matching file slave
cmd*/
#define SER2XFER_FINDNEXT       6       /* Find next matching file slave cmd*/
     unsigned short  cmd;           /* Slave command variable */
     short           stat;          /* Status of slave command */
```

```
    } PacketH;

typedef struct
{
PacketH         head;                           /* Packet header */
unsigned char   data[SER2XFER_PACKET_LEN];      /* Packet data buffer */
} Packet;

/*****************************************************************************/
/*=====COMMDRV screen sub-function equates                                    */
/*****************************************************************************/
#define COMMDRVW_CHARSCREENINIT         0
#define COMMDRVW_CHARSCREENRESIZE       1
#define COMMDRVW_CHARSCREENWRITE        2
#define COMMDRVW_CHARSCREENPAINT        3
#define COMMDRVW_CHARSCREENVSCROLL      4
#define COMMDRVW_CHARSCREENHSCROLL      5
#define COMMDRVW_CHARSCREENSETFOCUS     6
#define COMMDRVW_CHARSCREENKILLFOCUS    7
#define COMMDRVW_CHARSCREENCLEANUP      8

#define COMMDRVW_MAXROW                 25
#define COMMDRVW_MAXCOL                 80

#ifdef  MSWIN
/*****************************************************************/
/*=====CDRVRMAP & WINCDRV definitions                             */
/*****************************************************************/
/* Number of ports reserverd for old comm.drv */
#define OLDCNT  4

/* Menu items */
#define CDRVRMAP_MENU_ABOUT     (WM_USER+1)
#define CDRVRMAP_MENU_REMAP     (WM_USER+2)

#define CDRVRMAP_CID_UP         (WM_USER+50)
#define CDRVRMAP_CID_DOWN       (WM_USER+51)
#define CDRVRMAP_CID_TEXT       (WM_USER+52)
#define CDRVRMAP_DOIT           (WM_USER+53)
#define CDRVRMAP_PORT_UP        (WM_USER+54)
#define CDRVRMAP_PORT_DOWN      (WM_USER+55)
#define CDRVRMAP_PORT_TEXT      (WM_USER+56)
/*****************************************************************/
#endif

/*****************************************************************/
/*****************************************************************/
/*=====COMMDRV Specific VxD(WCSCCDRV.386)                         */
/*****************************************************************/
/*****************************************************************/
#define WCSCCDRV_Dev_ID         0x3229          /* Device ID */

#define VACTIVE_FLG             0x00000001L     /* */
#define VDISABLE_MODEMSIG       0x00000002L     /* */
```

```
#define VIGNORE_16550           0x00000004L     /* */
#define VNOCHANGE_MODEMSIG      0x00000008L     /* */
#define VLEAVE_RTS              0x00000010L     /* */
#define VLEAVE_DTR              0x00000020L     /* */
#define VXMTOFF_STATE           0x00000040L     /* */
#define VRCVOFF_STATE           0x00000080L     /* */
#define VSEND_XON               0x00000100L     /* */
#define VSEND_XOFF              0x00000200L     /* */
#define VSTOREALL               0x00000400L     /* */
#define VPARENTPORT             0x00000800L     /* */
#define VNINEBITTOGGLE          0x00001000L     /* */
#define VNINEBITTOGGLE2         0x00002000L     /* */
#define VNINEBITPROTOCOL        0x00004000L     /* */
#define VMULTIDROP              0x00008000L     /* */
#define VTIMESTAMPBYTES         0x00010000L     /* */
#define VLSRINT                 0x00020000L     /* */
#define VRCVINT                 0x00040000L     /* */
#define VXMTINT                 0x00080000L     /* */
#define VMSRINT                 0x00100000L     /* */

/*****************************************************************************/
/*=====VxD Port Control Structure                                             */
/*      (VPCBIdx - VTotalIn) == Portion returned for partial                 */
/*      structure read.                                                      */
/*****************************************************************************/
#ifdef MSNTDRV
#pragma pack(push)
#pragma pack(1)
#endif
struct vxd_port_param
{
unsigned long   VTotalIn;       /*Number of characters received.        */
unsigned long   VTotalOut;      /*Number of characters transmitted.     */
unsigned long   VInBufCnt;      /*Input buffer count                    */
unsigned long   VOutBufCnt;     /*Output buffer count                   */
unsigned long   VIRingSize;     /*Size of input buffer                  */
unsigned long   VORingSize;     /*Size of output buffer                 */
unsigned long   VMSRIntCnt;     /*Number of MSR ints                    */
unsigned long   VLSRIntCnt;     /*Number of LSR ints                    */
unsigned long   VXmtIntCnt;     /*Number of XMT ints                    */
unsigned long   VRcvIntCnt;     /*Number of RCV ints                    */
unsigned long   VRcvLostCnt;    /*Number of received characters lost.   */
unsigned long   VXmtLostCnt;    /*Number of transmit characters lost.   */
unsigned long   VOFlushCnt;     /*Output flush count                    */
unsigned long   VIFlushCnt;     /*Input flush count                     */
unsigned char   VMSRReg;        /*Current modem status register         */
unsigned char   VLSRReg;        /*Current line status register          */
unsigned char   VTypeUART;      /*Type of UART                          */
unsigned char   VFillerb1;      /*Filler                                */
unsigned short  VPort;          /*COMM-DRV port number                  */
unsigned short  VBase;          /*Base address                          */
unsigned short  VIrq;           /*IRQ                                   */
unsigned short  VBaud;          /*Baud rate                             */
unsigned short  VBreak;         /*Break                                 */
```

```
unsigned short  VLength;        /*Length                                    */
unsigned short  VParity;        /*Parity                                    */
unsigned short  VStopbit;       /*Stop bits                                 */
unsigned short  VProtocol;      /*Protocol                                  */
unsigned short  VCardseg;       /*Card segment                              */
unsigned char   VCharXon;       /*XON character to expect from remote       */
unsigned char   VCharXoff;      /*XOFF character to expect from remote      */
unsigned char   VCharXxon;      /*XON character to send to remote           */
unsigned char   VCharXxoff;     /*XOFF character to send to remote          */
unsigned long   VFlag;          /*Flag                                      */
unsigned long   VCurOverrunCnt; /*Ovrrun err cnt                            */
unsigned long   VCurParityCnt;  /*Parity err cnt                            */
unsigned long   VCurFramingCnt; /*Framing err cnt                           */
unsigned long   VCurBreakCnt;   /*Break err cnt                             */
unsigned long   VBytesRead;     /*Number of bytes read from VxD             */
unsigned long   VBytesWrite;    /*Number of bytes written to VxD            */
unsigned long   VXonIn;         /*Number of XON received                    */
unsigned long   VXoffIn;        /*Number of XOFF received                   */
unsigned long   VXonOut;        /*Number of XON sent                        */
unsigned long   VXoffOut;       /*Number of XOFF sent                       */
unsigned long   VFillerb2[10];  /*Room for growth                           */

unsigned short  VPCBIdx;        /*Index to this PCB                         */
unsigned short  VXmtCnt;        /*Number of bytes to xmit at once           */
unsigned long   VORingTop;      /*Start of output buffer ring               */
unsigned long   VORingBottom;   /*Bottom of output buffer ring+1            */
unsigned long   VORingInput;    /*Current input pointer to output buf       */
unsigned long   VORingOutput;   /*Current output pointer to output buf      */
unsigned long   VIRingTop;      /*Start of input buffer ring                */
unsigned long   VIRingBottom;   /*Bottom of input buffer ring+1             */
unsigned long   VIRingInput;    /*Current input pointer to input buf        */
unsigned long   VIRingOutput;   /*Current output pointer to input buf       */
unsigned long   VProtLocal;     /*Local protocol index                      */
unsigned long   VProtRemote;    /*Remote protocol index                     */
unsigned long   VInbufHigh;     /*Input buffer high threshold               */
unsigned long   VInbufLow;      /*Input buffer high threshold               */
unsigned long   VNextPCBPtr;    /*Pointer to next PCB                       */
unsigned char   VOldBaudReg[2]; /*Old Baud Rate Register                    */
unsigned char   VOldMCRReg;     /*Old Modem Control Register                */
unsigned char   VOldLCRReg;     /*Old Line Control Register                 */
unsigned char   VOldFCRReg;     /*Old FIFO Control Register                 */
unsigned char   VOldIERReg;     /*Old Interrupt Enable Register             */
unsigned char   VFillerb3[2];   /*Filler(Complete 32bit boundary)           */
unsigned long   VCtsOn;         /*# of Cts On transition                    */
unsigned long   VCtsOff;        /*# of Cts Off transition                   */
unsigned long   VCdOn;          /*# of Cd On transition                     */
unsigned long   VCdOff;         /*# of Cd Off transition                    */
unsigned long   VRiOn;          /*# of Ri On transition                     */
unsigned long   VRiOff;         /*# of Ri Off transition                    */
unsigned long   VDsrOn;         /*# of Dsr On transition                    */
unsigned long   VDsrOff;        /*# of Dsr Off transition                   */
unsigned long   VFillerb4[2];   /*Room for growth                           */
unsigned long   VPPCB;          /*Parent PCB                                */
};
```

```
#ifdef MSNTDRV
#pragma pack(pop)
#endif

/*****************************************************************************/
/*=====Index for WCSCCDRV.386 API functions                                   */
/*****************************************************************************/
#define VXDAPI_NT_IOCTL_OFFSET  0x800   /* */
#define VXDAPI_Version          0       /* */
#define VXDAPI_Setup            1       /* */
#define VXDAPI_GetPacket        2       /* */
#define VXDAPI_PutPacket        3       /* */
#define VXDAPI_Cleanup          4       /* */
#define VXDAPI_ViewPacket       5       /* */
#define VXDAPI_Flush            6       /* */
#define VXDAPI_GetPort          7       /* */
#define VXDAPI_GetMetric        8       /* */
#define VXDAPI_ModemCtl         9       /* */
#define VXDAPI_BaudDiv          10      /* */
#define VXDAPI_UARTReg          11      /* */

#ifdef  MSWIN32
#define ICTL1   FILE_DEVICE_UNKNOWN
#define ICTL2   VXDAPI_NT_IOCTL_OFFSET
#define ICTL3   METHOD_BUFFERED
#define ICTL4   FILE_ANY_ACCESS
#define ICTL(i) CTL_CODE((ICTL1),(ICTL2+i),(ICTL3),(ICTL4))

#define VXDAPI_Version_NT
CTL_CODE(FILE_DEVICE_UNKNOWN,0x800,METHOD_BUFFERED,FILE_ANY_ACCESS)
#define VXDAPI_Setup_NT
CTL_CODE(FILE_DEVICE_UNKNOWN,0x801,METHOD_BUFFERED,FILE_ANY_ACCESS)
#define VXDAPI_GetPacket_NT
CTL_CODE(FILE_DEVICE_UNKNOWN,0x802,METHOD_BUFFERED,FILE_ANY_ACCESS)
#define VXDAPI_PutPacket_NT
CTL_CODE(FILE_DEVICE_UNKNOWN,0x803,METHOD_BUFFERED,FILE_ANY_ACCESS)
#define VXDAPI_Cleanup_NT
CTL_CODE(FILE_DEVICE_UNKNOWN,0x804,METHOD_BUFFERED,FILE_ANY_ACCESS)
#define VXDAPI_ViewPacket_NT
CTL_CODE(FILE_DEVICE_UNKNOWN,0x805,METHOD_BUFFERED,FILE_ANY_ACCESS)
#define VXDAPI_Flush_NT
CTL_CODE(FILE_DEVICE_UNKNOWN,0x806,METHOD_BUFFERED,FILE_ANY_ACCESS)
#define VXDAPI_GetPort_NT
CTL_CODE(FILE_DEVICE_UNKNOWN,0x807,METHOD_BUFFERED,FILE_ANY_ACCESS)
#define VXDAPI_GetMetric_NT
CTL_CODE(FILE_DEVICE_UNKNOWN,0x808,METHOD_BUFFERED,FILE_ANY_ACCESS)
#define VXDAPI_ModemCtl_NT
CTL_CODE(FILE_DEVICE_UNKNOWN,0x809,METHOD_BUFFERED,FILE_ANY_ACCESS)
#define VXDAPI_BaudDiv_NT
CTL_CODE(FILE_DEVICE_UNKNOWN,0x80a,METHOD_BUFFERED,FILE_ANY_ACCESS)
#define VXDAPI_UARTReg_NT
CTL_CODE(FILE_DEVICE_UNKNOWN,0x80b,METHOD_BUFFERED,FILE_ANY_ACCESS)
#endif
```

```
/*************************************************************************/
/*************************************************************************/
/*=====External Declarations                                              */
/*************************************************************************/
/*************************************************************************/
#ifndef MSNTDRV
#ifdef  MSWIN
extern HINSTANCE        cdrvxf_hInst;
#endif

#ifdef  MSWIN32
extern CRITICAL_SECTION CSection;
#endif
#endif

/* COMM-DRV System Software Externals */
extern struct ser_rs232_routines FAR    *ser_rs232;
extern struct commdrv_ctl               *commdrv_ctl;
extern char                             commdrv_version[];
extern char FAR                         cdrvpath[];

/* File Transfer Externals */
extern unsigned char    char_ack[];
extern unsigned char    char_c[];
extern unsigned char    char_can[];
extern unsigned char    char_eot[];
extern unsigned char    char_g[];
extern unsigned char    char_nak[];
extern unsigned char    *cdrvxfer_htab;
extern int              cdrvxfer_debug;
extern int              ser2xfer_sflag;
extern unsigned long    cdrvxfer_rcvflag;
extern int              cdrvxfer_rcvretrycnt;
extern int              cdrvxfer_rcvrcvtimeout;
extern int              cdrvxfer_rcvxmttimeout;
extern unsigned long    cdrvxfer_xmtflag;
extern int              cdrvxfer_xmtretrycnt;
extern int              cdrvxfer_xmtrcvtimeout;
extern int              cdrvxfer_xmtxmttimeout;
extern int              cdrvxfer_monitorcd;

/* High Level Function Externals */
extern int      modem_pacetime;
extern int      modem_pluspacetime;
extern int      modem_waitresponse;
extern int      modem_waitconnect;
extern int      modem_waitecho;
extern int      modem_waitcommand;
extern int      modem_waitdtr;
extern char     *modem_cmdstate;
extern char     *modem_attention;
extern char     *modem_hangup;
extern char     *modem_echoon;
extern char     *modem_dialpulse;
```

```
extern char     *modem_dialtone;
extern char     *modem_ok;

#ifdef  CDRVDYNA
#define AUXPCBPTR(pcb)  (((struct aux_pparam FAR
*)((((long)(FP_SEG((pcb))))<<16)+FP_OFF((pcb)->auxpcb))))
#else
#define AUXPCBPTR(pcb)  (pcb->auxpcb)
#endif

/*************************************************************************/
/*************************************************************************/
/*=====Function prototypes                                               */
/*************************************************************************/
/*************************************************************************/

/*************************************************************************/
/*=====Function prototypes of the COMM-DRV low level functions.          */
/*************************************************************************/
/*     If dynamically linking to a WINDOWS DLL then                      */
/*     MSWIN & MSWINDLL must be defined before COMM.H is                 */
/*     included for WIN32 apps MSWIN, MSWINDLL, and MSWIN32              */
/*      must be defined before COMM.H.                                   */
/*                                                                       */
/*      If dynamically linking to the MS-DOS TSR then                    */
/*      COMMDRV_DRIVER must be defined before COMM.H is                  */
/*      included.                                                        */
/*************************************************************************/
#ifdef  COMMDRV_DRIVER
extern int CDECL ser_rs232_init(void);
#define ser_rs232_cleanup(p)            ser_rs232->cleanup((p))
#define ser_rs232_getbyte(p,c)          ser_rs232->getbyte((p),(c))
#define ser_rs232_getport(p,pc)         ser_rs232->getport((p),(pc))
#define ser_rs232_getregister(p,o,v)    ser_rs232->getregister((p),(o),(v))
#define ser_rs232_getstatus(p,s)        ser_rs232->getstatus((p),(s))
#define ser_rs232_putbyte(p,c)          ser_rs232->putbyte((p),(c))
#define ser_rs232_putregister(p,o,v)    ser_rs232->putregister((p),(o),(v))
#define ser_rs232_maxport()             ser_rs232->maxport()
#define ser_rs232_setup(p,pc)           ser_rs232->setup((p),(pc))
#define ser_rs232_putpacket(p,c,ch)     ser_rs232->putpacket((p),(c),(ch))
#define ser_rs232_getpacket(p,c,ch)     ser_rs232->getpacket((p),(c),(ch))
#define ser_rs232_flush(p,a)            ser_rs232->flush((p),(a))
#define ser_rs232_block(p,i,o)          ser_rs232->block((p),(i),(o))
#define ser_rs232_dtr_off(p)            ser_rs232->dtr_off((p))
#define ser_rs232_dtr_on(p)             ser_rs232->dtr_on((p))
#define ser_rs232_rts_off(p)            ser_rs232->rts_off((p))
#define ser_rs232_rts_on(p)             ser_rs232->rts_on((p))
#define ser_rs232_set_intfunc(p,q,r)    ser_rs232->set_intfunc((p),(q),(r))
#define ser_rs232_get_sdata()           ser_rs232->get_sdata()
#define ser_rs232_misc_func(p,c,v)      ser_rs232->misc_func((p),(c),(v))
#define ser_rs232_setbauddiv(p,q,r)     ser_rs232->setbauddiv((p),(q),(r))
#define ser_rs232_viewpacket(p,q,r)     ser_rs232->viewpacket((p),(q),(r))
#else
#ifdef  COMMDRV_ALL
```

```
extern int CDECL ser_rs232_init(void);
#define ser_rs232_cleanup(p)            ser_cleanup((p))
#define ser_rs232_getbyte(p,c)          ser_getbyte((p),(c))
#define ser_rs232_getport(p,pc)         ser_getport((p),(pc))
#define ser_rs232_getregister(p,o,v)    ser_getregister((p),(o),(v))
#define ser_rs232_getstatus(p,s)        ser_getstatus((p),(s))
#define ser_rs232_putbyte(p,c)          ser_putbyte((p),(c))
#define ser_rs232_putregister(p,o,v)    ser_putregister((p),(o),(v))
#define ser_rs232_maxport()             ser_maxport()
#define ser_rs232_setup(p,pc)           ser_setup((p),(pc))
#define ser_rs232_putpacket(p,c,ch)     ser_putpacket((p),(c),(ch))
#define ser_rs232_getpacket(p,c,ch)     ser_getpacket((p),(c),(ch))
#define ser_rs232_flush(p,a)            ser_flush((p),(a))
#define ser_rs232_block(p,i,o)          ser_block((p),(i),(o))
#define ser_rs232_dtr_off(p)            ser_dtr_off((p))
#define ser_rs232_dtr_on(p)             ser_dtr_on((p))
#define ser_rs232_rts_off(p).           ser_rts_off((p))
#define ser_rs232_rts_on(p)             ser_rts_on((p))
#define ser_rs232_set_intfunc(p,q,r)    ser_set_intfunc((p),(q),(r))
#define ser_rs232_get_sdata()           ser_get_sdata()
#define ser_rs232_misc_func(p,c,v)      ser_misc_func((p),(c),(v))
#define ser_rs232_setbauddiv(p,q,r)     ser_setbauddiv((p),(q),(r))
#define ser_rs232_viewpacket(p,q,r)     ser_viewpacket((p),(q),(r))
#else
#ifdef  MSWINDLL
int CDECL ser_rs232_init(void);
int FARPAS ser_rs232_cleanup(int port);
int FARPAS ser_rs232_getbyte(int port,unsigned char FAR *ch);
int FARPAS ser_rs232_getport(int port,struct port_param FAR *pcb);
int FARPAS ser_rs232_getregister(int port,int offset,int FAR *value);
int FARPAS ser_rs232_getstatus(int port,unsigned char FAR *stat);
int FARPAS ser_rs232_putbyte(int port,unsigned char FAR *ch);
int FARPAS ser_rs232_putregister(int port,int offset,int value);
int FARPAS ser_rs232_maxport(void);
int FARPAS ser_rs232_setup(int port,struct port_param FAR *pcb);
int FARPAS ser_rs232_putpacket(int port,int count,
           unsigned char FAR *ch);
int FARPAS ser_rs232_getpacket(int port,int count,
           unsigned char FAR *ch);
int FARPAS ser_rs232_flush(int port,int arg);
int FARPAS ser_rs232_block(int port,int input,int output);
int FARPAS ser_rs232_dtr_off(int port);
int FARPAS ser_rs232_dtr_on(int port);
int FARPAS ser_rs232_rts_off(int port);
int FARPAS ser_rs232_rts_on(int port);
int FARPAS ser_rs232_set_intfunc(int port,
            CDRVIFNC (FAR *(FAR * func))(short val,
            struct port_param FAR *p),short imask);
struct ser_rs232_sdata FAR * FARPAS ser_rs232_get_sdata(void);
int FARPAS ser_rs232_misc_func(int port,unsigned short cmd,
            unsigned long val);
int FARPAS ser_rs232_setbauddiv(int cardtype,int baudidx,
           unsigned bauddiv);
int FARPAS ser_rs232_viewpacket(int port,int count,
```

```
            unsigned char FAR *ch);


/* WINCDRV.DRV Specific */
int FARPAS GetCOMMDRVPort(int cid);
int FARPAS GetCOMMDRVPorts(int idx);
int FARPAS SetCOMMDRVPort(int cid,int port);
#else
int CDECL ser_rs232_init(void);
int FARC ser_rs232_cleanup(int port);
int FARC ser_rs232_getbyte(int port,unsigned char FAR *ch);
int FARC ser_rs232_getport(int port,struct port_param FAR *pcb);
int FARC ser_rs232_getregister(int port,int offset,int FAR *value);
int FARC ser_rs232_getstatus(int port,unsigned char FAR *stat);
int FARC ser_rs232_putbyte(int port,unsigned char FAR *ch);
int FARC ser_rs232_putregister(int port,int offset,int value);
int FARC ser_rs232_maxport(void);
int FARC ser_rs232_setup(int port,struct port_param FAR *pcb);
int FARC ser_rs232_putpacket(int port,int count,unsigned char FAR *ch);
int FARC ser_rs232_getpacket(int port,int count,unsigned char FAR *ch);
int FARC ser_rs232_flush(int port,int arg);
int FARC ser_rs232_block(int port,int input,int output);
int FARC ser_rs232_dtr_off(int port);
int FARC ser_rs232_dtr_on(int port);
int FARC ser_rs232_rts_off(int port);
int FARC ser_rs232_rts_on(int port);
int FARC ser_rs232_set_intfunc(int port,
          CDRVIFNC (FAR *(FAR *func))(short val,
          struct port_param FAR *p),short imask);
struct ser_rs232_sdata FAR * FARC ser_rs232_get_sdata(void);
int FARC ser_rs232_misc_func(int port,unsigned short cmd,unsigned long val);
int FARC ser_rs232_setbauddiv(int cardtype,int baudidx,unsigned bauddiv);
int FARC ser_rs232_viewpacket(int port,int count,unsigned char FAR *ch);
#endif
#endif
#endif

/*************************************************************************/
/*=====Interrupt routines prototypes                                     */
/*************************************************************************/
int FAR ser_rs232_ifnc_getpacket(struct port_param FAR *p,
char FAR *buf,int cnt);
int FAR ser_rs232_ifnc_putpacket(struct port_param FAR *p,
char FAR *buf,int cnt);
int FAR ser_rs232_ifnc_dtr_off(struct port_param FAR *p);
int FAR ser_rs232_ifnc_dtr_on(struct port_param FAR *p);
int FAR ser_rs232_ifnc_rts_off(struct port_param FAR *p);
int FAR ser_rs232_ifnc_rts_on(struct port_param FAR *p);

/*************************************************************************/
/*=====High level COMM-DRV function prototypes                           */
/*************************************************************************/
#ifdef  MSWINDLL
int FARPAS BytesInReceiveBuffer(int port);
```

```
int FARPAS BytesInTransmitBuffer(int port);
int FARPAS CdrvCheckTime(unsigned short FAR *timerblk);
unsigned short FARPAS CdrvCrc16(int count,char FAR *buf);
unsigned long FARPAS CdrvCrc32(int count,char FAR *buf);
int FARPAS CdrvDelay(int port,int timeout);
struct port_param FAR * FARPAS CdrvGetPcb(int port);
int FARPAS CdrvSetTime(int port,int timeout,unsigned short FAR *timerblk);
int FARPAS CdrvSetTimeoutFunction(int port,int (FAR *func)(int port));
int FARPAS CdrvSetTimerResolution(int port,int resolution);
void FARPAS cdrvxfer_gclose(struct cdrvxfer_ftpcb FAR *p);
int FARPAS cdrvxfer_getfiles(int mode,struct cdrvxfer_ftpcb FAR *p);
int FARPAS cdrvxfer_files(int port,int direction,int protocol,
            char FAR *fspec,int (FARC *fnc)(int chgflg,char FAR *fname,
            long bytcnt,int curblk,int errcnt,unsigned rate));
int FARPAS cdrvxfer_sendfiles(int mode,struct cdrvxfer_ftpcb FAR *p);
int FARPAS cdrvxfer_sfiles(int port,int direction,int protocol,
                     char FAR *fspec,int (FAR *fnc)(int chgflg,
                     char FAR *fname,long bytcnt,int curblk,
                     int errcnt,unsigned rate,
                     struct cdrvxfer_ftpcb FAR *ftpcb));
int FARPAS CdrvXferCreateDialog(struct cdrvxfer_ftpcb FAR *p);
int FARPAS CdrvXferDestroyDialog(struct cdrvxfer_ftpcb FAR *p);
int FARPAS CdrvXferUpdateDialog(struct cdrvxfer_ftpcb FAR *p);
int FARPAS DataStreamGetByte(int port,unsigned char FAR *ch);
int FARPAS DataStreamGetPacket(int port,int count,unsigned char FAR *ch);
int FARPAS Dial(int port,int mode,char FAR *telephone);
int FARPAS DtrOff(int port);
int FARPAS DtrOn(int port);
int FARPAS FileTransferDialog(int port,int mode);
int FARPAS FlushReceiveBuffer(int port);
int FARPAS FlushTransmitBuffer(int port);
int FARPAS GetByte(int port);
int FARPAS GetNumber(int port,int count,char FAR *str);
unsigned int FARPAS GetPaceTime(int port);
int FARPAS GetPacket(int port,int len,char FAR *pkt);
int FARPAS GetString(int port,int len,char FAR *str);
unsigned int FARPAS GetTimeout(int port);
int FARPAS InitializePort(int port,int subport,unsigned short addr,
            unsigned short irq,unsigned short cardtype,
            unsigned short cardseg,CDRVUINT inbuflen,
            CDRVUINT outbuflen,CDRVUINT flag);
int FARPAS IsAllDataOut(int port);
int FARPAS IsBreak(int port);
int FARPAS IsCarrierDetect(int port);
int FARPAS IsCts(int port);
int FARPAS IsDsr(int port);
int FARPAS IsFramingError(int port);
int FARPAS IsInputOverrun(int port);
int FARPAS IsOverrunError(int port);
int FARPAS IsParityError(int port);
int FARPAS IsPortAvailable(int port);
int FARPAS IsReceiveBufferEmpty(int port);
int FARPAS IsRing(int port);
int FARPAS IsTransmitBufferEmpty(int port);
```

```
int FARPAS ModemAttention(int port);
int FARPAS ModemAnswerMode(int port,int count);
int FARPAS ModemCommandState(int port);
int FARPAS ModemConnect(int port);
int FARPAS ModemForceAnswer(int port);
long FARPAS ModemGetCarrierSpeed(int port);
long FARPAS ModemGetConnectSpeed(int port);
int FARPAS ModemGetSRegister(int port,int SReg);
char FAR * FARPAS ModemGetString(int port,int code);
long FARPAS ModemGetValue(int port,int code);
int FARPAS ModemHangup(int port);
int FARPAS ModemInit(int port);
int FARPAS ModemModifyString(int port,int code,char FAR *str);
int FARPAS ModemModifyValue(int port,int code,long val);
int FARPAS ModemOffHook(int port);
int FARPAS ModemOnline(int port);
int FARPAS ModemSendCommand(int port,char FAR *cmd);
int FARPAS ModemSetSRegister(int port,int SReg, int val);
int FARPAS ModemSpeaker(int port, int OnOff);
int FARPAS ModemVolume(int port, int lev);
int FARPAS ModemWaitForCall(int port,int RingCnt,int Mode,
                  int (FARC *fnc)(int port));
int FARPAS ModemWaitForRing(int port, int cnt,int timeout);
int FARPAS PeekChar(int port);
int FARPAS PutByte(int port,char ch);
int FARPAS PutPacket(int port,int len,char FAR *pkt);
int FARPAS PutString(int port,char FAR *str);
int FARPAS ReceiveBufferSize(int port);
unsigned long FARPAS CdrvReturnStringAddress(void FAR * adr);
int FARPAS RtsOff(int port);
int FARPAS RtsOn(int port);
int FARPAS SendBreak(int port,int timeval);
int FARPAS SetBaud(int port,int baud);
int FARPAS SetDataStreamFunction(int port,void  (FARC *fnc)(int p,
                    int c, char FAR *h));
int FARPAS SetFlowControlCharacters(int port,int xoff,int xon,
            int xxon,int xxoff);
int FARPAS SetFlowControlThreshold(int port,int inbuf_low,int inbuf_high);
int FARPAS SetPaceTime(int port,int timeval);
int FARPAS SetPortCharacteristics(int port,CDRVUINT baud,
            unsigned short parity,unsigned short length,
            unsigned short stopbit, unsigned short protocol);
int FARPAS SetSpecialBehavior(int port,int cmd,unsigned long val1,
                    unsigned long val2);
int FARPAS SetTimeout(int port,int timeout);
int FARPAS SetXferParameters(unsigned int command,unsigned long value);
int FARPAS SpaceInReceiveBuffer(int port);
int FARPAS SpaceInTransmitBuffer(int port);
int FARPAS SpaceTransmitBuffer(int port);
int FARPAS TransferFiles(int cmd,int port,int direction,int protocol,
          char FAR *fspec, struct cdrvxfer_ftpcb FAR *p);
int FARPAS TransmitBufferSize(int port);
int FARPAS UnInitializePort(int port);
int FARPAS WaitFor(int port,int timeout,char FAR *out,char FAR *in);
```

```
int FARPAS WaitForFixed(int port,int timeout,char FAR *out,
                char FAR *in,int olen,int ilen);
int FARPAS WaitForPeek(int port,int timeout,char FAR *out,char FAR *in);
int FARPAS WaitForPeekFixed(int port,int timeout,char FAR *out,
                    char FAR *in,int olen,int ilen);
int FARPAS WaitForPeekTable(int port,int timeout,char FAR *out,
          char FAR * FAR *in);
int FARPAS WaitForPeekTableFixed(int port,int timeout,char FAR *out,
                      char FAR * FAR *in,int olen,int FAR *ilen);
int FARPAS WaitForTable(int port,int timeout,char FAR *out,char FAR * FAR
*in);
int FARPAS WaitForTableFixed(int port,int timeout,char FAR *out,
                    char FAR * FAR *in,int olen,int FAR *ilen);
#else
int CDECL BytesInReceiveBuffer(int port);
int CDECL BytesInTransmitBuffer(int port);
int CDECL CdrvCheckTime(unsigned short FAR *timerblk);
unsigned short CDECL CdrvCrc16(int count,char FAR *buf);
unsigned long CDECL CdrvCrc32(int count,char FAR *buf);
int CDECL CdrvDelay(int port,int timeout);
struct port_param FAR * CDECL CdrvGetPcb(int port);
int CDECL CdrvSetTime(int port,int timeout,unsigned short FAR *timerblk);
int CDECL CdrvSetTimeoutFunction(int port,int (FAR *func)(int port));
int CDECL CdrvSetTimerResolution(int port,int resolution);
void CDECL cdrvxfer_gclose(struct cdrvxfer_ftpcb FAR *p);
int CDECL cdrvxfer_getfiles(int mode,struct cdrvxfer_ftpcb FAR *p);
int CDECL cdrvxfer_files(int port,int direction,int protocol,
    char FAR *fspec,int (FAR *fnc)(int chgflg,char FAR *fname,long bytcnt,
    int curblk,int errcnt,unsigned rate));
int CDECL cdrvxfer_sendfiles(int mode,struct cdrvxfer_ftpcb FAR *p);
int CDECL cdrvxfer_sfiles(int port,int direction,int protocol,char FAR *fspec,
            int (FAR *fnc)(int chgflg,char FAR *fname,long bytcnt,
            int curblk,int errcnt,unsigned rate,
            struct cdrvxfer_ftpcb FAR *ftpcb));
int CDECL CdrvXferCreateDialog(struct cdrvxfer_ftpcb FAR *p);
int CDECL CdrvXferDestroyDialog(struct cdrvxfer_ftpcb FAR *p);
int CDECL CdrvXferUpdateDialog(struct cdrvxfer_ftpcb FAR *p);
int CDECL DataStreamGetByte(int port,unsigned char FAR *ch);
int CDECL DataStreamGetPacket(int port,int count,unsigned char FAR *ch);
int CDECL Dial(int port,int mode,char FAR *telephone);
int CDECL DtrOff(int port);
int CDECL DtrOn(int port);
int CDECL FileTransferDialog(int port,int mode);
int CDECL FlushReceiveBuffer(int port);
int CDECL FlushTransmitBuffer(int port);
int CDECL GetByte(int port);
int CDECL GetNumber(int port,int count,char FAR *str);
unsigned int CDECL GetPaceTime(int port);
int CDECL GetPacket(int port,int len,char FAR *pkt);
int CDECL GetString(int port,int len,char FAR *str);
unsigned int GetTimeout(int port);
int CDECL InitializePort(int port,int subport,unsigned short addr,
    unsigned short irq,unsigned short cardtype,unsigned short cardseg,
    CDRVUINT inbuflen,CDRVUINT outbuflen,CDRVUINT flag);
```

```
int CDECL IsAllDataOut(int port);
int CDECL IsBreak(int port);
int CDECL IsCarrierDetect(int port);
int CDECL IsCts(int port);
int CDECL IsDsr(int port);
int CDECL IsFramingError(int port);
int CDECL IsInputOverrun(int port);
int CDECL IsOverrunError(int port);
int CDECL IsParityError(int port);
int CDECL IsPortAvailable(int port);
int CDECL IsReceiveBufferEmpty(int port);
int CDECL IsRing(int port);
int CDECL IsTransmitBufferEmpty(int port);
int CDECL ModemAttention(int port);
int CDECL ModemAnswerMode(int port,int count);
int CDECL ModemCommandState(int port);
int CDECL ModemConnect(int port);
long CDECL ModemGetCarrierSpeed(int port);
long CDECL ModemGetConnectSpeed(int port);
int CDECL ModemForceAnswer(int port);
int CDECL ModemGetSRegister(int port,int SReg);
char FAR * CDECL ModemGetString(int port,int code);
long CDECL ModemGetValue(int port,int code);
int CDECL ModemHangup(int port);
int CDECL ModemInit(int port);
int CDECL ModemModifyString(int port,int code,char FAR *str);
int CDECL ModemModifyValue(int port,int code,long val);
int CDECL ModemOffHook(int port);
int CDECL ModemOnline(int port);
int CDECL ModemSendCommand(int port,char FAR *cmd);
int CDECL ModemSetSRegister(int port,int SReg, int val);
int CDECL ModemSpeaker(int port, int OnOff);
int CDECL ModemVolume(int port, int lev);
int CDECL ModemWaitForCall(int port,int RingCnt,int Mode,
int (FARC *fnc)(int port));
int CDECL ModemWaitForRing(int port, int cnt,int timeout);
int CDECL PeekChar(int port);
int CDECL PutByte(int port,char ch);
int CDECL PutPacket(int port,int len,char FAR *pkt);
int CDECL PutString(int port,char FAR *str);
int CDECL ReceiveBufferSize(int port);
unsigned long CDECL CdrvReturnStringAddress(void FAR * adr);
int CDECL RtsOff(int port);
int CDECL RtsOn(int port);
int CDECL SendBreak(int port,int timeval);
int CDECL SetBaud(int port,int baud);
int CDECL SetDataStreamFunction(int port,void   (FARC *fnc)(int p,
                    int c, char FAR *h));
int CDECL SetFlowControlCharacters(int port,int xoff,int xon,
     int xxon,int xxoff);
int CDECL SetFlowControlThreshold(int port,int inbuf_low,int inbuf_high);
int CDECL SetPaceTime(int port,int timeval);
int CDECL SetPortCharacteristics(int port,CDRVUINT baud,
     unsigned short parity,unsigned short length,unsigned short stopbit,
```

```
     unsigned short protocol);
int CDECL SetSpecialBehavior(int port,int cmd,unsigned long val1,
                    unsigned long val2);
int CDECL SetTimeout(int port,int timeout);
int CDECL SetXferParameters(unsigned int command,unsigned long value);
int CDECL SpaceInReceiveBuffer(int port);
int CDECL SpaceInTransmitBuffer(int port);
int CDECL SpaceTransmitBuffer(int port);
int CDECL TransferFiles(int cmd,int port,int direction,int protocol,
    char FAR *fspec, struct cdrvxfer_ftpcb FAR *p);
int CDECL TransmitBufferSize(int port);
int CDECL UnInitializePort(int port);
int CDECL WaitFor(int port,int timeout,char FAR *out,char FAR *in);
int CDECL WaitForFixed(int port,int timeout,char FAR *out,
                  char FAR *in,int olen,int ilen);
int CDECL WaitForPeek(int port,int timeout,char FAR *out,char FAR *in);
int CDECL WaitForPeekFixed(int port,int timeout,char FAR *out,
                   char FAR *in,int olen,int ilen);
int CDECL WaitForPeekTable(int port,int timeout,char FAR *out,
                   char FAR * FAR *in);
int CDECL WaitForPeekTableFixed(int port,int timeout,char FAR *out,
                    char FAR * FAR *in,int olen,int FAR *ilen);
int CDECL WaitForTable(int port,int timeout,char FAR *out,char FAR * FAR *in);
int CDECL WaitForTableFixed(int port,int timeout,char FAR *out,
                   char FAR * FAR *in,int olen,int FAR *ilen);
#endif

/*************************************************************************/
/*=====VxD prototypes                                                    */
/*************************************************************************/
#ifdef  MSWINDLL
int FARPAS WCSCVxDBreakOff(int id);
int FARPAS WCSCVxDBreakOn(int id);
int FARPAS WCSCVxDBytesInRcvBuf(int id);
int FARPAS WCSCVxDBytesInXmtBuf(int id);
int FARPAS WCSCVxDCleanup(int id);
int FARPAS WCSCVxDDtrOff(int id);
int FARPAS WCSCVxDDtrOn(int id);
int FARPAS WCSCVxDFlush(int id,int flag);
long FARPAS WCSCVxDGetMetrics(int id,int code);
int FARPAS WCSCVxDGetPacket(int id,int count,char FAR *pkt);
int FARPAS WCSCVxDGetPort(int id,struct vxd_port_param FAR *vpcb,int flag);
int FARPAS WCSCVxDGetRegister(int id,int offset,int FAR *value);
int FARPAS WCSCVxDInit(void);
int FARPAS WCSCVxDInitializePort(int FAR *id,unsigned short sp,
                    unsigned short adr,unsigned short irq,
                    unsigned short ct, unsigned short cs,
                    unsigned short flag);
int FARPAS WCSCVxDIsAllDataOut(int id);
int FARPAS WCSCVxDOpenPort(int sport,unsigned short addr,unsigned short irq,
                  unsigned short ctyp,unsigned short cseg,
                  unsigned short flg,unsigned pport);
int FARPAS WCSCVxDPutPacket(int id,int count,char FAR *pkt);
int FARPAS WCSCVxDPutRegister(int id,int offset,int value);
```

```
int FARPAS WCSCVxDRtsOff(int id);
int FARPAS WCSCVxDRtsOn(int id);
int FARPAS WCSCVxDSetBaudDiv(int bidx,int bdiv);
int FARPAS WCSCVxDSetPortCharacteristics(int id,unsigned short bd,
                    unsigned short pa,unsigned short len,
                    unsigned short st,unsigned short prot);
int FARPAS WCSCVxDSetup(struct port_param FAR *vpcb,int FAR *id);
int FARPAS WCSCVxDSizeOfRcvBuf(int id);
int FARPAS WCSCVxDSizeOfXmtBuf(int id);
int FARPAS WCSCVxDSpaceInRcvBuf(int id);
int FARPAS WCSCVxDSpaceInXmtBuf(int id);
int FARPAS WCSCVxDViewPacket(int id,int count,char FAR *pkt);
#else
int FAR WCSCVxDBreakOff(int id);
int FAR WCSCVxDBreakOn(int id);
int FAR WCSCVxDBytesInRcvBuf(int id);
int FAR WCSCVxDBytesInXmtBuf(int id);
int FAR WCSCVxDCleanup(int id);
int FAR WCSCVxDDtrOff(int id);
int FAR WCSCVxDDtrOn(int id);
int FAR WCSCVxDFlush(int id,int flag);
long FAR WCSCVxDGetMetrics(int id,int code);
int FAR WCSCVxDGetPacket(int id,int count,char FAR *pkt);
int FAR WCSCVxDGetPort(int id,struct vxd_port_param FAR *vpcb,int flag);
int FAR WCSCVxDGetRegister(int id,int offset,int FAR *value);
int FAR WCSCVxDInit(void);
int FAR WCSCVxDInitializePort(int FAR *id,unsigned short sp,
                    unsigned short adr,unsigned short irq,
                    unsigned short ct, unsigned short cs,
                    unsigned short flag);
int FAR WCSCVxDIsAllDataOut(int id);
int FAR WCSCVxDOpenPort(int sport,unsigned short addr,unsigned short irq,
unsigned short ctyp,unsigned short cseg,unsigned short flg,unsigned pport);
int FAR WCSCVxDPutPacket(int id,int count,char FAR *pkt);
int FAR WCSCVxDPutRegister(int id,int offset,int value);
int FAR WCSCVxDRtsOff(int id);
int FAR WCSCVxDRtsOn(int id);
int FAR WCSCVxDSetBaudDiv(int bidx,int bdiv);
int FAR WCSCVxDSetPortCharacteristics(int id,unsigned short bd,
                        unsigned short pa,unsigned short len,
                        unsigned short st,unsigned short prot);
int FAR WCSCVxDSetup(struct port_param FAR *vpcb,int FAR *id);
int FAR WCSCVxDSizeOfRcvBuf(int id);
int FAR WCSCVxDSizeOfXmtBuf(int id);
int FAR WCSCVxDSpaceInRcvBuf(int id);
int FAR WCSCVxDSpaceInXmtBuf(int id);
int FAR WCSCVxDViewPacket(int id,int count,char FAR *pkt);
#endif

#ifdef  MSWIN
#ifdef  MSWINDLL
int FARPAS CdrvScrCreate(HWND hWnd,int height,int width);
int FARPAS CdrvScrDestroy(HWND hWnd);
int FARPAS CdrvScrKillFocus(HWND hWnd);
```

```
int FARPAS CdrvScrResize(HWND hWnd);
int FARPAS CdrvScrPaint(HWND hWnd);
int FARPAS CdrvScrSetFocus(HWND hWnd);
int FARPAS CdrvScrWrite(HWND hWnd,char FAR *str,int len);
int FARPAS commdrvw_char_screen(int mode,HWND hWnd,WPARAM w,LPARAM l);
#else
int CdrvScrCreate(HWND hWnd,int height,int width);
int CdrvScrDestroy(HWND hWnd);
int CdrvScrKillFocus(HWND hWnd);
int CdrvScrPaint(HWND hWnd);
int CdrvScrResize(HWND hWnd);
int CdrvScrSetFocus(HWND hWnd);
int CdrvScrWrite(HWND hWnd,char FAR *str,int len);
int commdrvw_char_screen(int mode,HWND hWnd,WPARAM w,LPARAM l);
#endif
#endif

/*************************************************************************/
/*=====Internal prototyping of renamed COMM-DRV routines                 */
/*************************************************************************/
int FAR wWCSCVxDCleanup(int id);
int FAR wWCSCVxDFlush(int id,int flag);
long FAR wWCSCVxDGetMetric(int id,int validx);
int FAR wWCSCVxDGetPacket(int id,int count,char FAR *pkt);
int FAR wWCSCVxDGetPort(int id,struct vxd_port_param FAR *vpcb,int flag);
int FAR wWCSCVxDInit(void);
int FAR wWCSCVxDInitializePort(int FAR *id,int subport,int addr,int irq,
                    int cardtype,int cardseg,int flag);
int FAR wWCSCVxDModemCtl(int id, int command);
int FAR wWCSCVxDPutPacket(int id,int count,char FAR *pkt);
int FAR wWCSCVxDSetBaudDiv(int bidx,int bdiv);
int FAR wWCSCVxDSetup(struct port_param FAR *vpcb,int FAR *id);
int FAR wWCSCVxDUARTReg(int port,int offset,int value,int putget);
int FAR wWCSCVxDViewPacket(int id,int count,char FAR *pkt);

int FAR wser_rs232_cleanup(int port);
int FAR wser_rs232_getbyte(int port,unsigned char FAR *ch);
int FAR wser_rs232_getport(int port,struct port_param FAR *pcb);
int FAR wser_rs232_getregister(int port,int offset,int FAR *value);
int FAR wser_rs232_getstatus(int port,unsigned char FAR *stat);
int FAR wser_rs232_putbyte(int port,unsigned char FAR *ch);
int FAR wser_rs232_putregister(int port,int offset,int value);
int FAR wser_rs232_maxport(void);
int FAR wser_rs232_setup(int port,struct port_param FAR *pcb);
int FAR wser_rs232_putpacket(int port,int count,unsigned char FAR *ch);
int FAR wser_rs232_getpacket(int port,int count,unsigned char FAR *ch);
int FAR wser_rs232_flush(int port,int arg);
int FAR wser_rs232_block(int port,int input,int output);
int FAR wser_rs232_dtr_off(int port);
int FAR wser_rs232_dtr_on(int port);
int FAR wser_rs232_rts_off(int port);
int FAR wser_rs232_rts_on(int port);
int FAR wser_rs232_set_intfunc(int port,
          CDRVIFNC (FAR *(FAR *func))(short val,
```

```
          struct port_param FAR *p),short imask);
struct ser_rs232_sdata FAR * FAR wser_rs232_get_sdata(void);
int FAR wser_rs232_misc_func(int port,unsigned short cmd,unsigned long val);
int FAR wser_rs232_setbauddiv(int cardtype,int baudidx,unsigned bauddiv);
int FAR wser_rs232_viewpacket(int port,int count,unsigned char FAR *ch);
void FAR * FAR ser_rs232_get_routine_table(void);

int cdrvxfer_availfname(struct cdrvxfer_ftpcb FAR *p);
int cdrvxfer_build_fileheader(struct cdrvxfer_ftpcb FAR *p);
void cdrvxfer_cancel(struct cdrvxfer_ftpcb FAR *p);
unsigned long cdrvxfer_DFdateToXFdate(unsigned short dos_date,
unsigned short dos_time);
int cdrvxfer_fileinfo(int mode,struct cdrvxfer_fileinfo FAR *f);
void cdrvxfer_flush(struct cdrvxfer_ftpcb FAR *p,int flg);
int cdrvxfer_forceout(struct cdrvxfer_ftpcb FAR *p,char FAR *ch);
int cdrvxfer_getbyte(struct cdrvxfer_ftpcb FAR *p,char FAR *ch);
int cdrvxfer_parse_packet(struct cdrvxfer_ftpcb FAR *p);
int cdrvxfer_putbyte(struct cdrvxfer_ftpcb FAR *p,char FAR *ch);
void cdrvxymod_getfiles(int mode,struct cdrvxfer_ftpcb FAR *p);
void cdrvxymod_sendfiles(int mode,struct cdrvxfer_ftpcb FAR *p);
void cdrvzmod_getfiles(struct cdrvxfer_ftpcb FAR *p);
void cdrvzmod_sendfiles(struct cdrvxfer_ftpcb FAR *p);
void cdrvxfer_XFdateToDFdate(unsigned long xfer_date,
unsigned short FAR *dos_date, unsigned short FAR *dos_time);

/* Ring buffer prototypes */
extern int BytesInRing(struct RingCtl FAR *ctl);
extern int CreateRing(struct RingCtl FAR *ctl,int len);
extern int DestroyRing(struct RingCtl FAR *ctl);
extern int GetByteFromRing(struct RingCtl FAR *ctl);
extern int GetPacketFromRing(struct RingCtl FAR *ctl,int len,char FAR *ch);
extern int PurgeRing(struct RingCtl FAR *ctl);
extern int PutByteInRing(struct RingCtl FAR *ctl,int ch);
extern int PutPacketInRing(struct RingCtl FAR *ctl,int len,char FAR *ch);
extern int SpaceInRing(struct RingCtl FAR *ctl);
extern int ViewPacketInRing(struct RingCtl FAR *ctl,int len,char FAR *ch);

#ifdef  MSWIN
unsigned short ser_rs232_getsel(unsigned short seg);
unsigned short ser_rs232_retsel(unsigned short sel);
#endif

int ser2xfer_close(unsigned int port);
int ser2xfer_getfiles(unsigned int port,char *filespec);
int ser2xfer_open(unsigned int port,unsigned int adr,unsigned int irq,
                  unsigned int speed,unsigned int parity,
                  unsigned int length,unsigned int stopbit);
int ser2xfer_receive(unsigned int port,char *localfile,
                     char *remotefile);
int ser2xfer_send(unsigned int port,char *localfile,char *remotefile);
int ser2xfer_sendfiles(unsigned int port,char *filespec);
int ser2xfer_slave(unsigned int port);

/*=====*/
```

```
int  ser_cleanup(int  port);
int  ser_getbyte(int  port,unsigned char FAR *ch);
int  ser_getport(int  port,struct  port_param FAR *pcb);
int  ser_getregister(int  port,int  offset,int  FAR *value);
int  ser_getstatus(int  port,unsigned char FAR *status);
int  ser_putbyte(int  port,unsigned char FAR *ch);
int  ser_putregister(int  port,int  offset,int  value);
int  ser_maxport(void);
int  ser_setup(int  port,struct  port_param FAR *pcb);
int  ser_putpacket(int  port,int  count,unsigned char  FAR *ch);
int  ser_getpacket(int  port,int  count,unsigned char  FAR *ch);
int  ser_flush(int  port,int  arg);
int  ser_block(int  port,int  inblock,int  outblock);
int  ser_dtr_on(int  port);
int  ser_dtr_off(int  port);
int  ser_rts_on(int  port);
int  ser_rts_off(int  port);
int  ser_set_intfunc(int port,
     CDRVIFNC (FAR *(FAR *func))(short val,struct port_param FAR *p),
     short imask);
struct ser_rs232_sdata FAR *ser_get_sdata(void);
int ser_misc_func(int port,unsigned short cmd,
     unsigned long val);
int ser_setbauddiv(int cardtype,int baudidx,unsigned bauddiv);
int ser_viewpacket(int port,int count,unsigned char FAR *ch);
int  ser_mode(int  port,int  baud,int  parity,int  lngth,int  stopbit,
               int  brk,int  protocol,int  inblock,int  outblock);
int  ser_settype(unsigned int  type);


/* INT14 Routines */
int  FAR ser_int14_cleanup(int port);
int  FAR ser_int14_getflag(int port);
struct ser_int14_info  FAR * FAR ser_int14_getinfo(void);
void FAR ser_int14_install(void);
int  FAR ser_int14_remove(void);
int  FAR ser_int14_setflag(int port,int flag);

/* General */
char    FAR *CdrvRpad(char FAR *str,int len);
long fatol(char FAR *str);
char FAR *fatoo(unsigned long val,char FAR *buf);
char FAR *fltoa(long  val,char  FAR *str,int  radix);
int ffsubstr(char FAR *str,char FAR *sstr);
int fmemcmp(void FAR *ch1,void FAR *ch2,int count);
void FAR *fmemmove(void FAR *ch1,void FAR *ch2,int count);
void FAR *fmemset(void  FAR *ch,int  c,int  count);
unsigned long fotoul(char FAR *buf);
char FAR *fstrcat(char FAR *ch1,char FAR *ch2);
int fstrcmp(char  FAR *ch1,char  FAR *ch2);
char FAR *fstrcpy(char FAR *ch1,char FAR *ch2);
int fstrlen(char  FAR *ch);
char FAR *fstrupr(char  FAR *str);
int ftoupper(int  ch);
```

```
void FAR *cdrv_alloc(int siz);
int cdrv_free(void FAR *ptr);

#ifdef __cplusplus
}                       /* End of extern "C" { */
#endif  /*  __cplusplus */
/*****************************************************************************/
#endif
```

7.2.4 Source Code for COMM-DRV/LIB Functions Used

The COMM-DRV/LIB software package includes three floppy disks, which include source code for the entire package, as well as source code for many example applications. Installation of the package places 312 files into the \COMMDRV directory and 16 subdirectories occupying about 9 Mbytes. The following pages contain directory listings of the files provided.

The primary source-code information (for the functions used) is most likely in the following four directories:

- \COMMDRV Root
- \COMMDRV\SOURCES Source code files
- \COMMDRV\INC Include files
- \COMMDRV\LIB. Library files.

These directories provide a feel for the complete level of documentation provided with the package. This directory listing also provides insight into the level of complexity and large amount of extraneous software included in the package.

The source codes necessary for the portion of the COMM-DRV software used appear in \COMMDRV\SOURCES and \COMMDRV\INC. Not all the files in these directories are necessary for the computational block software. It is difficult to discover which files are extraneous. Those files that have been found necessary for DATA_ATT.EXE software are denoted with a "==>" on the left side. However, there is no one-to-one correlation between the COMM-DRV functions called, which are listed above, and the file names containing the source codes, which follow in the directories. The high-level COMM-DRV functions appear to be located in the *.C group of source codes. A detailed examination of all the source files is necessary to match all the functional calls with source code files. The source files contain code for multiple functions and do not index the contents in an initial comment. The source files contain references to subsequent subroutines. Each file in the COMM*.ASM family contains several interrelated assembly-language routines used to handle the COM ports and are highly likely to be many of these subsequent subroutine calls.

The following directory listings include all the files provided with the software. These were obtained with a Norton find file (FF) command listing after installing the software.

FF listing of the COMM-DRV/LIB software provided by WCSC

C:\COMMDRV

	.	<DIR>	5:08 pm	Sat	Mar	3 00
	..	<DIR>	5:08 pm	Sat	Mar	3 00
	deisl1.isu	14,520 bytes	5:11 pm	Sat	Mar	3 00
	cdrvlib.bmp	10,774 bytes	1:07 am	Tue	Mar	30 99
	readmecl.doc	31,922 bytes	12:25 am	Tue	Mar	30 99
	readmecl.txt	4,683 bytes	12:36 am	Tue	Mar	30 99
	cdrvlib.hlp	516,564 bytes	10:56 pm	Sun	Nov	30 97
	BC7	<DIR>	5:08 pm	Sat	Mar	3 00
	ACCESS	<DIR>	5:08 pm	Sat	Mar	3 00
	cdrvdl32.dll	31,232 bytes	1:09 am	Tue	Mar	30 99
	cdrvdll.dll	56,320 bytes	12:58 am	Tue	Mar	30 99
	cdrvhf.dll	32,768 bytes	12:59 am	Tue	Mar	30 99
	cdrvhf32.dll	32,256 bytes	1:10 am	Tue	Mar	30 99
	cdrvxf.dll	45,056 bytes	1:00 am	Tue	Mar	30 99
	cdrvxf32.dll	43,520 bytes	1:10 am	Tue	Mar	30 99
	commsc32.dll	18,432 bytes	1:10 am	Tue	Mar	30 99
	commscrw.dll	8,704 bytes	1:00 am	Tue	Mar	30 99
	boca1610.bin	3,228 bytes	6:49 pm	Tue	Mar	30 99
	xabios.bin	2,048 bytes	6:49 pm	Tue	Mar	30 99
	xacook.bin	6,144 bytes	6:49 pm	Tue	Mar	30 99
	BIN	<DIR>	5:09 pm	Sat	Mar	3 00
	EXAMPLES	<DIR>	5:09 pm	Sat	Mar	3 00
	INC	<DIR>	5:09 pm	Sat	Mar	3 00
	LIB	<DIR>	5:09 pm	Sat	Mar	3 00
	PRJMSEX4	<DIR>	5:10 pm	Sat	Mar	3 00
	PRJMSEX8	<DIR>	5:10 pm	Sat	Mar	3 00
	PRJMSEX9	<DIR>	5:10 pm	Sat	Mar	3 00
	PRJBOEX4	<DIR>	5:10 pm	Sat	Mar	3 00
	PRJBOEX8	<DIR>	5:10 pm	Sat	Mar	3 00
	PRJBOEX9	<DIR>	5:10 pm	Sat	Mar	3 00
	VBWIN	<DIR>	5:10 pm	Sat	Mar	3 00
	VBDOS	<DIR>	5:10 pm	Sat	Mar	3 00
	QB45	<DIR>	5:10 pm	Sat	Mar	3 00
	SOURCES	<DIR>	5:10 pm	Sat	Mar	3 00

C:\COMMDRV\SOURCES

	.	<DIR>	5:10 pm	Sat	Mar	3 00
	..	<DIR>	5:10 pm	Sat	Mar	3 00
	bioclk.asm	4,063 bytes	12:20 am	Sat	Oct	21 95
	biortc.asm	4,894 bytes	12:20 am	Sat	Oct	21 95
	biotimer.asm	7,672 bytes	12:20 am	Sat	Oct	21 95
==>	biovideo.asm	12,824 bytes	12:20 am	Sat	Oct	21 95
	cdrvifnc.asm	8,218 bytes	12:21 am	Sat	Oct	21 95
	cdrvpio.asm	15,346 bytes	12:21 am	Sat	Oct	21 95
	cnvatoi.asm	1,547 bytes	12:22 am	Sat	Oct	21 95
	cnvhtoi.asm	1,912 bytes	12:22 am	Sat	Oct	21 95
	cnvitoa.asm	1,565 bytes	12:22 am	Sat	Oct	21 95
	cnvitoh.asm	1,575 bytes	12:22 am	Sat	Oct	21 95
==>	commdrvc.asm	8,254 bytes	12:22 am	Sat	Oct	21 95
==>	commdv00.asm	40,029 bytes	11:37 am	Tue	Mar	23 99
==>	commdv01.asm	67,170 bytes	11:47 am	Tue	Apr	7 98
==>	commdv02.asm	46,718 bytes	12:22 am	Sat	Oct	21 95

```
==>     commdv03.asm    48,938 bytes  12:22 am  Sat Oct 21 95
==>     commdv04.asm    47,981 bytes  12:22 am  Sat Oct 21 95
==>     commdv05.asm    69,581 bytes  12:22 am  Sat Oct 21 95
==>     commdv06.asm    36,655 bytes  12:22 am  Sat Oct 21 95
==>     commdv07.asm    27,046 bytes  12:22 am  Sat Oct 21 95
==>     commdv08.asm    46,484 bytes   6:48 pm  Wed Mar 24 99
==>     commdv09.asm    21,614 bytes  12:22 am  Sat Oct 21 95
        dosdir.asm       4,365 bytes  12:23 am  Sat Oct 21 95
        dosenv.asm       2,201 bytes  12:23 am  Sat Oct 21 95
        dosfattr.asm     5,878 bytes  12:23 am  Sat Oct 21 95
        dosfilio.asm    12,098 bytes  12:23 am  Sat Oct 21 95
        dosfnext.asm     4,904 bytes  12:23 am  Sat Oct 21 95
        dosmem.asm       4,515 bytes  12:23 am  Sat Oct 21 95
        dostime.asm      4,950 bytes  12:23 am  Sat Oct 21 95
        dostsr.asm       7,726 bytes  12:23 am  Sat Oct 21 95
        dostsr1.asm      4,544 bytes  12:23 am  Sat Oct 21 95
        dostsr2.asm      7,878 bytes  12:23 am  Sat Oct 21 95
        dostsr3.asm      3,652 bytes  12:23 am  Sat Oct 21 95
        dosvect.asm      2,198 bytes  12:23 am  Sat Oct 21 95
        mscrnbuf.asm    15,016 bytes  12:24 am  Sat Oct 21 95
        mscrtslc.asm     4,399 bytes  12:24 am  Sat Oct 21 95
        mscsvec.asm      2,432 bytes  12:24 am  Sat Oct 21 95
        ser1cbuf.asm     4,769 bytes  12:25 am  Sat Oct 21 95
        ser1r232.asm    34,918 bytes  12:25 am  Sat Oct 21 95
        ser2in14.asm     5,860 bytes  12:25 am  Sat Oct 21 95
        sersr232.asm   124,918 bytes   8:46 pm  Wed Mar 24 99
        strdelim.asm     1,088 bytes  12:25 am  Sat Oct 21 95
        strupr.asm       1,187 bytes  12:25 am  Sat Oct 21 95
        timtime.asm      3,875 bytes  12:26 am  Sat Oct 21 95
        wcscvxd1.asm    26,529 bytes  11:37 am  Tue Mar 23 99
        cdrv09.c        19,703 bytes   4:47 pm  Sun Jan 26 97
        cdrvdll.c       11,204 bytes  12:20 am  Sat Oct 21 95
        cdrvdl32.c      72,569 bytes  11:28 pm  Mon Mar 22 99
        cdrvhf00.c      12,055 bytes   1:02 am  Fri Feb 21 97
        cdrvhf01.c       8,523 bytes  12:21 am  Sat Oct 21 95
        cdrvhf02.c      27,604 bytes  12:04 pm  Thu Apr  9 98
        cdrvhf03.c      11,137 bytes  12:04 am  Thu Oct  3 96
        cdrvhf04.c       5,742 bytes  12:21 am  Sat Oct 21 95
        cdrvhf05.c      13,777 bytes  12:21 am  Sat Oct 21 95
        cdrvhf06.c       1,969 bytes  12:21 am  Sat Oct 21 95
        cdrvhf07.c       2,705 bytes  12:21 am  Sat Oct 21 95
        cdrvhfnc.c      14,351 bytes  11:01 pm  Mon Feb 24 97
        cdrvlib.c        9,989 bytes  12:21 am  Sat Oct 21 95
        cdrvxf.c        11,496 bytes  11:37 am  Tue Mar 23 99
        cdrvxf01.c      33,200 bytes  10:11 am  Wed Aug 19 98
        cdrvxfer.c      30,645 bytes  10:43 am  Mon Jul 14 97
        cdrvxfeu.c      34,600 bytes  10:51 am  Mon Jul 14 97
        cdrvxfio.c      15,364 bytes  12:21 am  Sat Oct 21 95
        cdrvzmod.c      47,206 bytes  12:22 am  Sat Oct 21 95
        commdrvi.c       5,835 bytes  12:22 am  Sat Oct 21 95
        commscrw.c      17,604 bytes  12:23 am  Sat Oct 21 95
        commver.c          821 bytes  12:23 am  Sat Oct 21 95
        csi1drv.c       13,119 bytes  12:23 am  Sat Oct 21 95
        csi1fist.c       2,949 bytes  12:23 am  Sat Oct 21 95
```

```
csilpars.c       3,219 bytes  12:23 am  Sat Oct 21 95
csilpstr.c       4,169 bytes  12:23 am  Sat Oct 21 95
dosraw.c         2,101 bytes  12:23 am  Sat Oct 21 95
dostest.c          707 bytes   9:27 pm  Tue Mar 30 99
kscldinp.c       4,979 bytes  12:23 am  Sat Oct 21 95
kscledit.c      12,263 bytes  12:23 am  Sat Oct 21 95
ksclfid.c        3,068 bytes  12:23 am  Sat Oct 21 95
ksclfiu.c        3,056 bytes  12:23 am  Sat Oct 21 95
ksclinpt.c       3,922 bytes  12:23 am  Sat Oct 21 95
kscInfld.c       5,580 bytes  12:23 am  Sat Oct 21 95
kscIrbrk.c       2,284 bytes  12:23 am  Sat Oct 21 95
kscIrebk.c       2,261 bytes  12:23 am  Sat Oct 21 95
kscIrefd.c       2,877 bytes  12:23 am  Sat Oct 21 95
kscIrep.c        2,322 bytes  12:23 am  Sat Oct 21 95
kscIrfld.c       2,419 bytes  12:23 am  Sat Oct 21 95
ksc2mndv.c       6,894 bytes  12:23 am  Sat Oct 21 95
ksc3elab.c       2,650 bytes  12:23 am  Sat Oct 21 95
ksc3labl.c       2,374 bytes  12:23 am  Sat Oct 21 95
ksc3plab.c       2,102 bytes  12:24 am  Sat Oct 21 95
ksc4win.c        9,780 bytes  12:24 am  Sat Oct 21 95
kscsgpos.c       2,121 bytes  12:24 am  Sat Oct 21 95
kscsinit.c       3,428 bytes  12:24 am  Sat Oct 21 95
libtest.c           51 bytes   9:27 pm  Tue Mar 30 99
mkbsinit.c       4,499 bytes  12:24 am  Sat Oct 21 95
mkbsinky.c       4,151 bytes  12:24 am  Sat Oct 21 95
mkbskchk.c       5,409 bytes  12:24 am  Sat Oct 21 95
mkbskdef.c       3,633 bytes  12:24 am  Sat Oct 21 95
mkbstrap.c       4,436 bytes  12:24 am  Sat Oct 21 95
mkbswaky.c       1,940 bytes  12:24 am  Sat Oct 21 95
mkbswkey.c       2,590 bytes  12:24 am  Sat Oct 21 95
msccrc16.c       2,863 bytes  12:24 am  Sat Oct 21 95
msccrc32.c       3,748 bytes  12:24 am  Sat Oct 21 95
mscring.c       10,221 bytes  12:24 am  Sat Oct 21 95
scrinex.c       34,763 bytes   9:27 pm  Tue Mar 30 99
scrinex1.c       4,413 bytes   9:27 pm  Tue Mar 30 99
scrinex2.c       2,466 bytes   9:27 pm  Tue Mar 30 99
scrinex3.c       5,345 bytes   9:27 pm  Tue Mar 30 99
scrinver.c       1,043 bytes  12:24 am  Sat Oct 21 95
scrsbnd.c        3,859 bytes  12:24 am  Sat Oct 21 95
scrsbuf.c        9,684 bytes  12:24 am  Sat Oct 21 95
scrscolr.c       4,612 bytes  12:24 am  Sat Oct 21 95
scrscpos.c       8,005 bytes  12:24 am  Sat Oct 21 95
scrsdraw.c       5,808 bytes  12:24 am  Sat Oct 21 95
scrseras.c       5,146 bytes  12:24 am  Sat Oct 21 95
scrsinit.c       7,531 bytes  12:24 am  Sat Oct 21 95
scrsparm.c       4,896 bytes  12:24 am  Sat Oct 21 95
scrsprt.c        9,869 bytes  12:25 am  Sat Oct 21 95
scrsrefr.c       2,861 bytes  12:25 am  Sat Oct 21 95
scrsscrl.c       3,102 bytes  12:25 am  Sat Oct 21 95
scrssfch.c       2,740 bytes  12:25 am  Sat Oct 21 95
scrssr.c         5,160 bytes  12:25 am  Sat Oct 21 95
scrssvrs.c       3,903 bytes  12:25 am  Sat Oct 21 95
scrswrt.c        2,474 bytes  12:25 am  Sat Oct 21 95
serlprim.c      46,532 bytes  12:25 am  Sat Oct 21 95
```

```
        ser2xfer.c      26,770 bytes  12:25 am  Sat Oct 21 95
        strlgsft.c       2,417 bytes  12:25 am  Sat Oct 21 95
        str2chsf.c       2,168 bytes  12:25 am  Sat Oct 21 95
        str2lpad.c       2,072 bytes  12:25 am  Sat Oct 21 95
        str2rl.c         2,150 bytes  12:25 am  Sat Oct 21 95
        str2rlt.c        2,021 bytes  12:25 am  Sat Oct 21 95
        str2rpad.c       2,028 bytes  12:25 am  Sat Oct 21 95
        str2rt.c         2,191 bytes  12:25 am  Sat Oct 21 95
        str2stcs.c       2,253 bytes  12:25 am  Sat Oct 21 95
        str2utab.c       1,944 bytes  12:25 am  Sat Oct 21 95
        str3addo.c       2,442 bytes  12:25 am  Sat Oct 21 95
        str3cfc.c        2,242 bytes  12:25 am  Sat Oct 21 95
        str3dtag.c       2,633 bytes  12:25 am  Sat Oct 21 95
        str3gtim.c       2,682 bytes  12:25 am  Sat Oct 21 95
        str3lgss.c       2,700 bytes  12:25 am  Sat Oct 21 95
        str3sslg.c       2,209 bytes  12:25 am  Sat Oct 21 95
        str3vdat.c       2,986 bytes  12:25 am  Sat Oct 21 95
        strtest.c           26 bytes   9:27 pm  Tue Mar 30 99
        testcsi.c        1,507 bytes   9:27 pm  Tue Mar 30 99
        timstime.c       2,375 bytes  12:26 am  Sat Oct 21 95
        commdvxx.dbg    61,090 bytes   9:27 pm  Tue Mar 30 99
        cdrvdll.def      1,957 bytes  12:20 am  Sat Oct 21 95
        cdrvdl32.def       977 bytes   4:51 pm  Sat Aug  3 96
        cdrvhf.def       2,850 bytes  12:04 am  Thu Oct  3 96
        cdrvhf32.def     2,287 bytes  12:04 am  Thu Oct  3 96
        cdrvxf.def       1,150 bytes  12:21 am  Sat Oct 21 95
        cdrvxf32.def     1,023 bytes   8:21 am  Wed Aug 19 98
        commsc32.def       517 bytes  12:23 am  Sat Oct 21 95
        commscrw.def       563 bytes  12:23 am  Sat Oct 21 95
        commdrv1.ico     1,278 bytes  12:22 am  Sat Oct 21 95
        wcsc001.ico        766 bytes  12:26 am  Sat Oct 21 95
        cdrvdl32.rc      1,110 bytes  12:20 am  Sat Oct 21 95
        cdrvdll.rc       1,111 bytes  12:20 am  Sat Oct 21 95
        cdrvhf.rc        1,115 bytes  12:21 am  Sat Oct 21 95
        cdrvhf32.rc      1,118 bytes  12:21 am  Sat Oct 21 95
        cdrvxf.rc        2,076 bytes  12:21 am  Sat Oct 21 95
        cdrvxf32.rc      2,075 bytes  12:21 am  Sat Oct 21 95
        commsc32.rc      1,114 bytes  12:23 am  Sat Oct 21 95
        commscrw.rc      1,107 bytes  12:23 am  Sat Oct 21 95

C:\COMMDRV\INC
        .                  <DIR>      5:09 pm  Sat Mar  3 00
        ..                 <DIR>      5:09 pm  Sat Mar  3 00
==>     comm.h         123,194 bytes   4:05 pm  Tue Mar 23 99
        csi.h            1,710 bytes  12:19 am  Sat Oct 21 95
        dosdef.h        10,292 bytes  10:47 pm  Wed Aug 19 98
        fieldlen.h       1,518 bytes  12:19 am  Sat Oct 21 95
        general.h          982 bytes  12:19 am  Sat Oct 21 95
        keyboard.h       6,524 bytes  12:19 am  Sat Oct 21 95
        keybscrn.h       5,007 bytes  12:19 am  Sat Oct 21 95
        misc.h             571 bytes  12:19 am  Sat Oct 21 95
        msccrc.h           808 bytes  12:19 am  Sat Oct 21 95
        screenio.h       9,110 bytes  12:19 am  Sat Oct 21 95
        strings.h        1,573 bytes  12:19 am  Sat Oct 21 95
```

```
        timer.h           981 bytes  12:19 am  Sat Oct 21 95
==>     comm.inc       57,165 bytes   4:05 pm  Tue Mar 23 99
        device.inc      3,272 bytes  12:19 am  Sat Oct 21 95
        dosdef.inc      1,859 bytes  12:19 am  Sat Oct 21 95
        general.inc       446 bytes  12:19 am  Sat Oct 21 95

C:\COMMDRV\LIB
        .                   <DIR>      5:09 pm  Sat Mar  3 00
        ..                  <DIR>      5:09 pm  Sat Mar  3 00
        cdrvdl32.lib    17,194 bytes   1:09 am  Tue Mar 30 99
        cdrvdll.lib      3,072 bytes  12:59 am  Tue Mar 30 99
        cdrvhf.lib       8,704 bytes   1:00 am  Tue Mar 30 99
        cdrvhf32.lib    51,168 bytes   1:10 am  Tue Mar 30 99
        cdrvxf.lib       2,048 bytes   1:00 am  Tue Mar 30 99
        cdrvxf32.lib    20,532 bytes   1:10 am  Tue Mar 30 99
        commsc32.lib     5,894 bytes   1:10 am  Tue Mar 30 99
        commscrw.lib     2,048 bytes   1:00 am  Tue Mar 30 99
        commdml.lib     79,045 bytes  12:56 am  Tue Mar 30 99
        commdms.lib     74,437 bytes  12:51 am  Tue Mar 30 99
        commdrml.lib   147,811 bytes  12:57 am  Tue Mar 30 99
        commdrms.lib   143,715 bytes  12:54 am  Tue Mar 30 99
        commdwml.lib   180,597 bytes   1:01 am  Tue Mar 30 99
        commdwms.lib   172,403 bytes   1:01 am  Tue Mar 30 99
        libsml.lib      29,517 bytes  12:55 am  Tue Mar 30 99
        libsms.lib      27,469 bytes  12:51 am  Tue Mar 30 99
        commdbl.lib     90,624 bytes   1:04 am  Tue Mar 30 99
        commdbs.lib     87,040 bytes   1:04 am  Tue Mar 30 99
        commdrbl.lib   164,352 bytes   1:04 am  Tue Mar 30 99
        commdrbs.lib   160,768 bytes   1:04 am  Tue Mar 30 99
        commdwbl.lib   165,376 bytes   1:06 am  Tue Mar 30 99
        commdwbs.lib   159,744 bytes   1:06 am  Tue Mar 30 99
        libsbl.lib      34,816 bytes   1:03 am  Tue Mar 30 99
        libsbs.lib      33,280 bytes   1:03 am  Tue Mar 30 99
        libswml.lib     28,399 bytes  12:59 am  Tue Mar 30 99
        libswms.lib     25,839 bytes  12:58 am  Tue Mar 30 99
        scrinpml.lib    37,823 bytes  12:57 am  Tue Mar 30 99
        scrinpms.lib    33,727 bytes  12:53 am  Tue Mar 30 99
        scrinwml.lib    53,183 bytes   1:02 am  Tue Mar 30 99
        scrinwms.lib    45,503 bytes   1:02 am  Tue Mar 30 99
        libswbl.lib     28,672 bytes   1:07 am  Tue Mar 30 99
        libswbs.lib     27,136 bytes   1:07 am  Tue Mar 30 99
        scrinpbl.lib    44,544 bytes   1:05 am  Tue Mar 30 99
        scrinpbs.lib    39,424 bytes   1:05 am  Tue Mar 30 99
        scrinwbl.lib    50,688 bytes   1:08 am  Tue Mar 30 99
        scrinwbs.lib    44,544 bytes   1:08 am  Tue Mar 30 99

C:\COMMDRV\BC7
        .                   <DIR>      5:08 pm  Sat Mar  3 00
        ..                  <DIR>      5:08 pm  Sat Mar  3 00
        bc7comm.bas     26,124 bytes  12:14 am  Sat Oct 21 95
        bc7exam.bas      6,170 bytes  12:14 am  Sat Oct 21 95
        bc7exm2.bas      9,341 bytes  12:14 am  Sat Oct 21 95
        bc7make.bat      2,176 bytes  11:08 pm  Sun Apr 20 97
        bc7cdrv.lib    324,551 bytes  11:08 pm  Sun Apr 20 97
```

```
        bc7exam.exe     64,410 bytes  11:11 pm  Sun Apr 20 97
        bc7exm2.exe    106,068 bytes  11:11 pm  Sun Apr 20 97
        bc7cdrv2.lib   161,851 bytes  11:08 pm  Sun Apr 20 97
        bc7cdrv.qlb    138,290 bytes  11:08 pm  Sun Apr 20 97

C:\COMMDRV\ACCESS
        .                   <DIR>       5:08 pm  Sat Mar  3 00
        ..                  <DIR>       5:08 pm  Sat Mar  3 00
        accemake.bat        194 bytes   4:49 pm  Sat Aug  3 96
        accecdrv.mdb    327,680 bytes   4:49 pm  Sat Aug  3 96
        ac32cdrv.mdb    266,240 bytes   4:48 pm  Sat Aug  3 96

C:\COMMDRV\BIN
        .                   <DIR>       5:09 pm  Sat Mar  3 00
        ..                  <DIR>       5:09 pm  Sat Mar  3 00
        bexample.bat     26,856 bytes  12:14 am  Sat Oct 21 95
        makeblib.bat     18,948 bytes  12:14 am  Sat Oct 21 95
        makemlib.bat     18,026 bytes   9:21 am  Tue Jun 18 96
        mexample.bat     24,019 bytes   2:29 pm  Fri Feb 27 98

C:\COMMDRV\EXAMPLES
        .                   <DIR>       5:09 pm  Sat Mar  3 00
        ..                  <DIR>       5:09 pm  Sat Mar  3 00
        cdrvpioc.c        5,421 bytes  12:17 am  Sat Oct 21 95
        example4.c       12,380 bytes  12:18 am  Sat Oct 21 95
        example8.c       29,165 bytes  11:36 am  Tue Mar 23 99
        example9.c       28,412 bytes  10:08 pm  Sat Nov 11 95
        exferdlg.c       29,751 bytes  12:18 am  Sat Oct 21 95
        exintfnc.c       15,355 bytes   8:15 am  Fri Mar  7 97
        example8.def        927 bytes  12:18 am  Sat Oct 21 95
        example9.def      1,015 bytes  12:18 am  Sat Oct 21 95
        exferdlg.def        925 bytes  12:18 am  Sat Oct 21 95
        examples.h        3,203 bytes  12:18 am  Sat Oct 21 95
        cdrvpioc.exe     10,799 bytes  12:57 am  Tue Mar 30 99
        exam8w32.exe     38,400 bytes   1:11 am  Tue Mar 30 99
        example4.exe     97,363 bytes  12:57 am  Tue Mar 30 99
        example8.exe     26,112 bytes   1:03 am  Tue Mar 30 99
        example9.exe    110,080 bytes   1:03 am  Tue Mar 30 99
        exferdlg.exe     26,112 bytes   1:03 am  Tue Mar 30 99
        example8.ico        766 bytes  10:03 pm  Tue Mar 30 99
        example9.ico        766 bytes  10:03 pm  Tue Mar 30 99
        example8.rc       6,445 bytes  12:18 am  Sat Oct 21 95
        example9.rc       5,956 bytes  12:18 am  Sat Oct 21 95
        exferdlg.rc       5,989 bytes  12:18 am  Sat Oct 21 95

C:\COMMDRV\PRJMSEX4
        .                   <DIR>       5:10 pm  Sat Mar  3 00
        ..                  <DIR>       5:10 pm  Sat Mar  3 00
        example4.mak      1,817 bytes   7:25 pm  Tue Mar 30 99

C:\COMMDRV\PRJMSEX8
        .                   <DIR>       5:10 pm  Sat Mar  3 00
        ..                  <DIR>       5:10 pm  Sat Mar  3 00
        example8.mak      2,543 bytes   7:25 pm  Tue Mar 30 99
```

```
C:\COMMDRV\PRJMSEX9
        .                <DIR>        5:10 pm  Sat Mar  3 00
        ..               <DIR>        5:10 pm  Sat Mar  3 00
        example9.mak     2,438 bytes  7:25 pm  Tue Mar 30 99

C:\COMMDRV\PRJBOEX4
        .                <DIR>        5:10 pm  Sat Mar  3 00
        ..               <DIR>        5:10 pm  Sat Mar  3 00
        example4.ide    23,122 bytes  7:23 pm  Tue Mar 30 99
        example4.mak     2,663 bytes  7:23 pm  Tue Mar 30 99

C:\COMMDRV\PRJBOEX8
        .                <DIR>        5:10 pm  Sat Mar  3 00
        ..               <DIR>        5:10 pm  Sat Mar  3 00
        example8.ide    29,946 bytes  7:24 pm  Tue Mar 30 99
        example8.mak     4,364 bytes  7:24 pm  Tue Mar 30 99

C:\COMMDRV\PRJBOEX9
        .                <DIR>        5:10 pm  Sat Mar  3 00
        ..               <DIR>        5:10 pm  Sat Mar  3 00
        example9.ide    28,950 bytes  7:24 pm  Tue Mar 30 99
        example9.mak     3,082 bytes  7:24 pm  Tue Mar 30 99

C:\COMMDRV\VBWIN
        .                <DIR>        5:10 pm  Sat Mar  3 00
        ..               <DIR>        5:10 pm  Sat Mar  3 00
        vb32comm.bas    40,815 bytes 12:11 pm  Wed Aug 19 98
        vb32comm.frm    21,653 bytes 10:34 pm  Sun Oct 13 96
        vb32exam.mak       468 bytes 11:06 pm  Mon Jul 29 96
        vb32exam.exe    42,496 bytes  9:37 pm  Sat Oct 12 96
        vb32file.frm     2,482 bytes 11:06 pm  Mon Jul 29 96
        vb32port.frm     4,237 bytes 11:06 pm  Mon Jul 29 96
        vb32vxd.frm        974 bytes 11:32 am  Sat Dec 14 96
        vb32vxd.exe     18,944 bytes 12:11 am  Thu Jun 27 96
        vb32xfer.frm     3,824 bytes 11:06 pm  Mon Jul 29 96
        vbwicomm.bas    40,154 bytes 11:05 pm  Tue Mar  3 98
        vbwimake.bat       207 bytes 11:32 am  Sat Dec 14 96
        vbwicomm.frm    22,503 bytes 10:35 pm  Sun Oct 13 96
        vbwiexam.exe    37,104 bytes 11:39 pm  Sat Oct 12 96
        vbwifile.frm     2,482 bytes  4:51 pm  Sat Aug  3 96
        vbwiport.frm     7,102 bytes 10:35 pm  Sun Oct 13 96
        vbwiport.frx       153 bytes 12:23 am  Sun Oct 13 96
        vbwixfer.frm     3,824 bytes  4:51 pm  Sat Aug  3 96
        vbwiexam.mak       418 bytes  4:51 pm  Sat Aug  3 96

C:\COMMDRV\VBDOS
        .                <DIR>        5:10 pm  Sat Mar  3 00
        ..               <DIR>        5:10 pm  Sat Mar  3 00
        vbdocomm.bas    26,271 bytes 12:26 am  Sat Oct 21 95
        vbdomake.bat     1,953 bytes  9:23 am  Tue Jun 18 96
        vbdocomm.frm    29,921 bytes 10:34 pm  Sun Oct 13 96
        vbdoexam.exe   242,674 bytes  9:41 pm  Sun Oct 13 96
        vbdofile.frm     1,973 bytes 12:26 am  Sat Oct 21 95
```

```
        vbdoport.frm      5,057 bytes   10:34 pm  Sun Oct 13 96
        vbdoxfer.frm      2,986 bytes   12:26 am  Sat Oct 21 95
        vbdocdr2.lib    186,411 bytes    9:40 pm  Sun Oct 13 96
        vbdocdrv.lib    381,909 bytes    9:40 pm  Sun Oct 13 96
        vbdoexam.mak         56 bytes    9:36 pm  Sun Oct 13 96
        vbdocdrv.qlb    155,208 bytes    9:40 pm  Sun Oct 13 96

C:\COMMDRV\QB45
        .                  <DIR>         5:10 pm  Sat Mar  3 00
        ..                 <DIR>         5:10 pm  Sat Mar  3 00
        qb45comm.bas     18,936 bytes   12:20 am  Sat Oct 21 95
        qb45exam.bas      5,548 bytes   12:20 am  Sat Oct 21 95
        qb45exm2.bas     22,933 bytes   12:20 am  Sat Oct 21 95
        qb45make.bat      2,385 bytes    9:22 am  Tue Jun 18 96
        qb45cdrv.lib     79,149 bytes   10:52 pm  Tue Apr  9 96
        qb45exam.exe     60,974 bytes   10:53 pm  Tue Apr  9 96
        qb45exm2.exe     43,286 bytes   10:53 pm  Tue Apr  9 96
        qb45cdrv.qlb     60,064 bytes   10:52 pm  Tue Apr  9 96
```

7.3 Microsoft, Visual C++ Version 1.52 – C-Compiler

Microsoft Visual C++ version 1.52 C-Compiler is the last version to compile for DOS. The current version is 6.0. A CD-ROM containing the Microsoft Visual C++ version 1.52 C-Compiler can be obtained as a downgrade disk for about $20 by calling Microsoft parts at 1-800-621-7930 with the CD-ROM number from the back of a Microsoft Visual C++ version 6.0 CD-ROM.

7.3.1 Specifications for a C-Compiler

The C-compiler must compile the DATA_ATT.C source code into 16-bit executable code for the 80386SX processor running a DOS operating system.

7.3.2 Compiler Configuration and Parameter Settings

It is important to completely specify the compiler configuration and parameters to allow the monitoring party to recompile the complete source code (after thorough examination) into an identical byte-by-byte match of the executable code located in the PROMs. This documentation needs to be complete and cover all the methods and options used. It would be undesirable for problems due to insufficient specification to hinder the software authentication process.

Microsoft Visual C++ uses workspaces to specify the names of files to be compiled and resources to be used (e.g., header files and library files). It uses a project file for configuring and optimizing the compiler. All the information in the project file would be necessary to specify the compiler status and to duplicate the results. It would be desirable to provide all this information in both a human-readable and machine-readable form.

At a minimum, the compiler uses information regarding

- The target processor – 80386 CPU
- The target operating system – ROM-DOS version 6.22
- The memory model
- The method of encoding math functions – no math coprocessor
- The location and names of libraries
- The location and names of the source code files
- The order of compilation and thus object linking
- Compiler optimization settings.

Various compilers and programmers use different methods of providing this information.

- Environmental variables
- Option switches on the compiler call line
- Setup options when the compiler is installed
- Batch file to control compiling and linking
- Make file to control compiling and linking
- Project file to control compiling and linking via settings
- Workspace to control compiling and linking via resources.

GetPacket

Description-

This function attempts to get a packet of desired length. Unlike the low level function **ser_rs232_getpacket()**, it will read a partial packet. Likewise, unlike **ser_rs232_getpacket()**, the input buffer length specified during the **InitializePort()** or **ser_rs232_setup()** functions may be less than the requested packet size. This function will wait for up to the amount of time set by **SetTimeout()** before returning failure if the entire packet is not read.

Syntax-

count = GetPacket(port,len,pkt);

On Entry-

int port;

Port previously opened with **InitializePort()**.

int len;

Desired length of packet to read.

char *pkt;

Pointer to buffer to store the packet.

On Exit-

int count;

Length of packet actually read.

else

-1 If port was not initialized.

See Also-

GetByte()
GetString()

Example-

```
#include <comm.h>

int     count;
int     port=0;
char    buf[256];


if ((count = GetPacket(port,sizeof(buf),buf)) == -1)
        {
        printf("Port was not initialized\n");

        /* Take remedial action*/
        break;
        }
```

GetString

Description-

This function attempts to get a string from the serial port. A string is considered to be a set of characters delimited by a carriage return, a linefeed, or a NULL character. This function throws away said characters. The returned string is always null terminated. This function will wait for up to the amount of time set by **SetTimeout()** before returning if an entire string is not read. A partial string will be read if the time-out occurs before one of the above delimiting characters is read or if an attempt to read a string larger than the specified buffer is read.

Syntax-

count = GetString(port,len,str);

On Entry-

int port;

Port previously opened with **InitializePort()**.

int len;

Length of the buffer in which to place the string.

char *str;

Pointer to the buffer for the string.

On Exit-

int count;

Number of bytes returned.

See Also-

GetByte()
GetPacket()

Example-

```
#include <comm.h>

int     count;
int     port=0;
char    buf[256];


if ((count = GetString(port,sizeof(buf),buf)) == -1)
        {
        printf("Port was not initialized\n");

        /* Take remedial action*/
        break;
        }
```

GetTimeout

Description-

Get the current timeout value. This function is typically used before issuing the **SetTimeout()** value so that the previous value may be restored.

Syntax-

timeout = GetTimeout(port);

On Entry-

int port;

Port previously opened with **InitializePort()**.

On Exit-

unsigned int timeout;

The current timeout.

else

0xffff If port was not initialized.

See Also-

CdrvSetTimerResolution()
SetTimeout()

Example-

```
#include <comm.h>

unsigned int    oldtimeout;
unsigned int    newtimeout=1;
int             port=0;


/* Save old timeout */
if ((oldtimeout = GetTimeout(port)) == -1)
        {
        printf("Port not initialized\n");

        /* Take remedial action */
        }

/* Set a new timeout */
SetTimeout(port,newtimeout);

/* Restore the old pace time */
SetTimeout(port,oldtimeout);
```

InitializePort

Description-

This function initializes the serial port. This function unlike the low level function **ser_rs232_setup()** does not require allocating and initializing buffers or any port control blocks. This call by default set the port characteristics to 9600 baud, 8 data bits, 1 stopbit, no parity, no line protocol(no software or hardware flow control). The high level function **SetPortCharacteristics()** may be used to change the baud rate, and the other port characteristics.

NOTE: Before the application exits, it should make a successful call to **UnInitializePort()**. If this is not done, the system will certainly crash since interrupt vectors will be left pointing to non existent code.

Syntax-

stat = InitializePort(port,subport,addr,irq,cardtype,cardseg, inbuflen,outbuflen,flag);

On Entry-

int port;

Port to initialize. This is simply an integer that will be assigned to the serial port as described by the arguments to this function. Subsequently, all functions use this number to refer to the opened port. Virtually any number of ports may be opened concurrently.

int subport;

This value should be 0 unless a multiport card is being used or unless **cardtype = CARD_WINAPI**. If a multiport card is being used(and cardtype is not CARD_WINAPI), then the first serial port on the multiport card is 0. The second port is 1, and so on. If **cardtype = CARD_WINAPI**, then this value is used to specify which **COM** port is being used(0 for **COM1**, 1 for **COM2**, 2 for **COM3**, and so on).

unsigned short addr;

This value corresponds to the address of the UART on the serial card. For most dumb multiport cards, each UART on the card has a distinct address. Each port would have said distinct address. For most intelligent multiport cards, this value will be physical address that the PC uses to communicate with the multiport card. In that case all ports associated with the one card should use the same address. This value should be 0(zero) if **cardtype = CARD_WINAPI**.

unsigned short irq;

This is the IRQ the for the serial port/multiport card. If the particular card does not require using an IRQ, this value should be **16**. This value should be 0(zero) if **cardtype = CARD_WINAPI**.

unsigned short cardtype;

This corresponds to the type of multiport card or serial card being used. Values for this parameter are defined in the include files(**comm.h, comm.lnc**, etc.). The values are as follows:

CARD_NORMAL	Standard PC COM1, COM2, COM3, and COM4. Also used for most dumb cards on the market.
CARD_INTELH	Intel HUB 6 multiport non-intelligent card.
CARD_CYCLOMY	Cyclades ISA & PCI Yx Cards.
CARD_DIGCXI	Digiboard COMXi multiport cards.
CARD_BOCA1610	BOCA 1610
CARD_DIGPCX	Digiboard PCX
CARD_GTEK	GTEK
CARD_INT14	3rd party INT14H board
CARD_WCSCVXD	WCSC high speed VxD for 8250 family. Used within standard PC COM1, COM2, COM3, and COM4. Also used for most dumb cards on the market. This setting requires that the **COMM-DRV/VxD** product be installed.
CARD_WINAPI	Use Windows 3.x, Windows 95, or Window NT's built in driver. Note that when this type is selected, **subport** must be used to specify the **COM** port (e.g., **COM1 => subport = 0, COM2 => subport = 1**, and so on).

unsigned short cardseg;

For most generic serial cards and multiport cards, this value is 0. It is usually the segment address of the shared memory used by smart multiport cards. Additionally it has special meanings for dumb cards used with the **CARD_NORMAL** or **CARD_WCSCVXD** cardtype. For the **BOCA Dumb Multiport Port Cards**, the value should be **ffff**. For the **WCSC AST Compatible Four Port Multiport Card(PCCOM4 or LCS 8650)** the value should be **1bf** or **2bf** depending on whether the first port on the card is 1a0 or 2a0 respectively. For the **AST Four Port Multiport Cards** the value should be **f1bf** or **f2bf** depending on whether the first port on the card is 1a0 or 2a0 respectively. For the **Cyclades' CyclomY ISA** cards it is the segment address of the card(the CD1400). For the **Cyclades' CyclomY PCI** it is the slot number in which the card is located.

unsigned short inbuflen;

Length of input communication buffer.

unsigned short outbuflen;

Length of output communication buffer.

unsigned short flag;

Normally this value is zero. It may be initialized to the following flags that further modify the behavior of **COMMDRV**. Values for this parameter are defined in the include files(**comm.h**, **comm.inc**, etc.). Flags may be orred together.

CHAIN_INT	If this bit is set, then if the interrupt was not generated by this port, then the interrupt is chained to the interrupt handler installed before **COMM-DRV**.
DISABLE_MODEMSIG	If this bit is set, then the DTR & RTS signals are disabled on initialization.

IGNORE_16550	If this bit is set, then the 16550 UART is not enabled if detected.
LEAVE_DTR	If this bit is set, then the DTR signal will be left at the state it was at before initialization.
LEAVE_RTS	If this bit is set, then the RTS signal will be left at the state it was at before initialization.
NOCHANGE_MODEMSIG	If this bit is set, then the DTR/RTS signals are left in their pre-installation state.
STOREALL	If this bit is set, then for every byte that is received, there are four bytes written to the receive buffer. These bytes are the data, line status register, modem status register, and port the data is from in this order.
STORESOME	If this bit is set, then for every byte that is received, there are two bytes written to the receive buffer. These bytes are the data, and port the data is from in this order.
LOCKBAUD	If this bit is set, then baud rate commands will only have an effect when the port is first opened.
POLLONSTATUS	If this bit is set COMM-DRV will poll the specific hardware anytime the **ser_rs232_getstatus()** function is called internally.

MULTIDROP	Use the multidrop protocol when outputting data. When data is outputted with **PutPacket()**, the **RTS** signal is dropped after all data has left the UART. This commonly used with multidrop configurations with RS485 devices. This is only functional when **cardtype = CARD_WCSCVXD**.
NINEBITPROTOCOL	When data is outputted with the **PutPacket()** function, it forces the first byte transmitted to have the parity bit set. All characters sent thereafter are sent normally. This is commonly referred to as the nine bit protocol. This is only functional when **cardtype = CARD_WCSCVXD**.
CUSTOMFLAG1	If this bit is set and the Arnet Smartport is being used, then an initial reset of the Arnet device is ignored.
CUSTOMFLAG2	If this bit is set and the Digiboard PC/Xe is being used, then the board will operate in 64K mode.

On Exit-

```
int     stat;
```

This function returns one of the **COMM-DRV/LIB** errors. The error codes are defined in **Appendix C** (Returns **RS232ERR_NONE** if successful).

See Also-

IsPortAvailable()
SetFlowControlCharacters()
SetFlowControlThreshold()
SetPortCharacteristics()
UnInitializePort()

Example-

```
#include <comm.h>

int     port=0;
int     subport=0;
int     addr=0x3f8;
int     irq=4;
int     cardtype = CARD_NORMAL;
int     cardseg=0x0000;
int     inbuflen=2048;
int     outbuflen=1048;
int     flag=0x0000;
int     stat;

stat = InitializePort(port,subport,addr,irq,cardtype,cardseg, inbuflen,outbuflen,flag);

/* Save old timeout */
if (stat != RS232ERR_NONE)
        {
        printf("Error #%d initializing port\n",stat);

        /* Take remedial action */
        }
```

IsAllDataOut

Description-

Returns true if all data has been transmitted as well as all data has left the serial hardware. This function differs from **IsTransmitBufferEmpty()** given that even though the transmit buffer may be empty, their still could be data leaving the serial hardware.

Syntax-

stat = IsAllDataOut(port);

On Entry-

int port;

Port previously opened with **InitializePort()**.

Return-

int stat;

0 If all data has not been transmitted.
1 If all data has been transmitted.

See Also-

IsCarrierDetect()
IsCts()
IsDsr()
IsFramingError()
IsInputOverrun()
IsOverrunError()
IsParityError()
IsRing()

Example-

```
#include <comm.h>

int             port=0;
int             stat;

if ((stat = IsAllDataOut(port)) == 1)
        {
        printf("All data was transmitted\n");
        }
```

IsBreak

Description-

Returns true if a break signal was detected since the last call to this function.

Syntax-

stat = IsBreak(port);

On Entry-

int port;

Port previously opened with **InitializePort()**.

Return-

int stat;

0 If break is not detected.
1 If break is detected.

See Also-

IsCarrierDetect()
IsCts()
IsDsr()
IsFramingError()
IsInputOverrun()
IsOverrunError()
IsParityError()
IsRing()

Example-

```
#include <comm.h>

int             port=0;
int             stat;

if ((stat = IsBreak(port)) == 1)
        {
        printf("Break occurred\n");

        }
```

IsCarrierDetect

Description-

Returns true if carrier detect signal is asserted.

Syntax-

stat = IsCarrierDetect(port);

On Entry-

int port;

Port previously opened with **InitializePort()**.

Return-

int stat;

0 If carrier is not detected.
1 If carrier is detected.

See Also-

IsBreak()
IsCts()
IsDsr()
IsFramingError()
IsInputOverrun()
IsOverrunError()
IsParityError()
IsRing()

Example-

```
#include<comm.h>

int             port=0;
int             stat;

if ((stat = IsCarrierDetect(port)) == 1)
        {
        printf("Carrier is present\n");

        }
```

IsCts

Description-

Returns true if CTS signal is high.

Syntax-

```
stat = IsCts(port);
```

On Entry-

```
int             port;
```

Port previously opened with **InitializePort()**.

On Exit-

```
int             stat;
```

0	If CTS signal is low.
1	If CTS signal is high.

See Also-

IsBreak()
IsCarrierDetect()
IsDsr()
IsFramingError()
IsInputOverrun()
IsOverrunError()
IsParityError()
IsRing()

Example-

```
#include <comm.h>

int             port=0;
int             stat;

if ((stat = IsCts(port)) == 1)
        {
        printf("CTS is present\n");

        }
```

IsDsr

Description-

Returns true if DSR signal is high.

Syntax-

stat = IsDsr(port);

On Entry-

int port;

Port previously opened with **InitializePort()**.

On Exit-

int stat;

0 If DSR signal is low.
1 If DSR signal is high.

See Also-

IsBreak()
IsCarrierDetect()
IsCts
IsFramingError()
IsInputOverrun()
IsOverrunError()
IsParityError()
IsRing()

Example-

```
#include <comm.h>

int             port=0;
int             stat;

if ((stat = IsDsr(port)) == 1)
        {
        printf("Dsr is present\n");

        }
```

IsFramingError

Description-

Returns true if a framing error occurred since the last call to this function.

Syntax-

stat = IsFramingError(port);

On Entry-

int port;

Port previously opened with **InitializePort()**.

On Exit-

int stat;

0 If framing error did not occur.
1 If framing error occurred.

See Also-

IsBreak()
IsCarrierDetect()
IsCts
IsDsr()
IsInputOverrun()
IsOverrunError()
IsParityError()
IsRing()

Example-

```
#include <comm.h>

int             port=0;
int             stat;

if ((stat = IsFramingError(port)) == 1)
        {
        printf("Framing Error occurred\n");

        }
```

IsInputOverrun

Description-

Returns true if an overrun error occurred since the last call to this function. There are two types of overrun. One is a hardware overrun which means that the UART got another character before **COMM-DRV/LIB** was able to process the previous character. The second type of overrun occurs if the **COMM-DRV/LIB** ring buffers (the buffer whose buffer size was given in the **InitializePort()** function). This function returns the ring buffer overrun condition. The function **IsOverrunError()** returns the hardware overrun condition.

Syntax-

stat = IsInputOverrun(port);

On Entry-

int port;

Port previously opened with **InitializePort()**.

On Exit-

int stat;

0 If overrun error did not occur.
1 If overrun error occurred.

See Also-

IsBreak()
IsCarrierDetect()
IsCts
IsDsr()
IsFramingError()
IsOverrunError()
IsParityError()
IsRing()

Example-

```
#include <comm.h>

int             port=0;
int             stat;

if ((stat = IsInputOverrun(port)) == 1)
        {
        printf("Overrun Error occurred\n");

        }
```

IsOverrunError

Description-

Returns true if an overrun error occurred since the last call to this function. There are two types of overrun. One is a hardware overrun which means that the UART got another character before **COMM-DRV/LIB** was able to process the previous character. The second type of overrun occurs if the **COMM-DRV/LIB** ring buffers (the buffer whose buffer size was given in the **InitializePort()** function). This function returns the hardware overrun condition. The function **IsInputOverrun()** returns the ring buffer overrun condition.

Syntax-

stat = IsOverrunError(port);

On Entry-

int port;

Port previously opened with **InitializePort()**.

On Exit-

int stat;

0 If overrun error did not occur.
1 If overrun error occurred.

See Also-

IsBreak()
IsCarrierDetect()
IsCts
IsDsr()
IsFramingError()
IsInputOverrun()
IsParityError()
IsRing()

Example-

```
#include <comm.h>

int             port=0;
int             stat;

if ((stat = IsOverrunError(port)) == 1)
        {
        printf("Overrun Error occurred\n");

        }
```

IsParityError

Description-

Returns true if a parity error occurred since the last call to this function.

Syntax-

stat = IsParityError(port);

On Entry-

int port;

Port previously opened with **InitializePort()**.

On Exit-

int stat;

0 If parity error did not occur.
1 If parity error occurred.

See Also-

IsBreak()
IsCarrierDetect()
IsCts
IsDsr()
IsFramingError()
IsInputOverrun()
IsOverrunError()
IsRing()

Example-

```
#include <comm.h>

int             port=0;
int             stat;

if ((stat = IsParityError(port)) == 1)
        {
        printf("Parity Error occurred\n");

        }
```

IsPortAvailable

Description-

Returns 0 if port was already initialized either with the **InitIalizePort()** function.

Syntax-

stat = IsPortAvailable(port);

On Entry-

int port;

Port previously opened with **InitIalizePort()**.

On Exit-

int stat;

1 If port is available for initialization.

See Also-

InitIalizePort()
UnInitIalizePort()

Example-

```
#include <comm.h>

int     port=0;
int     stat;

if ((stat = IsPortAvailable(port)) == 0)
        {
        printf("Port is already in use\n");

        }
```


IsReceiveBufferEmpty

Description-

Returns true if the receive buffer is empty.

Syntax-

stat = IsReceiveBufferEmpty(port);

On Entry-

int port;

Port previously opened with **InitIalizePort()**.

On Exit-

int stat;

0 If receive buffer is not empty.
1 If receive buffer is empty.

See Also-

ReceiveBufferSize()
SpaceInReceiveBuffer()

Example-

```
#include <comm.h>

int     port=0;
int     stat;

if ((stat = IsReceiveBufferEmpty(port)) == 1)
        {
        printf("Buffer is empty\n");

        }
```

IsRing

Description-

Returns true if a ring signal was detected since the last call to this function.

Syntax-

stat = IsRing(port);

On Entry-

int port;

Port previously opened with **InitializePort()**.

Return-

int stat;

0 If ring was not detected.
1 If ring was detected.

See Also-

IsCarrierDetect()
IsCts()
IsDsr()
IsFramingError()
IsInputOverrun()
IsOverrunError()
IsParityError()

Example-

```
#include <comm.h>

int             port=0;
int             stat;

if ((stat = IsRing(port)) == 1)
        {
        printf("Ring occurred\n");

        }
```

IsTransmitBufferEmpty

Description-

Returns true if the transmit buffer is empty.

Syntax-

stat = IsTransmitBufferEmpty(port);

On Entry-

int port;

Port previously opened with **InitializePort()**.

On Exit-

int stat;

0 If transmit buffer is not empty.
1 If transmit buffer is empty.

See Also-

SpaceInTransmitBuffer()
TransmitBufferSize()

Example-

```
#include <comm.h>

int             port=0;
int             stat;

if ((stat = IsTransmitBufferEmpty(port)) == 1)
        {
        printf("Buffer is empty\n");

        }
```

PeekChar

Description-

Returns the next character from receive buffer non-destructively. This function will wait for up to the amount of time set by **SetTimeout()** before returning failure if a byte is not present.

Syntax-

byte = PeekChar(port);

On Entry-

int port;

Port previously opened with **InitializePort()**.

On Exit-

int byte;

The returned byte.

else

-1 If port not initialized or no byte to be read.

See Also-

GetByte()
GetPacket()
GetString()

Example-

```
#include <comm.h>

int             port=0;
int             byte;

if ((byte = PeekChar(port)) == -1)
        {
        printf("No bytes to read\n");

        }
```

PutByte

Description-

Queues a byte for transmission. Causes a byte to be transmitted. This function will wait for up to the amount of time set by **SetTimeout()** before returning failure if a byte could not be outputted.

Syntax-

stat = PutByte(port,ch);

On Entry-

int port;

Port previously opened with **InitializePort()**.

char ch;

Character to transmit.

On Exit-

int stat;

0 Data queued successfully for transmission.

else

-1 If port was not initialized or output buffer is full.

See Also-

PutPacket()
PutString()

Example-

```
#include <comm.h>

int             port=0;
char            byte="A";

if (PutByte(port,byte) == -1)
        {
        printf("Transmit buffer full\n");

        }
```

PutPacket

Description-

Queues a packet for transmission. This function attempts to output a packet of desired length. Unlike the low level function **ser_rs232_putpacket()**, it will output a partial packet. Likewise, the buffer length specified during the **InitializePort()**, or **ser_rs232_setup()** may be less than the desired packet size. This function will wait for up to the amount of time set by **SetTimeout()** before returning failure if the entire packet is not written. It will also send the characters out at the pace specified by the **SetPaceTime()** function.

Syntax-

count = PutPacket(port,len,pkt);

On Entry-

int port;

Port previously opened with **InitializePort()**.

int len;

Length of packet to write.

char *pkt;

Pointer to buffer to with packet to transmit.

On Exit-

int count;

Length of packet actually transmitted.

else

-1 If port not initialized.

See Also-

PutByte()
PutString()

Example-

```
#include <comm.h>

int             port=0;
char            buf[]="ABCDEF";

if (PutPacket(port,sizeof(buf),buf) != sizeof(buf))
        {
        printf("Entire packet not outputted\n");

        }
```

PutString

Description-

This function outputs a null terminated string over the serial port. All characters preceding the null character are outputted. The null character is not outputted. This function will wait for up to the amount of time set by **SetTimeout()** before returning failure if the entire string is not written. It will also send the characters out at the pace specified by the **SetPaceTime()** function.

Syntax-

count = PutString(port,str);

On Entry-

int port;

Port previously opened with **InitializePort()**.

char *str;

Pointer to null terminated string to output.

On Exit-

int count;

-1 If all characters outputted successfully.

else

Actual number of characters outputted.

See Also-

PutByte()
PutPacket()

Example-

```
#include <comm.h>

int             port=0;
char            buf[]="ABCDEF";

if (PutString(port,buf) != -1)
        {
        printf("Entire string not outputted\n");

        }
```

ReceiveBufferSize

Description-

Returns the receive buffer size.

Syntax-

count = ReceiveBufferSize(port);

On Entry-

int port;

Port previously opened with **InitializePort()**.

On Exit-

int count;

Receive buffer size.

else

-1 If port not initialized.

See Also-

BytesInReceiveBuffer()
IsReceiveBufferEmpty()
SpaceInReceiveBuffer()

Example-

```
#include <comm.h>

int             count;
int             port=0;

if ((count = ReceiveBufferSize(port)) == -1)
        {
        printf("Port was not initialized\n");

        /* Take remedial action */
        }
else
        printf("Number of bytes =>%d\n",count);
```

RtsOff

Description-

Turns the RTS signal off.

Syntax-

stat = RtsOff(port);

On Entry-

int port;

Port previously opened with **InitializePort()**.

On Exit-

int stat;

-1 If port was not initialized.

See Also-

DtrOn()
RtsOff()
RtsOn()

Example-

```
#include <comm.h>

int             port=0;
int             stat;

if ((stat = RtsOff(port)) == -1)
        {
        printf("Port not initialized\n");

        /* Take remedial action */
        }
```

RtsOn

Description-

Turns the RTS signal on.

Syntax-

stat = RtsOn(port);

On Entry-

int port;

Port previously opened with ***InitializePort()***.

On Exit-

int stat;

-1 If port was not initialized.

See Also-

DtrOff()
DtrOn()
RtsOff()

Example-

```
#include <comm.h>

int             port=0;
int             stat;

if ((stat = RtsOn(port)) == -1)
        {
        printf("Port not initialized\n");

        /* Take remedial action */
        }
```

SendBreak

Description-

Sends a break signal for a specific duration.

Syntax-

stat = SendBreak(port,timeval);

On Entry-

int port;

Port previously opened with **InitializePort()**.

int timeval;

Length of time in timer tics to enable the break signal. The resolution of the timer tic is set by the function **CdrvSetTimerResolution()**.

On Exit-

int stat;

0 If break sent.

else

-1 If port not initialized.

See Also-

DtrOff()
DtrOn()
RtsOff()
RtsOn()

Example-

```
#include <comm.h>

int             port=0;
int             stat;

if ((stat = SendBreak(port,20)) == -1)
        {
        printf("Port not initialized\n");

        /* Take remedial action */
        }
```

SetBaud

Description-

This function sets the baud rate on a particular *port*.

Syntax-

stat = SetBaud(port,baud);

On Entry-

int port;

Port previously opened with **InitializePort()**.

unsigned short baud;

BAUD110	110 baud
BAUD150	150 baud
BAUD300	300 baud
BAUD600	600 baud
BAUD1200	1200 baud
BAUD2400	2400 baud
BAUD4800	4800 baud
BAUD9600	9600 baud
BAUD19200	19200 baud
BAUD38400	38400 baud
BAUD57600	57600 baud
BAUD115200	115200 baud

On Exit-

int stat;

This function returns one of the **COMM-DRV/LIB** errors. The error codes are defined in **Appendix C** (Returns **RS232ERR_NONE** if successful).

See Also-

SetPortCharacteristics()

Example-

```
#include <comm.h>


int             port=0;
int             stat;
unsigned short  baud=BAUD9600;

if ((stat = SetBaud(port,baud)) != RS232ERR_NONE)
        {
        printf("Port not initialized\n");

        /* Take remedial action */
        }
```

SetDataStreamFunction

Description-

This function is used to install a function that should be called back with data that was read by some of the high level functions and modem functions. Some of these functions are the **WaitFor()** family of functions, **Dial()**, **ModemConnect()**, etc.

Syntax-

stat = SetDataStreamFunction(port,fnc);

On Entry-

int port;

Port previously opened with **InitializePort()**.

void (*fnc)(int port,int count,char *buf);

Pointer to the callback function. If this value is 0L then the callback is disabled.

int port;

Port causing the callback.

int count;

Number of bytes passed.

char *buf;

Pointer to buffer with the data.

On Exit-

int stat;

0 If no error.

See Also-

Dial(), ModemAnswerMode(), ModemAttention(), ModemConnect()
ModemGetCarrierSpeed(), ModemGetConnectSpeed()
ModemHangup(), ModemInit(), ModemModifyString()
ModemModifyValue(), ModemWaitForCall(), WaitFor()
WaitForFixed(), WaitForPeek(), WaitForPeekFixed()
WaitForPeekTable(), WaitForPeekTableFixed()
WaitForTable(), WaitForTableFixed()

Example-

```
#include <comm.h>

int     port=0;
int     stat;
int     timeout=90;
char    *out="Password=>";
char    *in="AlphaBeta";

void fnc(int port,int count,char *buf);
        {
        // Display data being read by WaitFor()
        printf("%*.*s",count,count,buf);
        return(0);
        }

main()
{
SetDataStreamFunction(port,fnc);
stat = WaitFor(port,timeout,out,in);
SetDataStreamFunction(port,0L);
        :
        :
}
```

SetFlowControlCharacters

Description-

This function sets the characters that will be used for software flow control.

Syntax-

stat = SetFlowControlCharacters(port,xoff,xon,xxon,xxoff);

On Entry-

int port;

Desired port.

int xoff;

The character that will be interpreted as an XOFF character from the remote..

int xon;

The character that will be interpreted as an XON character from the remote.

int xxoff;

The character that will be transmitted as an XOFF character to the remote.

int xxon;

The character that will be transmitted as an XON character to the remote.

On Exit-

int stat;

0 If successful.

See Also-

SetFlowControlThreshold()

Example-

```
#include <comm.h>

#define  XOFF   10
#define  XON    13


int          port=0;
int          stat;
int          xon     =XON;
int          xoff    =XFOFF;
int          xon;    =XON;
int          xoff    =XOFF;

if ((stat = SetFlowControlCharacters(port,xon,xoff,xxon,xxof)) == -1)
        {
        printf("Port not initialized\n");

        /* Take remedial action */
        }
```

SetFlowControlThreshold

Description-

This function sets the flow control threshold. This is used for both software flow control(XON/XOFF), and hardware flow control(RTS/CTS,DTR/DSR).

Syntax-

stat = SetFlowControlThreshold(port,inbuf_low,inbuf_high);

On Exit-

int port;

Desired port.

int inbuf_low;

When the input communication buffer reaches this value, an XON will be sent if software handshaking was enabled, DTR will be raised if DTR hardware handshake was selected, or RTS will be raised if RTS handshake was selected. This tells the remote that it may begin sending data again.

int inbuf_high;

When the input communication buffer reaches this value, an XOFF will be sent if software handshaking was enabled, DTR will be lowered if DTR hardware handshake was selected, or RTS will be lowered if RTS handshake was selected. This tells the remote to stop sending data.

On Exit-

int stat;

0 If successful.

See Also-

SetFlowControlCharacters()

Example-

```
#include <comm.h>

int             port=0;
int             stat;

if ((stat = SetFlowControlThreshold(port,50,100)) == -1)
        {
        printf("Port not initialized\n");

        /* Take remedial action */
        }
```

SetPaceTime

Description-

Sets the current inter-character pace time. All the high level functions that transmit data will delay the specified amount of time between each character.

Syntax-

stat = SetPaceTime(port,timeval);

On Entry-

int port;

Port previously opened with **InitializePort()**.

int timeval;

Time in tics to delay between characters. The resolution of the timer tic is set by the function **CdrvSetTimerResolution()**.

On Exit-

int stat;

0 If successful.

See Also-

GetPaceTime()

Example-

```
#include <comm.h>

int             port=0;
int             stat;

if ((stat = SetPaceTime(port,20)) == -1)
        {
        printf("Port not initialized\n");

        /* Take remedial action */
        }
```

SetPortCharacteristics

Description-

This function sets the port characteristics. It is used to change baud rate, parity, length, stopbits, and line protocol of the port. The constants used for the arguments are defined in the include files(comm.h, comm.inc, etc.).

Syntax-

stat = SetPortCharacteristics(port,baud,parity,length,stopbit,protocol);

On Entry-

int port;

Desired port.

unsigned short baud;

BAUD110	110 baud
BAUD150	150 baud
BAUD300	300 baud
BAUD600	600 baud
BAUD1200	1200 baud
BAUD2400	2400 baud
BAUD4800	4800 baud
BAUD9600	9600 baud
BAUD14400	14400 baud
BAUD19200	19200 baud
BAUD28800	28800 baud
BAUD38400	38400 baud
BAUD57600	57600 baud
BAUD115200	115200 baud

unsigned short parity;

PAR_NONE	None parity
PAR_ODD	Odd parity
PAR_EVEN	Even parity
PAR_SODD	Sticky odd(MARK Parity)
PAR_SEVEN	Sticky even(SPACE Parity)

unsigned short length;

LENGTH_5	5 bits
LENGTH_6	6 bits
LENGTH_7	7 bits
LENGTH_8	8 bits

unsigned short Stopbit;

STOPBIT_1	1 stopbit
STOPBIT_2	2 stopbits

unsigned short Protocol;

PROT_RTSRTS	Local CTS/RTS handshake-Remote CTS/RTS handshake.
PROT_RTSXON	Local CTS/RTS handshake-Remote XON/XOFF handshake.
PROT_RTSDTR	Local CTS/RTS handshake-Remote DSR/DTR handshake.
PROT_RTSNON	Local CTS/RTS handshake-Remote no handshake.
PROT_NONNON	Local no protocol-Remote no protocol.
PROT_NONXON	Local no protocol-XON/XOFF handshake.
PROT_XONNON	Local XOFF/XON protocol-Remote no protocol.
PROT_XONXON	Local XON/XOFF protocol-Remote XON/XOFF protocol.
PROT_DTRNON	Local DSR/DTR protocol-Remote no protocol.
PROT_DTRRTS	Local DSR/DTR protocol-Remote CTS/RTS protocol.
PROT_DTRDTR	Local DSR/DTR protocol-Remote DSR/DTR protocol.
PROT_DTRXON	Local DSR/DTR protocol-Remote DSR/DTR protocol.
PROT_NONRTS	Local no protocol-Remote CTS/RTS protocol.

PROT_NONDTR	Local no protocol-Remote DSR/DTR protocol.
PROT_XONRTS	Local XOFF/XON protocol-Remote CTS/RTS protocol.
PROT_XONDTR	Local XOFF/XON protocol-Remote DSR/DTR protocol.

The first three bytes specifies the protocol the local computer is using, while the last three bytes specifies the protocol the remote device is expected to be using. PROT_RTS???, PROT_DTR???, and PROT_XON??? means that when the local computer's input buffer is filling up, the local computer will de-assert RTS, DTR, or send an XOFF respectively to tell the remote device to stop sending. The local machine will re-assert RTS, DTR, or send an XON to make remote device restart transmission. PROT_???RTS, PROT_???DTR, and PROT_???XON means that when the remote device's input buffer is filling up, the remote device is expected to de-assert RTS, DTR, or send an XOFF respectively to tell the local computer to stop sending. The remote device is expected to re-assert RTS, DTR, or send an XON to make the local computer restart transmission.(Default PROT_RTSRTS).

On Exit-

```
int             stat;
```

This function returns one of the **COMM-DRV/LIB** errors. The error codes are defined in **Appendix C** (Returns **RS232ERR_NONE** if successful).

See Also-

InitializePort()
SetFlowControlCharacters()
SetFlowControlThreshold()
SetBaud()

Example-

```
#include <comm.h>

int             port=0;
int             stat;
int             baud=BAUD9600;
int             parity = PAR_NONE;
int             length=LENGTH_8;
int             stopbit=STOPBIT_1;
int             protocol = PROT_RTSRTS;

if ((stat = SetPortCharacteristics(port,baud,parity,length,stopbit,
protocol)) != RS232ERR_NONE))
        {
        printf("Error setting port characteristics\n");

        /* Take remedial action */
        }
```

SetSpecialBehavior

Description-

This function is used to change the behavior of several of the high level functions.

Syntax-

stat = SetSpecialBehavior(port,command,value1,value2);

On Entry-

int port;

Port previously opened with **InitiallzePort()**.

int command;

This variable is set to one of the following symbols. Note that **value1** and **value2** contain the values or symbols as explained next to the **command** symbol.

unsigned long value1;
unsigned long value2;

value1 and **value2** depend on the particular **command** selected.

command = SB_SetGetStringDelimiter

Sets a new set of delimiters that **GetString()** will use as line terminators. If the port is reopened, the default string behavior returns.

value1= Number of characters in the list of delimiters.
value2= Pointer to the array with the list of delimiters. Note that the array that this value points to must remain instantiated for the duration of the session .

Example:

```
#include  <comm.h>

char      NewDelim[]={'\0','*','\r','\t','\n'};
char      str[20];
int       port=0;

SetSpecialBehavior(port,SB_SetGetStringDelimiter,sizeof(NewDelim),NewDelim);
GetString(port,sizeof(str),str);
```

command = SB_DoNotCall2ndMsgLoop

Controls whether a secondary message loop will get called while functions that may be delayed are called(all functions that directly or indirectly call **CdrvCheckTime()**).

value1= **CDRV_TRUE** to disable secondary message loop calls by the **CdrvCheckTime()** function. This will prevent other programs from running in Windows 3.x unless the user uses a callback function installed with the function **CdrvSetTimeoutFunction()** that yields control. **CDRV_FALSE** will cause the secondary message loop to be called while functions with substantial delays are called.
value2= 0

Example:

```
#include  <comm.h>

int       port=0;

SetSpecialBehavior(port,SB_DoNotCall2ndMsgLoop,CDRV_TRUE,0);
```

command = SB_DisableTiming

Controls whether **CdrvCheckTime()** will actually keep time. If the timing is disabled, then **CdrvCheckTime()** will never cause a timeout unless a timeout function was installed that will effect the timeout.

value1= **CDRV_TRUE** to prevents the function **CdrvCheckTime()** from doing any timing. This will prevent time-outs from occurring unless the user uses a callback function installed with the function **CdrvSetTimeoutFunction()** that effects a timeout. **CDRV_FALSE** will cause timing to function again.

value2= 0

Example:

```
#include  <comm.h>

int       port=0;

SetSpecialBehavior(port,SB_DisableTiming,CDRV_TRUE,0);
```

command = SB_SwitchOnDelay

Controls whether **CdrvCheckTime()** will cause a context switch in a multitasking environment while waiting for a timeout. In effect will prevent wasted CPU cycles on functions like WaitFor() or any functions that may be delayed.

value1= **CDRV_TRUE** forces a context switch if when any function is waiting. **CDRV_FALSE** will stop the forcing of context switches on functions that may be waiting.

value2= 0

Example:

```
#include  <comm.h>

int       port=0;

SetSpecialBehavior(port,SB_SwitchOnDelay,CDRV_TRUE,0);
```


On Exit-

int stat

0 if no error, -1 if invalid **command**, -2 if invalid **value1**, -3 if invalid **value2**.

See Also-

Example-

See examples for each individual command.

SetTimeout

Description-

Sets the current transmit/receive time-outs. This is the amount of time that the high level functions that send and receive data will delay before returning an error, if they cannot send data (because output buffer may be full) or did not receive data respectively.

Syntax-

stat = SetTimeout(port,timeval);

On Entry-

int port;

Port previously opened with **InitializePort()**.

int timeval;

Time-out in timer tics. The resolution of the timer tic is set by the function **CdrvSetTimerResolution()**.

On Exit-

int stat;

0 If successful.

else

-1 If port not initialized.

See Also-

GetTimeout()

Example-

```
#include <comm.h>

int             port=0;
int             stat;

if ((stat = SetTimeout(port,20)) == -1)
        {
        printf("Port not initialized\n");

        /* Take remedial action */
        }
```

SpaceInReceiveBuffer

Description-

Returns space unused in receive buffer.

Syntax-

count = SpaceInReceiveBuffer(port);

On Entry-

int port;

Port previously opened with **InitializePort()**.

On Exit-

int count;

Space available in the receive buffer.

else

-1 If port was not initialize.

See Also-

BytesInReceiveBuffer()
ReceiveBufferSize()
IsReceiveBufferEmpty()

Example-

```
#include<comm.h>

int             count;
int             port=0;

if ((count = SpaceInReceiveBuffer(port)) == -1)
        {
        printf("Port was not initialized\n");

        /* Take remedial action */
        }
else
        printf("Number of bytes =>%d\n",count);
```

SpaceInTransmitBuffer

Description-

Returns space unused in transmit buffer.

Syntax-

count = SpaceInTransmitBuffer(port);

On Entry-

int port;

Port previously opened with **InitializePort()**.

On Exit-

int count;

Space available in the transmit buffer.

else

-1 If port is not active.

See Also-

BytesInTransmitBuffer()
TransmitBufferSize()
IsTransmitBufferEmpty()

Example-

```
#include <comm.h>

int             count;
int             port=0;

if ((count = SpaceInTransmitBuffer(port)) == -1)
        {
        printf("Port was not initialized\n");

        /* Take remedial action */
        }
else
        printf("Number of bytes =>%d\n",count);
```

TransmitBufferSize

Description-

Returns transmit buffer size.

Syntax-

count = TransmitBufferSize(port);

On Entry-

int port;

Port previously opened with **InitializePort()**.

On Exit-

int count;

Transmit buffer size.

else

-1 If port was not initialized.

See Also-

BytesInTransmitBuffer()
IsTransmitBufferEmpty()
SpaceInTransmitBuffer()

Example-

```
#include <comm.h>

int             count;
int             port=0;

if ((count = TransmitBufferSize(port)) == -1)
        {
        printf("Port was not initialized\n");

        /* Take remedial action */
        }
else
        printf("Number of bytes =>%d\n",count);
```

UnInitializePort

Description-

This function un-initializes a port that was previously setup with the **InitializePort()** function. It releases any resources used(memory etc.).

Syntax-

stat = UnInitializePort(port);

On Entry-

int port;

Port previously opened with **InitializePort()**.

On Exit-

int stat;

0 If successful.

See Also-

InitializePort()

Example-

```
#include <comm.h>

int     stat;
int     port=0;

if ((stat = UnInitializePort(port)) != 0)
        {
        printf("Error cleaning up port\n");

        /* Take remedial action */
        }
```


WaitFor

Description-

Outputs a null terminated string and waits for a specific response. The null is not outputted. Note that the string sent could be "" in effect sending nothing and simply waiting for a response. This function will wait for up to the amount of time set by **SetTimeout()** before returning failure if the entire string is not written. It will also send the characters out at the pace specified by the **SetPaceTime()** function. It will wait for up to the amount of time specified by the timeout parameter.

Syntax-

stat = WaitFor(port,timeout,out,in);

On Entry-

int port;

Port previously opened with **InitializePort()**.

int timeout;

Number of tics to wait for a response. The tic resolution may be set by the function **CdrvSetTimerResolution()**.

char *out;

Pointer to the string to output. It may be an empty string if all that is desired is the specific response.

char *in;

Pointer to the comparator. This is the string that the serial stream will be compared against. ***All the data up to the last character corresponding to the matching string is thrown away.***

On Exit-

int stat;

0 If successfully matched the string.

See Also-

WaitForFixed()
WaitForPeek()
WaitForPeekFixed()
WaitForPeekTable()
WaitForPeekTableFixed()
WaitForTable()
WaitForTableFixed()

Example-

```
#include <comm.h>

int             stat;
int             port=0;
int             timeout=90;

/* Example Logging to a BBS */
if ((stat = WaitFor(port,timeout,"\r","Username:")) != 0)
        {
        printf("Logging failure\n");

        /* Take remedial action */
        }
if ((stat = WaitFor(port,timeout,"USER1\r","Password:")) != 0)
        {
        printf("Logging failure\n");

        /* Take remedial action */
        }
if ((stat = PutString(port,"X34575\r")) != 0)
        {
        printf("Port not initialized\n");

        /* Take remedial action */
        }
```


WaitForFixed

Description-

Outputs a fixed length string and waits for a specific response. It differs from **WaitFor()** in that all characters including NULLs may be used. This function will wait for up to the amount of time set by **SetTimeout()** before returning failure if the entire string is not written. It will also send the characters out at the pace specified by the **SetPaceTime()** function. It will wait for up to the amount of time specified by the timeout parameter.

Syntax-

stat = WaitForFixed(port,timeout,out,in,outcnt,incnt);

On Entry-

int port;

Port previously opened with **InitiallzePort()**.

int timeout;

Number of tics to wait for a response. The tic resolution may be set by the function **CdrvSetTimerResolution()**.

char *out;

Pointer to the string to output. It may be an empty string if all that is desired is the specific response.

char *in;

Pointer to the comparator. This is the string that the serial stream will be compared against. ***All the data up to the last character corresponding to the matching string is thrown away.***

int outcnt;

Number of bytes in "out".

int incnt;

Number of bytes in "in".

On Exit-

int stat;

0 If successfully matched the string.

See Also-

WaitFor()
WaitForPeek()
WaitForPeekFixed()
WaitForPeekTable()
WaitForPeekTableFixed()
WaitForTable()
WaitForTableFixed()

Example-

```
#include <comm.h>

int             stat;
int             port=0;
int             timeout=90;
static char     out[]={'\r','\0','\t'};
static char     in[]={'\r','\n'}

/* Example Logging to a BBS */
if ((stat = WaitForFixed(port,timeout,out,in,sizeof(out),sizeof(in))) != 0)
        {
        printf("No Match Found\n");

        /* Take remedial action */
        }
```


WaitForPeek

Description-

Outputs a null terminated string and waits for a specific response. The null is not outputted. Note that the string sent could be "" in effect sending nothing and simply waiting for a response. This function will wait for up to the amount of time set by **SetTimeout()** before returning failure if the entire string is not written. It will also send the characters out at the pace specified by the **SetPaceTime()** function. It will wait for up to the amount of time specified by the timeout parameter. This function differs from **WaitFor()** in that it does not remove any data from the data stream. As such it can scan a block of data no larger than the receive buffer size that was set with the **InitializePort()** function.

Syntax-

stat = WaitForPeek(port,timeout,out,in);

On Entry-

int port;

Port previously opened with **InitializePort()**.

int timeout;

Number of tics to wait for a response. The tic resolution may be set by the function **CdrvSetTimerResolution()**.

char *out;

Pointer to the string to output. It may be an empty string if all that is desired is the specific response.

char *in;

Pointer to the comparator. This is the string that the serial stream will be compared against.

On Exit-

```
int             stat;
```

0 If successfully matched the string.

See Also-

WaitFor()
WaitForFixed()
WaitForPeek()
WaitForPeekFixed()
WaitForPeekTable()
WaitForPeekTableFixed()
WaitForTable()
WaitForTableFixed()

Example-

```
#include <comm.h>

int             stat;
int             port=0;
int             timeout=90;

if ((stat = WaitForPeek(port,timeout,"get squote\r","Stock Quote:")) != 0)
        {
        printf("Match not found\n");

        /* Take remedial action */
        }
```


WaitForPeekFixed

Description-

Outputs a fixed length string and waits for a specific response. This function differs from **WaitForPeek()** in that NULL characters are allowed in the string. This function will wait for up to the amount of time set by **SetTimeout()** before returning failure if the entire string is not written. It will also send the characters out at the pace specified by the ***SetPaceTime()*** *function. It will wait for up to the amount of time specified by the time-*out parameter. This function differs from **WaitForFixed()** in that it does not remove any data from the data stream. As such it can scan a block of data no larger than the receive buffer size that was set with the **InitializePort()** function.

Syntax-

```
stat = WaitForPeekFixed(port,timeout,out,in,outcnt,incnt);
```

On Entry-

```
int             port;
```

Port previously opened with **InitializePort()**.

```
int             timeout;
```

Number of tics to wait for a response. The tic resolution may be set by the function **CdrvSetTimerResolution()**.

```
char            *out;
```

Pointer to the string to output. It may be an empty string if all that is desired is the specific response.

```
char            *in;
```

Pointer to the comparator. This is the string that the serial stream will be compared against.

int outcnt;

Number of bytes in "out".

int incnt;

Number of bytes in "in".

On Exit-

int stat;

0 If successfully matched the string.

See Also-

WaitFor()
WaitForFixed()
WaitForPeek()
WaitForPeekTable()
WaitForPeekTableFixed()
WaitForTable()
WaitForTableFixed()

Example-

```
#include <comm.h>

int          stat;
int          port=0;
int          timeout=90;
static char  out[]={'\r','\0','\t'};
static char  in[]={'\r','\n'}

/* Example Logging to a BBS */
if ((stat = WaitForPeekFixed(port,timeout,out,in,sizeof(out),sizeof(in))) != 0)
        {
        printf("No Match Found\n");

        /* Take remedial action */
        }
```


WaitForPeekTable

Description-

Outputs a null terminated string and waits for a specific response that matches one of the strings in the passed table of strings. The null is not outputted. Note that the string sent could be "" in effect sending nothing and simply waiting for a response. This function will wait for up to the amount of time set by **SetTimeout()** before returning failure if the entire string is not written. It will also send the characters out at the pace specified by the **SetPaceTime()** function. It will wait for up to the amount of time specified by the timeout parameter. This function differs from **WaitForTable()** in that it does not remove any data from the data stream. As such it can scan a block of data no larger than the receive buffer size that was set with the **InitializePort()** function.

Syntax-

stat = WaitForPeekTable(port,timeout,out,in);

On Entry-

int port;

Port previously opened with **InitializePort()**.

int timeout;

Number of tics to wait for a response. The tic resolution may be set by the function **CdrvSetTimerResolution()**.

char *out;

Pointer to the string to output. It may be an empty string if all that is desired is the specific response.

char **in;

Pointer to an array of strings to compare against the incoming serial data. The last element in the array must be a null pointer.

On Exit-

int stat;

-1 If no match found.

If one of the strings match the incoming stream, then the return value corresponds to the index of the string that matched.

See Also-

WaitFor()
WaitForFixed()
WaitForPeek()
WaitForPeekFixed()
WaitForPeekTableFixed()
WaitForTable()
WaitForTableFixed()

Example-

```
#include   <comm.h>

int             stat;
int             port=0;
int             timeout=90;
static char     *In[]=
                        {
                        /* 0 */     "CARRIER",
                        /* 1 */     "CONNECT",
                        /* 2 */     (char *)0
                        };

if ((stat = WaitForPeekTable(port,timeout,"ATDT568-3334\r",in)) == -1)
        {
        printf("Match not found\n");

        /* Take remedial action */
        }
```



```
switch(stat)
        {
        case    0:
        printf("CARRIER found in stream\n");
        break;

        case    1:
        prinif("CONNECT found in stream\n");
        break;
        }
```

WaitForPeekTableFixed

Description-

Outputs a fixed length string and waits for a specific response that matches one of the strings in the passed table of strings. This function differs from **WaitForPeekTable()** in that NULLs are allowed in the strings. This function will wait for up to the amount of time set by **SetTimeout()** before returning failure if the entire string is not written. It will also send the characters out at the pace specified by the **SetPaceTime()** function. It will wait for up to the amount of time specified by the timeout parameter. This function differs from **WaitForTableFixed()** in that it does not remove any data from the data stream. As such it can scan a block of data no larger than the receive buffer size that was set with the **InitializePort()** function.

Syntax-

stat = WaitForPeekTableFixed(port,timeout,out,in,outcnt,incnt);

On Entry-

int port;

Port previously opened with **InitializePort()**.

int timeout;

Number of tics to wait for a response. The tic resolution may be set by the function **CdrvSetTimerResolution()**.

char *out;

Pointer to the string to output. It may be an empty string if all that is desired is the specific response.

char **in;

Pointer to an array of strings to compare against the incoming serial data. The last element in the array must be a null pointer.

int outcnt;

Number of bytes in "out".

int *incnt;

Pointer to an array of byte counts corresponding to each string define in the table "in".

On Exit-

int stat;

-1 If no match found.

If one of the strings match the incoming stream, then the return value corresponds to the index of the string that matched.

See Also-

WaitFor()
WaitForFixed()
WaitForPeek()
WaitForPeekFixed()
WaitForPeekTable()
WaitForTable()
WaitForTableFixed()

Example-

```
#include  <comm.h>

int             stat;
int             port=0;
int             timeout=90;
static char     out[]={'\r','\0','\t'};
static char     in1[]={'\0','\t'};
static char     in2[]="Alpha";
static char     in3[]="BETA";
static char     *in[]={in1,in2,in3,(char *)0};
static int      incnt[]={sizeof(in1),sizeof(in2),sizeof(in3),0};
int             outcnt=sizeof(out);

/* Example Logging to a BBS */
if ((stat = WaitForPeekTableFixed(port,timeout,out,in,outcnt,incnt)) != 0)
        {
        printf("No Match Found\n");

        /* Take remedial action */
        }
```


WaitForTable

Description-

Outputs a null terminated string and waits for a specific response that matches one of the strings in the passed table of strings. The null is not outputted. Note that the string sent could be "" in effect sending nothing and simply waiting for a response. This function will wait for up to the amount of time set by **SetTimeout()** before returning failure if the entire string is not written. It will also send the characters out at the pace specified by the **SetPaceTime()** function. It will wait for up to the amount of time specified by the timeout parameter.

Syntax-

stat = WaitForTable(port,timeout,out,in);

On Entry-

int port;

Port previously opened with **InitializePort()**.

int timeout;

Number of tics to wait for a response. The tic resolution may be set by the function **CdrvSetTimerResolution()**.

char *out;

Pointer to the string to output. It may be an empty string if all that is desired is the specific response.

char **in;

Pointer to an array of strings to compare against the incoming serial data. The last element in the array must be a null pointer. ***All the data up to the last character corresponding to the matching string is thrown away.***

On Exit-

int stat;

-1 If no match found.

If one of the strings match the incoming stream, then the return value corresponds to the index of the string that matched.

See Also-

WaitFor()
WaitForFixed()
WaitForPeek()
WaitForPeekFixed()
WaitForPeekTable()
WaitForPeekTableFixed()
WaitForTable()
WaitForTableFixed()

Example-

```
#include   <comm.h>

int                stat;
int                port=0;
int                timeout=90;
static char        *in[]=
                        {
                        /* 0 */     "CARRIER",
                        /* 1 */     "CONNECT",
                        /* 2 */     (char *)0
                        };

if ((stat = WaitForTable(port,timeout,"ATDT568-3334\r",in)) == -1)
        {
        printf("Match not found\n");

        /* Take remedial action */
        }
switch(stat)
        {
        case    0:
        printf("CARRIER found in stream\n");
        break;

        case    1:
        printf("CONNECT found in stream\n");
        break;
        }
```

WaitForTableFixed

Description-

Outputs a fixed length string and waits for a specific response that matches one of the strings in the passed table of strings. This function differs from **WaitForTable()** in that NULL characters are allowed in the string. This function will wait for up to the amount of time set by **SetTimeout()** before returning failure if the entire string is not written. It will also send the characters out at the pace specified by the **SetPaceTime()** function. It will wait for up to the amount of time specified by the timeout parameter.

Syntax-

stat = WaitForTableFixed(port,timeout,out,in,outcnt,incnt);

On Entry-

int port;

Port previously opened with **InitializePort()**.

int timeout;

Number of tics to wait for a response. The tic resolution may be set by the function **CdrvSetTimerResolution()**.

char *out;

Pointer to the string to output. It may be an empty string if all that is desired is the specific response.

char **in;

Pointer to an array of strings to compare against the incoming serial data. The last element in the array must be a null pointer. ***All the data up to the last character corresponding to the matching string is thrown away.***

int outcnt;

Number of bytes in "out".

int *incnt;

Pointer to an array of byte counts corresponding to each string define in the table "**in**".

On Exit-

int stat;

-1 If no match found.

If one of the strings match the incoming stream, then the return value corresponds to the index of the string that matched.

See Also-

WaitFor()
WaitForFixed()
WaitForPeek()
WaitForPeekFixed()
WaitForPeekTable()
WaitForPeekTableFixed()
WaitForTable()

Example-

```
#include   <comm.h>

int             stat;
int             port=0;
int             timeout=90;
static char     out[]={'\r','\0','\t'};
static char     in1[]={'\0','\t'};
static char     in2[]="Alpha";
static char     in3[]="BETA";
static char     *in[]={in1,in2,in3,(char *)0};
static int      incnt[]={sizeof(in1),sizeof(in2),sizeof(in3),0};
int             outcnt=sizeof(out);

/* Example Logging to a BBS */
if ((stat = WaitForTableFixed(port,timeout,out,in,outcnt,incnt)) != 0)
        {
        printf("No Match Found\n");

        /* Take remedial action */
        }
```

Modem Functions

These functions are the COMM-DRV/LIB modem functions. These functions provide the standard Hayes command set to the modem.

Dial()	Dials using the modem with the Hayes command set.
ModemAnswerMode()	Puts modem in answer mode.
ModemAttention()	Puts modem in command state.
ModemCommandState()	Puts the modem in the command state.
ModemConnect()	Returns true if modem connection attained.
ModemForceAnswer()	Forces modem to answer the phone.
ModemGetCarrierSpeed()	Returns carrier speed.
ModemGetConnectSpeed()	Returns connect speed.
ModemGetSRegister()	Get modem's selected S register.
ModemGetString()	Returns the particular modem string.
ModemGetValue()	Returns the particular modem string.
ModemHangup()	Hang-up modem connection.
ModemInit()	Set modem initialization string.
ModemModifyString()	Modify modem string.
ModemModifyValue()	Modify modem parameters.
ModemOffHook()	Take the modem line off hook.
ModemOnline()	Puts the modem back online from command state.
ModemSendCommand()	Sends a command to the modem.
ModemSetSRegister()	Writes to modem's selected S register.
ModemSpeaker()	Turns modem speaker on/off.
ModemVolume()	Adjust modem speaker volume.
ModemWaitForCall()	Wait for a call.
ModemWaitForRing()	Waits for the modem's phone line to ring.

Dial

Description-

Dials the specified telephone number. It is assumed that a Hayes compatible modem is being used. This function forces the modem back to the command state and then dials.

Syntax-

stat = Dial(port,mode,telephone);

On Entry-

int port;

Port previously opened with **InitializePort()**.

int mode;

0 Pulse dialing.
1 Tone dialing.

char far *telephone;

String with telephone number.

On Exit-

int stat;

0 If the telephone number successfully dialed.

else

-1 if the modem did not respond or the port was not initialized.

See Also-

ModemAnswerMode()
ModemAttention()
ModemConnect()
ModemHangup()
ModemModifyString()

Example-

```
#include <comm.h>

int     stat;
int     port=0;
int     mode=1;
char    *telephone="1-713-568-6401";

if ((stat = Dial(port,mode,telephone)) == -1)
        {
        printf("Modem not responding\n");

        /* Take corrective action */
        }
```


ModemAnswerMode

Description-

Puts the modem in auto answer mode. This command assumes the modem is in the command state. The modem remains in the command state at the end of this function.

Syntax-

stat = ModemAnswerMode(port,ring_count);

On Entry-

int port;

Port previously opened with **InitializePort()**.

int ring_count;

Number of rings before modem answers the phone.

On Exit-

int stat;

-1 If port was not initialized.

See Also-

ModemCommandState()
ModemConnect()
ModemGetCarrierSpeed()
ModemGetConnectSpeed()
ModemHangup()
ModemInit()
ModemModifyString()

Example-

```
#include <comm.h>

int             port=0;
int             stat;

if ((stat = ModemAnswerMode(port,1)) == -1)
        {
        printf("Error putting modem in answer mode.\n");

        /* Take remedial action */
        }
```


ModemAttention

Description-

This function puts the modem in the command state & verbose mode. It first does this by sending "+++". If the modem does not respond to "+++" it will toggle the DTR signal. If the modem still does not respond, it returns an error. This function can be used to determine if a modem is on a particular port.

Syntax-

stat = ModemAttention(port);

On Entry-

int port;

Port previously opened with **InitializePort()**.

On Exit-

int stat;

0 If successful.

else

-1 If port was not initialized or modem does not respond.

See Also-

Dial()
ModemAnswerMode()
ModemConnect()
ModemGetCarrierSpeed()
ModemGetConnectSpeed()
ModemHangup()
ModemInit()
ModemModifyString()

Example-

```
#include <comm.h>

int             port=0;
int             stat;

if ((stat = ModemAttention(port)) == -1)
        {
        printf("Error getting modem's attention.\n");

        /* Take remedial action */
        }
```


ModemCommandState

Description-

This function puts the modem in the command state. It does so by sending the modem escape sequence. The modem can be returned to the online state by issuing the **ModemOnline()** function.

Syntax-

stat = ModemCommandState(port);

On Entry-

int port;

Port previously opened with **InitializePort()**.

On Exit-

int stat;

0 If successful.

else

-1 If error.

See Also-

ModemOnline()
ModemSendCommand()

Example-

```
#include <comm.h>

int             port=0;
int             stat;
char            cmd="R1&B1";

if ((stat = ModemCommandState(port)) != 0)
        {
        printf("Error putting modem in command state\n");

        /* Take remedial action */
        }
if ((stat = ModemSendCommand(port,cmd)) != 0)
        {
        printf("Error sending command to modem\n");

        /* Take remedial action */
        }
```


ModemConnect

Description-

Returns true if the modem connection was successful. This call is generally made after a call to **Dial()** if programmer is initiating a session. Otherwise it may be called simply to detect when the modem has successfully connected to another modem. It first scans for the message **"CONNECT"** (or the connect token set with the **ModemModifyString()** function) from the port. If it does not get it will check to see if the carrier detect signal is present. If the **"CONNECT"** string or the carrier detect signal is present then it assumes that the modem is connected to another modem and returns successful. The modem is in the online state after this function returns.

Note: After some modems connect, they switch to the carrier speed(baud rate between two modems over the telephone line). As an example, if you are connected to your modem at 9600 baud(baud rate between the computer and the modem), and it connects to a 2400 baud modem, after it connects, and sends the connect message, the modem may switch its baud to 2400. If a modem of this type is used, the programmer must then change the baud to 2400 baud with the **SetBaud()** function. The carrier speed may be queried with the **ModemGetCarrierSpeed()**.

Syntax-

```
stat = ModemConnect(port);
```

On Entry-

```
int             port;
```

Port previously opened with **InitializePort()**.

On Exit-

```
int             stat;

        0       If successful.

        else

        -1      If port was not initialized.
```

-2	If modem not responding.
-3	If no dial tone
-4	If number dialed is busy.

See Also-

Dial()
ModemAnswerMode()
ModemAttention()
ModemGetCarrierSpeed()
ModemGetConnectSpeed()
ModemHangup()
ModemInit()
ModemModifyString()

Example-

```
#include <comm.h>

int             port=0;
int             stat;
int             mode=1;
char            *telephone="1-713-568-6401";

if ((stat = ModemAttention(port)) == -1)
        {
        printf("Error getting modem's attention.\n");

        /* Take remedial action */
        }

if ((stat = Dial(port,mode,telephone)) == -1)
        {
        printf("Modem not responding\n");

        /* Take corrective action */
        }

if ((stat = (ModemConnect(port)) != 0)
        {
```



```
switch(stat)
        {
        case    -1:
        printf("Port not initialized .\n");
        break;

        case    -2:
        printf("Modem not responding .\n");
        break

        case    -3:
        printf("No Dial Tone.\n");
        break;

        case    -4:
        printf("Line Busy .\n");
        break;

        default:
        break;
        }

/* Take remedial action */
}
```

ModemForceAnswer

Description-

Forces the modem to answer the phone. This command assumes the modem is in the command state. The modem remains in the command state at the end of this function.

Syntax-

stat = ModemForceAnswer(port);

On Entry-

int port;

Port previously opened with **InitializePort()**.

On Exit-

int stat;

0 If modem successfully connected to another modem.

else

-1 If error.

See Also-

ModemCommandState()
ModemWaitForRing()

Example-

```
#include <comm.h>

int     port=0;
int     stat;
int     RingCount=3;
int     Timeout=180;


if ((stat = ModemWaitForRing(port,RingCount,Timeout)) != 0)
        {
        printf("Time-out waiting for a ring\n");

        /* Take remedial action */
        }
if ((stat = ModemForceAnswer(port)) != 0)
        {
        printf("Modems could not connect\n");

        /* Take remedial action */
        }
```

ModemGetCarrierSpeed

Description-

Returns the carrier speed at which the modem last connected(baud rate between the two modems over the telephone line). The speed returned by this function is only valid if a previous call to **ModemConnect()** was successful.

Syntax-

speed =ModemGetCarrierSpeed(port);

On Entry-

int port;

Port previously opened with **InitializePort()**.

On Exit-

long speed;

The actual carrier speed that was returned by the modem the last time it successfully connected to another modem. This value is a positive decimal value. If a -1 is returned then the port was not initialized.

See Also-

Dial()
ModemAnswerMode()
ModemAttention()
ModemConnect()
ModemGetConnectSpeed()
ModemHangup()
ModemInit()
ModemModifyString()

Example-

```
#include <comm.h>

int         port=0;
int         stat;
int         mode=1;
long        speed;
char        *telephone="1-713-568-6401";

if ((stat = ModemAttention(port)) == -1)
        {
        printf("Error getting modem's attention.\n");

        /* Take remedial action */
        }

if ((stat = Dial(port,mode,telephone)) == -1)
        {
        printf("Modem not responding\n");

        /* Take remedial action */
        }

if ((stat = (ModemConnect(port)) != 0)
        {
        printf("Error connecting to modem\n");

        /* Take remedial action */
        }

if ((speed = ModemGetCarrierSpeed(port)) != -1L)
        {
        switch(speed)
                {
                case    2400:
                SetBaud(port,BAUD2400);
                break;
                        :
```

```
        :

        case    9600:
        SetBaud(port,BAUD9600);
        break;

        default:
        break;
        }

/* Continue with application */
}
```


ModemGetConnectSpeed

Description-

Returns the connect speed at which the modem last connected(baud rate between the computer and the modem). The speed returned by this function is only valid if a previous call to **ModemConnect()** was successful.

Syntax-

speed =ModemGetConnectSpeed(port);

On Entry-

int port;

Port previously opened with **InitializePort()**.

On Exit-

long speed;

The actual connect speed that was returned by the modem the last time it successfully connected to another modem. This value is a positive decimal value. If a -1 is returned then the port was not initialized.

See Also-

Dial()
ModemAnswerMode()
ModemAttention()
ModemConnect()
ModemGetCarrierSpeed()
ModemHangup()
ModemInit()
ModemModifyString()

Example-

```
#include <comm.h>

int             port=0;
int             stat;
int             mode=1;
long            speed;
char            *telephone="1-713-568-6401";

if ((stat = ModemInit(port)) == -1)
        {
        printf("Error initializing modem\n");

        /* Take remedial action */
        }

if ((stat = Dial(port,mode,telephone)) == -1)
        {
        printf("Modem not responding\n");

        /* Take remedial action */
        }

if ((stat = (ModemConnect(port)) != 0)
        {
        printf("Error connecting to modem\n");

        /* Take remedial action */
        }

if ((speed = ModemGetCarrierSpeed(port)) == -1L)
        {
        printf("Port not initialized\n");
        /* Take remedial action */
        }

if (speed !=ModemGetConnectSpeed(port))
        printf("Connect & Carrier Speeds are different\n");
```


ModemGetSRegister

Description-

This function returns the S register requested from the modem on the opened port. This command assumes the modem is in the command state. The modem remains in the command state at the end of this function.

Syntax-

stat = ModemGetSRegister(port,SReg);

On Entry-

int port;

Port previously opened with **InitializePort()**.

int SReg;

S register number to get (0, 1, 2, 3, ...).

On Exit-

int stat;

Value of the selected S Register.

else

-1 If error.

See Also-

ModemSetSRegister()

Example-

```
#include <comm.h>

int             port=0;
int             oldSReg;

//Get S00
if ((oldSReg = ModemGetSRegister(port,0)) != 0)
        {
        printf("Error getting S Register\n");

        /* Take remedial action */
        }
// Set S00
if ((stat = ModemSetSRegister(port,0,1)) != 0)
        {
        printf("Error setting S Register\n");

        /* Take remedial action */
        }
```


ModemGetString

Description-

This function retrieves the current string setting for the requested modem string.

Syntax-

str = ModemGetString(port,string_code);

On Entry-

int port;

Port previously opened with **InitializePort()**.

int string_code;

Code corresponding to string to retrieve as follows.

0	Command prefix send string(e.g.,"AT").
1	Modem initialization send string(e.g., "V1Q0").
2	Tone dial send string(e.g., "DT").
3	Pulse dial send string(e.g., "DD").
4	Reset send string(e.g., "Z").
5	Escape send string(e.g., "+++").
6	Hang-up send string(e.g.,"H").
7	Command suffix send string(e.g., "\r").
8	Successful acknowledge receive string(e.g.,"OK").
9	Error acknowledgment receive string(e.g.,"ERROR").
10	Connect acknowledge receive string(e.g.,"CONNECT").
11	Connect failure receive string(e.g., "NO CARRIER").
12	Telephone busy message string(e.g., "BUSY).
13	No dial tone message string(e.g., "NO DIALTONE").
14	Verbose mode command(e.g., "V1Q0").
15	Answer mode command(e.g., "S0=").
16	Carrier speed message string(e.g., "CARRIER").
17	Ring modem receive string(e.g.,"RING").
18	Command to force modem to answer(e.g., "A").

On Exit-

char *str;

Pointer to the modem string else NULL if error.

See Also-

ModemGetValue()
ModemModifyString()
ModemModifyValue()

Example-

```
#include <comm.h>

int     port=0;
int     str;

if ((str = ModemGetString(port,10)) != NULL)
        printf("Modem Connect String=>%s\n",str);
```


ModemGetValue

Description-

This function retrieves the current setting for the requested modem value.

Syntax-

value = ModemGetValue(port,value_code);

On Entry-

int port;

Port previously opened with **InitializePort()**.

int value_code;

Code corresponding to value to retrieve as follows.

0	Pacetime used to send command strings to the modem(Default=1/18 sec).
1	Pacetime used when sending the +++ or other escape code to the modem(Default=2/18 sec).
2	Time to wait for any response from the modem(Default=2 sec).
3	Time to wait for the modem to connect to another modem(Defaultt=30 sec).
4	Time to wait for character echo from the modem(Default=2 sec).
5	Unused.
6	Time interval between dropping and raising DTR to reset the modem.
7	Time interval to wait for the "RING" message from modem.

On Exit-

long value;

Retrieved value.

See Also-

ModemGetString()
ModemModifyString()
ModemModifyValue()

Example-

```
#include <comm.h>

int             port=0;
long            value;

if ((value = ModemGetValue(port,0)) != -1)
        printf("Current Pacetime Is => %ld\n",value);
```


ModemHangup

Description-

Hangs up the modem. This function moves the modem from the online state to the command state.

Syntax-

```
stat = ModemHangup(port);
```

On Entry-

```
int             port;
```

Port desired.

On Exit-

```
int             stat;
```

0 If hang-up is successful.

See Also-

Dial()
ModemAttention()
ModemAnswerMode()
ModemConnect()
ModemGetCarrierSpeed()
ModemGetConnectSpeed()
ModemInit()
ModemModifyString()

Example-

```
#include <comm.h>

int             port=0;
int             stat;

if ((stat = ModemHangup(port)) != 0)
        {
        printf("Error hanging up modem\n");

        /* Take remedial action */
        }
```


ModemInit

Description-

This function sends the modem initialization string(the modem initialization string may be changed with the **ModemModifyString()** function). The default initialization string used by **COMM-DRV/LIB** should work with most modems. It forces the modem to return verbose result codes from which the connect baud rate(baud rate between the computer and the modem) and the carrier baud rate(baud rate between the two modems) are retrieved. The modem is left in the command state when this function returns.

Syntax-

```
stat = ModemInit(port);
```

On Entry-

int port;

Port previously opened with **InitializePort()**.

On Exit-

int stat;

0 If successful.

else

-1 If port was not initialized or modem does not respond.

See Also-

Dial()
ModemAnswerMode()
ModemAttention()
ModemConnect()
ModemGetCarrierSpeed()
ModemGetConnectSpeed()
ModemHangup()
ModemModifyString()

Example-

```
#include <comm.h>

int             port=0;
int             stat;

if ((stat = ModemInit(port)) == -1)
        {
        printf("Initializing modem\n");

        /* Take remedial action */
        }
```


ModemModifyString

Description-

Changes the different command strings sent to the modem. The default strings used by **COMM-DRV/LIB** is works with most modems. The application programmer may create a customization windows to satisfy users with non-standard Hayes compatible modems.

Syntax-

```
stat = ModemModifyString(port,string_code,command_string);
```

On Entry-

```
int             port;
```

Port previously opened with **InitializePort()**.

int string_code;

Code corresponding to string to change as follows.

0	Command prefix send string(e.g.,"AT").
1	Modem initialization send string(e.g., "V1Q0").
2	Tone dial send string(e.g., "DT").
3	Pulse dial send string(e.g., "DD").
4	Reset send string(e.g., "Z").
5	Escape send string(e.g., "+++").
6	Hang-up send string(e.g.,"H").
7	Command suffix send string(e.g., "\r").
8	Successful acknowledge receive string(e.g.,"OK").
9	Error acknowledgment receive string(e.g.,"ERROR").
10	Connect acknowledge receive string(e.g.,"CONNECT").
11	Connect failure receive string(e.g., "NO CARRIER").
12	Telephone busy message string(e.g., "BUSY).
13	No dial tone message string(e.g., "NO DIALTONE").
14	Verbose mode command(e.g., "V1Q0").
15	Answer mode command(e.g., "S0=").
16	Carrier speed message string(e.g., "CARRIER").
17	Ring modem receive string(e.g.,"RING").
18	Command to force modem to answer(e.g., "A").

char *command_string;

String to send to modem for corresponding string_code.

On Exit-

int stat;

-1	if string too long or port not initialized.

See Also-

ModemGetString()
ModemGetValue()
ModemModifyValue()

Example-

```
#include <comm.h>

int             port=0;
int             stat;

if ((stat = ModemModifyString(port,0,"Z")) != 0)
        {
        printf("Error Changing Modem String\n");

        /* Take remedial action */
        }
```

ModemModifyValue

Description-

Changes different numeric values(e.g., time-outs) used by modem functions. The default values used by **COMM-DRV/LIB** works with most modems. The application programmer may create a customization window to satisfy users with non-standard Hayes compatible modems.

Syntax-

stat = ModemModifyValue(port,value_code,value);

On Entry-

int port;

Port previously opened with **InitializePort()**.

int value_code;

Code corresponding to value to change as follows.

0	Pacetime used to send command strings to the modem(Default=1/18 sec).
1	Pacetime used when sending the +++ or other escape code to the modem(Default=2/18 sec).
2	Time to wait for any response from the modem(Default=2 sec).
3	Time to wait for the modem to connect to another modem(Defaultt=30 sec).
4	Time to wait for character echo from the modem(Default=2 sec).
5	Unused.
6	Time interval between dropping and raising DTR to reset the modem.
7	Time interval to wait for the "RING" message from modem.

long value;

New value corresponding value_code.

On Exit-

int stat;

-1 If port not initialized or invalid code.

See Also-

ModemGetString()
ModemGetValue()
ModemModifyString()

Example-

```
#include <comm.h>

int     port=0;
int     stat;

if ((stat = ModemModifyValue(port,0,2)) != 0)
        {
        printf("Error Changing Modem value\n");

        /* Take remedial action */
        }
```

ModemOffHook

Description-

This function takes the phone off hook. This command assumes the modem is in the command state. The modem remains in the command state at the end of this function.

Syntax-

stat = ModemOffHook(port);

On Entry-

int port;

Port previously opened with **InitializePort()**.

On Exit-

int stat;

0 If successful.

else

-1 If error.

See Also-

Dial()
ModemAnswerMode()
ModemAttention()
ModemCommandState()
ModemConnect()
ModemGetCarrierSpeed()
ModemGetConnectSpeed()
ModemHangup()
ModemModifyString()

Example-

```
#include <comm.h>

int             port=0;
int             stat;

if ((stat = ModemOffHook(port)) != 0)
        {
        printf("Error taking modem off hook\n");

        /* Take remedial action */
        }
```

ModemOnline

Description-

This function takes the phone off hook. This command assumes the modem is in the command state. The modem remains in the command state at the end of this function.

Syntax-

stat = ModemOnline(port);

On Entry-

int port;

Port previously opened with **InitializePort()**.

On Exit-

int stat;

0 If successful.

else

-1 If error.

See Also-

Dial()
ModemAnswerMode()
ModemAttention()
ModemCommandState()
ModemConnect()
ModemGetCarrierSpeed()
ModemGetConnectSpeed()
ModemHangup()
ModemModifyString()

Example-

```
#include <comm.h>

int             port=0;
int             stat;

if ((stat = ModemOnline(port)) == -1)
        {
        printf("Error putting modem back online\n");

        /* Take remedial action */
        }
```

ModemSendCommand

Description-

This function sends a command to the modem. *This command assumes the modem is in* the command state. The modem remains in the command state at the end of this function.

Syntax-

stat = ModemSendCommand(port,command);

On Entry-

int port;

Port previously opened with **InitializePort()**.

char *command;

Null terminated command to send to the modem.

On Exit-

int stat;

0 If successful.

else

-1 If error.

See Also-

ModemCommandState()

Example-

```
#include <comm.h>

int             port=0;
int             stat;
char            *command="V0Q1"

if ((stat = ModemSendCommand(port,command)) == -1)
        {
        printf("Error sending modem command\n");

        /* Take remedial action */
        }
```

ModemSetSRegister

Description-

This function sets the S register requested for the modem on the opened port. This command assumes the modem is in the command state. The modem remains in the command state at the end of this function.

Syntax-

stat = ModemSetSRegister(port,Sreg,value);

On Entry-

int port;

Port previously opened with **InitializePort()**.

int SReg;

S register number to set (0, 1, 2, 3, ...).

int value;

Value to set.

On Exit-

int stat;

0 If successful.

else

-1 If error.

See Also-

ModemCommandState()
ModemGetSRegister()

Example-

```
#include <comm.h>

int             port=0;
int             oldSReg;

//Get S00
if ((oldSReg = ModemGetSRegister(port,0)) != 0)
        {
        printf("Error getting S Register\n");

        /* Take remedial action */
        }
// Set S00
if ((stat = ModemSetSRegister(port,0,1)) != 0)
        {
        printf("Error setting S Register\n");

        /* Take remedial action */
        }
```

ModemSpeaker

Description-

This function turns the modem speaker on or off. This command assumes the modem is in the command state. The modem remains in the command state at the end of this function.

Syntax-

stat = ModemSpeaker(port,OnOff);

On Entry-

int port;

Port previously opened with **InitIalizePort()**.

int OnOff;

0 = Off.
1 = On.

On Exit-

int stat;

0 If successful.

else

-1 If error.

See Also-

ModemVolume()

Example-

```
#include <comm.h>

int             port=0;
int             stat;

//Turn the speaker on the modem on
if ((stat = ModemSpeaker(port,1)) == -1)
        {
        printf("Error turning speaker on\n");

        /* Take remedial action */
        }
```

ModemVolume

Description-

This function sets the modem's speaker volume. This command assumes the modem is in the command state. The modem remains in the command state at the end of this function.

Syntax-

stat = ModemVolume(port,VLevel);

On Entry-

int port;

Port previously opened with **InitializePort()**.

int VLevel;

0 = Low.
1 = Medium.
2 = High.

On Exit-

int stat;

0 If successful.

else

-1 If error.

See Also-

ModemSpeaker()

Example-

```
#include <comm.h>

int             port=0;
int             stat;

//Turn the speaker volume on modem to medium
if ((stat = ModemVolume(port,1)) == -1)
        {
        printf("Error setting speaker volume\n");

        /* Take remedial action */
        }
```

7.3.3 Content of Visual C++ Version 1.52 CD-ROM

The following is a directory listing of the software found on the Microsoft Visual C++ Version 1.52 CD-ROM. Some source code is included on this CD-ROM.

```
FF-File Find, Advanced Edition 4.50, (C) Copr 1987-88, Peter Norton

I:\
        referral.txt      1,043 bytes   2:04 pm  Tue Apr 30 96
        VC152               <DIR>      11:19 am  Mon Jun 10 96
        VC41                <DIR>      12:18 pm  Thu Jun 13 96

I:\VC152
        .                   <DIR>      11:19 am  Mon Jun 10 96
        ..                  <DIR>      11:23 am  Mon Jun 10 96
        bookinfo.wri     24,960 bytes   4:26 pm  Tue Jan 23 96
        INSIDE2             <DIR>      10:06 am  Mon Jun 10 96
        MSVC15              <DIR>      11:19 am  Mon Jun 10 96
        MSVCCDK             <DIR>      11:23 am  Mon Jun 10 96

I:\VC152\INSIDE2
        .                   <DIR>      10:06 am  Mon Jun 10 96
        ..                  <DIR>      11:19 am  Mon Jun 10 96
        infoview.inf         73 bytes   1:35 pm  Tue Dec 19 95
        infovw16.exe    579,856 bytes   4:35 pm  Mon Dec 18 95
        inside2.aux     118,594 bytes   9:46 am  Thu Jan 25 96
        inside2.cac     280,933 bytes   9:46 am  Thu Jan 25 96
        inside2.idx      41,065 bytes   9:46 am  Thu Jan 25 96
        inside2.mvb     634,889 bytes   9:46 am  Thu Jan 25 96
        iv2api.dll        6,352 bytes   4:35 pm  Mon Dec 18 95
        iv2tool.exe       6,160 bytes   4:35 pm  Mon Dec 18 95
        mscomstf.dll     78,064 bytes   4:35 pm  Mon Dec 18 95
        mscpydis.dll     11,552 bytes   4:35 pm  Mon Dec 18 95
        mscuistf.dll     27,952 bytes   4:35 pm  Mon Dec 18 95
        msdetstf.dll     25,232 bytes   4:35 pm  Mon Dec 18 95
        msdnlib.hlp     146,970 bytes   3:14 pm  Fri Jan  5 96
        msdntool.exe      6,160 bytes   4:35 pm  Mon Dec 18 95
        msin.inf             72 bytes   1:35 pm  Tue Dec 19 95
        msinsstf.dll     67,440 bytes   4:35 pm  Mon Dec 18 95
        mspress.ico         766 bytes   3:09 pm  Fri Jan  5 96
        msshlstf.dll     14,416 bytes   4:35 pm  Mon Dec 18 95
        msuilstf.dll      6,384 bytes   4:35 pm  Mon Dec 18 95
        relnotes.hlp     16,205 bytes   3:14 pm  Fri Jan  5 96
        SAMPLES             <DIR>      10:19 am  Mon Jun 10 96
        setup.exe        54,144 bytes   7:11 pm  Tue Dec 19 95
        setup.inf           973 bytes  11:10 am  Wed Jan 24 96
        setup.ini         1,586 bytes   3:14 pm  Fri Jan  5 96

I:\VC152\INSIDE2\SAMPLES
        .                   <DIR>      10:19 am  Mon Jun 10 96
        ..                  <DIR>      10:06 am  Mon Jun 10 96
        EX03A               <DIR>      10:07 am  Mon Jun 10 96
        EX04A               <DIR>      10:08 am  Mon Jun 10 96
        EX04B               <DIR>      10:08 am  Mon Jun 10 96
        EX04C               <DIR>      10:10 am  Mon Jun 10 96
        EX05A               <DIR>      10:11 am  Mon Jun 10 96
        EX05B               <DIR>      10:13 am  Mon Jun 10 96
        EX05C               <DIR>      10:14 am  Mon Jun 10 96
        EX06A               <DIR>      10:14 am  Mon Jun 10 96
```

```
        EX07A                <DIR>      10:15 am  Mon Jun 10 96
        EX07B                <DIR>      10:16 am  Mon Jun 10 96
        EX14A                <DIR>      10:17 am  Mon Jun 10 96
        EX15A                <DIR>      10:18 am  Mon Jun 10 96
        EX15B                <DIR>      10:18 am  Mon Jun 10 96
        EX16A                <DIR>      10:19 am  Mon Jun 10 96

I:\VC152\INSIDE2\SAMPLES\EX03A
        .                    <DIR>      10:07 am  Mon Jun 10 96
        ..                   <DIR>      10:19 am  Mon Jun 10 96
        ex03a.cpp          3,435 bytes  12:00 am  Thu Apr  7 94
        ex03a.def            347 bytes  12:00 am  Thu Apr  7 94
        ex03a.exe      1,253,256 bytes  12:00 am  Tue Jun 21 94
        ex03a.h              807 bytes  12:00 am  Thu Apr  7 94
        ex03a.mak          3,215 bytes  12:00 am  Mon Jun 20 94
        ex03a.obj          6,051 bytes  12:00 am  Tue Jun 21 94
        ex03a.rc           6,881 bytes  12:00 am  Thu Apr  7 94
        ex03a.res         10,726 bytes  12:00 am  Tue Jun 21 94
        ex03a.vcw             90 bytes  12:00 am  Tue Jun 21 94
        ex03a.wsp            170 bytes  12:00 am  Tue Jun 21 94
        ex03adoc.cpp       1,705 bytes  12:00 am  Thu Apr  7 94
        ex03adoc.h           931 bytes  12:00 am  Thu Apr  7 94
        ex03adoc.obj       3,859 bytes  12:00 am  Tue Jun 21 94
        ex03avw.cpp        1,966 bytes  12:00 am  Thu Apr  7 94
        ex03avw.h          1,057 bytes  12:00 am  Thu Apr  7 94
        ex03avw.obj        5,062 bytes  12:00 am  Tue Jun 21 94
        mainfrm.cpp        2,447 bytes  12:00 am  Thu Apr  7 94
        mainfrm.h            978 bytes  12:00 am  Thu Apr  7 94
        mainfrm.obj        5,155 bytes  12:00 am  Tue Jun 21 94
        RES                  <DIR>      10:07 am  Mon Jun 10 96
        resource.h           359 bytes  12:00 am  Thu Apr  7 94
        stdafx.cpp           204 bytes  12:00 am  Thu Apr  7 94
        stdafx.h             283 bytes  12:00 am  Thu Apr  7 94
        stdafx.obj           276 bytes  12:00 am  Tue Jun 21 94

I:\VC152\INSIDE2\SAMPLES\EX03A\RES
        .                    <DIR>      10:07 am  Mon Jun 10 96
        ..                   <DIR>      10:07 am  Mon Jun 10 96
        ex03a.ico            768 bytes  12:00 am  Thu Apr  7 94
        ex03a.rc2          1,538 bytes  12:00 am  Thu Apr  7 94
        toolbar.bmp        1,200 bytes  12:00 am  Thu Apr  7 94

I:\VC152\INSIDE2\SAMPLES\EX04A
        .                    <DIR>      10:08 am  Mon Jun 10 96
        ..                   <DIR>      10:19 am  Mon Jun 10 96
        ex04a.clw          1,890 bytes  12:00 am  Tue Apr 19 94
        ex04a.cpp          3,325 bytes  12:00 am  Tue Apr 19 94
        ex04a.def            347 bytes  12:00 am  Tue Apr 19 94
        ex04a.exe      1,253,580 bytes  12:00 am  Mon Jun 20 94
        ex04a.h              812 bytes  12:00 am  Tue Apr 19 94
        ex04a.mak          3,187 bytes  12:00 am  Mon Jun 20 94
        ex04a.obj          6,057 bytes  12:00 am  Mon Jun 20 94
        ex04a.rc           6,921 bytes  12:00 am  Tue Apr 19 94
        ex04a.res         10,758 bytes  12:00 am  Mon Jun 20 94
```

```
        ex04a.vcw             90 bytes   12:00 am  Mon Jun 20 94
        ex04a.wsp            170 bytes   12:00 am  Mon Jun 20 94
        ex04adoc.cpp       1,704 bytes   12:00 am  Tue Apr 19 94
        ex04adoc.h           933 bytes   12:00 am  Tue Apr 19 94
        ex04adoc.obj       3,859 bytes   12:00 am  Mon Jun 20 94
        ex04avw.cpp        1,872 bytes   12:00 am  Tue Apr 19 94
        ex04avw.h          1,053 bytes   12:00 am  Tue Apr 19 94
        ex04avw.obj        5,319 bytes   12:00 am  Mon Jun 20 94
        mainfrm.cpp        2,461 bytes   12:00 am  Tue Apr 19 94
        mainfrm.h            980 bytes   12:00 am  Tue Apr 19 94
        mainfrm.obj        5,155 bytes   12:00 am  Mon Jun 20 94
        readme.txt         4,342 bytes   12:00 am  Tue Apr 19 94
        RES                  <DIR>       10:08 am  Mon Jun 10 96
        resource.h           360 bytes   12:00 am  Tue Apr 19 94
        stdafx.cpp           204 bytes   12:00 am  Tue Apr 19 94
        stdafx.h             299 bytes   12:00 am  Tue Apr 19 94
        stdafx.obj           276 bytes   12:00 am  Mon Jun 20 94

I:\VC152\INSIDE2\SAMPLES\EX04A\RES
        .                    <DIR>       10:08 am  Mon Jun 10 96
        ..                   <DIR>       10:08 am  Mon Jun 10 96
        ex04a.ico            768 bytes   12:00 am  Tue Apr 19 94
        ex04a.rc2          1,538 bytes   12:00 am  Tue Apr 19 94
        toolbar.bmp        1,200 bytes   12:00 am  Tue Apr 19 94

I:\VC152\INSIDE2\SAMPLES\EX04B
        .                    <DIR>       10:08 am  Mon Jun 10 96
        ..                   <DIR>       10:19 am  Mon Jun 10 96
        ex04b.clw          1,890 bytes   12:00 am  Tue Apr 19 94
        ex04b.cpp          3,325 bytes   12:00 am  Tue Apr 19 94
        ex04b.def            347 bytes   12:00 am  Tue Apr 19 94
        ex04b.exe      1,254,116 bytes   12:00 am  Tue Jun 21 94
        ex04b.h              812 bytes   12:00 am  Tue Apr 19 94
        ex04b.mak          3,187 bytes   12:00 am  Mon Jun 20 94
        ex04b.obj          6,057 bytes   12:00 am  Tue Jun 21 94
        ex04b.pdb        158,230 bytes   12:00 am  Tue Apr 19 94
        ex04b.rc           6,921 bytes   12:00 am  Tue Apr 19 94
        ex04b.res         10,758 bytes   12:00 am  Tue Jun 21 94
        ex04b.vcw             90 bytes   12:00 am  Tue Jun 21 94
        ex04b.wsp            170 bytes   12:00 am  Tue Jun 21 94
        ex04bdoc.cpp       1,704 bytes   12:00 am  Tue Apr 19 94
        ex04bdoc.h           933 bytes   12:00 am  Tue Apr 19 94
        ex04bdoc.obj       3,859 bytes   12:00 am  Tue Jun 21 94
        ex04bvw.cpp        2,970 bytes   12:00 am  Tue Apr 19 94
        ex04bvw.h          1,219 bytes   12:00 am  Tue Apr 19 94
        ex04bvw.obj        6,733 bytes   12:00 am  Tue Jun 21 94
        mainfrm.cpp        2,461 bytes   12:00 am  Tue Apr 19 94
        mainfrm.h            980 bytes   12:00 am  Tue Apr 19 94
        mainfrm.obj        5,155 bytes   12:00 am  Tue Jun 21 94
        readme.txt         4,342 bytes   12:00 am  Tue Apr 19 94
        RES                  <DIR>       10:08 am  Mon Jun 10 96
        resource.h           360 bytes   12:00 am  Tue Apr 19 94
        stdafx.cpp           204 bytes   12:00 am  Tue Apr 19 94
        stdafx.h             299 bytes   12:00 am  Tue Apr 19 94
```

```
        stdafx.obj         276 bytes  12:00 am  Tue Jun 21 94

I:\VC152\INSIDE2\SAMPLES\EX04B\RES
        .                  <DIR>      10:08 am  Mon Jun 10 96
        ..                 <DIR>      10:08 am  Mon Jun 10 96
        ex04b.ico          768 bytes  12:00 am  Tue Apr 19 94
        ex04b.rc2        1,538 bytes  12:00 am  Tue Apr 19 94
        toolbar.bmp      1,200 bytes  12:00 am  Tue Apr 19 94

I:\VC152\INSIDE2\SAMPLES\EX04C
        .                  <DIR>      10:10 am  Mon Jun 10 96
        ..                 <DIR>      10:19 am  Mon Jun 10 96
        ex04c.clw        1,890 bytes  12:00 am  Tue Apr 19 94
        ex04c.cpp        3,325 bytes  12:00 am  Tue Apr 19 94
        ex04c.def          347 bytes  12:00 am  Tue Apr 19 94
        ex04c.exe    1,273,468 bytes  12:00 am  Mon Jun 20 94
        ex04c.h            812 bytes  12:00 am  Tue Apr 19 94
        ex04c.mak        3,187 bytes  12:00 am  Mon Jun 20 94
        ex04c.obj        6,057 bytes  12:00 am  Mon Jun 20 94
        ex04c.rc         6,921 bytes  12:00 am  Tue Apr 19 94
        ex04c.res       10,758 bytes  12:00 am  Mon Jun 20 94
        ex04c.vcw           90 bytes  12:00 am  Mon Jun 20 94
        ex04c.wsp          170 bytes  12:00 am  Mon Jun 20 94
        ex04cdoc.cpp     1,704 bytes  12:00 am  Tue Apr 19 94
        ex04cdoc.h         933 bytes  12:00 am  Tue Apr 19 94
        ex04cdoc.obj     3,859 bytes  12:00 am  Mon Jun 20 94
        ex04cvw.cpp      3,104 bytes  12:00 am  Tue Apr 19 94
        ex04cvw.h        1,069 bytes  12:00 am  Tue Apr 19 94
        ex04cvw.obj      5,919 bytes  12:00 am  Mon Jun 20 94
        mainfrm.cpp      2,461 bytes  12:00 am  Tue Apr 19 94
        mainfrm.h          980 bytes  12:00 am  Tue Apr 19 94
        mainfrm.obj      5,155 bytes  12:00 am  Mon Jun 20 94
        readme.txt       4,342 bytes  12:00 am  Tue Apr 19 94
        RES                <DIR>      10:10 am  Mon Jun 10 96
        resource.h         360 bytes  12:00 am  Tue Apr 19 94
        stdafx.cpp         204 bytes  12:00 am  Tue Apr 19 94
        stdafx.h           299 bytes  12:00 am  Tue Apr 19 94
        stdafx.obj         276 bytes  12:00 am  Mon Jun 20 94

I:\VC152\INSIDE2\SAMPLES\EX04C\RES
        .                  <DIR>      10:10 am  Mon Jun 10 96
        ..                 <DIR>      10:10 am  Mon Jun 10 96
        ex04c.ico          768 bytes  12:00 am  Tue Apr 19 94
        ex04c.rc2        1,538 bytes  12:00 am  Tue Apr 19 94
        toolbar.bmp      1,200 bytes  12:00 am  Tue Apr 19 94

I:\VC152\INSIDE2\SAMPLES\EX05A
        .                  <DIR>      10:11 am  Mon Jun 10 96
        ..                 <DIR>      10:19 am  Mon Jun 10 96
        ex05a.cpp        3,350 bytes  12:00 am  Thu Apr  7 94
        ex05a.def          347 bytes  12:00 am  Thu Apr  7 94
        ex05a.exe    1,340,160 bytes  12:00 am  Mon Jun 20 94
        ex05a.h            807 bytes  12:00 am  Thu Apr  7 94
        ex05a.mak        2,949 bytes  12:00 am  Mon Jun 20 94
```

```
        ex05a.obj         6,100 bytes   12:00 am  Mon Jun 20 94
        ex05a.rc          7,502 bytes   12:00 am  Thu Apr  7 94
        ex05a.res        11,715 bytes   12:00 am  Mon Jun 20 94
        ex05a.vcw            90 bytes   12:00 am  Mon Jun 20 94
        ex05a.wsp           170 bytes   12:00 am  Mon Jun 20 94
        ex05adoc.cpp      1,705 bytes   12:00 am  Thu Apr  7 94
        ex05adoc.h          931 bytes   12:00 am  Thu Apr  7 94
        ex05adoc.obj      3,859 bytes   12:00 am  Mon Jun 20 94
        ex05avw.cpp       3,781 bytes   12:00 am  Thu Apr  7 94
        ex05avw.h         1,454 bytes   12:00 am  Thu Apr  7 94
        ex05avw.obj       6,525 bytes   12:00 am  Mon Jun 20 94
        mainfrm.cpp       2,447 bytes   12:00 am  Thu Apr  7 94
        mainfrm.h           978 bytes   12:00 am  Thu Apr  7 94
        mainfrm.obj       5,155 bytes   12:00 am  Mon Jun 20 94
        RES                 <DIR>       10:11 am  Mon Jun 10 96
        resource.h          359 bytes   12:00 am  Thu Apr  7 94
        stdafx.cpp          204 bytes   12:00 am  Thu Apr  7 94
        stdafx.h            283 bytes   12:00 am  Thu Apr  7 94
        stdafx.obj          276 bytes   12:00 am  Mon Jun 20 94

I:\VC152\INSIDE2\SAMPLES\EX05A\RES
        .                   <DIR>       10:11 am  Mon Jun 10 96
        ..                  <DIR>       10:11 am  Mon Jun 10 96
        ex05a.ico           768 bytes   12:00 am  Thu Apr  7 94
        ex05a.rc2         1,538 bytes   12:00 am  Thu Apr  7 94
        toolbar.bmp       1,200 bytes   12:00 am  Thu Apr  7 94

I:\VC152\INSIDE2\SAMPLES\EX05B
        .                   <DIR>       10:13 am  Mon Jun 10 96
        ..                  <DIR>       10:19 am  Mon Jun 10 96
        ex05b.cpp         3,350 bytes   12:00 am  Mon May 23 94
        ex05b.def           347 bytes   12:00 am  Thu Apr  7 94
        ex05b.exe     1,340,448 bytes   12:00 am  Mon Jun 20 94
        ex05b.h             807 bytes   12:00 am  Mon May 23 94
        ex05b.mak         3,215 bytes   12:00 am  Mon Jun 20 94
        ex05b.obj         6,100 bytes   12:00 am  Mon Jun 20 94
        ex05b.rc          7,502 bytes   12:00 am  Thu Apr  7 94
        ex05b.res        11,715 bytes   12:00 am  Mon Jun 20 94
        ex05b.vcw            90 bytes   12:00 am  Mon Jun 20 94
        ex05b.wsp           170 bytes   12:00 am  Mon Jun 20 94
        ex05bdoc.cpp      1,705 bytes   12:00 am  Mon May 23 94
        ex05bdoc.h          931 bytes   12:00 am  Mon May 23 94
        ex05bdoc.obj      3,859 bytes   12:00 am  Mon Jun 20 94
        ex05bvw.cpp       4,660 bytes   12:00 am  Mon May 23 94
        ex05bvw.h         1,379 bytes   12:00 am  Mon May 23 94
        ex05bvw.obj       6,909 bytes   12:00 am  Mon Jun 20 94
        mainfrm.cpp       2,447 bytes   12:00 am  Mon May 23 94
        mainfrm.h           978 bytes   12:00 am  Mon May 23 94
        mainfrm.obj       5,155 bytes   12:00 am  Mon Jun 20 94
        RES                 <DIR>       10:13 am  Mon Jun 10 96
        resource.h          359 bytes   12:00 am  Mon May 23 94
        stdafx.cpp          204 bytes   12:00 am  Mon May 23 94
        stdafx.h            283 bytes   12:00 am  Mon May 23 94
        stdafx.obj          276 bytes   12:00 am  Mon Jun 20 94
```

```
I:\VC152\INSIDE2\SAMPLES\EX05B\RES
        .                   <DIR>       10:13 am  Mon Jun 10 96
        ..                  <DIR>       10:13 am  Mon Jun 10 96
        ex05b.ico             768 bytes  12:00 am  Thu Apr  7 94
        ex05b.rc2           1,538 bytes  12:00 am  Thu Apr  7 94
        toolbar.bmp         1,200 bytes  12:00 am  Thu Apr  7 94

I:\VC152\INSIDE2\SAMPLES\EX05C
        .                   <DIR>       10:14 am  Mon Jun 10 96
        ..                  <DIR>       10:19 am  Mon Jun 10 96
        ex05c.clw           1,890 bytes  12:00 am  Wed Apr 20 94
        ex05c.cpp           3,325 bytes  12:00 am  Wed Apr 20 94
        ex05c.def             347 bytes  12:00 am  Wed Apr 20 94
        ex05c.exe       1,273,984 bytes  12:00 am  Mon Jun 20 94
        ex05c.h               812 bytes  12:00 am  Wed Apr 20 94
        ex05c.mak           3,187 bytes  12:00 am  Mon Jun 20 94
        ex05c.obj           6,057 bytes  12:00 am  Mon Jun 20 94
        ex05c.rc            6,921 bytes  12:00 am  Wed Apr 20 94
        ex05c.res          10,758 bytes  12:00 am  Mon Jun 20 94
        ex05c.vcw              90 bytes  12:00 am  Mon Jun 20 94
        ex05c.wsp             170 bytes  12:00 am  Mon Jun 20 94
        ex05cdoc.cpp        1,704 bytes  12:00 am  Wed Apr 20 94
        ex05cdoc.h            933 bytes  12:00 am  Wed Apr 20 94
        ex05cdoc.obj        3,859 bytes  12:00 am  Mon Jun 20 94
        ex05cvw.cpp         3,911 bytes  12:00 am  Tue May 24 94
        ex05cvw.h           1,287 bytes  12:00 am  Wed Apr 20 94
        ex05cvw.obj         7,816 bytes  12:00 am  Mon Jun 20 94
        mainfrm.cpp         2,461 bytes  12:00 am  Wed Apr 20 94
        mainfrm.h             980 bytes  12:00 am  Wed Apr 20 94
        mainfrm.obj         5,155 bytes  12:00 am  Mon Jun 20 94
        readme.txt          4,342 bytes  12:00 am  Wed Apr 20 94
        RES                 <DIR>        10:14 am  Mon Jun 10 96
        resource.h            360 bytes  12:00 am  Wed Apr 20 94
        stdafx.cpp            204 bytes  12:00 am  Wed Apr 20 94
        stdafx.h              299 bytes  12:00 am  Wed Apr 20 94
        stdafx.obj            276 bytes  12:00 am  Mon Jun 20 94

I:\VC152\INSIDE2\SAMPLES\EX05C\RES
        .                   <DIR>       10:14 am  Mon Jun 10 96
        ..                  <DIR>       10:14 am  Mon Jun 10 96
        ex05c.ico             768 bytes  12:00 am  Wed Apr 20 94
        ex05c.rc2           1,538 bytes  12:00 am  Wed Apr 20 94
        toolbar.bmp         1,200 bytes  12:00 am  Wed Apr 20 94

I:\VC152\INSIDE2\SAMPLES\EX06A
        .                   <DIR>       10:14 am  Mon Jun 10 96
        ..                  <DIR>       10:19 am  Mon Jun 10 96
        ex06a.aps          22,987 bytes  12:00 am  Thu Apr  7 94
        ex06a.clw           3,154 bytes  12:00 am  Thu Apr  7 94
        ex06a.cpp           3,255 bytes  12:00 am  Thu Apr  7 94
        ex06a.def             347 bytes  12:00 am  Thu Apr  7 94
        ex06a.exe         282,352 bytes  12:00 am  Mon Jun 20 94
        ex06a.h               807 bytes  12:00 am  Thu Apr  7 94
```

```
ex06a.mak          3,488 bytes   12:00 am   Mon Jun 20 94
ex06a.obj          5,910 bytes   12:00 am   Mon Jun 20 94
ex06a.rc          10,749 bytes   12:00 am   Thu Apr  7 94
ex06a.res         11,640 bytes   12:00 am   Mon Jun 20 94
ex06a.vcw             90 bytes   12:00 am   Mon Jun 20 94
ex06a.wsp            170 bytes   12:00 am   Mon Jun 20 94
ex06adlg.cpp       3,815 bytes   12:00 am   Thu Apr  7 94
ex06adlg.h         1,137 bytes   12:00 am   Thu Apr  7 94
ex06adlg.obj       5,051 bytes   12:00 am   Mon Jun 20 94
ex06adoc.cpp       1,705 bytes   12:00 am   Thu Apr  7 94
ex06adoc.h           931 bytes   12:00 am   Thu Apr  7 94
ex06adoc.obj       3,744 bytes   12:00 am   Mon Jun 20 94
ex06avw.cpp        2,727 bytes   12:00 am   Thu Apr  7 94
ex06avw.h            972 bytes   12:00 am   Thu Apr  7 94
ex06avw.obj        5,998 bytes   12:00 am   Mon Jun 20 94
mainfrm.cpp        2,447 bytes   12:00 am   Thu Apr  7 94
mainfrm.h            978 bytes   12:00 am   Thu Apr  7 94
mainfrm.obj        5,044 bytes   12:00 am   Mon Jun 20 94
RES                  <DIR>       10:14 am   Mon Jun 10 96
resource.h         1,588 bytes   12:00 am   Thu Apr  7 94
stdafx.cpp           204 bytes   12:00 am   Thu Apr  7 94
stdafx.h             227 bytes   12:00 am   Thu Apr  7 94
stdafx.obj           276 bytes   12:00 am   Mon Jun 20 94

I:\VC152\INSIDE2\SAMPLES\EX06A\RES
.                    <DIR>       10:14 am   Mon Jun 10 96
..                   <DIR>       10:14 am   Mon Jun 10 96
ex06a.ico            768 bytes   12:00 am   Thu Apr  7 94
ex06a.rc2          1,631 bytes   12:00 am   Thu Apr  7 94
toolbar.bmp        1,200 bytes   12:00 am   Thu Apr  7 94

I:\VC152\INSIDE2\SAMPLES\EX07A
.                    <DIR>       10:15 am   Mon Jun 10 96
..                   <DIR>       10:19 am   Mon Jun 10 96
ex07a.aps         13,878 bytes   12:00 am   Thu Apr  7 94
ex07a.cpp          3,255 bytes   12:00 am   Thu Apr  7 94
ex07a.def            347 bytes   12:00 am   Thu Apr  7 94
ex07a.exe      1,269,464 bytes   12:00 am   Mon Jun 20 94
ex07a.h              807 bytes   12:00 am   Thu Apr  7 94
ex07a.mak          3,453 bytes   12:00 am   Mon Jun 20 94
ex07a.obj          6,051 bytes   12:00 am   Mon Jun 20 94
ex07a.rc           7,907 bytes   12:00 am   Thu Apr  7 94
ex07a.res         10,858 bytes   12:00 am   Mon Jun 20 94
ex07a.vcw             90 bytes   12:00 am   Mon Jun 20 94
ex07a.wsp            170 bytes   12:00 am   Mon Jun 20 94
ex07adlg.cpp       1,499 bytes   12:00 am   Tue May 24 94
ex07adlg.h           763 bytes   12:00 am   Thu Apr  7 94
ex07adlg.obj       3,646 bytes   12:00 am   Mon Jun 20 94
ex07adoc.cpp       1,705 bytes   12:00 am   Thu Apr  7 94
ex07adoc.h           931 bytes   12:00 am   Thu Apr  7 94
ex07adoc.obj       3,859 bytes   12:00 am   Mon Jun 20 94
ex07avw.cpp        2,379 bytes   12:00 am   Fri Apr  8 94
ex07avw.h          1,173 bytes   12:00 am   Thu Apr  7 94
ex07avw.obj        5,795 bytes   12:00 am   Mon Jun 20 94
```

```
        mainfrm.cpp       2,447 bytes  12:00 am  Thu Apr  7 94
        mainfrm.h           978 bytes  12:00 am  Thu Apr  7 94
        mainfrm.obj       5,155 bytes  12:00 am  Mon Jun 20 94
        RES                 <DIR>      10:15 am  Mon Jun 10 96
        resource.h          625 bytes  12:00 am  Fri Apr  8 94
        stdafx.cpp          204 bytes  12:00 am  Thu Apr  7 94
        stdafx.h            283 bytes  12:00 am  Thu Apr  7 94
        stdafx.obj          276 bytes  12:00 am  Mon Jun 20 94

I:\VC152\INSIDE2\SAMPLES\EX07A\RES
        .                   <DIR>      10:15 am  Mon Jun 10 96
        ..                  <DIR>      10:15 am  Mon Jun 10 96
        ex07a.ico           768 bytes  12:00 am  Thu Apr  7 94
        ex07a.rc2         1,538 bytes  12:00 am  Thu Apr  7 94
        toolbar.bmp       1,200 bytes  12:00 am  Thu Apr  7 94

I:\VC152\INSIDE2\SAMPLES\EX07B
        .                   <DIR>      10:16 am  Mon Jun 10 96
        ..                  <DIR>      10:19 am  Mon Jun 10 96
        ex07b.aps        13,596 bytes  12:00 am  Thu Apr 21 94
        ex07b.cpp         3,255 bytes  12:00 am  Thu Apr  7 94
        ex07b.def           347 bytes  12:00 am  Thu Apr  7 94
        ex07b.exe     1,255,072 bytes  12:00 am  Mon Jun 20 94
        ex07b.h             807 bytes  12:00 am  Thu Apr  7 94
        ex07b.mak         3,360 bytes  12:00 am  Mon Jun 20 94
        ex07b.obj         6,047 bytes  12:00 am  Mon Jun 20 94
        ex07b.rc          7,528 bytes  12:00 am  Thu Apr  7 94
        ex07b.res        10,726 bytes  12:00 am  Mon Jun 20 94
        ex07b.vcw           90 bytes  12:00 am  Mon Jun 20 94
        ex07b.wsp           170 bytes  12:00 am  Mon Jun 20 94
        ex07bdlg.cpp      2,213 bytes  12:00 am  Fri May 13 94
        ex07bdlg.h          693 bytes  12:00 am  Thu Apr  7 94
        ex07bdlg.obj      4,158 bytes  12:00 am  Mon Jun 20 94
        ex07bdoc.cpp      1,705 bytes  12:00 am  Thu Apr  7 94
        ex07bdoc.h          931 bytes  12:00 am  Thu Apr  7 94
        ex07bdoc.obj      3,859 bytes  12:00 am  Mon Jun 20 94
        ex07bvw.cpp       2,126 bytes  12:00 am  Fri Apr  8 94
        ex07bvw.h           972 bytes  12:00 am  Thu Apr  7 94
        ex07bvw.obj       5,871 bytes  12:00 am  Mon Jun 20 94
        mainfrm.cpp       2,447 bytes  12:00 am  Thu Apr  7 94
        mainfrm.h           978 bytes  12:00 am  Thu Apr  7 94
        mainfrm.obj       5,155 bytes  12:00 am  Mon Jun 20 94
        RES                 <DIR>      10:16 am  Mon Jun 10 96
        resource.h          527 bytes  12:00 am  Thu Apr  7 94
        stdafx.cpp          204 bytes  12:00 am  Thu Apr  7 94
        stdafx.h            283 bytes  12:00 am  Thu Apr  7 94
        stdafx.obj          276 bytes  12:00 am  Mon Jun 20 94

I:\VC152\INSIDE2\SAMPLES\EX07B\RES
        .                   <DIR>      10:16 am  Mon Jun 10 96
        ..                  <DIR>      10:16 am  Mon Jun 10 96
        ex07b.ico           768 bytes  12:00 am  Thu Apr  7 94
        ex07b.rc2         1,538 bytes  12:00 am  Thu Apr  7 94
        toolbar.bmp       1,200 bytes  12:00 am  Thu Apr  7 94
```

```
I:\VC152\INSIDE2\SAMPLES\EX14A
        .                   <DIR>       10:17 am  Mon Jun 10 96
        ..                  <DIR>       10:19 am  Mon Jun 10 96
        ex14a.aps        13,578 bytes   12:00 am  Tue Apr 12 94
        ex14a.clw         1,911 bytes   12:00 am  Tue Apr 12 94
        ex14a.cpp         3,277 bytes   12:00 am  Thu Apr  7 94
        ex14a.def           347 bytes   12:00 am  Thu Apr  7 94
        ex14a.exe     1,256,220 bytes   12:00 am  Mon Jun 20 94
        ex14a.h             807 bytes   12:00 am  Thu Apr  7 94
        ex14a.mak         3,423 bytes   12:00 am  Mon Jun 20 94
        ex14a.obj         6,051 bytes   12:00 am  Mon Jun 20 94
        ex14a.rc          6,881 bytes   12:00 am  Thu Apr  7 94
        ex14a.res        10,726 bytes   12:00 am  Mon Jun 20 94
        ex14a.vcw           90 bytes    12:00 am  Mon Jun 20 94
        ex14a.wsp           170 bytes   12:00 am  Mon Jun 20 94
        ex14adoc.cpp      1,705 bytes   12:00 am  Thu Apr  7 94
        ex14adoc.h          931 bytes   12:00 am  Thu Apr  7 94
        ex14adoc.obj      3,859 bytes   12:00 am  Mon Jun 20 94
        ex14avw.cpp       1,643 bytes   12:00 am  Thu Apr  7 94
        ex14avw.h         1,057 bytes   12:00 am  Thu Apr  7 94
        ex14avw.obj       4,820 bytes   12:00 am  Mon Jun 20 94
        mainfrm.cpp       2,504 bytes   12:00 am  Thu Apr  7 94
        mainfrm.h           985 bytes   12:00 am  Thu Apr  7 94
        mainfrm.obj       5,197 bytes   12:00 am  Mon Jun 20 94
        persist.cpp       5,127 bytes   12:00 am  Thu Apr  7 94
        persist.h           644 bytes   12:00 am  Thu Apr  7 94
        persist.obj       6,940 bytes   12:00 am  Mon Jun 20 94
        RES                 <DIR>       10:17 am  Mon Jun 10 96
        resource.h          359 bytes   12:00 am  Thu Apr  7 94
        stdafx.cpp          204 bytes   12:00 am  Thu Apr  7 94
        stdafx.h            283 bytes   12:00 am  Thu Apr  7 94
        stdafx.obj          276 bytes   12:00 am  Mon Jun 20 94

I:\VC152\INSIDE2\SAMPLES\EX14A\RES
        .                   <DIR>       10:17 am  Mon Jun 10 96
        ..                  <DIR>       10:17 am  Mon Jun 10 96
        ex14a.ico           768 bytes   12:00 am  Thu Apr  7 94
        ex14a.rc2         1,538 bytes   12:00 am  Thu Apr  7 94
        toolbar.bmp       1,200 bytes   12:00 am  Thu Apr  7 94

I:\VC152\INSIDE2\SAMPLES\EX15A
        .                   <DIR>       10:18 am  Mon Jun 10 96
        ..                  <DIR>       10:19 am  Mon Jun 10 96
        ex15a.aps        21,005 bytes   12:00 am  Tue Apr 12 94
        ex15a.bsc       421,011 bytes   12:00 am  Tue May  3 94
        ex15a.clw         2,140 bytes   12:00 am  Tue Apr 12 94
        ex15a.cpp         3,525 bytes   12:00 am  Tue Apr 12 94
        ex15a.def           347 bytes   12:00 am  Tue Apr 12 94
        ex15a.exe     1,297,056 bytes   12:00 am  Mon Jun 20 94
        ex15a.h             812 bytes   12:00 am  Tue Apr 12 94
        ex15a.mak         3,445 bytes   12:00 am  Mon Jun 20 94
        ex15a.obj         6,057 bytes   12:00 am  Mon Jun 20 94
        ex15a.pdb       158,902 bytes   12:00 am  Tue May  3 94
```

```
ex15a.rc          7,800 bytes   12:00 am   Tue Apr 12 94
ex15a.res        10,882 bytes   12:00 am   Mon Jun 20 94
ex15a.vcw           90 bytes    12:00 am   Mon Jun 20 94
ex15a.wsp          170 bytes    12:00 am   Mon Jun 20 94
ex15adoc.cpp     1,946 bytes    12:00 am   Tue Apr 12 94
ex15adoc.h       1,009 bytes    12:00 am   Tue Apr 12 94
ex15adoc.obj     4,969 bytes    12:00 am   Mon Jun 20 94
ex15avw.cpp      2,841 bytes    12:00 am   Tue May  3 94
ex15avw.h        1,312 bytes    12:00 am   Tue Apr 12 94
ex15avw.obj      7,531 bytes    12:00 am   Mon Jun 20 94
mainfrm.cpp      2,626 bytes    12:00 am   Tue May  3 94
mainfrm.h          980 bytes    12:00 am   Tue Apr 12 94
mainfrm.obj      5,155 bytes    12:00 am   Mon Jun 20 94
readme.txt       4,342 bytes    12:00 am   Tue Apr 12 94
RES                <DIR>        10:18 am   Mon Jun 10 96
resource.h         713 bytes    12:00 am   Tue Apr 12 94
stdafx.cpp         204 bytes    12:00 am   Tue Apr 12 94
stdafx.h           299 bytes    12:00 am   Tue Apr 12 94
stdafx.obj         276 bytes    12:00 am   Mon Jun 20 94
student.cpp        286 bytes    12:00 am   Tue May  3 94
student.h          990 bytes    12:00 am   Tue May 31 94
student.obj      1,153 bytes    12:00 am   Mon Jun 20 94

I:\VC152\INSIDE2\SAMPLES\EX15A\RES
.                  <DIR>        10:18 am   Mon Jun 10 96
..                 <DIR>        10:18 am   Mon Jun 10 96
ex15a.ico          768 bytes    12:00 am   Tue Apr 12 94
ex15a.rc2        1,538 bytes    12:00 am   Tue Apr 12 94
toolbar.bmp      1,200 bytes    12:00 am   Tue Apr 12 94

I:\VC152\INSIDE2\SAMPLES\EX15B
.                  <DIR>        10:18 am   Mon Jun 10 96
..                 <DIR>        10:19 am   Mon Jun 10 96
ex15b.aps       21,653 bytes    12:00 am   Tue Apr 12 94
ex15b.clw        2,081 bytes    12:00 am   Tue Apr 12 94
ex15b.cpp        3,350 bytes    12:00 am   Thu Apr  7 94
ex15b.def          347 bytes    12:00 am   Thu Apr  7 94
ex15b.exe      296,192 bytes    12:00 am   Mon Jun 20 94
ex15b.h            807 bytes    12:00 am   Thu Apr  7 94
ex15b.mak        3,492 bytes    12:00 am   Mon Jun 20 94
ex15b.obj        5,944 bytes    12:00 am   Mon Jun 20 94
ex15b.rc         7,795 bytes    12:00 am   Tue Apr 12 94
ex15b.res       11,241 bytes    12:00 am   Mon Jun 20 94
ex15b.vcw           90 bytes    12:00 am   Mon Jun 20 94
ex15b.wsp          170 bytes    12:00 am   Mon Jun 20 94
mainfrm.cpp      2,676 bytes    12:00 am   Tue May  3 94
mainfrm.h          978 bytes    12:00 am   Thu Apr  7 94
mainfrm.obj      5,050 bytes    12:00 am   Mon Jun 20 94
RES                <DIR>        10:19 am   Mon Jun 10 96
resource.h         952 bytes    12:00 am   Thu Apr  7 94
stdafx.cpp         204 bytes    12:00 am   Thu Apr  7 94
stdafx.h           283 bytes    12:00 am   Thu Apr  7 94
stdafx.obj         276 bytes    12:00 am   Mon Jun 20 94
student.cpp        457 bytes    12:00 am   Thu Apr  7 94
```

```
        student.h           406 bytes   12:00 am  Wed Apr 13 94
        student.obj       2,576 bytes   12:00 am  Mon Jun 20 94
        studoc.cpp        2,511 bytes   12:00 am  Tue Apr 12 94
        studoc.h          1,010 bytes   12:00 am  Tue May  3 94
        studoc.obj        4,827 bytes   12:00 am  Mon Jun 20 94
        stuview.cpp       8,618 bytes   12:00 am  Tue May 31 94
        stuview.h         2,073 bytes   12:00 am  Tue May 31 94
        stuview.obj      10,882 bytes   12:00 am  Mon Jun 20 94

I:\VC152\INSIDE2\SAMPLES\EX15B\RES
        .                   <DIR>       10:19 am  Mon Jun 10 96
        ..                  <DIR>       10:18 am  Mon Jun 10 96
        ex15b.ico           768 bytes   12:00 am  Thu Apr  7 94
        ex15b.rc2         1,631 bytes   12:00 am  Thu Apr  7 94
        toolbar.bmp       1,558 bytes   12:00 am  Tue Apr 12 94

I:\VC152\INSIDE2\SAMPLES\EX16A
        .                   <DIR>       10:19 am  Mon Jun 10 96
        ..                  <DIR>       10:19 am  Mon Jun 10 96
        ex16a.aps        21,751 bytes   12:00 am  Wed Apr 13 94
        ex16a.cpp         3,328 bytes   12:00 am  Thu Apr  7 94
        ex16a.def           347 bytes   12:00 am  Thu Apr  7 94
        ex16a.exe       296,928 bytes   12:00 am  Mon Jun 20 94
        ex16a.h             807 bytes   12:00 am  Thu Apr  7 94
        ex16a.mak         3,355 bytes   12:00 am  Mon Jun 20 94
        ex16a.obj         6,081 bytes   12:00 am  Mon Jun 20 94
        ex16a.rc          8,206 bytes   12:00 am  Wed Apr 13 94
        ex16a.res        11,305 bytes   12:00 am  Mon Jun 20 94
        ex16a.vcw            90 bytes   12:00 am  Mon Jun 20 94
        ex16a.wsp           170 bytes   12:00 am  Mon Jun 20 94
        mainfrm.cpp       2,665 bytes   12:00 am  Thu Apr  7 94
        mainfrm.h         1,026 bytes   12:00 am  Thu Apr  7 94
        mainfrm.obj       5,342 bytes   12:00 am  Mon Jun 20 94
        RES                 <DIR>       10:19 am  Mon Jun 10 96
        resource.h          934 bytes   12:00 am  Thu Apr  7 94
        stdafx.cpp          204 bytes   12:00 am  Thu Apr  7 94
        stdafx.h            283 bytes   12:00 am  Thu Apr  7 94
        stdafx.obj          276 bytes   12:00 am  Mon Jun 20 94
        student.cpp         500 bytes   12:00 am  Thu Apr  7 94
        student.h         1,013 bytes   12:00 am  Tue May 31 94
        student.obj       2,675 bytes   12:00 am  Mon Jun 20 94
        studoc.cpp        2,738 bytes   12:00 am  Thu Apr  7 94
        studoc.h            973 bytes   12:00 am  Thu Apr  7 94
        studoc.obj        4,929 bytes   12:00 am  Mon Jun 20 94
        stuview.cpp       8,626 bytes   12:00 am  Thu Apr  7 94
        stuview.h         2,080 bytes   12:00 am  Thu Apr  7 94
        stuview.obj      11,275 bytes   12:00 am  Mon Jun 20 94

I:\VC152\INSIDE2\SAMPLES\EX16A\RES
        .                   <DIR>       10:19 am  Mon Jun 10 96
        ..                  <DIR>       10:19 am  Mon Jun 10 96
        ex16a.ico           768 bytes   12:00 am  Thu Apr  7 94
        ex16a.rc2         1,631 bytes   12:00 am  Thu Apr  7 94
        toolbar.bmp       1,558 bytes   12:00 am  Wed Apr 13 94
```

I:\VC152\MSVC15

.	<DIR>	11:19 am	Mon Jun 10 96
..	<DIR>	11:19 am	Mon Jun 10 96
BIN	<DIR>	10:24 am	Mon Jun 10 96
DEBUG	<DIR>	10:27 am	Mon Jun 10 96
HELP	<DIR>	10:43 am	Mon Jun 10 96
INCLUDE	<DIR>	10:44 am	Mon Jun 10 96
LIB	<DIR>	10:47 am	Mon Jun 10 96
MFC	<DIR>	11:00 am	Mon Jun 10 96
MSQUERY	<DIR>	11:02 am	Mon Jun 10 96
NODEBUG	<DIR>	11:02 am	Mon Jun 10 96
OLE2	<DIR>	11:05 am	Mon Jun 10 96
PEN	<DIR>	11:12 am	Mon Jun 10 96
PHARLAP	<DIR>	11:12 am	Mon Jun 10 96
PSS-SAMP	<DIR>	11:12 am	Mon Jun 10 96
readme.txt	1,257 bytes	2:00 pm	Fri Sep 16 94
REDIST	<DIR>	11:14 am	Mon Jun 10 96
SAMPLES	<DIR>	11:19 am	Mon Jun 10 96
setup.exe	671,744 bytes	6:10 am	Fri Jan 13 95
setup.inf	131,671 bytes	2:10 pm	Fri Jan 13 95
SOURCE	<DIR>	11:19 am	Mon Jun 10 96
TCPIP	<DIR>	11:19 am	Mon Jun 10 96
wsetup.hlp	69,439 bytes	2:00 pm	Fri Sep 16 94

I:\VC152\MSVC15\BIN

.	<DIR>	10:24 am	Mon Jun 10 96
..	<DIR>	11:19 am	Mon Jun 10 96
appwzlib.dll	115,904 bytes	2:00 pm	Fri Sep 16 94
apstudio.exe	900,928 bytes	2:00 pm	Fri Sep 16 94
atrm1111.fnt	6,446 bytes	2:00 pm	Fri Sep 16 94
autoload.mak	24 bytes	2:00 pm	Fri Sep 16 94
bscmake.exe	116,224 bytes	2:00 pm	Fri Sep 16 94
c1.err	54,596 bytes	2:00 pm	Fri Sep 16 94
c13216.exe	333,312 bytes	2:00 pm	Fri Sep 16 94
c1xx3216.exe	552,960 bytes	2:00 pm	Fri Sep 16 94
c23.err	4,563 bytes	2:00 pm	Fri Sep 16 94
c23216.exe	493,056 bytes	2:00 pm	Fri Sep 16 94
c23pcd.exe	332,800 bytes	2:00 pm	Fri Sep 16 94
c33216.exe	244,224 bytes	2:00 pm	Fri Sep 16 94
cl.def	1,717 bytes	2:00 pm	Fri Sep 16 94
cl.err	1,756 bytes	2:00 pm	Fri Sep 16 94
cl.exe	91,648 bytes	2:00 pm	Fri Sep 16 94
cl.msg	4,279 bytes	2:00 pm	Fri Sep 16 94
clonedbg.dll	16,384 bytes	2:00 pm	Fri Sep 16 94
contents.exe	19,904 bytes	2:00 pm	Fri Sep 16 94
cv.exe	490,768 bytes	2:00 pm	Fri Sep 16 94
cv.ico	1,086 bytes	2:00 pm	Fri Sep 16 94
cvpack.exe	169,472 bytes	2:10 pm	Fri Jan 13 95
cvw.exe	14,336 bytes	2:00 pm	Fri Sep 16 94
cvw1.386	5,194 bytes	2:00 pm	Fri Sep 16 94
cvw4.exe	431,616 bytes	2:00 pm	Fri Sep 16 94
cvwin.dll	12,304 bytes	2:00 pm	Fri Sep 16 94
d2n.bat	23 bytes	2:00 pm	Fri Sep 16 94

```
dbwin.dll        17,408 bytes   2:00 pm  Fri Sep 16 94
dbwin.exe        16,384 bytes   2:00 pm  Fri Sep 16 94
ddespy.exe       29,184 bytes   2:00 pm  Fri Sep 16 94
defo2v.dll       36,112 bytes   2:00 pm  Fri Sep 16 94
dfview.exe       79,792 bytes   2:00 pm  Fri Sep 16 94
disp200.dll      13,824 bytes   2:00 pm  Fri Sep 16 94
disptest.exe    868,608 bytes   2:00 pm  Fri Sep 16 94
dobjview.exe     69,392 bytes   2:00 pm  Fri Sep 16 94
dosxnt.386        9,343 bytes   2:00 pm  Fri Sep 16 94
dosxnt.exe      393,942 bytes   2:00 pm  Fri Sep 16 94
eed1can.dll      97,784 bytes   2:00 pm  Fri Sep 16 94
eed1cxx.dll     112,604 bytes   2:00 pm  Fri Sep 16 94
eew0can.dll      98,112 bytes   2:00 pm  Fri Sep 16 94
eew0cxx.dll     112,736 bytes   2:00 pm  Fri Sep 16 94
eew0cxxv.dll    112,768 bytes   2:10 pm  Fri Jan 13 95
emd1d1.dll       71,130 bytes   2:00 pm  Fri Sep 16 94
emw0w0.dll       75,392 bytes   2:00 pm  Fri Sep 16 94
emw0w0v.dll      75,392 bytes   2:00 pm  Fri Sep 16 94
exehdr.exe       67,072 bytes   2:00 pm  Fri Sep 16 94
fcount.bat          179 bytes   2:00 pm  Fri Sep 16 94
fcountw.bat         187 bytes   2:00 pm  Fri Sep 16 94
fcover.bat          180 bytes   2:00 pm  Fri Sep 16 94
fcoverw.bat         188 bytes   2:00 pm  Fri Sep 16 94
fontedit.exe     58,976 bytes   2:00 pm  Fri Sep 16 94
fontedit.hlp     16,124 bytes   2:00 pm  Fri Sep 16 94
fsample.bat         187 bytes   2:00 pm  Fri Sep 16 94
ftengine.dll     43,520 bytes   2:00 pm  Fri Sep 16 94
ftime.bat           179 bytes   2:00 pm  Fri Sep 16 94
ftimew.bat          187 bytes   2:00 pm  Fri Sep 16 94
ftui.dll         68,096 bytes   2:00 pm  Fri Sep 16 94
guidgen.exe      31,680 bytes   2:00 pm  Fri Sep 16 94
hc.bat              81 bytes   2:00 pm  Fri Sep 16 94
hc30.exe        133,835 bytes   2:00 pm  Fri Sep 16 94
hc31.err          7,463 bytes   2:00 pm  Fri Sep 16 94
hc31.exe        174,439 bytes   2:00 pm  Fri Sep 16 94
hdxdll.dll       51,640 bytes   2:00 pm  Fri Sep 16 94
heapwalk.exe     45,184 bytes   2:00 pm  Fri Sep 16 94
hook.dll          5,120 bytes   2:00 pm  Fri Sep 16 94
implib.exe       70,144 bytes   2:00 pm  Fri Sep 16 94
irotview.exe  1,307,476 bytes   2:00 pm  Fri Sep 16 94
lcount.bat          179 bytes   2:00 pm  Fri Sep 16 94
lcountw.bat         187 bytes   2:00 pm  Fri Sep 16 94
lcover.bat          180 bytes   2:00 pm  Fri Sep 16 94
lcoverw.bat         188 bytes   2:00 pm  Fri Sep 16 94
lib.exe         134,144 bytes   2:00 pm  Fri Sep 16 94
link.exe        364,544 bytes   2:10 pm  Fri Jan 13 95
lrpcspy.exe      31,744 bytes   2:00 pm  Fri Sep 16 94
lsample.bat         187 bytes   2:00 pm  Fri Sep 16 94
ltime.bat           179 bytes   2:00 pm  Fri Sep 16 94
ltimew.bat          187 bytes   2:00 pm  Fri Sep 16 94
makehm.exe       22,033 bytes   2:10 pm  Fri Jan 13 95
mapsym.exe       33,001 bytes   2:00 pm  Fri Sep 16 94
mark.exe         21,751 bytes   2:00 pm  Fri Sep 16 94
markmidi.exe      8,243 bytes   2:00 pm  Fri Sep 16 94
```

```
mfcappwz.exe   272,128 bytes   2:00 pm  Fri Sep 16 94
mfcclswz.dll   418,928 bytes   2:00 pm  Fri Sep 16 94
mktyplib.exe    54,784 bytes   2:00 pm  Fri Sep 16 94
mmd.386          9,273 bytes   2:00 pm  Fri Sep 16 94
mpc.exe        105,472 bytes   2:00 pm  Fri Sep 16 94
mrbc.exe        48,707 bytes   2:00 pm  Fri Sep 16 94
mscopts.dll    105,984 bytes   2:00 pm  Fri Sep 16 94
msole2.bas       5,004 bytes   2:00 pm  Fri Sep 16 94
msole2.vbx      10,560 bytes   2:00 pm  Fri Sep 16 94
msolevbx.dll   144,512 bytes   2:00 pm  Fri Sep 16 94
msvc.exe       775,760 bytes   2:10 pm  Fri Jan 13 95
msvchelp.dll    16,384 bytes   2:00 pm  Fri Sep 16 94
msvcver.dll      3,888 bytes   2:00 pm  Fri Sep 16 94
mvapi.dll        5,120 bytes   2:00 pm  Fri Sep 16 94
n2d.bat            451 bytes   6:45 pm  Sat Sep 30 95
nmake.exe      128,512 bytes   2:00 pm  Fri Sep 16 94
nmaker.exe      94,747 bytes   2:00 pm  Fri Sep 16 94
nmd1pcd.dll     38,514 bytes   2:00 pm  Fri Sep 16 94
nmw0pcd.dll     41,952 bytes   2:00 pm  Fri Sep 16 94
ntspawn.exe     20,992 bytes   2:00 pm  Fri Sep 16 94
oem08.fon        4,752 bytes   2:00 pm  Fri Sep 16 94
oem10.fon        5,264 bytes   2:00 pm  Fri Sep 16 94
ole2free.exe    43,232 bytes   2:00 pm  Fri Sep 16 94
ole2view.exe   140,176 bytes   2:00 pm  Fri Sep 16 94
ole2view.hlp    65,406 bytes   2:00 pm  Fri Sep 16 94
oled2n.bat          26 bytes   2:00 pm  Fri Sep 16 94
olen2d.bat          26 bytes   2:00 pm  Fri Sep 16 94
oleswtch.bat     3,349 bytes   2:00 pm  Fri Sep 16 94
plist.exe      139,776 bytes   2:00 pm  Fri Sep 16 94
prep.exe       119,808 bytes   2:00 pm  Fri Sep 16 94
profile.exe     87,754 bytes   2:00 pm  Fri Sep 16 94
profilew.exe    71,680 bytes   2:00 pm  Fri Sep 16 94
q23.exe        208,384 bytes   2:00 pm  Fri Sep 16 94
qwin.hlp        31,010 bytes   2:00 pm  Fri Sep 16 94
rc.exe          44,099 bytes   2:00 pm  Fri Sep 16 94
rcdll.dll      167,424 bytes   2:00 pm  Fri Sep 16 94
rcpp.exe        99,499 bytes   2:00 pm  Fri Sep 16 94
sbrpack.exe     14,379 bytes   2:00 pm  Fri Sep 16 94
shd1.dll        42,536 bytes   2:00 pm  Fri Sep 16 94
shed.exe        96,768 bytes   2:00 pm  Fri Sep 16 94
shed.hlp        85,476 bytes   2:00 pm  Fri Sep 16 94
shw0.dll        43,984 bytes   2:00 pm  Fri Sep 16 94
shw0v.dll       43,984 bytes   2:00 pm  Fri Sep 16 94
spy.exe         27,136 bytes   2:00 pm  Fri Sep 16 94
stress.dll      50,400 bytes   2:00 pm  Fri Sep 16 94
stress.exe      50,304 bytes   2:00 pm  Fri Sep 16 94
stress.hlp      22,185 bytes   2:00 pm  Fri Sep 16 94
stress.ini       1,216 bytes   2:00 pm  Fri Sep 16 94
stresshk.dll     4,549 bytes   2:00 pm  Fri Sep 16 94
stresslg.dll    10,831 bytes   2:00 pm  Fri Sep 16 94
sysincl.dat      1,512 bytes  11:18 pm  Fri Sep 29 95
templdef.exe    28,989 bytes   2:10 pm  Fri Jan 13 95
tibrowse.exe    13,120 bytes   2:00 pm  Fri Sep 16 94
tld1loc.dll     29,400 bytes   2:00 pm  Fri Sep 16 94
```

```
tlw0loc.dll     100,448 bytes   2:00 pm  Fri Sep 16 94
tlw0ncl.dll     108,640 bytes   2:00 pm  Fri Sep 16 94
trace.exe        25,246 bytes   2:00 pm  Fri Sep 16 94
tracer.exe       34,464 bytes   2:10 pm  Fri Jan 13 95
vb.hlp        1,895,620 bytes   2:00 pm  Fri Sep 16 94
vgasys.fnt        5,441 bytes   2:00 pm  Fri Sep 16 94
VIDEO               <DIR>      10:24 am  Mon Jun 10 96
vmb.386           9,273 bytes   2:00 pm  Fri Sep 16 94
wdeb386.exe     103,078 bytes   2:00 pm  Fri Sep 16 94
winstub.exe         610 bytes   2:00 pm  Fri Sep 16 94
wintee.exe       25,799 bytes   2:00 pm  Fri Sep 16 94
wintee32.exe     37,888 bytes   2:00 pm  Fri Sep 16 94
winteer.exe      20,205 bytes   2:00 pm  Fri Sep 16 94
wps.exe          23,040 bytes   2:00 pm  Fri Sep 16 94
wx.exe           13,077 bytes   2:00 pm  Fri Sep 16 94
wxsrvr.exe       17,920 bytes   2:00 pm  Fri Sep 16 94
zoomin.exe        6,464 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\BIN\VIDEO
.                   <DIR>      10:24 am  Mon Jun 10 96
..                  <DIR>      10:24 am  Mon Jun 10 96
evrxvesa.com      4,228 bytes   2:00 pm  Fri Sep 16 94
htvesa.com        3,502 bytes   2:00 pm  Fri Sep 16 94
msherc.com        6,947 bytes   2:00 pm  Fri Sep 16 94
vesa1a1b.exe      3,095 bytes   2:00 pm  Fri Sep 16 94
vesa1c.exe        2,844 bytes   2:00 pm  Fri Sep 16 94
vesa1d.exe        3,525 bytes   2:00 pm  Fri Sep 16 94
vvesa.com         9,715 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\DEBUG
.                   <DIR>      10:27 am  Mon Jun 10 96
..                  <DIR>      11:19 am  Mon Jun 10 96
compobj.dll     247,808 bytes   2:00 pm  Fri Sep 16 94
compobj.sym       4,612 bytes   2:00 pm  Fri Sep 16 94
gdi.exe         230,912 bytes   2:00 pm  Fri Sep 16 94
gdi.sym           6,004 bytes   2:00 pm  Fri Sep 16 94
krnl286.exe     135,404 bytes   2:00 pm  Fri Sep 16 94
krnl286.sym       4,308 bytes   2:00 pm  Fri Sep 16 94
krnl386.exe      92,816 bytes   2:00 pm  Fri Sep 16 94
krnl386.sym       4,340 bytes   2:00 pm  Fri Sep 16 94
mfc250d.dll   1,914,140 bytes  11:08 pm  Wed Apr 26 95
mfc250d.sym     139,444 bytes  11:22 pm  Wed Apr 26 95
mfcd250d.dll    534,644 bytes  11:20 pm  Wed Apr 26 95
mfcd250d.sym     10,324 bytes  11:22 pm  Wed Apr 26 95
mfcn250d.dll    427,028 bytes  11:22 pm  Wed Apr 26 95
mfcn250d.sym      6,404 bytes  11:22 pm  Wed Apr 26 95
mfco250d.dll    986,416 bytes  11:17 pm  Wed Apr 26 95
mfco250d.sym     59,060 bytes  11:22 pm  Wed Apr 26 95
mmsystem.dll     66,656 bytes   2:00 pm  Fri Sep 16 94
mmsystem.sym      5,428 bytes   2:00 pm  Fri Sep 16 94
ole2.dll        615,424 bytes   2:00 pm  Fri Sep 16 94
ole2.sym          4,180 bytes   2:00 pm  Fri Sep 16 94
ole2conv.dll     57,328 bytes   2:00 pm  Fri Sep 16 94
ole2conv.sym      7,748 bytes   2:00 pm  Fri Sep 16 94
```

```
        ole2disp.dll   941,368 bytes   2:00 pm  Fri Sep 16 94
        ole2disp.sym    48,676 bytes   2:00 pm  Fri Sep 16 94
        ole2nls.dll    656,224 bytes   2:00 pm  Fri Sep 16 94
        ole2nls.sym      4,500 bytes   2:00 pm  Fri Sep 16 94
        ole2prox.dll    90,112 bytes   2:00 pm  Fri Sep 16 94
        ole2prox.sym       164 bytes   2:00 pm  Fri Sep 16 94
        readme.txt       1,399 bytes   2:00 pm  Fri Sep 16 94
        storage.dll    522,752 bytes   2:00 pm  Fri Sep 16 94
        storage.sym     52,708 bytes   2:00 pm  Fri Sep 16 94
        typelib.dll  1,176,428 bytes   2:00 pm  Fri Sep 16 94
        typelib.sym     74,148 bytes   2:00 pm  Fri Sep 16 94
        user.exe       280,896 bytes   2:00 pm  Fri Sep 16 94
        user.sym        11,620 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\HELP
        .                <DIR>        10:43 am  Mon Jun 10 96
        ..               <DIR>        11:19 am  Mon Jun 10 96
        apstudio.hlp   361,428 bytes   2:00 pm  Fri Sep 16 94
        bscmake.wri     27,776 bytes   2:00 pm  Fri Sep 16 94
        cdsetup.hlp     69,439 bytes   2:00 pm  Fri Sep 16 94
        config.wri      12,672 bytes   2:00 pm  Fri Sep 16 94
        contents.hlp    49,690 bytes   2:00 pm  Fri Sep 16 94
        cv.hlp         181,223 bytes   2:00 pm  Fri Sep 16 94
        cvpack.wri       9,728 bytes   2:00 pm  Fri Sep 16 94
        errata.wri      32,000 bytes   2:00 pm  Fri Sep 16 94
        errors.hlp     748,796 bytes   2:00 pm  Fri Sep 16 94
        hlptab.dll       1,048 bytes   2:00 pm  Fri Sep 16 94
        implib.wri       6,144 bytes   2:00 pm  Fri Sep 16 94
        lib.wri         35,840 bytes   2:00 pm  Fri Sep 16 94
        mcistrwh.hlp   153,600 bytes   2:00 pm  Fri Sep 16 94
        mfc.hlp      2,738,544 bytes   2:10 pm  Fri Jan 13 95
        mfcext.hlp     174,705 bytes   2:10 pm  Fri Jan 13 95
        mfcnotes.hlp   399,970 bytes   2:00 pm  Fri Sep 16 94
        mfcsamp.hlp    116,780 bytes   2:00 pm  Fri Sep 16 94
        moveapi.wri     29,056 bytes   2:00 pm  Fri Sep 16 94
        mscopts.hlp    187,856 bytes   2:00 pm  Fri Sep 16 94
        mscxx.hlp    1,740,175 bytes   2:00 pm  Fri Sep 16 94
        msquery.hlp    439,138 bytes   2:00 pm  Fri Sep 16 94
        msvc.hlp       466,296 bytes   2:00 pm  Fri Sep 16 94
        msvchelp.idx   129,162 bytes   2:10 pm  Fri Jan 13 95
        nmake.wri      165,120 bytes   2:00 pm  Fri Sep 16 94
        nthost.gid       8,628 bytes   2:01 pm  Thu Apr 25 96
        nthost.hlp      15,783 bytes   2:00 pm  Fri Sep 16 94
        odbcapi.hlp    451,663 bytes   2:00 pm  Fri Sep 16 94
        ole2ui.hlp     101,756 bytes   2:00 pm  Fri Sep 16 94
        olesdkv1.hlp 1,744,999 bytes   2:00 pm  Fri Sep 16 94
        olesdkv2.hlp   722,634 bytes   2:00 pm  Fri Sep 16 94
        penapiwh.hlp   207,513 bytes   2:00 pm  Fri Sep 16 94
        printdll.dll    16,384 bytes   2:00 pm  Fri Sep 16 94
        probrpt.wri      4,736 bytes   2:00 pm  Fri Sep 16 94
        pssqh.hlp        1,021 bytes   2:00 pm  Fri Sep 16 94
        psswh.hlp       96,931 bytes   2:00 pm  Fri Sep 16 94
        readme.hlp      14,593 bytes   2:00 pm  Fri Sep 16 94
        readme.wri      70,528 bytes   2:10 pm  Fri Jan 13 95
```

```
samples.hlp      70,391 bytes   2:00 pm  Fri Sep 16 94
technote.dll     14,080 bytes   2:00 pm  Fri Sep 16 94
technote.ico        766 bytes   2:00 pm  Fri Sep 16 94
tips.hlp        185,537 bytes   2:00 pm  Fri Sep 16 94
tips.wri        130,304 bytes   2:00 pm  Fri Sep 16 94
tools.hlp        15,796 bytes   2:00 pm  Fri Sep 16 94
tools.wri        24,448 bytes   2:00 pm  Fri Sep 16 94
vcbks15.hdx   1,578,988 bytes   2:00 pm  Fri Sep 16 94
vcbks15.hlp  20,322,482 bytes   2:00 pm  Fri Sep 16 94
vcbks15.ind  15,581,184 bytes   2:00 pm  Fri Sep 16 94
vcsdk15.hlp  16,610,031 bytes   2:00 pm  Fri Sep 16 94
win31mwh.hlp    348,160 bytes   2:00 pm  Fri Sep 16 94
win31wh.hlp   3,386,753 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\INCLUDE
.                 <DIR>        10:44 am  Mon Jun 10 96
..                <DIR>        11:19 am  Mon Jun 10 96
assert.h            590 bytes   2:00 pm  Fri Sep 16 94
bios.h            5,839 bytes   2:00 pm  Fri Sep 16 94
cderr.h           2,256 bytes   2:00 pm  Fri Sep 16 94
cmacros.inc      19,341 bytes   2:00 pm  Fri Sep 16 94
cobjps.h          2,553 bytes   2:00 pm  Fri Sep 16 94
coguid.h          3,170 bytes   2:00 pm  Fri Sep 16 94
colordlg.h        1,804 bytes   2:00 pm  Fri Sep 16 94
commdlg.h        11,784 bytes   2:00 pm  Fri Sep 16 94
compobj.h        35,315 bytes   2:00 pm  Fri Sep 16 94
conio.h           1,609 bytes   2:00 pm  Fri Sep 16 94
cpl.h             5,810 bytes   2:00 pm  Fri Sep 16 94
ctype.h           3,502 bytes   2:00 pm  Fri Sep 16 94
custcntl.h        3,808 bytes   2:00 pm  Fri Sep 16 94
dde.h             4,656 bytes   2:00 pm  Fri Sep 16 94
ddeml.h          15,726 bytes   2:00 pm  Fri Sep 16 94
direct.h          1,105 bytes   2:00 pm  Fri Sep 16 94
dispatch.h       38,963 bytes   2:00 pm  Fri Sep 16 94
dlgs.h            5,796 bytes   2:00 pm  Fri Sep 16 94
dos.h             8,320 bytes   2:00 pm  Fri Sep 16 94
drivinit.h           57 bytes   2:00 pm  Fri Sep 16 94
dvobj.h          16,177 bytes   2:00 pm  Fri Sep 16 94
errno.h           1,700 bytes   2:00 pm  Fri Sep 16 94
fcntl.h           1,609 bytes   2:00 pm  Fri Sep 16 94
fgraph.fd        14,001 bytes   2:00 pm  Fri Sep 16 94
fgraph.fi        13,704 bytes   2:00 pm  Fri Sep 16 94
float.h           7,313 bytes   2:00 pm  Fri Sep 16 94
fstream.h         4,328 bytes   2:00 pm  Fri Sep 16 94
graph.h          17,570 bytes   2:00 pm  Fri Sep 16 94
initguid.h        1,447 bytes   2:00 pm  Fri Sep 16 94
io.h              3,999 bytes   2:00 pm  Fri Sep 16 94
iomanip.h         4,182 bytes   2:00 pm  Fri Sep 16 94
ios.h             6,096 bytes   2:00 pm  Fri Sep 16 94
iostream.h        1,668 bytes   2:00 pm  Fri Sep 16 94
istream.h         5,568 bytes   2:00 pm  Fri Sep 16 94
limits.h          1,700 bytes   2:00 pm  Fri Sep 16 94
locale.h          1,593 bytes   2:00 pm  Fri Sep 16 94
lzdos.h             120 bytes   2:00 pm  Fri Sep 16 94
```

```
lzexpand.h       3,645 bytes   2:00 pm  Fri Sep 16 94
malloc.h         4,212 bytes   2:00 pm  Fri Sep 16 94
mapi.h           7,263 bytes   2:10 pm  Fri Jan 13 95
math.h           8,773 bytes   2:00 pm  Fri Sep 16 94
memory.h         1,957 bytes   2:00 pm  Fri Sep 16 94
mmsystem.h      80,077 bytes   2:00 pm  Fri Sep 16 94
mmsystem.inc    60,312 bytes   2:00 pm  Fri Sep 16 94
moniker.h        9,673 bytes   2:00 pm  Fri Sep 16 94
new.h            1,140 bytes   2:00 pm  Fri Sep 16 94
odbcinst.h       4,470 bytes   2:00 pm  Fri Sep 16 94
odbcver.h        1,004 bytes   2:00 pm  Fri Sep 16 94
ole.h           25,791 bytes   2:00 pm  Fri Sep 16 94
olelcls.h        6,579 bytes   2:00 pm  Fri Sep 16 94
ole2.h          45,740 bytes   2:00 pm  Fri Sep 16 94
ole2dbg.h          553 bytes   2:00 pm  Fri Sep 16 94
ole2ui.h        35,007 bytes   2:10 pm  Fri Jan 13 95
ole2ver.h          103 bytes   2:00 pm  Fri Sep 16 94
oleguid.h        3,870 bytes   2:00 pm  Fri Sep 16 94
olenls.h        21,841 bytes   2:00 pm  Fri Sep 16 94
olestd.h        29,566 bytes   2:00 pm  Fri Sep 16 94
ostream.h        4,322 bytes   2:00 pm  Fri Sep 16 94
penwin.h        26,280 bytes   2:00 pm  Fri Sep 16 94
penwoem.h        2,713 bytes   2:00 pm  Fri Sep 16 94
pgchart.h        9,540 bytes   2:00 pm  Fri Sep 16 94
print.h         10,493 bytes   2:00 pm  Fri Sep 16 94
process.h        3,912 bytes   2:00 pm  Fri Sep 16 94
scode.h         11,051 bytes   2:00 pm  Fri Sep 16 94
scrnsave.h       6,739 bytes   2:00 pm  Fri Sep 16 94
search.h         1,443 bytes   2:00 pm  Fri Sep 16 94
setjmp.h           918 bytes   2:00 pm  Fri Sep 16 94
share.h            751 bytes   2:00 pm  Fri Sep 16 94
shellapi.h       2,857 bytes   2:00 pm  Fri Sep 16 94
signal.h         1,503 bytes   2:00 pm  Fri Sep 16 94
sql.h            8,546 bytes   2:00 pm  Fri Sep 16 94
sqlext.h        39,348 bytes   2:00 pm  Fri Sep 16 94
stdarg.h         1,221 bytes   2:00 pm  Fri Sep 16 94
stddef.h         1,474 bytes   2:00 pm  Fri Sep 16 94
stdio.h          7,816 bytes   2:00 pm  Fri Sep 16 94
stdiostr.h       1,387 bytes   2:00 pm  Fri Sep 16 94
stdlib.h         7,992 bytes   2:00 pm  Fri Sep 16 94
storage.h       14,085 bytes   2:00 pm  Fri Sep 16 94
streamb.h        4,480 bytes   2:00 pm  Fri Sep 16 94
stress.h         1,928 bytes   2:00 pm  Fri Sep 16 94
string.h         5,600 bytes   2:00 pm  Fri Sep 16 94
strstrea.h       2,680 bytes   2:00 pm  Fri Sep 16 94
SYS                <DIR>      10:44 am  Mon Jun 10 96
sysincl.dat      1,470 bytes   2:00 pm  Fri Sep 16 94
tchar.h          6,889 bytes   2:00 pm  Fri Sep 16 94
time.h           2,977 bytes   2:00 pm  Fri Sep 16 94
toolhelp.h      14,817 bytes   2:00 pm  Fri Sep 16 94
toolhelp.inc     8,768 bytes   2:00 pm  Fri Sep 16 94
varargs.h        1,274 bytes   2:00 pm  Fri Sep 16 94
variant.h        9,687 bytes   2:00 pm  Fri Sep 16 94
ver.h            9,360 bytes   2:00 pm  Fri Sep 16 94
```

```
        vmemory.h         1,526 bytes   2:00 pm  Fri Sep 16 94
        wfext.h           2,914 bytes   2:00 pm  Fri Sep 16 94
        windows.h       151,579 bytes   2:00 pm  Fri Sep 16 94
        windows.inc      65,122 bytes   2:00 pm  Fri Sep 16 94
        windowsx.h       63,745 bytes   2:00 pm  Fri Sep 16 94
        winmem32.h          837 bytes   2:00 pm  Fri Sep 16 94
        winsock.h        31,960 bytes   2:10 pm  Fri Jan 13 95

I:\VC152\MSVC15\INCLUDE\SYS
        .                   <DIR>       10:44 am  Mon Jun 10 96
        ..                  <DIR>       10:44 am  Mon Jun 10 96
        locking.h           793 bytes   2:00 pm  Fri Sep 16 94
        stat.h            2,248 bytes   2:00 pm  Fri Sep 16 94
        timeb.h           1,162 bytes   2:00 pm  Fri Sep 16 94
        types.h           1,103 bytes   2:00 pm  Fri Sep 16 94
        utime.h           1,373 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\LIB
        .                   <DIR>       10:47 am  Mon Jun 10 96
        ..                  <DIR>       11:19 am  Mon Jun 10 96
        binmode.obj         127 bytes   2:00 pm  Fri Sep 16 94
        cdllcaw.lib     301,183 bytes   2:00 pm  Fri Sep 16 94
        cdllcew.lib     323,045 bytes   2:00 pm  Fri Sep 16 94
        clibc7.lib      411,873 bytes   2:00 pm  Fri Sep 16 94
        clibca.lib      384,879 bytes   2:00 pm  Fri Sep 16 94
        clibcaw.lib     315,715 bytes   2:00 pm  Fri Sep 16 94
        clibcawq.lib    469,371 bytes   2:00 pm  Fri Sep 16 94
        clibce.lib      424,673 bytes   2:00 pm  Fri Sep 16 94
        clibcew.lib     337,577 bytes   2:00 pm  Fri Sep 16 94
        clibcewq.lib    489,221 bytes   2:00 pm  Fri Sep 16 94
        cnocrtdw.lib      2,069 bytes   2:00 pm  Fri Sep 16 94
        cnocrtw.lib       2,581 bytes   2:00 pm  Fri Sep 16 94
        commdlg.lib       2,048 bytes   2:00 pm  Fri Sep 16 94
        commode.obj         116 bytes   2:00 pm  Fri Sep 16 94
        compobj.lib      22,353 bytes   2:00 pm  Fri Sep 16 94
        cp437.obj           221 bytes   2:00 pm  Fri Sep 16 94
        crtcom.lib        4,633 bytes   2:00 pm  Fri Sep 16 94
        ddeml.lib         3,072 bytes   2:00 pm  Fri Sep 16 94
        fastadd.obj         505 bytes   2:10 pm  Fri Jan 13 95
        fileinfo.obj        119 bytes   2:00 pm  Fri Sep 16 94
        FONT                <DIR>       10:47 am  Mon Jun 10 96
        graphics.lib     98,259 bytes   2:00 pm  Fri Sep 16 94
        ldllcaw.lib     308,353 bytes   2:00 pm  Fri Sep 16 94
        ldllcew.lib     331,239 bytes   2:00 pm  Fri Sep 16 94
        libw.lib         80,351 bytes   2:00 pm  Fri Sep 16 94
        llibc7.lib      423,171 bytes   2:00 pm  Fri Sep 16 94
        llibca.lib      396,743 bytes   2:00 pm  Fri Sep 16 94
        llibcaw.lib     321,861 bytes   2:00 pm  Fri Sep 16 94
        llibcawq.lib    476,029 bytes   2:00 pm  Fri Sep 16 94
        llibce.lib      435,971 bytes   2:00 pm  Fri Sep 16 94
        llibcew.lib     345,259 bytes   2:00 pm  Fri Sep 16 94
        llibcewq.lib    497,415 bytes   2:00 pm  Fri Sep 16 94
        lnocrtdw.lib      2,069 bytes   2:00 pm  Fri Sep 16 94
        lnocrtw.lib       2,581 bytes   2:00 pm  Fri Sep 16 94
```

```
lseekchk.obj        119 bytes   2:00 pm  Fri Sep 16 94
lzexpand.lib      2,048 bytes   2:00 pm  Fri Sep 16 94
lzexpc.lib       26,249 bytes   2:00 pm  Fri Sep 16 94
lzexpl.lib       26,249 bytes   2:00 pm  Fri Sep 16 94
lzexpm.lib       23,177 bytes   2:00 pm  Fri Sep 16 94
lzexps.lib       22,665 bytes   2:00 pm  Fri Sep 16 94
mdllcaw.lib     292,337 bytes   2:00 pm  Fri Sep 16 94
mdllcew.lib     314,711 bytes   2:00 pm  Fri Sep 16 94
mlibc7.lib      402,009 bytes   2:00 pm  Fri Sep 16 94
mlibca.lib      376,613 bytes   2:00 pm  Fri Sep 16 94
mlibcaw.lib     303,279 bytes   2:00 pm  Fri Sep 16 94
mlibcawq.lib    457,447 bytes   2:00 pm  Fri Sep 16 94
mlibce.lib      414,809 bytes   2:00 pm  Fri Sep 16 94
mlibcew.lib     326,165 bytes   2:00 pm  Fri Sep 16 94
mlibcewq.lib    477,809 bytes   2:00 pm  Fri Sep 16 94
mmsystem.lib     12,800 bytes   2:00 pm  Fri Sep 16 94
mnocrtdw.lib      2,069 bytes   2:00 pm  Fri Sep 16 94
mnocrtw.lib       2,581 bytes   2:00 pm  Fri Sep 16 94
movetr.lib        9,241 bytes   2:00 pm  Fri Sep 16 94
novesa.obj           94 bytes   2:00 pm  Fri Sep 16 94
odbc.lib          5,120 bytes   2:00 pm  Fri Sep 16 94
odbcinst.lib      3,072 bytes   2:00 pm  Fri Sep 16 94
oldnames.lib      5,647 bytes   2:00 pm  Fri Sep 16 94
ole2.lib         42,155 bytes   2:00 pm  Fri Sep 16 94
ole2disp.lib     13,963 bytes   2:00 pm  Fri Sep 16 94
ole2nls.lib       2,048 bytes   2:00 pm  Fri Sep 16 94
olecli.lib        6,144 bytes   2:00 pm  Fri Sep 16 94
olesvr.lib        2,048 bytes   2:00 pm  Fri Sep 16 94
patchdll.bat        534 bytes   2:10 pm  Fri Jan 13 95
pcdm.lib         21,559 bytes   2:00 pm  Fri Sep 16 94
penwin.lib        8,192 bytes   2:00 pm  Fri Sep 16 94
pgchart.lib      46,707 bytes   2:00 pm  Fri Sep 16 94
scrnsave.lib     11,805 bytes   2:00 pm  Fri Sep 16 94
sdllcaw.lib     285,679 bytes   2:00 pm  Fri Sep 16 94
sdllcew.lib     307,541 bytes   2:00 pm  Fri Sep 16 94
setargv.obj         142 bytes   2:00 pm  Fri Sep 16 94
shell.lib         2,048 bytes   2:00 pm  Fri Sep 16 94
slibc7.lib      391,207 bytes   2:00 pm  Fri Sep 16 94
slibca.lib      365,279 bytes   2:00 pm  Fri Sep 16 94
slibcaw.lib     297,645 bytes   2:00 pm  Fri Sep 16 94
slibcawq.lib    450,789 bytes   2:00 pm  Fri Sep 16 94
slibce.lib      404,519 bytes   2:00 pm  Fri Sep 16 94
slibcew.lib     319,507 bytes   2:00 pm  Fri Sep 16 94
slibcewq.lib    470,639 bytes   2:00 pm  Fri Sep 16 94
snocrtdw.lib      2,069 bytes   2:00 pm  Fri Sep 16 94
snocrtw.lib       2,581 bytes   2:00 pm  Fri Sep 16 94
storage.lib       2,099 bytes   2:00 pm  Fri Sep 16 94
stress.lib        2,048 bytes   2:00 pm  Fri Sep 16 94
toolhelp.lib      3,584 bytes   2:00 pm  Fri Sep 16 94
txtonly.obj         553 bytes   2:00 pm  Fri Sep 16 94
typelib.lib       2,048 bytes   2:00 pm  Fri Sep 16 94
varstck.obj         170 bytes   2:00 pm  Fri Sep 16 94
ver.lib           2,048 bytes   2:00 pm  Fri Sep 16 94
verc.lib         11,313 bytes   2:00 pm  Fri Sep 16 94
```

```
        verl.lib          11,825 bytes   2:00 pm  Fri Sep 16 94
        verm.lib          11,313 bytes   2:00 pm  Fri Sep 16 94
        vers.lib          11,313 bytes   2:00 pm  Fri Sep 16 94
        win87em.lib        2,089 bytes   2:00 pm  Fri Sep 16 94
        winmem32.lib       2,048 bytes   2:00 pm  Fri Sep 16 94
        winsock.lib        4,096 bytes   2:10 pm  Fri Jan 13 95

I:\VC152\MSVC15\LIB\FONT
        .                    <DIR>       10:47 am  Mon Jun 10 96
        ..                   <DIR>       10:47 am  Mon Jun 10 96
        helvb.fon         50,880 bytes   2:00 pm  Fri Sep 16 94
        modern.fon         7,584 bytes   2:00 pm  Fri Sep 16 94
        roman.fon         11,120 bytes   2:00 pm  Fri Sep 16 94
        script.fon        10,304 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\MFC
        .                    <DIR>       11:00 am  Mon Jun 10 96
        ..                   <DIR>       11:19 am  Mon Jun 10 96
        circ3.vbx         16,832 bytes   2:00 pm  Fri Sep 16 94
        grid.vbx          45,136 bytes   2:00 pm  Fri Sep 16 94
        INCLUDE              <DIR>       10:48 am  Mon Jun 10 96
        LIB                  <DIR>       10:52 am  Mon Jun 10 96
        readme.txt         8,815 bytes   2:00 pm  Fri Sep 16 94
        SAMPLES              <DIR>       11:00 am  Mon Jun 10 96
        SRC                  <DIR>       11:01 am  Mon Jun 10 96
        vc.lic               414 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\MFC\INCLUDE
        .                    <DIR>       10:48 am  Mon Jun 10 96
        ..                   <DIR>       11:00 am  Mon Jun 10 96
        afx.h             39,016 bytes  10:05 pm  Wed Apr 26 95
        afx.inl           13,326 bytes  10:00 pm  Mon Apr 10 95
        afxcoll.h         29,566 bytes  10:01 pm  Mon Apr 10 95
        afxcoll.inl       19,027 bytes  10:01 pm  Mon Apr 10 95
        afxdb.h           22,862 bytes  10:01 pm  Mon Apr 10 95
        afxdb.inl          4,257 bytes  10:01 pm  Mon Apr 10 95
        afxdb.rc           2,885 bytes  10:01 pm  Mon Apr 10 95
        afxdd_.h           3,020 bytes  10:01 pm  Mon Apr 10 95
        afxdisp.h         16,819 bytes  10:01 pm  Mon Apr 10 95
        afxdlgs.h         18,362 bytes   9:56 pm  Thu Apr 20 95
        afxdlgs.inl        4,787 bytes   9:56 pm  Thu Apr 20 95
        afxdllx.h          4,653 bytes  10:01 pm  Mon Apr 10 95
        afxdll_.h          9,735 bytes  10:01 pm  Mon Apr 10 95
        afxext.h          37,662 bytes  10:01 pm  Mon Apr 10 95
        afxext.inl         4,791 bytes  10:01 pm  Mon Apr 10 95
        afxhelp.hm         9,006 bytes  10:01 pm  Mon Apr 10 95
        afxmsg_.h         23,918 bytes  10:01 pm  Mon Apr 10 95
        afxodlgs.h         9,912 bytes  10:01 pm  Mon Apr 10 95
        afxole.h          55,140 bytes  10:01 pm  Mon Apr 10 95
        afxole.inl         7,415 bytes  10:01 pm  Mon Apr 10 95
        afxolecl.rc        2,279 bytes  10:01 pm  Mon Apr 10 95
        afxolesv.rc        1,418 bytes  10:01 pm  Mon Apr 10 95
        afxpen.h           3,173 bytes  10:01 pm  Mon Apr 10 95
        afxpen.inl         2,803 bytes  10:01 pm  Mon Apr 10 95
```

```
        afxprint.rc       2,592 bytes   10:01 pm  Mon Apr 10 95
        afxpriv.h        12,788 bytes   10:01 pm  Mon Apr 10 95
        afxres.h         25,389 bytes   10:01 pm  Mon Apr 10 95
        afxres.rc         5,871 bytes   10:01 pm  Mon Apr 10 95
        afxsock.h         9,244 bytes   10:01 pm  Mon Apr 10 95
        afxsock.inl       3,245 bytes   10:01 pm  Mon Apr 10 95
        afxver_.h         6,729 bytes   10:01 pm  Mon Apr 10 95
        afxv_dll.h        2,856 bytes   10:01 pm  Mon Apr 10 95
        afxv_dos.h        1,874 bytes   10:01 pm  Mon Apr 10 95
        afxwin.h        104,649 bytes    6:01 pm  Mon Apr 24 95
        afxwin1.inl      36,899 bytes   10:01 pm  Mon Apr 10 95
        afxwin2.inl      44,189 bytes   10:01 pm  Mon Apr 10 95
        help.cur            326 bytes   10:01 pm  Mon Apr 10 95
        magnify.cur         326 bytes   10:01 pm  Mon Apr 10 95
        move4way.cur        530 bytes   10:01 pm  Mon Apr 10 95
        nodrop.cur          326 bytes   10:01 pm  Mon Apr 10 95
        psscroll.bmp      1,412 bytes   10:01 pm  Mon Apr 10 95
        sarrows.cur         326 bytes   10:01 pm  Mon Apr 10 95
        splith.cur          326 bytes   10:01 pm  Mon Apr 10 95
        splitv.cur          326 bytes   10:01 pm  Mon Apr 10 95
        trck4way.cur        530 bytes   10:01 pm  Mon Apr 10 95
        trcknesw.cur        530 bytes   10:01 pm  Mon Apr 10 95
        trckns.cur          530 bytes   10:01 pm  Mon Apr 10 95
        trcknwse.cur        530 bytes   10:01 pm  Mon Apr 10 95
        trckwe.cur          530 bytes   10:01 pm  Mon Apr 10 95
        winres.h          9,371 bytes   10:01 pm  Mon Apr 10 95

I:\VC152\MSVC15\MFC\LIB
        .                   <DIR>       10:52 am  Mon Jun 10 96
        ..                  <DIR>       11:00 am  Mon Jun 10 96
        cafxcr.lib      138,159 bytes   11:31 pm  Wed Apr 26 95
        cafxcrd.lib     506,647 bytes   11:30 pm  Wed Apr 26 95
        cafxcw.lib      964,583 bytes   12:02 am  Thu Apr 27 95
        cafxcwd.lib   3,228,781 bytes   12:13 am  Thu Apr 27 95
        lafxcr.lib      142,255 bytes   11:29 pm  Wed Apr 26 95
        lafxcrd.lib     517,911 bytes   11:28 pm  Wed Apr 26 95
        lafxcw.lib    1,062,577 bytes   11:04 pm  Wed Apr 26 95
        lafxcwd.lib   3,392,851 bytes   10:56 pm  Wed Apr 26 95
        lafxdw.lib    1,012,717 bytes   11:46 pm  Wed Apr 26 95
        lafxdwd.lib   3,307,631 bytes   11:37 pm  Wed Apr 26 95
        mafxcr.lib      131,503 bytes   11:27 pm  Wed Apr 26 95
        mafxcrd.lib     493,847 bytes   11:25 pm  Wed Apr 26 95
        mafxcw.lib    1,001,137 bytes   10:50 pm  Wed Apr 26 95
        mafxcwd.lib   3,278,165 bytes   10:42 pm  Wed Apr 26 95
        mfc250.lib      366,080 bytes   11:16 pm  Wed Apr 26 95
        mfc250d.lib     945,152 bytes   11:09 pm  Wed Apr 26 95
        mfcd250.lib      23,552 bytes   11:21 pm  Wed Apr 26 95
        mfcd250d.lib     28,160 bytes   11:20 pm  Wed Apr 26 95
        mfcn250.lib      14,336 bytes   11:22 pm  Wed Apr 26 95
        mfcn250d.lib     16,384 bytes   11:22 pm  Wed Apr 26 95
        mfco250.lib     165,888 bytes   11:20 pm  Wed Apr 26 95
        mfco250d.lib    194,560 bytes   11:17 pm  Wed Apr 26 95
        mfcoleui.lib     24,064 bytes   10:33 pm  Wed Apr 26 95
        safxcr.lib      123,311 bytes   11:25 pm  Wed Apr 26 95
```

```
        safxcrd.lib    481,559 bytes  11:23 pm  Wed Apr 26 95
        safxcw.lib     897,511 bytes  11:54 pm  Wed Apr 26 95
        safxcwd.lib  3,113,581 bytes  12:07 am  Thu Apr 27 95

I:\VC152\MSVC15\MFC\SAMPLES
        .                   <DIR>      11:00 am  Mon Jun 10 96
        ..                  <DIR>      11:00 am  Mon Jun 10 96
        APSTUDIO            <DIR>      10:53 am  Mon Jun 10 96
        AUTOCLIK            <DIR>      10:53 am  Mon Jun 10 96
        BIN                 <DIR>      10:54 am  Mon Jun 10 96
        CALCDRIV            <DIR>      10:54 am  Mon Jun 10 96
        CATALOG             <DIR>      10:55 am  Mon Jun 10 96
        CHKBOOK             <DIR>      10:55 am  Mon Jun 10 96
        CONTAIN             <DIR>      10:55 am  Mon Jun 10 96
        CTRLBARS            <DIR>      10:55 am  Mon Jun 10 96
        CTRLTEST            <DIR>      10:55 am  Mon Jun 10 96
        DIBLOOK             <DIR>      10:55 am  Mon Jun 10 96
        DLLHUSK             <DIR>      10:55 am  Mon Jun 10 96
        DLLTRACE            <DIR>      10:56 am  Mon Jun 10 96
        DRAWCLI             <DIR>      10:56 am  Mon Jun 10 96
        DYNABIND            <DIR>      10:56 am  Mon Jun 10 96
        ENROLL              <DIR>      10:57 am  Mon Jun 10 96
        HELLO               <DIR>      10:57 am  Mon Jun 10 96
        HELLOAPP            <DIR>      10:57 am  Mon Jun 10 96
        HIERSVR             <DIR>      10:57 am  Mon Jun 10 96
        MAKEHM              <DIR>      10:57 am  Mon Jun 10 96
        MDI                 <DIR>      10:57 am  Mon Jun 10 96
        mksample.bat      3,428 bytes  10:11 pm  Mon Apr 10 95
        MULTIPAD            <DIR>      10:57 am  Mon Jun 10 96
        OCLIENT             <DIR>      10:58 am  Mon Jun 10 96
        PROPDLG             <DIR>      10:58 am  Mon Jun 10 96
        sample_.mak       3,927 bytes  10:11 pm  Mon Apr 10 95
        SCRIBBLE            <DIR>      10:59 am  Mon Jun 10 96
        SPEAKN              <DIR>      10:59 am  Mon Jun 10 96
        stdafx.cpp          568 bytes  10:11 pm  Mon Apr 10 95
        stdafx.h            673 bytes  10:11 pm  Mon Apr 10 95
        SUPERPAD            <DIR>      10:59 am  Mon Jun 10 96
        TEMPLDEF            <DIR>      10:59 am  Mon Jun 10 96
        TRACER              <DIR>      10:59 am  Mon Jun 10 96
        TRACKER             <DIR>      11:00 am  Mon Jun 10 96
        VBCHART             <DIR>      11:00 am  Mon Jun 10 96
        VBCIRCLE            <DIR>      11:00 am  Mon Jun 10 96
        VIEWEX              <DIR>      11:00 am  Mon Jun 10 96

I:\VC152\MSVC15\MFC\SAMPLES\APSTUDIO
        .                   <DIR>      10:53 am  Mon Jun 10 96
        ..                  <DIR>      11:00 am  Mon Jun 10 96
        commdlg.rc       11,652 bytes  10:11 pm  Mon Apr 10 95
        common.res      313,788 bytes  10:11 pm  Mon Apr 10 95
        indicate.rc         682 bytes  10:11 pm  Mon Apr 10 95
        prompts.rc        4,332 bytes  10:11 pm  Mon Apr 10 95

I:\VC152\MSVC15\MFC\SAMPLES\AUTOCLIK
        .                   <DIR>      10:53 am  Mon Jun 10 96
```

```
..                  <DIR>       11:00 am  Mon Jun 10 96
autodriv.frm      5,446 bytes    2:10 pm  Fri Jan 13 95
autodriv.mak         76 bytes    2:10 pm  Fri Jan 13 95
STEP1               <DIR>       10:53 am  Mon Jun 10 96
STEP2               <DIR>       10:53 am  Mon Jun 10 96
STEP3               <DIR>       10:53 am  Mon Jun 10 96

I:\VC152\MSVC15\MFC\SAMPLES\AUTOCLIK\STEP1
.                   <DIR>       10:53 am  Mon Jun 10 96
..                  <DIR>       10:53 am  Mon Jun 10 96
autocdoc.cpp      2,157 bytes    2:10 pm  Fri Jan 13 95
autocdoc.h        1,131 bytes    2:10 pm  Fri Jan 13 95
autoclik.clw      2,861 bytes    2:10 pm  Fri Jan 13 95
autoclik.cpp      5,367 bytes    2:10 pm  Fri Jan 13 95
autoclik.def        352 bytes    2:10 pm  Fri Jan 13 95
autoclik.h          890 bytes    2:10 pm  Fri Jan 13 95
autoclik.mak      2,817 bytes    2:10 pm  Fri Jan 13 95
autoclik.rc      10,231 bytes    2:10 pm  Fri Jan 13 95
autoclik.reg      1,060 bytes    2:10 pm  Fri Jan 13 95
autocvw.cpp       2,344 bytes    2:10 pm  Fri Jan 13 95
autocvw.h         1,171 bytes    2:10 pm  Fri Jan 13 95
dialogs.cpp         940 bytes    2:10 pm  Fri Jan 13 95
dialogs.h           650 bytes    2:10 pm  Fri Jan 13 95
mainfrm.cpp       2,477 bytes    2:10 pm  Fri Jan 13 95
mainfrm.h           944 bytes    2:10 pm  Fri Jan 13 95
readme.txt        4,699 bytes    2:10 pm  Fri Jan 13 95
RES                 <DIR>       10:53 am  Mon Jun 10 96
resource.h          711 bytes    2:10 pm  Fri Jan 13 95
stdafx.cpp          204 bytes    2:10 pm  Fri Jan 13 95
stdafx.h            358 bytes    2:10 pm  Fri Jan 13 95

I:\VC152\MSVC15\MFC\SAMPLES\AUTOCLIK\STEP1\RES
.                   <DIR>       10:53 am  Mon Jun 10 96
..                  <DIR>       10:53 am  Mon Jun 10 96
autocdoc.ico        768 bytes    2:10 pm  Fri Jan 13 95
autoclik.ico        768 bytes    2:10 pm  Fri Jan 13 95
autoclik.rc2      1,553 bytes    2:10 pm  Fri Jan 13 95
toolbar.bmp       1,200 bytes    2:10 pm  Fri Jan 13 95

I:\VC152\MSVC15\MFC\SAMPLES\AUTOCLIK\STEP2
.                   <DIR>       10:53 am  Mon Jun 10 96
..                  <DIR>       10:53 am  Mon Jun 10 96
autocdoc.cpp      3,130 bytes    2:10 pm  Fri Jan 13 95
autocdoc.h        1,216 bytes    2:10 pm  Fri Jan 13 95
autoclik.clw      2,861 bytes    2:10 pm  Fri Jan 13 95
autoclik.cpp      5,367 bytes    2:10 pm  Fri Jan 13 95
autoclik.def        352 bytes    2:10 pm  Fri Jan 13 95
autoclik.h          890 bytes    2:10 pm  Fri Jan 13 95
autoclik.mak      2,817 bytes    2:10 pm  Fri Jan 13 95
autoclik.rc      10,231 bytes    2:10 pm  Fri Jan 13 95
autoclik.reg      1,060 bytes    2:10 pm  Fri Jan 13 95
autocvw.cpp       2,344 bytes    2:10 pm  Fri Jan 13 95
autocvw.h         1,171 bytes    2:10 pm  Fri Jan 13 95
dialogs.cpp         940 bytes    2:10 pm  Fri Jan 13 95
```

```
        dialogs.h             650 bytes   2:10 pm  Fri Jan 13 95
        mainfrm.cpp         2,477 bytes   2:10 pm  Fri Jan 13 95
        mainfrm.h             944 bytes   2:10 pm  Fri Jan 13 95
        readme.txt          4,699 bytes   2:10 pm  Fri Jan 13 95
        RES                   <DIR>      10:53 am  Mon Jun 10 96
        resource.h            711 bytes   2:10 pm  Fri Jan 13 95
        stdafx.cpp            204 bytes   2:10 pm  Fri Jan 13 95
        stdafx.h              358 bytes   2:10 pm  Fri Jan 13 95

I:\VC152\MSVC15\MFC\SAMPLES\AUTOCLIK\STEP2\RES
        .                     <DIR>      10:53 am  Mon Jun 10 96
        ..                    <DIR>      10:53 am  Mon Jun 10 96
        autocdoc.ico          768 bytes   2:10 pm  Fri Jan 13 95
        autoclik.ico          768 bytes   2:10 pm  Fri Jan 13 95
        autoclik.rc2        1,553 bytes   2:10 pm  Fri Jan 13 95
        toolbar.bmp         1,200 bytes   2:10 pm  Fri Jan 13 95

I:\VC152\MSVC15\MFC\SAMPLES\AUTOCLIK\STEP3
        .                     <DIR>      10:53 am  Mon Jun 10 96
        ..                    <DIR>      10:53 am  Mon Jun 10 96
        autocdoc.cpp        3,691 bytes   2:10 pm  Fri Jan 13 95
        autocdoc.h          1,301 bytes   2:10 pm  Fri Jan 13 95
        autoclik.clw        2,977 bytes   2:10 pm  Fri Jan 13 95
        autoclik.cpp        5,367 bytes   2:10 pm  Fri Jan 13 95
        autoclik.def          352 bytes   2:10 pm  Fri Jan 13 95
        autoclik.h            890 bytes   2:10 pm  Fri Jan 13 95
        autoclik.mak        2,966 bytes   2:10 pm  Fri Jan 13 95
        autoclik.rc        10,226 bytes   2:10 pm  Fri Jan 13 95
        autoclik.reg        1,060 bytes   2:10 pm  Fri Jan 13 95
        autocpnt.cpp        1,540 bytes   2:10 pm  Fri Jan 13 95
        autocpnt.h            814 bytes   2:10 pm  Fri Jan 13 95
        autocvw.cpp         2,344 bytes   2:10 pm  Fri Jan 13 95
        autocvw.h           1,171 bytes   2:10 pm  Fri Jan 13 95
        dialogs.cpp           940 bytes   2:10 pm  Fri Jan 13 95
        dialogs.h             650 bytes   2:10 pm  Fri Jan 13 95
        mainfrm.cpp         2,477 bytes   2:10 pm  Fri Jan 13 95
        mainfrm.h             944 bytes   2:10 pm  Fri Jan 13 95
        readme.txt          4,699 bytes   2:10 pm  Fri Jan 13 95
        RES                   <DIR>      10:53 am  Mon Jun 10 96
        resource.h            711 bytes   2:10 pm  Fri Jan 13 95
        stdafx.cpp            204 bytes   2:10 pm  Fri Jan 13 95
        stdafx.h              358 bytes   2:10 pm  Fri Jan 13 95

I:\VC152\MSVC15\MFC\SAMPLES\AUTOCLIK\STEP3\RES
        .                     <DIR>      10:53 am  Mon Jun 10 96
        ..                    <DIR>      10:53 am  Mon Jun 10 96
        autocdoc.ico          768 bytes   2:10 pm  Fri Jan 13 95
        autoclik.ico          768 bytes   2:10 pm  Fri Jan 13 95
        autoclik.rc2        1,553 bytes   2:10 pm  Fri Jan 13 95
        toolbar.bmp         1,200 bytes   2:10 pm  Fri Jan 13 95

I:\VC152\MSVC15\MFC\SAMPLES\BIN
        .                     <DIR>      10:54 am  Mon Jun 10 96
        ..                    <DIR>      11:00 am  Mon Jun 10 96
```

```
        autoclik.exe   139,040 bytes   2:10 pm  Fri Jan 13 95
        calcdriv.exe    75,856 bytes   2:10 pm  Fri Jan 13 95
        catalog.exe    124,400 bytes   9:46 pm  Wed Mar  1 95
        chkbook.exe    127,856 bytes   2:10 pm  Fri Jan 13 95
        contain.exe    181,152 bytes   2:10 pm  Fri Jan 13 95
        ctrlbars.exe    70,976 bytes   2:10 pm  Fri Jan 13 95
        ctrltest.exe    86,688 bytes   2:10 pm  Fri Jan 13 95
        diblook.exe    115,072 bytes   2:10 pm  Fri Jan 13 95
        dllhusk.exe     12,240 bytes   2:10 pm  Fri Jan 13 95
        drawcli.exe    203,488 bytes   2:10 pm  Fri Jan 13 95
        dynabind.exe   158,160 bytes   9:47 pm  Wed Mar  1 95
        enroll.exe     156,768 bytes   9:49 pm  Wed Mar  1 95
        hello.exe       38,608 bytes   2:10 pm  Fri Jan 13 95
        hello1.exe   1,309,188 bytes   2:10 pm  Fri Jan 13 95
        helloapp.exe    34,425 bytes   2:10 pm  Fri Jan 13 95
        hiersvr.exe    207,696 bytes   2:10 pm  Fri Jan 13 95
        hiersvr.hlp     38,628 bytes   2:10 pm  Fri Jan 13 95
        mdi.exe         57,968 bytes   2:10 pm  Fri Jan 13 95
        multipad.exe    95,056 bytes   2:10 pm  Fri Jan 13 95
        muscroll.dll    11,776 bytes   2:10 pm  Fri Jan 13 95
        oclient.exe    197,088 bytes   2:10 pm  Fri Jan 13 95
        oclient.hlp     39,533 bytes   2:10 pm  Fri Jan 13 95
        propdlg.exe    129,408 bytes   2:10 pm  Fri Jan 13 95
        scribble.exe   215,152 bytes   2:10 pm  Fri Jan 13 95
        scribble.hlp    38,493 bytes   2:10 pm  Fri Jan 13 95
        speakn.exe     226,448 bytes   2:10 pm  Fri Jan 13 95
        superpad.exe   233,312 bytes   2:10 pm  Fri Jan 13 95
        testdll1.dll    11,776 bytes   2:10 pm  Fri Jan 13 95
        testdll2.dll     9,216 bytes   2:10 pm  Fri Jan 13 95
        tracer.dll   1,345,844 bytes   2:10 pm  Fri Jan 13 95
        tracer.exe      34,464 bytes   2:10 pm  Fri Jan 13 95
        tracker.exe    116,224 bytes   2:10 pm  Fri Jan 13 95
        vbchart.exe    157,536 bytes   2:10 pm  Fri Jan 13 95
        vbcircle.exe    87,312 bytes   2:10 pm  Fri Jan 13 95
        viewex.exe     111,232 bytes   2:10 pm  Fri Jan 13 95

I:\VC152\MSVC15\MFC\SAMPLES\CALCDRIV
        .                 <DIR>       10:54 am  Mon Jun 10 96
        ..                <DIR>       11:00 am  Mon Jun 10 96
        calcdriv.clw    1,108 bytes   10:11 pm  Mon Apr 10 95
        calcdriv.cpp    5,420 bytes   10:11 pm  Mon Apr 10 95
        calcdriv.def      783 bytes   10:11 pm  Mon Apr 10 95
        calcdriv.h      2,248 bytes   10:11 pm  Mon Apr 10 95
        calcdriv.mak    2,403 bytes   10:11 pm  Mon Apr 10 95
        calcdriv.rc     3,072 bytes   10:11 pm  Mon Apr 10 95
        calctype.cpp    2,399 bytes   10:11 pm  Mon Apr 10 95
        calctype.h        962 bytes   10:11 pm  Mon Apr 10 95
        makefile          517 bytes   10:11 pm  Mon Apr 10 95
        RES               <DIR>       10:54 am  Mon Jun 10 96
        resource.h        842 bytes   10:11 pm  Mon Apr 10 95
        stdafx.cpp        644 bytes   10:11 pm  Mon Apr 10 95
        stdafx.h          798 bytes   10:11 pm  Mon Apr 10 95

I:\VC152\MSVC15\MFC\SAMPLES\CALCDRIV\RES
```

```
        .                   <DIR>       10:54 am  Mon Jun 10 96
        ..                  <DIR>       10:54 am  Mon Jun 10 96
        calcdriv.ico          766 bytes  10:11 pm  Mon Apr 10 95
        calcdriv.rc2        1,553 bytes  10:11 pm  Mon Apr 10 95

I:\VC152\MSVC15\MFC\SAMPLES\CATALOG
        .                   <DIR>       10:55 am  Mon Jun 10 96
        ..                  <DIR>       11:00 am  Mon Jun 10 96
        cataldoc.cpp        1,951 bytes  10:11 pm  Mon Apr 10 95
        cataldoc.h          1,336 bytes  10:11 pm  Mon Apr 10 95
        catalog.clw         2,443 bytes  10:11 pm  Mon Apr 10 95
        catalog.cpp         3,706 bytes  10:11 pm  Mon Apr 10 95
        catalog.def           780 bytes  10:11 pm  Mon Apr 10 95
        catalog.h           1,266 bytes  10:11 pm  Mon Apr 10 95
        catalog.mak         2,916 bytes  10:11 pm  Mon Apr 10 95
        catalog.rc          7,784 bytes  10:11 pm  Mon Apr 10 95
        catalvw.cpp         3,903 bytes  10:11 pm  Mon Apr 10 95
        catalvw.h           1,825 bytes  10:11 pm  Mon Apr 10 95
        columnst.cpp        3,966 bytes  10:11 pm  Mon Apr 10 95
        columnst.h          1,595 bytes  10:11 pm  Mon Apr 10 95
        mainfrm.cpp         2,946 bytes  10:11 pm  Mon Apr 10 95
        mainfrm.h           1,420 bytes  10:11 pm  Mon Apr 10 95
        makefile              565 bytes  10:11 pm  Mon Apr 10 95
        RES                 <DIR>       10:55 am  Mon Jun 10 96
        resource.h            799 bytes  10:11 pm  Mon Apr 10 95
        stdafx.cpp            644 bytes  10:11 pm  Mon Apr 10 95
        stdafx.h              792 bytes  10:11 pm  Mon Apr 10 95
        tableset.cpp        3,568 bytes  10:11 pm  Mon Apr 10 95
        tableset.h          1,416 bytes  10:11 pm  Mon Apr 10 95

I:\VC152\MSVC15\MFC\SAMPLES\CATALOG\RES
        .                   <DIR>       10:55 am  Mon Jun 10 96
        ..                  <DIR>       10:55 am  Mon Jun 10 96
        catalog.ico           766 bytes  10:11 pm  Mon Apr 10 95
        catalog.rc2         1,548 bytes  10:11 pm  Mon Apr 10 95
        toolbar.bmp         1,328 bytes  10:11 pm  Mon Apr 10 95

I:\VC152\MSVC15\MFC\SAMPLES\CHKBOOK
        .                   <DIR>       10:55 am  Mon Jun 10 96
        ..                  <DIR>       11:00 am  Mon Jun 10 96
        bookvw.cpp          4,066 bytes  10:11 pm  Mon Apr 10 95
        bookvw.h            1,895 bytes  10:11 pm  Mon Apr 10 95
        checkdoc.cpp        7,786 bytes  10:11 pm  Mon Apr 10 95
        checkdoc.h          2,854 bytes  10:11 pm  Mon Apr 10 95
        checkvw.cpp         5,089 bytes  10:11 pm  Mon Apr 10 95
        checkvw.h           1,965 bytes  10:11 pm  Mon Apr 10 95
        chkbkfrm.cpp          938 bytes  10:11 pm  Mon Apr 10 95
        chkbkfrm.h          1,263 bytes  10:11 pm  Mon Apr 10 95
        chkbook.clw         3,337 bytes  10:11 pm  Mon Apr 10 95
        chkbook.cpp         7,245 bytes  10:11 pm  Mon Apr 10 95
        chkbook.def           694 bytes  10:11 pm  Mon Apr 10 95
        chkbook.h           1,398 bytes  10:11 pm  Mon Apr 10 95
        chkbook.mak         3,270 bytes  10:11 pm  Mon Apr 10 95
        chkbook.rc         11,814 bytes  10:11 pm  Mon Apr 10 95
```

```
        dollcent.cpp      5,955 bytes   10:11 pm  Mon Apr 10 95
        dollcent.h          935 bytes   10:11 pm  Mon Apr 10 95
        fxrecdoc.cpp      6,040 bytes   10:11 pm  Mon Apr 10 95
        fxrecdoc.h        2,859 bytes   10:11 pm  Mon Apr 10 95
        mainfrm.cpp       3,684 bytes   10:11 pm  Mon Apr 10 95
        mainfrm.h         1,435 bytes   10:11 pm  Mon Apr 10 95
        makefile            584 bytes   10:11 pm  Mon Apr 10 95
        RES                 <DIR>       10:55 am  Mon Jun 10 96
        resource.h        1,473 bytes   10:11 pm  Mon Apr 10 95
        rowview.cpp      10,514 bytes   10:11 pm  Mon Apr 10 95
        rowview.h         2,971 bytes   10:11 pm  Mon Apr 10 95
        stdafx.cpp          643 bytes   10:11 pm  Mon Apr 10 95
        stdafx.h            693 bytes   10:11 pm  Mon Apr 10 95

I:\VC152\MSVC15\MFC\SAMPLES\CHKBOOK\RES
        .                   <DIR>       10:55 am  Mon Jun 10 96
        ..                  <DIR>       10:55 am  Mon Jun 10 96
        checkvw.ico         766 bytes   10:11 pm  Mon Apr 10 95
        chkbook.ico         766 bytes   10:11 pm  Mon Apr 10 95
        doc.ico             766 bytes   10:11 pm  Mon Apr 10 95
        toolbar.bmp         958 bytes   10:11 pm  Mon Apr 10 95

I:\VC152\MSVC15\MFC\SAMPLES\CONTAIN
        .                   <DIR>       10:55 am  Mon Jun 10 96
        ..                  <DIR>       11:00 am  Mon Jun 10 96
        STEP1               <DIR>       10:55 am  Mon Jun 10 96
        STEP2               <DIR>       10:55 am  Mon Jun 10 96

I:\VC152\MSVC15\MFC\SAMPLES\CONTAIN\STEP1
        .                   <DIR>       10:55 am  Mon Jun 10 96
        ..                  <DIR>       10:55 am  Mon Jun 10 96
        cntritem.cpp      3,216 bytes    2:10 pm  Fri Jan 13 95
        cntritem.h        1,268 bytes    2:10 pm  Fri Jan 13 95
        contain.clw       3,354 bytes    2:10 pm  Fri Jan 13 95
        contain.cpp       4,147 bytes    2:10 pm  Fri Jan 13 95
        contain.def         349 bytes    2:10 pm  Fri Jan 13 95
        contain.h           826 bytes    2:10 pm  Fri Jan 13 95
        contain.mak       2,796 bytes    2:10 pm  Fri Jan 13 95
        contain.rc       11,964 bytes    2:10 pm  Fri Jan 13 95
        contain.reg         714 bytes    2:10 pm  Fri Jan 13 95
        contrdoc.cpp      2,584 bytes    2:10 pm  Fri Jan 13 95
        contrdoc.h          952 bytes    2:10 pm  Fri Jan 13 95
        contrvw.cpp       8,650 bytes    2:10 pm  Fri Jan 13 95
        contrvw.h         2,327 bytes    2:10 pm  Fri Jan 13 95
        mainfrm.cpp       2,478 bytes    2:10 pm  Fri Jan 13 95
        mainfrm.h           946 bytes    2:10 pm  Fri Jan 13 95
        readme.txt        5,053 bytes    2:10 pm  Fri Jan 13 95
        RES                 <DIR>       10:55 am  Mon Jun 10 96
        resource.h          552 bytes    2:10 pm  Fri Jan 13 95
        stdafx.cpp          204 bytes    2:10 pm  Fri Jan 13 95
        stdafx.h            402 bytes    2:10 pm  Fri Jan 13 95

I:\VC152\MSVC15\MFC\SAMPLES\CONTAIN\STEP1\RES
        .                   <DIR>       10:55 am  Mon Jun 10 96
```

```
..                  <DIR>       10:55 am  Mon Jun 10 96
contain.ico           768 bytes   2:10 pm  Fri Jan 13 95
contain.rc2         1,548 bytes   2:10 pm  Fri Jan 13 95
contrdoc.ico          768 bytes   2:10 pm  Fri Jan 13 95
toolbar.bmp         1,200 bytes   2:10 pm  Fri Jan 13 95

I:\VC152\MSVC15\MFC\SAMPLES\CONTAIN\STEP2
.                   <DIR>       10:55 am  Mon Jun 10 96
..                  <DIR>       10:55 am  Mon Jun 10 96
cntritem.cpp        3,864 bytes   2:10 pm  Fri Jan 13 95
cntritem.h          1,341 bytes   2:10 pm  Fri Jan 13 95
contain.clw         3,354 bytes   2:10 pm  Fri Jan 13 95
contain.cpp         4,147 bytes   2:10 pm  Fri Jan 13 95
contain.def           349 bytes   2:10 pm  Fri Jan 13 95
contain.h             826 bytes   2:10 pm  Fri Jan 13 95
contain.mak         2,796 bytes   2:10 pm  Fri Jan 13 95
contain.rc         11,964 bytes   2:10 pm  Fri Jan 13 95
contain.reg           714 bytes   2:10 pm  Fri Jan 13 95
contrdoc.cpp        2,584 bytes   2:10 pm  Fri Jan 13 95
contrdoc.h          1,014 bytes   2:10 pm  Fri Jan 13 95
contrvw.cpp        10,436 bytes   2:10 pm  Fri Jan 13 95
contrvw.h           2,506 bytes   2:10 pm  Fri Jan 13 95
mainfrm.cpp         2,478 bytes   2:10 pm  Fri Jan 13 95
mainfrm.h             946 bytes   2:10 pm  Fri Jan 13 95
readme.txt          5,053 bytes   2:10 pm  Fri Jan 13 95
RES                 <DIR>       10:55 am  Mon Jun 10 96
resource.h            552 bytes   2:10 pm  Fri Jan 13 95
stdafx.cpp            204 bytes   2:10 pm  Fri Jan 13 95
stdafx.h              402 bytes   2:10 pm  Fri Jan 13 95

I:\VC152\MSVC15\MFC\SAMPLES\CONTAIN\STEP2\RES
.                   <DIR>       10:55 am  Mon Jun 10 96
..                  <DIR>       10:55 am  Mon Jun 10 96
contain.ico           768 bytes   2:10 pm  Fri Jan 13 95
contain.rc2         1,548 bytes   2:10 pm  Fri Jan 13 95
contrdoc.ico          768 bytes   2:10 pm  Fri Jan 13 95
toolbar.bmp         1,200 bytes   2:10 pm  Fri Jan 13 95

I:\VC152\MSVC15\MFC\SAMPLES\CTRLBARS
.                   <DIR>       10:55 am  Mon Jun 10 96
..                  <DIR>       11:00 am  Mon Jun 10 96
ctrlbars.clw        1,727 bytes  10:11 pm  Mon Apr 10 95
ctrlbars.cpp        3,167 bytes  10:11 pm  Mon Apr 10 95
ctrlbars.def          762 bytes  10:11 pm  Mon Apr 10 95
ctrlbars.h          1,178 bytes  10:11 pm  Mon Apr 10 95
ctrlbars.mak        2,370 bytes  10:11 pm  Mon Apr 10 95
ctrlbars.rc         8,433 bytes  10:11 pm  Mon Apr 10 95
mainfrm.cpp        13,130 bytes  10:11 pm  Mon Apr 10 95
mainfrm.h           2,768 bytes  10:11 pm  Mon Apr 10 95
makefile              561 bytes  10:11 pm  Mon Apr 10 95
palette.cpp         6,657 bytes  10:11 pm  Mon Apr 10 95
palette.h           1,896 bytes  10:11 pm  Mon Apr 10 95
RES                 <DIR>       10:55 am  Mon Jun 10 96
resource.h          1,899 bytes  10:11 pm  Mon Apr 10 95
```

```
        stdafx.cpp          639 bytes  10:11 pm  Mon Apr 10 95
        stdafx.h            719 bytes  10:11 pm  Mon Apr 10 95

I:\VC152\MSVC15\MFC\SAMPLES\CTRLBARS\RES
        .                   <DIR>      10:55 am  Mon Jun 10 96
        ..                  <DIR>      10:55 am  Mon Jun 10 96
        ctrlbars.ico        766 bytes  10:11 pm  Mon Apr 10 95
        palette.bmp       1,558 bytes  10:11 pm  Mon Apr 10 95
        styles.bmp          838 bytes  10:11 pm  Mon Apr 10 95
        toolbar.bmp       1,438 bytes  10:11 pm  Mon Apr 10 95

I:\VC152\MSVC15\MFC\SAMPLES\CTRLTEST
        .                   <DIR>      10:55 am  Mon Jun 10 96
        ..                  <DIR>      11:00 am  Mon Jun 10 96
        bbutton.cpp       5,581 bytes  10:11 pm  Mon Apr 10 95
        ctrltest.clw      5,108 bytes  10:11 pm  Mon Apr 10 95
        ctrltest.cpp      3,375 bytes  10:11 pm  Mon Apr 10 95
        ctrltest.def        595 bytes  10:11 pm  Mon Apr 10 95
        ctrltest.h        2,355 bytes  10:11 pm  Mon Apr 10 95
        ctrltest.mak      3,896 bytes  10:11 pm  Mon Apr 10 95
        ctrltest.rc       8,344 bytes  10:11 pm  Mon Apr 10 95
        custlist.cpp      4,890 bytes  10:11 pm  Mon Apr 10 95
        custmenu.cpp      4,866 bytes  10:11 pm  Mon Apr 10 95
        derpen.cpp        4,742 bytes  10:11 pm  Mon Apr 10 95
        dertest.cpp       4,697 bytes  10:11 pm  Mon Apr 10 95
        dlgpen.cpp        2,888 bytes  10:11 pm  Mon Apr 10 95
        featpen.cpp       5,087 bytes  10:11 pm  Mon Apr 10 95
        makefile            659 bytes  10:11 pm  Mon Apr 10 95
        muscroll.dll     11,776 bytes  10:11 pm  Mon Apr 10 95
        muscroll.h        2,185 bytes  10:11 pm  Mon Apr 10 95
        paredit.cpp       3,873 bytes  10:11 pm  Mon Apr 10 95
        paredit.h         2,088 bytes  10:11 pm  Mon Apr 10 95
        paredit2.cpp      4,354 bytes  10:11 pm  Mon Apr 10 95
        RES                 <DIR>      10:55 am  Mon Jun 10 96
        resource.h        1,489 bytes  10:11 pm  Mon Apr 10 95
        spin.cpp            883 bytes  10:11 pm  Mon Apr 10 95
        spin.h            2,049 bytes  10:11 pm  Mon Apr 10 95
        spintest.cpp      3,030 bytes  10:11 pm  Mon Apr 10 95
        stdafx.cpp          495 bytes  10:11 pm  Mon Apr 10 95
        stdafx.h            483 bytes  10:11 pm  Mon Apr 10 95
        subtest.cpp       2,152 bytes  10:11 pm  Mon Apr 10 95
        wclstest.cpp      2,863 bytes  10:11 pm  Mon Apr 10 95

I:\VC152\MSVC15\MFC\SAMPLES\CTRLTEST\RES
        .                   <DIR>      10:55 am  Mon Jun 10 96
        ..                  <DIR>      10:55 am  Mon Jun 10 96
        canceld.bmp       1,366 bytes  10:11 pm  Mon Apr 10 95
        cancelf.bmp       1,366 bytes  10:11 pm  Mon Apr 10 95
        cancelu.bmp       1,366 bytes  10:11 pm  Mon Apr 10 95
        ctrltest.ico        766 bytes  10:11 pm  Mon Apr 10 95
        ctrltest.rc2      4,353 bytes  10:11 pm  Mon Apr 10 95
        imageld.bmp       1,398 bytes  10:11 pm  Mon Apr 10 95
        imagelf.bmp       1,398 bytes  10:11 pm  Mon Apr 10 95
        imagelu.bmp       1,398 bytes  10:11 pm  Mon Apr 10 95
```

```
        image2d.bmp       2,038 bytes   10:11 pm  Mon Apr 10 95
        image2f.bmp       2,038 bytes   10:11 pm  Mon Apr 10 95
        image2u.bmp       2,038 bytes   10:11 pm  Mon Apr 10 95
        nextd.bmp           678 bytes   10:11 pm  Mon Apr 10 95
        nextf.bmp           678 bytes   10:11 pm  Mon Apr 10 95
        nextu.bmp           678 bytes   10:11 pm  Mon Apr 10 95
        nextx.bmp           678 bytes   10:11 pm  Mon Apr 10 95
        okd.bmp           1,366 bytes   10:11 pm  Mon Apr 10 95
        okf.bmp           1,366 bytes   10:11 pm  Mon Apr 10 95
        oku.bmp           1,366 bytes   10:11 pm  Mon Apr 10 95
        otherids.h          579 bytes   10:11 pm  Mon Apr 10 95
        prevd.bmp           678 bytes   10:11 pm  Mon Apr 10 95
        prevf.bmp           678 bytes   10:11 pm  Mon Apr 10 95
        prevu.bmp           678 bytes   10:11 pm  Mon Apr 10 95
        prevx.bmp           678 bytes   10:11 pm  Mon Apr 10 95

I:\VC152\MSVC15\MFC\SAMPLES\DIBLOOK
        .                   <DIR>       10:55 am  Mon Jun 10 96
        ..                  <DIR>       11:00 am  Mon Jun 10 96
        dibapi.cpp       14,390 bytes   10:11 pm  Mon Apr 10 95
        dibapi.h          1,671 bytes   10:11 pm  Mon Apr 10 95
        dibdoc.cpp        4,848 bytes   10:11 pm  Mon Apr 10 95
        dibdoc.h          1,481 bytes   10:11 pm  Mon Apr 10 95
        diblook.clw       2,326 bytes   10:11 pm  Mon Apr 10 95
        diblook.cpp       4,180 bytes   10:11 pm  Mon Apr 10 95
        diblook.def         775 bytes   10:11 pm  Mon Apr 10 95
        diblook.h         1,161 bytes   10:11 pm  Mon Apr 10 95
        diblook.mak       2,749 bytes   10:11 pm  Mon Apr 10 95
        diblook.rc        9,172 bytes   10:11 pm  Mon Apr 10 95
        dibview.cpp       6,048 bytes   10:11 pm  Mon Apr 10 95
        dibview.h         1,572 bytes   10:11 pm  Mon Apr 10 95
        mainfrm.cpp       3,364 bytes   10:11 pm  Mon Apr 10 95
        mainfrm.h         1,189 bytes   10:11 pm  Mon Apr 10 95
        makefile            542 bytes   10:11 pm  Mon Apr 10 95
        myfile.cpp        5,842 bytes   10:11 pm  Mon Apr 10 95
        RES                 <DIR>       10:55 am  Mon Jun 10 96
        resource.h          526 bytes   10:11 pm  Mon Apr 10 95
        stdafx.cpp          643 bytes   10:11 pm  Mon Apr 10 95
        stdafx.h            672 bytes   10:11 pm  Mon Apr 10 95

I:\VC152\MSVC15\MFC\SAMPLES\DIBLOOK\RES
        .                   <DIR>       10:55 am  Mon Jun 10 96
        ..                  <DIR>       10:55 am  Mon Jun 10 96
        dibdoc.ico          766 bytes   10:11 pm  Mon Apr 10 95
        diblook.ico         766 bytes   10:11 pm  Mon Apr 10 95
        toolbar.bmp       1,200 bytes   10:11 pm  Mon Apr 10 95

I:\VC152\MSVC15\MFC\SAMPLES\DLLHUSK
        .                   <DIR>       10:55 am  Mon Jun 10 96
        ..                  <DIR>       11:00 am  Mon Jun 10 96
        dllhusk.clw       1,604 bytes   10:11 pm  Mon Apr 10 95
        dllhusk.cpp       6,853 bytes   10:11 pm  Mon Apr 10 95
        dllhusk.def         775 bytes   10:11 pm  Mon Apr 10 95
        dllhusk.h         1,426 bytes   10:11 pm  Mon Apr 10 95
```

```
        dllhusk.rc        4,409 bytes   10:11 pm  Mon Apr 10 95
        mainfrm.cpp       3,177 bytes   10:11 pm  Mon Apr 10 95
        mainfrm.h         1,405 bytes   10:11 pm  Mon Apr 10 95
        makefile          2,308 bytes   10:11 pm  Mon Apr 10 95
        RES                 <DIR>       10:55 am  Mon Jun 10 96
        resource.h          802 bytes   10:11 pm  Mon Apr 10 95
        stdafx.cpp          643 bytes   10:11 pm  Mon Apr 10 95
        stdafx.h            734 bytes   10:11 pm  Mon Apr 10 95
        testdll1.clw      1,850 bytes   10:11 pm  Mon Apr 10 95
        testdll1.cpp      4,930 bytes   10:11 pm  Mon Apr 10 95
        testdll1.def        940 bytes   10:11 pm  Mon Apr 10 95
        testdll1.h        1,856 bytes   10:11 pm  Mon Apr 10 95
        testdll1.rc       4,895 bytes   10:11 pm  Mon Apr 10 95
        testdll2.clw        659 bytes   10:11 pm  Mon Apr 10 95
        testdll2.cpp      5,950 bytes   10:11 pm  Mon Apr 10 95
        testdll2.def      1,443 bytes   10:11 pm  Mon Apr 10 95
        testdll2.h        2,222 bytes   10:11 pm  Mon Apr 10 95
        testdll2.rc       2,278 bytes   10:11 pm  Mon Apr 10 95
        testres1.h          686 bytes   10:11 pm  Mon Apr 10 95
        testres2.h          490 bytes   10:11 pm  Mon Apr 10 95

I:\VC152\MSVC15\MFC\SAMPLES\DLLHUSK\RES
        .                   <DIR>       10:55 am  Mon Jun 10 96
        ..                  <DIR>       10:55 am  Mon Jun 10 96
        dllhusk.ico         766 bytes   10:11 pm  Mon Apr 10 95
        hello.ico           766 bytes   10:11 pm  Mon Apr 10 95
        listout.ico         766 bytes   10:11 pm  Mon Apr 10 95
        texttype.ico        766 bytes   10:11 pm  Mon Apr 10 95
        toolbar.bmp       1,200 bytes   10:11 pm  Mon Apr 10 95
        vbcircle.ico        766 bytes   10:11 pm  Mon Apr 10 95

I:\VC152\MSVC15\MFC\SAMPLES\DLLTRACE
        .                   <DIR>       10:56 am  Mon Jun 10 96
        ..                  <DIR>       11:00 am  Mon Jun 10 96
        hello1.clw          844 bytes   10:11 pm  Mon Apr 10 95
        hello1.cpp        3,263 bytes   10:11 pm  Mon Apr 10 95
        hello1.def          722 bytes   10:11 pm  Mon Apr 10 95
        hello1.ico          766 bytes   10:11 pm  Mon Apr 10 95
        hello1.mak        1,961 bytes   10:11 pm  Mon Apr 10 95
        hello1.rc         2,261 bytes   10:11 pm  Mon Apr 10 95
        makefile          1,955 bytes   10:11 pm  Mon Apr 10 95
        resource.h          406 bytes   10:11 pm  Mon Apr 10 95
        traceapi.h          894 bytes   10:11 pm  Mon Apr 10 95
        tracer.clw          767 bytes   10:11 pm  Mon Apr 10 95
        tracer.cpp        4,194 bytes   10:11 pm  Mon Apr 10 95
        tracer.def          809 bytes   10:11 pm  Mon Apr 10 95
        tracer.mak        2,018 bytes   10:11 pm  Mon Apr 10 95
        tracer.rc         2,070 bytes   10:11 pm  Mon Apr 10 95
        traceres.h          586 bytes   10:11 pm  Mon Apr 10 95

I:\VC152\MSVC15\MFC\SAMPLES\DRAWCLI
        .                   <DIR>       10:56 am  Mon Jun 10 96
        ..                  <DIR>       11:00 am  Mon Jun 10 96
        cntritem.cpp      4,305 bytes   10:11 pm  Mon Apr 10 95
```

```
        cntritem.h       1,726 bytes   10:11 pm  Mon Apr 10 95
        drawcli.clw      4,915 bytes   10:11 pm  Mon Apr 10 95
        drawcli.cpp      4,659 bytes   10:11 pm  Mon Apr 10 95
        drawcli.def        780 bytes   10:11 pm  Mon Apr 10 95
        drawcli.h        1,262 bytes   10:11 pm  Mon Apr 10 95
        drawcli.mak      3,354 bytes   10:11 pm  Mon Apr 10 95
        drawcli.rc      16,979 bytes   10:11 pm  Mon Apr 10 95
        drawdoc.cpp      5,162 bytes   10:11 pm  Mon Apr 10 95
        drawdoc.h        1,818 bytes   10:11 pm  Mon Apr 10 95
        drawobj.cpp     22,081 bytes   10:11 pm  Mon Apr 10 95
        drawobj.h        4,589 bytes   10:11 pm  Mon Apr 10 95
        drawtool.cpp    10,850 bytes   10:11 pm  Mon Apr 10 95
        drawtool.h       2,850 bytes   10:11 pm  Mon Apr 10 95
        drawvw.cpp      27,050 bytes   10:11 pm  Mon Apr 10 95
        drawvw.h         4,876 bytes   10:11 pm  Mon Apr 10 95
        mainfrm.cpp      3,044 bytes   10:11 pm  Mon Apr 10 95
        mainfrm.h        1,306 bytes   10:11 pm  Mon Apr 10 95
        makefile           599 bytes   10:11 pm  Mon Apr 10 95
        rectdlg.cpp      1,459 bytes   10:11 pm  Mon Apr 10 95
        rectdlg.h        1,178 bytes   10:11 pm  Mon Apr 10 95
        RES                <DIR>       10:56 am  Mon Jun 10 96
        resource.h       1,683 bytes   10:11 pm  Mon Apr 10 95
        splitfrm.cpp     1,421 bytes   10:11 pm  Mon Apr 10 95
        splitfrm.h       1,302 bytes   10:11 pm  Mon Apr 10 95
        stdafx.cpp         644 bytes   10:11 pm  Mon Apr 10 95
        stdafx.h           850 bytes   10:11 pm  Mon Apr 10 95

I:\VC152\MSVC15\MFC\SAMPLES\DRAWCLI\RES
        .                  <DIR>       10:56 am  Mon Jun 10 96
        ..                 <DIR>       10:56 am  Mon Jun 10 96
        drawcli.ico        766 bytes   10:11 pm  Mon Apr 10 95
        drawcli.rc2      1,548 bytes   10:11 pm  Mon Apr 10 95
        drawdoc.ico        768 bytes   10:11 pm  Mon Apr 10 95
        pencil.cur         326 bytes   10:11 pm  Mon Apr 10 95
        toolbar.bmp      2,158 bytes   10:11 pm  Mon Apr 10 95

I:\VC152\MSVC15\MFC\SAMPLES\DYNABIND
        .                  <DIR>       10:56 am  Mon Jun 10 96
        ..                 <DIR>       11:00 am  Mon Jun 10 96
        addfield.cpp     2,518 bytes   10:12 pm  Mon Apr 10 95
        addfield.h       1,108 bytes   10:12 pm  Mon Apr 10 95
        columnst.cpp     3,966 bytes   10:12 pm  Mon Apr 10 95
        columnst.h       1,595 bytes   10:12 pm  Mon Apr 10 95
        coursset.cpp       984 bytes   10:12 pm  Mon Apr 10 95
        coursset.h         655 bytes   10:12 pm  Mon Apr 10 95
        dynabind.clw     3,208 bytes   10:12 pm  Mon Apr 10 95
        dynabind.def       361 bytes   10:12 pm  Mon Apr 10 95
        dynabind.mak     3,154 bytes   10:12 pm  Mon Apr 10 95
        dynabind.rc      9,559 bytes   10:12 pm  Mon Apr 10 95
        enroldoc.cpp     1,492 bytes   10:12 pm  Mon Apr 10 95
        enroldoc.h         923 bytes   10:12 pm  Mon Apr 10 95
        enroll.cpp       3,348 bytes   10:12 pm  Mon Apr 10 95
        enroll.h           819 bytes   10:12 pm  Mon Apr 10 95
        mainfrm.cpp      2,442 bytes   10:12 pm  Mon Apr 10 95
```

```
        mainfrm.h             841 bytes   10:12 pm  Mon Apr 10 95
        makefile              590 bytes   10:12 pm  Mon Apr 10 95
        RES                     <DIR>     10:56 am  Mon Jun 10 96
        resource.h          1,167 bytes   10:12 pm  Mon Apr 10 95
        sectform.cpp        8,232 bytes   10:12 pm  Mon Apr 10 95
        sectform.h          1,965 bytes   10:12 pm  Mon Apr 10 95
        sectset.cpp         2,143 bytes   10:12 pm  Mon Apr 10 95
        sectset.h           1,031 bytes   10:12 pm  Mon Apr 10 95
        stdafx.cpp            204 bytes   10:12 pm  Mon Apr 10 95
        stdafx.h              352 bytes   10:12 pm  Mon Apr 10 95

I:\VC152\MSVC15\MFC\SAMPLES\DYNABIND\RES
        .                       <DIR>     10:56 am  Mon Jun 10 96
        ..                      <DIR>     10:56 am  Mon Jun 10 96
        enroll.ico            766 bytes   10:12 pm  Mon Apr 10 95
        enroll.rc2          1,543 bytes   10:12 pm  Mon Apr 10 95
        idr_main.bmp        1,678 bytes   10:12 pm  Mon Apr 10 95
        toolbar.bmp         1,678 bytes   10:12 pm  Mon Apr 10 95

I:\VC152\MSVC15\MFC\SAMPLES\ENROLL
        .                       <DIR>     10:57 am  Mon Jun 10 96
        ..                      <DIR>     11:00 am  Mon Jun 10 96
        stdreg.mdb        131,072 bytes    2:10 pm  Fri Jan 13 95
        STEP1                   <DIR>     10:56 am  Mon Jun 10 96
        STEP2                   <DIR>     10:56 am  Mon Jun 10 96
        STEP3                   <DIR>     10:57 am  Mon Jun 10 96
        STEP4                   <DIR>     10:57 am  Mon Jun 10 96

I:\VC152\MSVC15\MFC\SAMPLES\ENROLL\STEP1
        .                       <DIR>     10:56 am  Mon Jun 10 96
        ..                      <DIR>     10:57 am  Mon Jun 10 96
        enroldoc.cpp        1,469 bytes    2:10 pm  Fri Jan 13 95
        enroldoc.h            896 bytes    2:10 pm  Fri Jan 13 95
        enroll.clw          2,837 bytes    2:10 pm  Fri Jan 13 95
        enroll.cpp          3,325 bytes    2:10 pm  Fri Jan 13 95
        enroll.def            346 bytes    2:10 pm  Fri Jan 13 95
        enroll.h              819 bytes    2:10 pm  Fri Jan 13 95
        enroll.mak          2,694 bytes    2:10 pm  Fri Jan 13 95
        enroll.rc           8,978 bytes    2:10 pm  Fri Jan 13 95
        mainfrm.cpp         2,507 bytes    2:10 pm  Fri Jan 13 95
        mainfrm.h             982 bytes    2:10 pm  Fri Jan 13 95
        readme.txt          4,188 bytes    2:10 pm  Fri Jan 13 95
        RES                     <DIR>     10:56 am  Mon Jun 10 96
        resource.h            843 bytes    2:10 pm  Fri Jan 13 95
        sectform.cpp        3,088 bytes    2:10 pm  Fri Jan 13 95
        sectform.h          1,544 bytes    2:10 pm  Fri Jan 13 95
        sectset.cpp         1,122 bytes    2:10 pm  Fri Jan 13 95
        sectset.h             718 bytes    2:10 pm  Fri Jan 13 95
        stdafx.cpp            204 bytes    2:10 pm  Fri Jan 13 95
        stdafx.h              352 bytes    2:10 pm  Fri Jan 13 95

I:\VC152\MSVC15\MFC\SAMPLES\ENROLL\STEP1\RES
        .                       <DIR>     10:56 am  Mon Jun 10 96
        ..                      <DIR>     10:56 am  Mon Jun 10 96
```

```
        enroll.ico          768 bytes   2:10 pm  Fri Jan 13 95
        enroll.rc2        1,543 bytes   2:10 pm  Fri Jan 13 95
        toolbar.bmp       1,678 bytes   2:10 pm  Fri Jan 13 95

I:\VC152\MSVC15\MFC\SAMPLES\ENROLL\STEP2
        .                   <DIR>       10:56 am  Mon Jun 10 96
        ..                  <DIR>       10:57 am  Mon Jun 10 96
        coursset.cpp        984 bytes   2:10 pm  Fri Jan 13 95
        coursset.h          655 bytes   2:10 pm  Fri Jan 13 95
        enroldoc.cpp      1,492 bytes   2:10 pm  Fri Jan 13 95
        enroldoc.h          923 bytes   2:10 pm  Fri Jan 13 95
        enroll.clw        3,078 bytes   2:10 pm  Fri Jan 13 95
        enroll.cpp        3,348 bytes   2:10 pm  Fri Jan 13 95
        enroll.def          346 bytes   2:10 pm  Fri Jan 13 95
        enroll.h            819 bytes   2:10 pm  Fri Jan 13 95
        enroll.mak        2,843 bytes   2:10 pm  Fri Jan 13 95
        enroll.rc         9,029 bytes   2:10 pm  Fri Jan 13 95
        mainfrm.cpp       2,507 bytes   2:10 pm  Fri Jan 13 95
        mainfrm.h           982 bytes   2:10 pm  Fri Jan 13 95
        readme.txt        4,188 bytes   2:10 pm  Fri Jan 13 95
        RES                 <DIR>       10:56 am  Mon Jun 10 96
        resource.h          842 bytes   2:10 pm  Fri Jan 13 95
        sectform.cpp      4,123 bytes   2:10 pm  Fri Jan 13 95
        sectform.h        1,472 bytes   2:10 pm  Fri Jan 13 95
        sectset.cpp       1,264 bytes   2:10 pm  Fri Jan 13 95
        sectset.h           748 bytes   2:10 pm  Fri Jan 13 95
        stdafx.cpp          204 bytes   2:10 pm  Fri Jan 13 95
        stdafx.h            352 bytes   2:10 pm  Fri Jan 13 95

I:\VC152\MSVC15\MFC\SAMPLES\ENROLL\STEP2\RES
        .                   <DIR>       10:56 am  Mon Jun 10 96
        ..                  <DIR>       10:56 am  Mon Jun 10 96
        enroll.ico          768 bytes   2:10 pm  Fri Jan 13 95
        enroll.rc2        1,543 bytes   2:10 pm  Fri Jan 13 95
        toolbar.bmp       1,678 bytes   2:10 pm  Fri Jan 13 95

I:\VC152\MSVC15\MFC\SAMPLES\ENROLL\STEP3
        .                   <DIR>       10:57 am  Mon Jun 10 96
        ..                  <DIR>       10:57 am  Mon Jun 10 96
        coursset.cpp        984 bytes   2:10 pm  Fri Jan 13 95
        coursset.h          655 bytes   2:10 pm  Fri Jan 13 95
        enroldoc.cpp      1,492 bytes   2:10 pm  Fri Jan 13 95
        enroldoc.h          923 bytes   2:10 pm  Fri Jan 13 95
        enroll.clw        3,130 bytes   2:10 pm  Fri Jan 13 95
        enroll.cpp        3,348 bytes   2:10 pm  Fri Jan 13 95
        enroll.def          346 bytes   2:10 pm  Fri Jan 13 95
        enroll.h            819 bytes   2:10 pm  Fri Jan 13 95
        enroll.mak        2,843 bytes   2:10 pm  Fri Jan 13 95
        enroll.rc         9,096 bytes   2:10 pm  Fri Jan 13 95
        mainfrm.cpp       2,582 bytes   2:10 pm  Fri Jan 13 95
        mainfrm.h           982 bytes   2:10 pm  Fri Jan 13 95
        readme.txt        4,188 bytes   2:10 pm  Fri Jan 13 95
        RES                 <DIR>       10:57 am  Mon Jun 10 96
        resource.h          983 bytes   2:10 pm  Fri Jan 13 95
```

```
        sectform.cpp     6,006 bytes   2:10 pm  Fri Jan 13 95
        sectform.h       1,668 bytes   2:10 pm  Fri Jan 13 95
        sectset.cpp      1,264 bytes   2:10 pm  Fri Jan 13 95
        sectset.h          748 bytes   2:10 pm  Fri Jan 13 95
        stdafx.cpp         204 bytes   2:10 pm  Fri Jan 13 95
        stdafx.h           352 bytes   2:10 pm  Fri Jan 13 95

I:\VC152\MSVC15\MFC\SAMPLES\ENROLL\STEP3\RES
        .                  <DIR>       10:57 am  Mon Jun 10 96
        ..                 <DIR>       10:57 am  Mon Jun 10 96
        enroll.ico         768 bytes   2:10 pm  Fri Jan 13 95
        enroll.rc2       1,543 bytes   2:10 pm  Fri Jan 13 95
        toolbar.bmp      1,558 bytes   2:10 pm  Fri Jan 13 95

I:\VC152\MSVC15\MFC\SAMPLES\ENROLL\STEP4
        .                  <DIR>       10:57 am  Mon Jun 10 96
        ..                 <DIR>       10:57 am  Mon Jun 10 96
        addform.cpp      2,684 bytes   2:10 pm  Fri Jan 13 95
        addform.h          804 bytes   2:10 pm  Fri Jan 13 95
        coursset.cpp       892 bytes   2:10 pm  Fri Jan 13 95
        coursset.h         584 bytes   2:10 pm  Fri Jan 13 95
        crsform.cpp      2,691 bytes   2:10 pm  Fri Jan 13 95
        crsform.h        1,036 bytes   2:10 pm  Fri Jan 13 95
        enroldoc.cpp     1,969 bytes   2:10 pm  Fri Jan 13 95
        enroldoc.h       1,286 bytes   2:10 pm  Fri Jan 13 95
        enroll.clw       3,416 bytes   2:10 pm  Fri Jan 13 95
        enroll.cpp       3,368 bytes   2:10 pm  Fri Jan 13 95
        enroll.def         350 bytes   2:10 pm  Fri Jan 13 95
        enroll.h           898 bytes   2:10 pm  Fri Jan 13 95
        enroll.mak       3,140 bytes   2:10 pm  Fri Jan 13 95
        enroll.rc        9,858 bytes   2:10 pm  Fri Jan 13 95
        mainfrm.cpp      4,254 bytes   2:10 pm  Fri Jan 13 95
        mainfrm.h        1,102 bytes   2:10 pm  Fri Jan 13 95
        readme.txt       4,188 bytes   2:10 pm  Fri Jan 13 95
        RES                <DIR>       10:57 am  Mon Jun 10 96
        resource.h       1,441 bytes   2:10 pm  Fri Jan 13 95
        sectform.cpp     5,959 bytes   2:10 pm  Fri Jan 13 95
        sectform.h       1,635 bytes   2:10 pm  Fri Jan 13 95
        sectset.cpp      1,144 bytes   2:10 pm  Fri Jan 13 95
        sectset.h          677 bytes   2:10 pm  Fri Jan 13 95
        stdafx.cpp         204 bytes   2:10 pm  Fri Jan 13 95
        stdafx.h           352 bytes   2:10 pm  Fri Jan 13 95

I:\VC152\MSVC15\MFC\SAMPLES\ENROLL\STEP4\RES
        .                  <DIR>       10:57 am  Mon Jun 10 96
        ..                 <DIR>       10:57 am  Mon Jun 10 96
        enroll.ico         768 bytes   2:10 pm  Fri Jan 13 95
        enroll.rc2       1,543 bytes   2:10 pm  Fri Jan 13 95
        toolbar.bmp      1,558 bytes   2:10 pm  Fri Jan 13 95

I:\VC152\MSVC15\MFC\SAMPLES\HELLO
        .                  <DIR>       10:57 am  Mon Jun 10 96
        ..                 <DIR>       11:00 am  Mon Jun 10 96
        hello.clw          884 bytes   10:12 pm  Mon Apr 10 95
```

```
        hello.cpp          3,595 bytes   10:12 pm  Mon Apr 10 95
        hello.def            700 bytes   10:12 pm  Mon Apr 10 95
        hello.h            1,481 bytes   10:12 pm  Mon Apr 10 95
        hello.ico            766 bytes   10:12 pm  Mon Apr 10 95
        hello.mak          2,040 bytes   10:12 pm  Mon Apr 10 95
        hello.rc           2,290 bytes   10:12 pm  Mon Apr 10 95
        makefile             481 bytes   10:12 pm  Mon Apr 10 95
        resource.h           439 bytes   10:12 pm  Mon Apr 10 95
        stdafx.cpp           495 bytes   10:12 pm  Mon Apr 10 95
        stdafx.h             441 bytes   10:12 pm  Mon Apr 10 95

I:\VC152\MSVC15\MFC\SAMPLES\HELLOAPP
        .                    <DIR>       10:57 am  Mon Jun 10 96
        ..                   <DIR>       11:00 am  Mon Jun 10 96
        helloapp.cpp       1,107 bytes   10:12 pm  Mon Apr 10 95
        helloapp.def         652 bytes   10:12 pm  Mon Apr 10 95
        helloapp.mak       1,741 bytes   10:12 pm  Mon Apr 10 95
        makefile             740 bytes   10:12 pm  Mon Apr 10 95

I:\VC152\MSVC15\MFC\SAMPLES\HIERSVR
        .                    <DIR>       10:57 am  Mon Jun 10 96
        ..                   <DIR>       11:00 am  Mon Jun 10 96
        hiersvr.clw        7,218 bytes   10:12 pm  Mon Apr 10 95
        hiersvr.cpp        4,535 bytes   10:12 pm  Mon Apr 10 95
        hiersvr.def          694 bytes   10:12 pm  Mon Apr 10 95
        hiersvr.h          1,156 bytes   10:12 pm  Mon Apr 10 95
        hiersvr.hpj        2,726 bytes   10:12 pm  Mon Apr 10 95
        hiersvr.mak        3,039 bytes   10:12 pm  Mon Apr 10 95
        hiersvr.rc        20,641 bytes   10:12 pm  Mon Apr 10 95
        hiersvr.reg        1,969 bytes   10:12 pm  Mon Apr 10 95
        HLP                  <DIR>       10:57 am  Mon Jun 10 96
        ipframe.cpp        3,316 bytes   10:12 pm  Mon Apr 10 95
        ipframe.h          1,483 bytes   10:12 pm  Mon Apr 10 95
        mainfrm.cpp        3,189 bytes   10:12 pm  Mon Apr 10 95
        mainfrm.h          1,314 bytes   10:12 pm  Mon Apr 10 95
        makefile             573 bytes   10:12 pm  Mon Apr 10 95
        makehelp.bat         870 bytes   10:12 pm  Mon Apr 10 95
        mfcclass.hie       2,716 bytes   10:12 pm  Mon Apr 10 95
        mfcclass.txt       1,902 bytes   10:12 pm  Mon Apr 10 95
        RES                  <DIR>       10:57 am  Mon Jun 10 96
        resource.h         1,873 bytes   10:12 pm  Mon Apr 10 95
        stdafx.cpp           643 bytes   10:12 pm  Mon Apr 10 95
        stdafx.h             670 bytes   10:12 pm  Mon Apr 10 95
        svrdoc.cpp         6,938 bytes   10:12 pm  Mon Apr 10 95
        svrdoc.h           1,912 bytes   10:12 pm  Mon Apr 10 95
        svritem.cpp       20,362 bytes   10:12 pm  Mon Apr 10 95
        svritem.h          4,340 bytes   10:12 pm  Mon Apr 10 95
        svrview.cpp       28,303 bytes   10:12 pm  Mon Apr 10 95
        svrview.h          4,500 bytes   10:12 pm  Mon Apr 10 95
        zoomdlg.cpp        1,040 bytes   10:12 pm  Mon Apr 10 95
        zoomdlg.h            652 bytes   10:12 pm  Mon Apr 10 95

I:\VC152\MSVC15\MFC\SAMPLES\HIERSVR\HLP
        .                    <DIR>       10:57 am  Mon Jun 10 96
```

```
..                  <DIR>       10:57 am  Mon Jun 10 96
afxcore.rtf      47,264 bytes   10:12 pm  Mon Apr 10 95
afxolesv.rtf      4,688 bytes   10:12 pm  Mon Apr 10 95
afxprint.rtf     11,360 bytes   10:12 pm  Mon Apr 10 95
appexit.bmp       2,272 bytes   10:12 pm  Mon Apr 10 95
bullet.bmp          144 bytes   10:12 pm  Mon Apr 10 95
curarw2.bmp         320 bytes   10:12 pm  Mon Apr 10 95
curarw4.bmp         576 bytes   10:12 pm  Mon Apr 10 95
curhelp.bmp         512 bytes   10:12 pm  Mon Apr 10 95
editcopy.bmp        512 bytes   10:12 pm  Mon Apr 10 95
editcut.bmp         512 bytes   10:12 pm  Mon Apr 10 95
editpast.bmp        512 bytes   10:12 pm  Mon Apr 10 95
editundo.bmp        512 bytes   10:12 pm  Mon Apr 10 95
filenew.bmp         576 bytes   10:12 pm  Mon Apr 10 95
fileopen.bmp        576 bytes   10:12 pm  Mon Apr 10 95
fileprnt.bmp        512 bytes   10:12 pm  Mon Apr 10 95
filesave.bmp        512 bytes   10:12 pm  Mon Apr 10 95
hlpsbar.bmp       7,168 bytes   10:12 pm  Mon Apr 10 95
hlptbar.bmp       2,368 bytes   10:12 pm  Mon Apr 10 95
scmax.bmp           512 bytes   10:12 pm  Mon Apr 10 95
scmenu.bmp        2,144 bytes   10:12 pm  Mon Apr 10 95
scmin.bmp           512 bytes   10:12 pm  Mon Apr 10 95

I:\VC152\MSVC15\MFC\SAMPLES\HIERSVR\RES
.                   <DIR>       10:57 am  Mon Jun 10 96
..                  <DIR>       10:57 am  Mon Jun 10 96
hiersvr.ico         766 bytes   10:12 pm  Mon Apr 10 95
itoolbar.bmp        718 bytes   10:12 pm  Mon Apr 10 95
svrdoc.ico          766 bytes   10:12 pm  Mon Apr 10 95
toolbar.bmp       1,200 bytes   10:12 pm  Mon Apr 10 95

I:\VC152\MSVC15\MFC\SAMPLES\MAKEHM
.                   <DIR>       10:57 am  Mon Jun 10 96
..                  <DIR>       11:00 am  Mon Jun 10 96
makefile          1,178 bytes   10:12 pm  Mon Apr 10 95
makehm.cpp        8,588 bytes   10:12 pm  Mon Apr 10 95
makehm.mak        1,525 bytes   10:12 pm  Mon Apr 10 95

I:\VC152\MSVC15\MFC\SAMPLES\MDI
.                   <DIR>       10:57 am  Mon Jun 10 96
..                  <DIR>       11:00 am  Mon Jun 10 96
bounce.cpp        7,162 bytes   10:12 pm  Mon Apr 10 95
bounce.h          2,023 bytes   10:12 pm  Mon Apr 10 95
hello.cpp         3,171 bytes   10:12 pm  Mon Apr 10 95
hello.h           1,295 bytes   10:12 pm  Mon Apr 10 95
hello.ico           766 bytes   10:12 pm  Mon Apr 10 95
mainfrm.cpp       1,870 bytes   10:12 pm  Mon Apr 10 95
mainfrm.h           952 bytes   10:12 pm  Mon Apr 10 95
makefile            510 bytes   10:12 pm  Mon Apr 10 95
mdi.clw           2,026 bytes   10:12 pm  Mon Apr 10 95
mdi.cpp           2,202 bytes   10:12 pm  Mon Apr 10 95
mdi.def             684 bytes   10:12 pm  Mon Apr 10 95
mdi.h             1,255 bytes   10:12 pm  Mon Apr 10 95
mdi.ico             766 bytes   10:12 pm  Mon Apr 10 95
```

```
        mdi.mak           2,397 bytes   10:12 pm  Mon Apr 10 95
        mdi.rc            5,044 bytes   10:12 pm  Mon Apr 10 95
        resource.h        1,053 bytes   10:12 pm  Mon Apr 10 95
        stdafx.cpp          643 bytes   10:12 pm  Mon Apr 10 95
        stdafx.h            721 bytes   10:12 pm  Mon Apr 10 95

I:\VC152\MSVC15\MFC\SAMPLES\MULTIPAD
        .                   <DIR>       10:57 am  Mon Jun 10 96
        ..                  <DIR>       11:00 am  Mon Jun 10 96
        makefile            487 bytes   10:12 pm  Mon Apr 10 95
        multipad.clw      2,298 bytes   10:12 pm  Mon Apr 10 95
        multipad.cpp      2,765 bytes   10:12 pm  Mon Apr 10 95
        multipad.def        697 bytes   10:12 pm  Mon Apr 10 95
        multipad.h        1,210 bytes   10:12 pm  Mon Apr 10 95
        multipad.mak      2,094 bytes   10:12 pm  Mon Apr 10 95
        multipad.rc       9,328 bytes   10:12 pm  Mon Apr 10 95
        RES                 <DIR>       10:57 am  Mon Jun 10 96
        resource.h          528 bytes   10:12 pm  Mon Apr 10 95
        stdafx.cpp          643 bytes   10:12 pm  Mon Apr 10 95
        stdafx.h            672 bytes   10:12 pm  Mon Apr 10 95

I:\VC152\MSVC15\MFC\SAMPLES\MULTIPAD\RES
        .                   <DIR>       10:57 am  Mon Jun 10 96
        ..                  <DIR>       10:57 am  Mon Jun 10 96
        multipad.ico        766 bytes   10:12 pm  Mon Apr 10 95
        paddoc.ico          766 bytes   10:12 pm  Mon Apr 10 95
        toolbar.bmp       1,198 bytes   10:12 pm  Mon Apr 10 95

I:\VC152\MSVC15\MFC\SAMPLES\OCLIENT
        .                   <DIR>       10:58 am  Mon Jun 10 96
        ..                  <DIR>       11:00 am  Mon Jun 10 96
        frame.cpp         3,223 bytes   10:12 pm  Mon Apr 10 95
        frame.h           1,242 bytes   10:12 pm  Mon Apr 10 95
        HLP                 <DIR>       10:58 am  Mon Jun 10 96
        maindoc.cpp       5,089 bytes   10:12 pm  Mon Apr 10 95
        maindoc.h         1,506 bytes   10:12 pm  Mon Apr 10 95
        mainview.cpp     30,890 bytes   10:12 pm  Mon Apr 10 95
        mainview.h        4,454 bytes   10:12 pm  Mon Apr 10 95
        makefile            558 bytes   10:12 pm  Mon Apr 10 95
        makehelp.bat        870 bytes   10:12 pm  Mon Apr 10 95
        oclient.clw       3,689 bytes   10:12 pm  Mon Apr 10 95
        oclient.cpp       4,088 bytes   10:12 pm  Mon Apr 10 95
        oclient.def         780 bytes   10:12 pm  Mon Apr 10 95
        oclient.h         1,079 bytes   10:12 pm  Mon Apr 10 95
        oclient.hpj       2,727 bytes   10:12 pm  Mon Apr 10 95
        oclient.mak       2,973 bytes   10:12 pm  Mon Apr 10 95
        oclient.rc       13,825 bytes   10:12 pm  Mon Apr 10 95
        oclient.reg       1,262 bytes   10:12 pm  Mon Apr 10 95
        rectitem.cpp      5,930 bytes   10:12 pm  Mon Apr 10 95
        rectitem.h        1,751 bytes   10:12 pm  Mon Apr 10 95
        RES                 <DIR>       10:58 am  Mon Jun 10 96
        resource.h        1,355 bytes   10:12 pm  Mon Apr 10 95
        splitfra.cpp      1,436 bytes   10:12 pm  Mon Apr 10 95
        splitfra.h        1,311 bytes   10:12 pm  Mon Apr 10 95
```

```
        stdafx.cpp          497 bytes  10:12 pm  Mon Apr 10 95
        stdafx.h            693 bytes  10:12 pm  Mon Apr 10 95

I:\VC152\MSVC15\MFC\SAMPLES\OCLIENT\HLP
        .                   <DIR>      10:58 am  Mon Jun 10 96
        ..                  <DIR>      10:58 am  Mon Jun 10 96
        afxcore.rtf      47,264 bytes  10:12 pm  Mon Apr 10 95
        afxolecl.rtf      7,920 bytes  10:12 pm  Mon Apr 10 95
        afxprint.rtf     11,360 bytes  10:12 pm  Mon Apr 10 95
        appexit.bmp       2,272 bytes  10:12 pm  Mon Apr 10 95
        bullet.bmp          144 bytes  10:12 pm  Mon Apr 10 95
        curarw2.bmp         320 bytes  10:12 pm  Mon Apr 10 95
        curarw4.bmp         576 bytes  10:12 pm  Mon Apr 10 95
        curhelp.bmp         512 bytes  10:12 pm  Mon Apr 10 95
        editcopy.bmp        512 bytes  10:12 pm  Mon Apr 10 95
        editcut.bmp         512 bytes  10:12 pm  Mon Apr 10 95
        editpast.bmp        512 bytes  10:12 pm  Mon Apr 10 95
        editundo.bmp        512 bytes  10:12 pm  Mon Apr 10 95
        filenew.bmp         576 bytes  10:12 pm  Mon Apr 10 95
        fileopen.bmp        576 bytes  10:12 pm  Mon Apr 10 95
        fileprnt.bmp        512 bytes  10:12 pm  Mon Apr 10 95
        filesave.bmp        512 bytes  10:12 pm  Mon Apr 10 95
        hlpsbar.bmp       7,168 bytes  10:12 pm  Mon Apr 10 95
        hlptbar.bmp       2,368 bytes  10:12 pm  Mon Apr 10 95
        scmax.bmp           512 bytes  10:12 pm  Mon Apr 10 95
        scmenu.bmp        2,144 bytes  10:12 pm  Mon Apr 10 95
        scmin.bmp           512 bytes  10:12 pm  Mon Apr 10 95

I:\VC152\MSVC15\MFC\SAMPLES\OCLIENT\RES
        .                   <DIR>      10:58 am  Mon Jun 10 96
        ..                  <DIR>      10:58 am  Mon Jun 10 96
        oclient.ico         766 bytes  10:12 pm  Mon Apr 10 95
        toolbar.bmp       1,200 bytes  10:12 pm  Mon Apr 10 95

I:\VC152\MSVC15\MFC\SAMPLES\PROPDLG
        .                   <DIR>      10:58 am  Mon Jun 10 96
        ..                  <DIR>      11:00 am  Mon Jun 10 96
        colorpge.cpp      2,328 bytes  10:12 pm  Mon Apr 10 95
        colorpge.h          778 bytes  10:12 pm  Mon Apr 10 95
        mainfrm.cpp       2,494 bytes  10:12 pm  Mon Apr 10 95
        mainfrm.h           915 bytes  10:12 pm  Mon Apr 10 95
        makefile            603 bytes  10:12 pm  Mon Apr 10 95
        preview.cpp       1,260 bytes  10:12 pm  Mon Apr 10 95
        preview.h           415 bytes  10:12 pm  Mon Apr 10 95
        propdlg.clw       3,641 bytes  10:12 pm  Mon Apr 10 95
        propdlg.cpp       3,436 bytes  10:12 pm  Mon Apr 10 95
        propdlg.def         774 bytes  10:12 pm  Mon Apr 10 95
        propdlg.h           854 bytes  10:12 pm  Mon Apr 10 95
        propdlg.mak       3,443 bytes  10:12 pm  Mon Apr 10 95
        propdlg.rc       15,763 bytes  10:12 pm  Mon Apr 10 95
        propsht.cpp       2,406 bytes  10:12 pm  Mon Apr 10 95
        propsht.h           714 bytes  10:12 pm  Mon Apr 10 95
        propsht2.cpp      1,719 bytes  10:12 pm  Mon Apr 10 95
        propsht2.h          737 bytes  10:12 pm  Mon Apr 10 95
```

```
        RES                 <DIR>       10:58 am  Mon Jun 10 96
        resource.h        1,139 bytes   10:12 pm  Mon Apr 10 95
        shapedoc.cpp      2,122 bytes   10:12 pm  Mon Apr 10 95
        shapedoc.h        1,019 bytes   10:12 pm  Mon Apr 10 95
        shapeobj.cpp      2,209 bytes   10:12 pm  Mon Apr 10 95
        shapeobj.h          787 bytes   10:12 pm  Mon Apr 10 95
        shapesvw.cpp      7,175 bytes   10:12 pm  Mon Apr 10 95
        shapesvw.h        1,710 bytes   10:12 pm  Mon Apr 10 95
        stdafx.cpp          205 bytes   10:12 pm  Mon Apr 10 95
        stdafx.h            299 bytes   10:12 pm  Mon Apr 10 95
        stylepge.cpp      1,960 bytes   10:12 pm  Mon Apr 10 95
        stylepge.h          783 bytes   10:12 pm  Mon Apr 10 95

I:\VC152\MSVC15\MFC\SAMPLES\PROPDLG\RES
        .                   <DIR>       10:58 am  Mon Jun 10 96
        ..                  <DIR>       10:58 am  Mon Jun 10 96
        propdlg.ico         766 bytes   10:12 pm  Mon Apr 10 95
        propdlg.rc2         379 bytes   10:12 pm  Mon Apr 10 95
        shapedoc.ico        766 bytes   10:12 pm  Mon Apr 10 95
        toolbar.bmp       1,198 bytes   10:12 pm  Mon Apr 10 95

I:\VC152\MSVC15\MFC\SAMPLES\SCRIBBLE
        .                   <DIR>       10:59 am  Mon Jun 10 96
        ..                  <DIR>       11:00 am  Mon Jun 10 96
        STEP0               <DIR>       10:58 am  Mon Jun 10 96
        STEP1               <DIR>       10:58 am  Mon Jun 10 96
        STEP2               <DIR>       10:58 am  Mon Jun 10 96
        STEP3               <DIR>       10:58 am  Mon Jun 10 96
        STEP4               <DIR>       10:58 am  Mon Jun 10 96
        STEP5               <DIR>       10:58 am  Mon Jun 10 96
        STEP6               <DIR>       10:59 am  Mon Jun 10 96
        STEP7               <DIR>       10:59 am  Mon Jun 10 96

I:\VC152\MSVC15\MFC\SAMPLES\SCRIBBLE\STEP0
        .                   <DIR>       10:58 am  Mon Jun 10 96
        ..                  <DIR>       10:59 am  Mon Jun 10 96
        mainfrm.cpp       2,913 bytes    2:10 pm  Fri Jan 13 95
        mainfrm.h         1,378 bytes    2:10 pm  Fri Jan 13 95
        readme.txt        4,010 bytes    2:10 pm  Fri Jan 13 95
        RES                 <DIR>       10:58 am  Mon Jun 10 96
        resource.h          403 bytes    2:10 pm  Fri Jan 13 95
        scribble.clw      2,236 bytes    2:10 pm  Fri Jan 13 95
        scribble.cpp      4,417 bytes    2:10 pm  Fri Jan 13 95
        scribble.def        779 bytes    2:10 pm  Fri Jan 13 95
        scribble.h        1,266 bytes    2:10 pm  Fri Jan 13 95
        scribble.mak      2,599 bytes    2:10 pm  Fri Jan 13 95
        scribble.rc       9,147 bytes    2:10 pm  Fri Jan 13 95
        scribdoc.cpp      2,141 bytes    2:10 pm  Fri Jan 13 95
        scribdoc.h        1,302 bytes    2:10 pm  Fri Jan 13 95
        scribvw.cpp       2,720 bytes    2:10 pm  Fri Jan 13 95
        scribvw.h         1,623 bytes    2:10 pm  Fri Jan 13 95
        stdafx.cpp          639 bytes    2:10 pm  Fri Jan 13 95
        stdafx.h            734 bytes    2:10 pm  Fri Jan 13 95
```

```
I:\VC152\MSVC15\MFC\SAMPLES\SCRIBBLE\STEP0\RES
        .                   <DIR>      10:58 am  Mon Jun 10 96
        ..                  <DIR>      10:58 am  Mon Jun 10 96
        scribble.ico          766 bytes   2:10 pm  Fri Jan 13 95
        scribble.rc2        1,646 bytes   2:10 pm  Fri Jan 13 95
        scribdoc.ico          766 bytes   2:10 pm  Fri Jan 13 95
        toolbar.bmp         1,200 bytes   2:10 pm  Fri Jan 13 95

I:\VC152\MSVC15\MFC\SAMPLES\SCRIBBLE\STEP1
        .                   <DIR>      10:58 am  Mon Jun 10 96
        ..                  <DIR>      10:59 am  Mon Jun 10 96
        mainfrm.cpp         2,913 bytes   2:10 pm  Fri Jan 13 95
        mainfrm.h           1,378 bytes   2:10 pm  Fri Jan 13 95
        readme.txt          4,281 bytes   2:10 pm  Fri Jan 13 95
        RES                 <DIR>      10:58 am  Mon Jun 10 96
        resource.h            403 bytes   2:10 pm  Fri Jan 13 95
        scribble.clw        2,322 bytes   2:10 pm  Fri Jan 13 95
        scribble.cpp        4,417 bytes   2:10 pm  Fri Jan 13 95
        scribble.def          779 bytes   2:10 pm  Fri Jan 13 95
        scribble.h          1,266 bytes   2:10 pm  Fri Jan 13 95
        scribble.mak        2,599 bytes   2:10 pm  Fri Jan 13 95
        scribble.rc         9,147 bytes   2:10 pm  Fri Jan 13 95
        scribdoc.cpp        4,693 bytes   2:10 pm  Fri Jan 13 95
        scribdoc.h          2,810 bytes   2:10 pm  Fri Jan 13 95
        scribvw.cpp         5,056 bytes   2:10 pm  Fri Jan 13 95
        scribvw.h           1,937 bytes   2:10 pm  Fri Jan 13 95
        stdafx.cpp            639 bytes   2:10 pm  Fri Jan 13 95
        stdafx.h              734 bytes   2:10 pm  Fri Jan 13 95

I:\VC152\MSVC15\MFC\SAMPLES\SCRIBBLE\STEP1\RES
        .                   <DIR>      10:58 am  Mon Jun 10 96
        ..                  <DIR>      10:58 am  Mon Jun 10 96
        scribble.ico          766 bytes   2:10 pm  Fri Jan 13 95
        scribble.rc2        1,646 bytes   2:10 pm  Fri Jan 13 95
        scribdoc.ico          766 bytes   2:10 pm  Fri Jan 13 95
        toolbar.bmp         1,200 bytes   2:10 pm  Fri Jan 13 95

I:\VC152\MSVC15\MFC\SAMPLES\SCRIBBLE\STEP2
        .                   <DIR>      10:58 am  Mon Jun 10 96
        ..                  <DIR>      10:59 am  Mon Jun 10 96
        mainfrm.cpp         2,954 bytes   2:10 pm  Fri Jan 13 95
        mainfrm.h           1,378 bytes   2:10 pm  Fri Jan 13 95
        readme.txt          4,281 bytes   2:10 pm  Fri Jan 13 95
        RES                 <DIR>      10:58 am  Mon Jun 10 96
        resource.h            618 bytes   2:10 pm  Fri Jan 13 95
        scribble.clw        2,322 bytes   2:10 pm  Fri Jan 13 95
        scribble.cpp        4,417 bytes   2:10 pm  Fri Jan 13 95
        scribble.def          779 bytes   2:10 pm  Fri Jan 13 95
        scribble.h          1,266 bytes   2:10 pm  Fri Jan 13 95
        scribble.mak        2,599 bytes   2:10 pm  Fri Jan 13 95
        scribble.rc         9,538 bytes   2:10 pm  Fri Jan 13 95
        scribdoc.cpp        6,377 bytes   2:10 pm  Fri Jan 13 95
        scribdoc.h          3,154 bytes   2:10 pm  Fri Jan 13 95
        scribvw.cpp         5,056 bytes   2:10 pm  Fri Jan 13 95
```

```
        scribvw.h         1,937 bytes   2:10 pm  Fri Jan 13 95
        stdafx.cpp          639 bytes   2:10 pm  Fri Jan 13 95
        stdafx.h            734 bytes   2:10 pm  Fri Jan 13 95

I:\VC152\MSVC15\MFC\SAMPLES\SCRIBBLE\STEP2\RES
        .                   <DIR>       10:58 am  Mon Jun 10 96
        ..                  <DIR>       10:58 am  Mon Jun 10 96
        scribble.ico        766 bytes   2:10 pm  Fri Jan 13 95
        scribble.rc2      1,646 bytes   2:10 pm  Fri Jan 13 95
        scribdoc.ico        766 bytes   2:10 pm  Fri Jan 13 95
        toolbar.bmp       1,318 bytes   2:10 pm  Fri Jan 13 95

I:\VC152\MSVC15\MFC\SAMPLES\SCRIBBLE\STEP3
        .                   <DIR>       10:58 am  Mon Jun 10 96
        ..                  <DIR>       10:59 am  Mon Jun 10 96
        mainfrm.cpp       2,954 bytes   2:10 pm  Fri Jan 13 95
        mainfrm.h         1,378 bytes   2:10 pm  Fri Jan 13 95
        pendlg.cpp        1,980 bytes   2:10 pm  Fri Jan 13 95
        pendlg.h            999 bytes   2:10 pm  Fri Jan 13 95
        readme.txt        4,281 bytes   2:10 pm  Fri Jan 13 95
        RES                 <DIR>       10:58 am  Mon Jun 10 96
        resource.h          798 bytes   2:10 pm  Fri Jan 13 95
        scribble.clw      2,843 bytes   2:10 pm  Fri Jan 13 95
        scribble.cpp      4,417 bytes   2:10 pm  Fri Jan 13 95
        scribble.def        779 bytes   2:10 pm  Fri Jan 13 95
        scribble.h        1,266 bytes   2:10 pm  Fri Jan 13 95
        scribble.mak      2,729 bytes   2:10 pm  Fri Jan 13 95
        scribble.rc      10,207 bytes   2:10 pm  Fri Jan 13 95
        scribdoc.cpp      7,008 bytes   2:10 pm  Fri Jan 13 95
        scribdoc.h        3,184 bytes   2:10 pm  Fri Jan 13 95
        scribvw.cpp       5,056 bytes   2:10 pm  Fri Jan 13 95
        scribvw.h         1,937 bytes   2:10 pm  Fri Jan 13 95
        stdafx.cpp          639 bytes   2:10 pm  Fri Jan 13 95
        stdafx.h            734 bytes   2:10 pm  Fri Jan 13 95

I:\VC152\MSVC15\MFC\SAMPLES\SCRIBBLE\STEP3\RES
        .                   <DIR>       10:58 am  Mon Jun 10 96
        ..                  <DIR>       10:58 am  Mon Jun 10 96
        scribble.ico        766 bytes   2:10 pm  Fri Jan 13 95
        scribble.rc2      1,646 bytes   2:10 pm  Fri Jan 13 95
        scribdoc.ico        766 bytes   2:10 pm  Fri Jan 13 95
        toolbar.bmp       1,318 bytes   2:10 pm  Fri Jan 13 95

I:\VC152\MSVC15\MFC\SAMPLES\SCRIBBLE\STEP4
        .                   <DIR>       10:58 am  Mon Jun 10 96
        ..                  <DIR>       10:59 am  Mon Jun 10 96
        mainfrm.cpp       2,954 bytes   2:10 pm  Fri Jan 13 95
        mainfrm.h         1,378 bytes   2:10 pm  Fri Jan 13 95
        pendlg.cpp        1,980 bytes   2:10 pm  Fri Jan 13 95
        pendlg.h            999 bytes   2:10 pm  Fri Jan 13 95
        readme.txt        4,281 bytes   2:10 pm  Fri Jan 13 95
        RES                 <DIR>       10:58 am  Mon Jun 10 96
        resource.h          798 bytes   2:10 pm  Fri Jan 13 95
        scribble.clw      2,960 bytes   2:10 pm  Fri Jan 13 95
```

```
        scribble.cpp     4,445 bytes   2:10 pm  Fri Jan 13 95
        scribble.def       779 bytes   2:10 pm  Fri Jan 13 95
        scribble.h       1,266 bytes   2:10 pm  Fri Jan 13 95
        scribble.mak     2,873 bytes   2:10 pm  Fri Jan 13 95
        scribble.rc     10,207 bytes   2:10 pm  Fri Jan 13 95
        scribdoc.cpp     8,250 bytes   2:10 pm  Fri Jan 13 95
        scribdoc.h       3,479 bytes   2:10 pm  Fri Jan 13 95
        scribfrm.cpp     1,419 bytes   2:10 pm  Fri Jan 13 95
        scribfrm.h       1,180 bytes   2:10 pm  Fri Jan 13 95
        scribvw.cpp      7,873 bytes   2:10 pm  Fri Jan 13 95
        scribvw.h        2,129 bytes   2:10 pm  Fri Jan 13 95
        stdafx.cpp         639 bytes   2:10 pm  Fri Jan 13 95
        stdafx.h           734 bytes   2:10 pm  Fri Jan 13 95

I:\VC152\MSVC15\MFC\SAMPLES\SCRIBBLE\STEP4\RES
        .                  <DIR>       10:58 am  Mon Jun 10 96
        ..                 <DIR>       10:58 am  Mon Jun 10 96
        scribble.ico       766 bytes   2:10 pm  Fri Jan 13 95
        scribble.rc2     1,646 bytes   2:10 pm  Fri Jan 13 95
        scribdoc.ico       766 bytes   2:10 pm  Fri Jan 13 95
        toolbar.bmp      1,318 bytes   2:10 pm  Fri Jan 13 95

I:\VC152\MSVC15\MFC\SAMPLES\SCRIBBLE\STEP5
        .                  <DIR>       10:58 am  Mon Jun 10 96
        ..                 <DIR>       10:59 am  Mon Jun 10 96
        mainfrm.cpp      2,954 bytes   2:10 pm  Fri Jan 13 95
        mainfrm.h        1,378 bytes   2:10 pm  Fri Jan 13 95
        pendlg.cpp       1,980 bytes   2:10 pm  Fri Jan 13 95
        pendlg.h           999 bytes   2:10 pm  Fri Jan 13 95
        readme.txt       4,281 bytes   2:10 pm  Fri Jan 13 95
        RES                <DIR>       10:58 am  Mon Jun 10 96
        resource.h         798 bytes   2:10 pm  Fri Jan 13 95
        scribble.clw     2,960 bytes   2:10 pm  Fri Jan 13 95
        scribble.cpp     4,445 bytes   2:10 pm  Fri Jan 13 95
        scribble.def       779 bytes   2:10 pm  Fri Jan 13 95
        scribble.h       1,266 bytes   2:10 pm  Fri Jan 13 95
        scribble.mak     2,873 bytes   2:10 pm  Fri Jan 13 95
        scribble.rc     10,207 bytes   2:10 pm  Fri Jan 13 95
        scribdoc.cpp     8,292 bytes   2:10 pm  Fri Jan 13 95
        scribdoc.h       3,574 bytes   2:10 pm  Fri Jan 13 95
        scribfrm.cpp     1,419 bytes   2:10 pm  Fri Jan 13 95
        scribfrm.h       1,180 bytes   2:10 pm  Fri Jan 13 95
        scribvw.cpp     10,383 bytes   2:10 pm  Fri Jan 13 95
        scribvw.h        2,307 bytes   2:10 pm  Fri Jan 13 95
        stdafx.cpp         639 bytes   2:10 pm  Fri Jan 13 95
        stdafx.h           734 bytes   2:10 pm  Fri Jan 13 95

I:\VC152\MSVC15\MFC\SAMPLES\SCRIBBLE\STEP5\RES
        .                  <DIR>       10:58 am  Mon Jun 10 96
        ..                 <DIR>       10:58 am  Mon Jun 10 96
        scribble.ico       766 bytes   2:10 pm  Fri Jan 13 95
        scribble.rc2     1,646 bytes   2:10 pm  Fri Jan 13 95
        scribdoc.ico       766 bytes   2:10 pm  Fri Jan 13 95
        toolbar.bmp      1,318 bytes   2:10 pm  Fri Jan 13 95
```

```
I:\VC152\MSVC15\MFC\SAMPLES\SCRIBBLE\STEP6
.                   <DIR>       10:59 am  Mon Jun 10 96
..                  <DIR>       10:59 am  Mon Jun 10 96
HLP                 <DIR>       10:59 am  Mon Jun 10 96
mainfrm.cpp       3,269 bytes    2:10 pm  Fri Jan 13 95
mainfrm.h         1,378 bytes    2:10 pm  Fri Jan 13 95
makehelp.bat        890 bytes    2:10 pm  Fri Jan 13 95
pendlg.cpp        1,980 bytes    2:10 pm  Fri Jan 13 95
pendlg.h            999 bytes    2:10 pm  Fri Jan 13 95
readme.txt        4,845 bytes    2:10 pm  Fri Jan 13 95
RES                 <DIR>       10:59 am  Mon Jun 10 96
resource.h          798 bytes    2:10 pm  Fri Jan 13 95
scribble.clw      2,960 bytes    2:10 pm  Fri Jan 13 95
scribble.cpp      4,445 bytes    2:10 pm  Fri Jan 13 95
scribble.def        779 bytes    2:10 pm  Fri Jan 13 95
scribble.h        1,266 bytes    2:10 pm  Fri Jan 13 95
scribble.hpj      2,724 bytes    2:10 pm  Fri Jan 13 95
scribble.mak      2,873 bytes    2:10 pm  Fri Jan 13 95
scribble.rc      10,579 bytes    2:10 pm  Fri Jan 13 95
scribdoc.cpp      8,292 bytes    2:10 pm  Fri Jan 13 95
scribdoc.h        3,574 bytes    2:10 pm  Fri Jan 13 95
scribfrm.cpp      1,419 bytes    2:10 pm  Fri Jan 13 95
scribfrm.h        1,180 bytes    2:10 pm  Fri Jan 13 95
scribvw.cpp      10,383 bytes    2:10 pm  Fri Jan 13 95
scribvw.h         2,307 bytes    2:10 pm  Fri Jan 13 95
stdafx.cpp          639 bytes    2:10 pm  Fri Jan 13 95
stdafx.h            734 bytes    2:10 pm  Fri Jan 13 95

I:\VC152\MSVC15\MFC\SAMPLES\SCRIBBLE\STEP6\HLP
.                   <DIR>       10:59 am  Mon Jun 10 96
..                  <DIR>       10:59 am  Mon Jun 10 96
afxcore.rtf      47,264 bytes    2:10 pm  Fri Jan 13 95
afxprint.rtf     11,360 bytes    2:10 pm  Fri Jan 13 95
appexit.bmp       2,272 bytes    2:10 pm  Fri Jan 13 95
bullet.bmp          144 bytes    2:10 pm  Fri Jan 13 95
curarw2.bmp         320 bytes    2:10 pm  Fri Jan 13 95
curarw4.bmp         576 bytes    2:10 pm  Fri Jan 13 95
curhelp.bmp         512 bytes    2:10 pm  Fri Jan 13 95
editcopy.bmp        512 bytes    2:10 pm  Fri Jan 13 95
editcut.bmp         512 bytes    2:10 pm  Fri Jan 13 95
editpast.bmp        512 bytes    2:10 pm  Fri Jan 13 95
editundo.bmp        512 bytes    2:10 pm  Fri Jan 13 95
filenew.bmp         576 bytes    2:10 pm  Fri Jan 13 95
fileopen.bmp        576 bytes    2:10 pm  Fri Jan 13 95
fileprnt.bmp        512 bytes    2:10 pm  Fri Jan 13 95
filesave.bmp        512 bytes    2:10 pm  Fri Jan 13 95
hlpsbar.bmp       7,168 bytes    2:10 pm  Fri Jan 13 95
hlptbar.bmp       2,368 bytes    2:10 pm  Fri Jan 13 95
pen.rtf           6,500 bytes    2:10 pm  Fri Jan 13 95
scmax.bmp           512 bytes    2:10 pm  Fri Jan 13 95
scmenu.bmp        2,144 bytes    2:10 pm  Fri Jan 13 95
scmin.bmp           512 bytes    2:10 pm  Fri Jan 13 95
scribble.hm         598 bytes    2:00 pm  Fri Sep 16 94
```

```
I:\VC152\MSVC15\MFC\SAMPLES\SCRIBBLE\STEP6\RES
        .                   <DIR>      10:59 am  Mon Jun 10 96
        ..                  <DIR>      10:59 am  Mon Jun 10 96
        scribble.ico          766 bytes   2:10 pm  Fri Jan 13 95
        scribble.rc2        1,646 bytes   2:10 pm  Fri Jan 13 95
        scribdoc.ico          766 bytes   2:10 pm  Fri Jan 13 95
        toolbar.bmp         1,318 bytes   2:10 pm  Fri Jan 13 95

I:\VC152\MSVC15\MFC\SAMPLES\SCRIBBLE\STEP7
        .                   <DIR>      10:59 am  Mon Jun 10 96
        ..                  <DIR>      10:59 am  Mon Jun 10 96
        HLP                 <DIR>      10:59 am  Mon Jun 10 96
        ipframe.cpp         3,202 bytes   2:10 pm  Fri Jan 13 95
        ipframe.h           1,284 bytes   2:10 pm  Fri Jan 13 95
        mainfrm.cpp         3,269 bytes   2:10 pm  Fri Jan 13 95
        mainfrm.h           1,378 bytes   2:10 pm  Fri Jan 13 95
        makehelp.bat          890 bytes   2:10 pm  Fri Jan 13 95
        pendlg.cpp          1,980 bytes   2:10 pm  Fri Jan 13 95
        pendlg.h              999 bytes   2:10 pm  Fri Jan 13 95
        readme.txt          3,903 bytes   2:10 pm  Fri Jan 13 95
        RES                 <DIR>      10:59 am  Mon Jun 10 96
        resource.h            933 bytes   2:10 pm  Fri Jan 13 95
        scribble.clw        2,960 bytes   2:10 pm  Fri Jan 13 95
        scribble.cpp        5,913 bytes   2:10 pm  Fri Jan 13 95
        scribble.def          779 bytes   2:10 pm  Fri Jan 13 95
        scribble.h          1,339 bytes   2:10 pm  Fri Jan 13 95
        scribble.hpj        2,724 bytes   2:10 pm  Fri Jan 13 95
        scribble.mak        3,276 bytes   2:10 pm  Fri Jan 13 95
        scribble.rc        15,195 bytes   2:10 pm  Fri Jan 13 95
        scribble.reg        2,108 bytes   2:10 pm  Fri Jan 13 95
        scribdoc.cpp        9,269 bytes   2:10 pm  Fri Jan 13 95
        scribdoc.h          3,822 bytes   2:10 pm  Fri Jan 13 95
        scribfrm.cpp        1,419 bytes   2:10 pm  Fri Jan 13 95
        scribfrm.h          1,180 bytes   2:10 pm  Fri Jan 13 95
        scribitm.cpp        3,195 bytes   2:10 pm  Fri Jan 13 95
        scribitm.h          1,206 bytes   2:10 pm  Fri Jan 13 95
        scribvw.cpp        11,439 bytes   2:10 pm  Fri Jan 13 95
        scribvw.h           2,475 bytes   2:10 pm  Fri Jan 13 95
        stdafx.cpp            639 bytes   2:10 pm  Fri Jan 13 95
        stdafx.h              782 bytes   2:10 pm  Fri Jan 13 95

I:\VC152\MSVC15\MFC\SAMPLES\SCRIBBLE\STEP7\HLP
        .                   <DIR>      10:59 am  Mon Jun 10 96
        ..                  <DIR>      10:59 am  Mon Jun 10 96
        afxcore.rtf        47,264 bytes   2:10 pm  Fri Jan 13 95
        afxprint.rtf       11,360 bytes   2:10 pm  Fri Jan 13 95
        appexit.bmp         2,272 bytes   2:10 pm  Fri Jan 13 95
        bullet.bmp            144 bytes   2:10 pm  Fri Jan 13 95
        curarw2.bmp           320 bytes   2:10 pm  Fri Jan 13 95
        curarw4.bmp           576 bytes   2:10 pm  Fri Jan 13 95
        curhelp.bmp           512 bytes   2:10 pm  Fri Jan 13 95
        editcopy.bmp          512 bytes   2:10 pm  Fri Jan 13 95
        editcut.bmp           512 bytes   2:10 pm  Fri Jan 13 95
```

```
        editpast.bmp          512 bytes   2:10 pm  Fri Jan 13 95
        editundo.bmp          512 bytes   2:10 pm  Fri Jan 13 95
        filenew.bmp           576 bytes   2:10 pm  Fri Jan 13 95
        fileopen.bmp          576 bytes   2:10 pm  Fri Jan 13 95
        fileprnt.bmp          512 bytes   2:10 pm  Fri Jan 13 95
        filesave.bmp          512 bytes   2:10 pm  Fri Jan 13 95
        hlpsbar.bmp         7,168 bytes   2:10 pm  Fri Jan 13 95
        hlptbar.bmp         2,368 bytes   2:10 pm  Fri Jan 13 95
        pen.rtf             6,500 bytes   2:10 pm  Fri Jan 13 95
        scmax.bmp             512 bytes   2:10 pm  Fri Jan 13 95
        scmenu.bmp          2,144 bytes   2:10 pm  Fri Jan 13 95
        scmin.bmp             512 bytes   2:10 pm  Fri Jan 13 95
        scribble.hm           598 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\MFC\SAMPLES\SCRIBBLE\STEP7\RES
        .                     <DIR>       10:59 am  Mon Jun 10 96
        ..                    <DIR>       10:59 am  Mon Jun 10 96
        itoolbar.bmp          838 bytes   2:10 pm  Fri Jan 13 95
        scribble.ico          766 bytes   2:10 pm  Fri Jan 13 95
        scribble.rc2        1,646 bytes   2:10 pm  Fri Jan 13 95
        scribdoc.ico          766 bytes   2:10 pm  Fri Jan 13 95
        toolbar.bmp         1,318 bytes   2:10 pm  Fri Jan 13 95

I:\VC152\MSVC15\MFC\SAMPLES\SPEAKN
        .                     <DIR>       10:59 am  Mon Jun 10 96
        ..                    <DIR>       11:00 am  Mon Jun 10 96
        lessons.rc          1,283 bytes   10:12 pm  Mon Apr 10 95
        makefile              506 bytes   10:12 pm  Mon Apr 10 95
        RES                   <DIR>       10:59 am  Mon Jun 10 96
        resource.h          1,201 bytes   10:12 pm  Mon Apr 10 95
        sounds.rc             962 bytes   10:12 pm  Mon Apr 10 95
        speakn.clw          1,138 bytes   10:12 pm  Mon Apr 10 95
        speakn.cpp          8,415 bytes   10:12 pm  Mon Apr 10 95
        speakn.def            712 bytes   10:12 pm  Mon Apr 10 95
        speakn.h            2,665 bytes   10:12 pm  Mon Apr 10 95
        speakn.mak          2,084 bytes   10:12 pm  Mon Apr 10 95
        speakn.rc           3,384 bytes   10:12 pm  Mon Apr 10 95
        stdafx.cpp            512 bytes   10:12 pm  Mon Apr 10 95
        stdafx.h              670 bytes   10:12 pm  Mon Apr 10 95

I:\VC152\MSVC15\MFC\SAMPLES\SPEAKN\RES
        .                     <DIR>       10:59 am  Mon Jun 10 96
        ..                    <DIR>       10:59 am  Mon Jun 10 96
        correct.wav         9,622 bytes   10:12 pm  Mon Apr 10 95
        dog.dib             4,900 bytes   10:12 pm  Mon Apr 10 95
        dog.wav            25,144 bytes   10:12 pm  Mon Apr 10 95
        face01.ico            766 bytes   10:12 pm  Mon Apr 10 95
        face02.ico            766 bytes   10:12 pm  Mon Apr 10 95
        face03.ico            766 bytes   10:12 pm  Mon Apr 10 95
        face04.ico            766 bytes   10:12 pm  Mon Apr 10 95
        frog.dib            3,236 bytes   10:12 pm  Mon Apr 10 95
        frog.wav            5,772 bytes   10:12 pm  Mon Apr 10 95
        giveup.wav         18,258 bytes   10:12 pm  Mon Apr 10 95
        goodbye.wav         8,476 bytes   10:12 pm  Mon Apr 10 95
```

```
incorrec.wav      9,996 bytes  10:12 pm  Mon Apr 10 95
intro.dib        14,856 bytes  10:12 pm  Mon Apr 10 95
pig.dib           6,112 bytes  10:12 pm  Mon Apr 10 95
pig.wav          18,488 bytes  10:12 pm  Mon Apr 10 95
question.wav     10,430 bytes  10:12 pm  Mon Apr 10 95
replayd.bmp       1,014 bytes  10:12 pm  Mon Apr 10 95
replayf.bmp       1,014 bytes  10:12 pm  Mon Apr 10 95
replayu.bmp       1,014 bytes  10:12 pm  Mon Apr 10 95
speakn.ico          766 bytes  10:12 pm  Mon Apr 10 95
train.dib         8,492 bytes  10:12 pm  Mon Apr 10 95
train.wav        21,054 bytes  10:12 pm  Mon Apr 10 95
welcome.wav      18,664 bytes  10:12 pm  Mon Apr 10 95

I:\VC152\MSVC15\MFC\SAMPLES\SUPERPAD
.                   <DIR>      10:59 am  Mon Jun 10 96
..                  <DIR>      11:00 am  Mon Jun 10 96
aboutbox.cpp      6,170 bytes  10:13 pm  Mon Apr 10 95
aboutbox.h        2,300 bytes  10:13 pm  Mon Apr 10 95
ipframe.cpp       2,812 bytes  10:13 pm  Mon Apr 10 95
ipframe.h         1,245 bytes  10:13 pm  Mon Apr 10 95
linkitem.cpp      1,376 bytes  10:13 pm  Mon Apr 10 95
linkitem.h          878 bytes  10:13 pm  Mon Apr 10 95
mainfrm.cpp       3,804 bytes  10:13 pm  Mon Apr 10 95
mainfrm.h         1,036 bytes  10:13 pm  Mon Apr 10 95
makefile            631 bytes  10:13 pm  Mon Apr 10 95
paddoc.cpp        2,364 bytes  10:13 pm  Mon Apr 10 95
paddoc.h          1,258 bytes  10:13 pm  Mon Apr 10 95
padframe.cpp      2,031 bytes  10:13 pm  Mon Apr 10 95
padframe.h        1,133 bytes  10:13 pm  Mon Apr 10 95
paditem.cpp       5,503 bytes  10:13 pm  Mon Apr 10 95
paditem.h         1,251 bytes  10:13 pm  Mon Apr 10 95
padview.cpp      17,646 bytes  10:13 pm  Mon Apr 10 95
padview.h         2,824 bytes  10:13 pm  Mon Apr 10 95
pageset.cpp       4,215 bytes  10:13 pm  Mon Apr 10 95
pageset.h         1,764 bytes  10:13 pm  Mon Apr 10 95
RES                 <DIR>      10:59 am  Mon Jun 10 96
resource.h        2,214 bytes  10:13 pm  Mon Apr 10 95
stdafx.cpp          497 bytes  10:13 pm  Mon Apr 10 95
stdafx.h            653 bytes  10:13 pm  Mon Apr 10 95
superpad.clw      6,575 bytes  10:13 pm  Mon Apr 10 95
superpad.cpp      5,214 bytes  10:13 pm  Mon Apr 10 95
superpad.def        693 bytes  10:13 pm  Mon Apr 10 95
superpad.h        1,315 bytes  10:13 pm  Mon Apr 10 95
superpad.mak      3,641 bytes  10:13 pm  Mon Apr 10 95
superpad.rc      18,134 bytes  10:13 pm  Mon Apr 10 95
superpad.reg      1,979 bytes  10:13 pm  Mon Apr 10 95
tabstop.cpp       1,425 bytes  10:13 pm  Mon Apr 10 95
tabstop.h         1,122 bytes  10:13 pm  Mon Apr 10 95
waitcur.h           887 bytes  10:13 pm  Mon Apr 10 95

I:\VC152\MSVC15\MFC\SAMPLES\SUPERPAD\RES
.                   <DIR>      10:59 am  Mon Jun 10 96
..                  <DIR>      10:59 am  Mon Jun 10 96
itoolbar.bmp        718 bytes  10:13 pm  Mon Apr 10 95
```

```
        paddoc.ico            766 bytes   10:13 pm  Mon Apr 10 95
        superpad.ico          766 bytes   10:13 pm  Mon Apr 10 95
        toolbar.bmp         1,198 bytes   10:13 pm  Mon Apr 10 95

I:\VC152\MSVC15\MFC\SAMPLES\TEMPLDEF
        .                     <DIR>       10:59 am  Mon Jun 10 96
        ..                    <DIR>       11:00 am  Mon Jun 10 96
        afxcoll1.h          1,761 bytes   10:13 pm  Mon Apr 10 95
        afxcoll1.inl          469 bytes   10:13 pm  Mon Apr 10 95
        afxcoll2.h            285 bytes   10:13 pm  Mon Apr 10 95
        afxcoll2.inl          107 bytes   10:13 pm  Mon Apr 10 95
        array.ctt          13,528 bytes   10:13 pm  Mon Apr 10 95
        list.ctt           19,322 bytes   10:13 pm  Mon Apr 10 95
        makefile            1,204 bytes   10:13 pm  Mon Apr 10 95
        map.ctt            14,792 bytes   10:13 pm  Mon Apr 10 95
        map_s.ctt          14,439 bytes   10:13 pm  Mon Apr 10 95
        mkcoll.bat          3,640 bytes   10:13 pm  Mon Apr 10 95
        templdef.cpp       14,307 bytes   10:13 pm  Mon Apr 10 95
        templdef.mak        1,543 bytes   10:13 pm  Mon Apr 10 95

I:\VC152\MSVC15\MFC\SAMPLES\TRACER
        .                     <DIR>       10:59 am  Mon Jun 10 96
        ..                    <DIR>       11:00 am  Mon Jun 10 96
        makefile              483 bytes   10:13 pm  Mon Apr 10 95
        resource.h            756 bytes   10:13 pm  Mon Apr 10 95
        stdafx.h              611 bytes   10:13 pm  Mon Apr 10 95
        tracer.clw            832 bytes   10:13 pm  Mon Apr 10 95
        tracer.cpp          3,055 bytes   10:13 pm  Mon Apr 10 95
        tracer.def            603 bytes   10:13 pm  Mon Apr 10 95
        tracer.ico            766 bytes   10:13 pm  Mon Apr 10 95
        tracer.mak          1,905 bytes   10:13 pm  Mon Apr 10 95
        tracer.rc           2,447 bytes   10:13 pm  Mon Apr 10 95

I:\VC152\MSVC15\MFC\SAMPLES\TRACKER
        .                     <DIR>       11:00 am  Mon Jun 10 96
        ..                    <DIR>       11:00 am  Mon Jun 10 96
        dialogs.cpp         2,157 bytes   10:13 pm  Mon Apr 10 95
        dialogs.h           1,716 bytes   10:13 pm  Mon Apr 10 95
        mainfrm.cpp         3,135 bytes   10:13 pm  Mon Apr 10 95
        mainfrm.h           1,387 bytes   10:13 pm  Mon Apr 10 95
        makefile              535 bytes   10:13 pm  Mon Apr 10 95
        RES                   <DIR>       11:00 am  Mon Jun 10 96
        resource.h          1,129 bytes   10:13 pm  Mon Apr 10 95
        stdafx.cpp            644 bytes   10:13 pm  Mon Apr 10 95
        stdafx.h              760 bytes   10:13 pm  Mon Apr 10 95
        trackapp.cpp        4,068 bytes   10:13 pm  Mon Apr 10 95
        trackapp.h          1,266 bytes   10:13 pm  Mon Apr 10 95
        trackdoc.cpp        5,915 bytes   10:13 pm  Mon Apr 10 95
        trackdoc.h          1,947 bytes   10:13 pm  Mon Apr 10 95
        tracker.clw         3,481 bytes   10:13 pm  Mon Apr 10 95
        tracker.def           780 bytes   10:13 pm  Mon Apr 10 95
        tracker.mak         2,776 bytes   10:13 pm  Mon Apr 10 95
        tracker.rc         11,630 bytes   10:13 pm  Mon Apr 10 95
        trackvw.cpp         8,733 bytes   10:13 pm  Mon Apr 10 95
```

```
        trackvw.h         1,857 bytes   10:13 pm  Mon Apr 10 95

 I:\VC152\MSVC15\MFC\SAMPLES\TRACKER\RES
        .                   <DIR>       11:00 am  Mon Jun 10 96
        ..                  <DIR>       11:00 am  Mon Jun 10 96
        toolbar.bmp       1,918 bytes   10:13 pm  Mon Apr 10 95
        trackdoc.ico        768 bytes   10:13 pm  Mon Apr 10 95
        tracker.ico         766 bytes   10:13 pm  Mon Apr 10 95
        tracker.rc2       1,548 bytes   10:13 pm  Mon Apr 10 95

 I:\VC152\MSVC15\MFC\SAMPLES\VBCHART
        .                   <DIR>       11:00 am  Mon Jun 10 96
        ..                  <DIR>       11:00 am  Mon Jun 10 96
        chartdoc.cpp      4,568 bytes   10:13 pm  Mon Apr 10 95
        chartdoc.h        1,100 bytes   10:13 pm  Mon Apr 10 95
        chartvw.cpp      21,866 bytes   10:13 pm  Mon Apr 10 95
        chartvw.h         3,013 bytes   10:13 pm  Mon Apr 10 95
        gridentr.cpp     15,232 bytes   10:13 pm  Mon Apr 10 95
        gridentr.h        2,627 bytes   10:13 pm  Mon Apr 10 95
        mainfrm.cpp       4,375 bytes   10:13 pm  Mon Apr 10 95
        mainfrm.h         1,284 bytes   10:13 pm  Mon Apr 10 95
        makefile            547 bytes   10:13 pm  Mon Apr 10 95
        RES                 <DIR>       11:00 am  Mon Jun 10 96
        resource.h          706 bytes   10:13 pm  Mon Apr 10 95
        stdafx.cpp          497 bytes   10:13 pm  Mon Apr 10 95
        stdafx.h            673 bytes   10:13 pm  Mon Apr 10 95
        vbchart.clw       2,932 bytes   10:13 pm  Mon Apr 10 95
        vbchart.cpp       4,946 bytes   10:13 pm  Mon Apr 10 95
        vbchart.def         776 bytes   10:13 pm  Mon Apr 10 95
        vbchart.h         1,415 bytes   10:13 pm  Mon Apr 10 95
        vbchart.mak       2,642 bytes   10:13 pm  Mon Apr 10 95
        vbchart.rc        9,737 bytes   10:13 pm  Mon Apr 10 95

 I:\VC152\MSVC15\MFC\SAMPLES\VBCHART\RES
        .                   <DIR>       11:00 am  Mon Jun 10 96
        ..                  <DIR>       11:00 am  Mon Jun 10 96
        chartdoc.ico        766 bytes   10:13 pm  Mon Apr 10 95
        toolbar.bmp         838 bytes   10:13 pm  Mon Apr 10 95
        vbchart.ico         766 bytes   10:13 pm  Mon Apr 10 95

 I:\VC152\MSVC15\MFC\SAMPLES\VBCIRCLE
        .                   <DIR>       11:00 am  Mon Jun 10 96
        ..                  <DIR>       11:00 am  Mon Jun 10 96
        dialog.cpp        4,888 bytes   10:13 pm  Mon Apr 10 95
        dialog.h          1,547 bytes   10:13 pm  Mon Apr 10 95
        frame.cpp         3,148 bytes   10:13 pm  Mon Apr 10 95
        frame.h           1,013 bytes   10:13 pm  Mon Apr 10 95
        makefile            520 bytes   10:13 pm  Mon Apr 10 95
        resource.h          898 bytes   10:13 pm  Mon Apr 10 95
        stdafx.cpp          497 bytes   10:13 pm  Mon Apr 10 95
        stdafx.h            673 bytes   10:13 pm  Mon Apr 10 95
        vbcircle.clw      1,978 bytes   10:13 pm  Mon Apr 10 95
        vbcircle.cpp      2,558 bytes   10:13 pm  Mon Apr 10 95
        vbcircle.def        716 bytes   10:13 pm  Mon Apr 10 95
```

```
        vbcircle.h        1,348 bytes   10:13 pm  Mon Apr 10 95
        vbcircle.ico        766 bytes   10:13 pm  Mon Apr 10 95
        vbcircle.mak      2,349 bytes   10:13 pm  Mon Apr 10 95
        vbcircle.rc       7,897 bytes   10:13 pm  Mon Apr 10 95

I:\VC152\MSVC15\MFC\SAMPLES\VIEWEX
        .                   <DIR>       11:00 am  Mon Jun 10 96
        ..                  <DIR>       11:00 am  Mon Jun 10 96
        enterdlg.cpp      1,343 bytes   10:13 pm  Mon Apr 10 95
        enterdlg.h        1,021 bytes   10:13 pm  Mon Apr 10 95
        inputvw.cpp       7,316 bytes   10:13 pm  Mon Apr 10 95
        inputvw.h         1,630 bytes   10:13 pm  Mon Apr 10 95
        maindoc.cpp       1,994 bytes   10:13 pm  Mon Apr 10 95
        maindoc.h         1,189 bytes   10:13 pm  Mon Apr 10 95
        makefile            548 bytes   10:13 pm  Mon Apr 10 95
        resource.h        1,063 bytes   10:13 pm  Mon Apr 10 95
        simpvw.cpp        3,275 bytes   10:13 pm  Mon Apr 10 95
        simpvw.h          2,223 bytes   10:13 pm  Mon Apr 10 95
        splitter.cpp      4,320 bytes   10:13 pm  Mon Apr 10 95
        splitter.h        1,832 bytes   10:13 pm  Mon Apr 10 95
        stdafx.cpp          497 bytes   10:13 pm  Mon Apr 10 95
        stdafx.h            687 bytes   10:13 pm  Mon Apr 10 95
        viewex.clw        2,476 bytes   10:13 pm  Mon Apr 10 95
        viewex.cpp        4,318 bytes   10:13 pm  Mon Apr 10 95
        viewex.def          693 bytes   10:13 pm  Mon Apr 10 95
        viewex.h          1,297 bytes   10:13 pm  Mon Apr 10 95
        viewex.ico          766 bytes   10:13 pm  Mon Apr 10 95
        viewex.mak        2,759 bytes   10:13 pm  Mon Apr 10 95
        viewex.rc         4,727 bytes   10:13 pm  Mon Apr 10 95

I:\VC152\MSVC15\MFC\SRC
        .                   <DIR>       11:01 am  Mon Jun 10 96
        ..                  <DIR>       11:00 am  Mon Jun 10 96
        afx.ini           1,090 bytes   10:13 pm  Mon Apr 10 95
        afxabort.cpp        894 bytes   10:13 pm  Mon Apr 10 95
        afxasert.cpp      3,050 bytes   10:13 pm  Mon Apr 10 95
        afxdll.asm        3,539 bytes   10:13 pm  Mon Apr 10 95
        afxdll.obj          447 bytes   11:15 pm  Wed Apr 26 95
        afxdlld.obj         665 bytes   11:08 pm  Wed Apr 26 95
        afxinl1.cpp       1,330 bytes   10:13 pm  Mon Apr 10 95
        afxinl2.cpp         828 bytes   10:13 pm  Mon Apr 10 95
        afxinl3.cpp         828 bytes   10:13 pm  Mon Apr 10 95
        afxmem.cpp       24,148 bytes   10:13 pm  Mon Apr 10 95
        afxtrace.cpp      4,680 bytes   10:13 pm  Mon Apr 10 95
        afxver.cpp          998 bytes   10:13 pm  Mon Apr 10 95
        appcore.cpp      14,135 bytes   10:13 pm  Mon Apr 10 95
        appdata.cpp       1,103 bytes   10:13 pm  Mon Apr 10 95
        appdlg.cpp        7,075 bytes   10:13 pm  Mon Apr 10 95
        appgray.cpp       3,773 bytes   10:13 pm  Mon Apr 10 95
        apphelp.cpp       2,565 bytes   10:13 pm  Mon Apr 10 95
        apphelpx.cpp      1,953 bytes    6:02 pm  Mon Apr 24 95
        appinit.cpp       6,807 bytes   10:13 pm  Mon Apr 10 95
        appprnt.cpp       3,966 bytes   10:13 pm  Mon Apr 10 95
        appterm.cpp       3,200 bytes   10:13 pm  Mon Apr 10 95
```

```
appui.cpp        14,898 bytes   9:57 pm  Thu Apr 20 95
appui2.cpp        3,806 bytes  10:13 pm  Mon Apr 10 95
arccore.cpp       9,385 bytes  10:13 pm  Mon Apr 10 95
arcex.cpp         2,196 bytes  10:13 pm  Mon Apr 10 95
arcobj.cpp        6,029 bytes  10:13 pm  Mon Apr 10 95
array_b.cpp       6,972 bytes  10:13 pm  Mon Apr 10 95
array_d.cpp       7,011 bytes  10:13 pm  Mon Apr 10 95
array_o.cpp       7,026 bytes  10:13 pm  Mon Apr 10 95
array_p.cpp       6,625 bytes  10:13 pm  Mon Apr 10 95
array_s.cpp       7,665 bytes  10:13 pm  Mon Apr 10 95
array_u.cpp       6,619 bytes  10:13 pm  Mon Apr 10 95
array_w.cpp       6,972 bytes  10:13 pm  Mon Apr 10 95
auxdata.cpp       5,925 bytes  10:13 pm  Mon Apr 10 95
auxdata.h         6,773 bytes  10:13 pm  Mon Apr 10 95
auxvars.h         3,285 bytes  10:13 pm  Mon Apr 10 95
barcore.cpp      26,801 bytes  10:13 pm  Mon Apr 10 95
bardlg.cpp        2,908 bytes  10:13 pm  Mon Apr 10 95
bartool.cpp      27,115 bytes  10:13 pm  Mon Apr 10 95
cmdtarg.cpp      14,042 bytes  10:13 pm  Mon Apr 10 95
ctlstate.cpp      7,390 bytes  10:13 pm  Mon Apr 10 95
dbcore.cpp       71,449 bytes  10:13 pm  Mon Apr 10 95
dbflt.cpp         6,578 bytes  10:13 pm  Mon Apr 10 95
dbrfx.cpp        47,991 bytes  10:13 pm  Mon Apr 10 95
dbview.cpp       14,221 bytes  10:13 pm  Mon Apr 10 95
dcmeta.cpp       10,086 bytes  10:13 pm  Mon Apr 10 95
dcprev.cpp       25,362 bytes  10:13 pm  Mon Apr 10 95
dlgclr.cpp        3,692 bytes  10:13 pm  Mon Apr 10 95
dlgcore.cpp      15,966 bytes  10:13 pm  Mon Apr 10 95
dlgdata.cpp      14,514 bytes  10:13 pm  Mon Apr 10 95
dlgfile.cpp       8,175 bytes  10:13 pm  Mon Apr 10 95
dlgfloat.cpp      3,698 bytes  10:13 pm  Mon Apr 10 95
dlgfnt.cpp        3,404 bytes  10:13 pm  Mon Apr 10 95
dlgfr.cpp         3,872 bytes  10:13 pm  Mon Apr 10 95
dlgprnt.cpp       6,967 bytes  10:13 pm  Mon Apr 10 95
dlgprop.cpp      59,078 bytes   6:02 pm  Mon Apr 24 95
dllctl.cpp       17,701 bytes  10:13 pm  Mon Apr 10 95
dlldb.cpp         1,215 bytes  10:13 pm  Mon Apr 10 95
dllinit.cpp      18,085 bytes  10:13 pm  Mon Apr 10 95
dllnet.cpp        1,216 bytes  10:13 pm  Mon Apr 10 95
dllole.cpp        1,449 bytes  10:13 pm  Mon Apr 10 95
doccore.cpp      17,691 bytes  10:13 pm  Mon Apr 10 95
docmapi.cpp       6,204 bytes  10:13 pm  Mon Apr 10 95
docmulti.cpp      5,955 bytes  10:13 pm  Mon Apr 10 95
docsingl.cpp      6,570 bytes  10:13 pm  Mon Apr 10 95
doctempl.cpp     11,625 bytes  10:13 pm  Mon Apr 10 95
dumpcont.cpp      4,855 bytes  10:13 pm  Mon Apr 10 95
dumpflt.cpp       1,112 bytes  10:13 pm  Mon Apr 10 95
dumpinit.cpp      2,493 bytes  10:13 pm  Mon Apr 10 95
dumpout.cpp       2,566 bytes  10:13 pm  Mon Apr 10 95
elements.h        2,441 bytes  10:13 pm  Mon Apr 10 95
except.cpp        8,401 bytes  10:13 pm  Mon Apr 10 95
filecore.cpp     11,485 bytes  10:13 pm  Mon Apr 10 95
filemem.cpp       6,182 bytes  10:13 pm  Mon Apr 10 95
fileshrd.cpp      2,655 bytes  10:13 pm  Mon Apr 10 95
```

```
filest.cpp        5,163 bytes   10:13 pm  Mon Apr 10 95
filetxt.cpp       7,750 bytes   10:13 pm  Mon Apr 10 95
filex.cpp         6,172 bytes   10:13 pm  Mon Apr 10 95
list_o.cpp        9,677 bytes   10:13 pm  Mon Apr 10 95
list_p.cpp        9,212 bytes   10:13 pm  Mon Apr 10 95
list_s.cpp        9,997 bytes   10:13 pm  Mon Apr 10 95
lvbxcw.lib       64,585 bytes   10:34 pm  Wed Apr 26 95
lvbxew.lib       64,579 bytes   10:36 pm  Wed Apr 26 95
makefile         12,498 bytes   10:13 pm  Mon Apr 10 95
map_pp.cpp        7,428 bytes   10:13 pm  Mon Apr 10 95
map_pw.cpp        7,448 bytes   10:13 pm  Mon Apr 10 95
map_so.cpp        8,686 bytes   10:13 pm  Mon Apr 10 95
map_sp.cpp        8,007 bytes   10:13 pm  Mon Apr 10 95
map_ss.cpp        8,906 bytes   10:13 pm  Mon Apr 10 95
map_wo.cpp        8,119 bytes   10:13 pm  Mon Apr 10 95
map_wp.cpp        7,445 bytes   10:13 pm  Mon Apr 10 95
mfc250.def       95,936 bytes   10:07 pm  Wed Apr 26 95
mfc250d.def     150,839 bytes   10:07 pm  Wed Apr 26 95
mfcd250.def       7,859 bytes   10:13 pm  Mon Apr 10 95
mfcd250d.def      9,875 bytes   10:13 pm  Mon Apr 10 95
mfcdb.mak         2,508 bytes   10:13 pm  Mon Apr 10 95
mfcdb.rc          3,030 bytes   10:07 pm  Wed Apr 26 95
mfcdll.clw        4,148 bytes   10:13 pm  Mon Apr 10 95
mfcdll.mak        3,643 bytes   10:13 pm  Mon Apr 10 95
mfcdll.rc         3,244 bytes   10:07 pm  Wed Apr 26 95
mfcn250.def       4,723 bytes   10:13 pm  Mon Apr 10 95
mfcn250d.def      5,877 bytes   10:13 pm  Mon Apr 10 95
mfcnet.mak        2,470 bytes   10:13 pm  Mon Apr 10 95
mfcnet.rc         2,976 bytes   10:07 pm  Wed Apr 26 95
mfco250.def      53,719 bytes   10:13 pm  Mon Apr 10 95
mfco250d.def     62,114 bytes   10:13 pm  Mon Apr 10 95
mfcocx.mak        2,712 bytes   10:13 pm  Mon Apr 10 95
mfcocx1.mak       2,897 bytes   10:13 pm  Mon Apr 10 95
mfcocxd.mak       2,732 bytes    1:10 am  Thu Apr 27 95
mfcole.mak        2,513 bytes   10:13 pm  Mon Apr 10 95
mfcole.rc         3,063 bytes   10:07 pm  Wed Apr 26 95
mvbxcw.lib       61,513 bytes   10:34 pm  Wed Apr 26 95
objcore.cpp       6,907 bytes   10:13 pm  Mon Apr 10 95
ocd25.def         7,854 bytes    3:54 pm  Wed May 10 95
ocd25d.def        9,871 bytes   10:13 pm  Mon Apr 10 95
olebar.cpp        4,944 bytes   10:13 pm  Mon Apr 10 95
olecallf.obj        476 bytes   10:41 pm  Wed Apr 26 95
olecalln.obj        228 bytes   11:53 pm  Wed Apr 26 95
olecall_.asm      3,178 bytes   10:13 pm  Mon Apr 10 95
olecli1.cpp      56,215 bytes   10:13 pm  Mon Apr 10 95
olecli2.cpp      39,800 bytes   10:13 pm  Mon Apr 10 95
olecli3.cpp      15,846 bytes   10:13 pm  Mon Apr 10 95
oledata.cpp         908 bytes   10:13 pm  Mon Apr 10 95
oledisp1.cpp     37,996 bytes   10:07 pm  Wed Apr 26 95
oledisp2.cpp     11,312 bytes   10:13 pm  Mon Apr 10 95
oledlgs1.cpp     34,791 bytes   10:13 pm  Mon Apr 10 95
oledlgs2.cpp      4,738 bytes   10:13 pm  Mon Apr 10 95
oledll.cpp        1,884 bytes   10:13 pm  Mon Apr 10 95
oledobj1.cpp      7,665 bytes   10:13 pm  Mon Apr 10 95
```

```
oledobj2.cpp    21,365 bytes  10:13 pm  Mon Apr 10 95
oledoc1.cpp     20,668 bytes  10:13 pm  Mon Apr 10 95
oledoc2.cpp      1,964 bytes  10:13 pm  Mon Apr 10 95
oledrop1.cpp     6,975 bytes  10:13 pm  Mon Apr 10 95
oledrop2.cpp    13,088 bytes  10:13 pm  Mon Apr 10 95
oleenum.cpp      5,320 bytes  10:13 pm  Mon Apr 10 95
olefact.cpp      9,572 bytes  10:13 pm  Mon Apr 10 95
oleimpl.h        8,710 bytes  10:13 pm  Mon Apr 10 95
oleinit.cpp     13,722 bytes  10:13 pm  Mon Apr 10 95
oleipfrm.cpp    13,616 bytes  10:13 pm  Mon Apr 10 95
olelink.cpp     18,592 bytes  10:13 pm  Mon Apr 10 95
olelock1.cpp     1,885 bytes  10:13 pm  Mon Apr 10 95
olelock2.cpp     1,607 bytes  10:13 pm  Mon Apr 10 95
olemisc.cpp     21,498 bytes  10:13 pm  Mon Apr 10 95
olemsgf.cpp      8,653 bytes  10:13 pm  Mon Apr 10 95
olereg.cpp       7,130 bytes  10:13 pm  Mon Apr 10 95
olestrm.cpp     10,875 bytes  10:13 pm  Mon Apr 10 95
olesvr1.cpp     73,123 bytes  10:13 pm  Mon Apr 10 95
olesvr2.cpp     37,374 bytes  10:13 pm  Mon Apr 10 95
oletsvr.cpp      3,225 bytes  10:13 pm  Mon Apr 10 95
oleui1.cpp       4,616 bytes  10:13 pm  Mon Apr 10 95
oleui2.cpp       4,236 bytes  10:13 pm  Mon Apr 10 95
oleunk.cpp      11,306 bytes  10:13 pm  Mon Apr 10 95
penctrl.cpp      2,175 bytes  10:13 pm  Mon Apr 10 95
plex.cpp         1,226 bytes  10:13 pm  Mon Apr 10 95
plex.h           1,003 bytes  10:13 pm  Mon Apr 10 95
readme.txt       8,442 bytes  10:13 pm  Mon Apr 10 95
sockcore.cpp    25,770 bytes  10:13 pm  Mon Apr 10 95
stdafx.h         2,013 bytes  10:13 pm  Mon Apr 10 95
strcore1.cpp     9,201 bytes  10:13 pm  Mon Apr 10 95
strcore2.cpp     1,920 bytes  10:13 pm  Mon Apr 10 95
strex.cpp        9,779 bytes  10:07 pm  Wed Apr 26 95
timecore.cpp     5,823 bytes  10:13 pm  Mon Apr 10 95
tracedat.h       3,510 bytes  10:13 pm  Mon Apr 10 95
trckrect.cpp    20,520 bytes  10:13 pm  Mon Apr 10 95
validadd.cpp     4,485 bytes  10:13 pm  Mon Apr 10 95
vbctrl.cpp       8,835 bytes  10:13 pm  Mon Apr 10 95
vbctrl.h         1,783 bytes  10:13 pm  Mon Apr 10 95
vbddx.cpp        5,537 bytes  10:13 pm  Mon Apr 10 95
vbddxf.cpp       1,432 bytes  10:13 pm  Mon Apr 10 95
vbfloat.cpp      1,610 bytes  10:13 pm  Mon Apr 10 95
viewcore.cpp    11,676 bytes  10:13 pm  Mon Apr 10 95
viewedit.cpp    30,954 bytes  10:13 pm  Mon Apr 10 95
viewform.cpp     6,454 bytes  10:13 pm  Mon Apr 10 95
viewprev.cpp    29,450 bytes  10:13 pm  Mon Apr 10 95
viewprnt.cpp     7,300 bytes  10:13 pm  Mon Apr 10 95
viewscrl.cpp    22,601 bytes  10:13 pm  Mon Apr 10 95
winbtn.cpp       4,834 bytes  10:13 pm  Mon Apr 10 95
wincore.cpp     49,571 bytes  10:13 pm  Mon Apr 10 95
winctrl.cpp      7,676 bytes  10:13 pm  Mon Apr 10 95
winfrm.cpp      42,704 bytes  10:13 pm  Mon Apr 10 95
winfrmx.cpp      9,747 bytes   6:02 pm  Mon Apr 24 95
wingdi.cpp      24,295 bytes  10:13 pm  Mon Apr 10 95
wingdix.cpp      3,501 bytes  10:13 pm  Mon Apr 10 95
```

```
        winhand.cpp       6,253 bytes   10:13 pm  Mon Apr 10 95
        winhand_.h        3,273 bytes   10:13 pm  Mon Apr 10 95
        winmain.cpp       3,211 bytes   10:13 pm  Mon Apr 10 95
        winmdi.cpp       27,071 bytes   10:13 pm  Mon Apr 10 95
        winmenu.cpp       2,531 bytes   10:13 pm  Mon Apr 10 95
        winsplit.cpp     51,111 bytes   10:13 pm  Mon Apr 10 95
        winstr.cpp        2,947 bytes    9:57 pm  Thu Apr 20 95
        winutil.cpp       7,800 bytes   10:13 pm  Mon Apr 10 95

I:\VC152\MSVC15\MSQUERY
        .                   <DIR>       11:02 am  Mon Jun 10 96
        ..                  <DIR>       11:19 am  Mon Jun 10 96
        ccapi200.dll     46,400 bytes    2:00 pm  Fri Sep 16 94
        ccard200.exe    127,424 bytes    2:00 pm  Fri Sep 16 94
        cirdeb.inf           64 bytes    2:00 pm  Fri Sep 16 94
        cuecard2.dll      2,944 bytes    2:00 pm  Fri Sep 16 94
        msquery.cue     209,228 bytes    2:00 pm  Fri Sep 16 94
        msquery.exe     622,464 bytes    2:00 pm  Fri Sep 16 94
        msquery.hlp     437,165 bytes    2:00 pm  Fri Sep 16 94
        NWIND               <DIR>       11:02 am  Mon Jun 10 96
        qryintl.dll      35,391 bytes    2:00 pm  Fri Sep 16 94
        winrfs.dll       14,509 bytes    2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\MSQUERY\NWIND
        .                   <DIR>       11:02 am  Mon Jun 10 96
        ..                  <DIR>       11:02 am  Mon Jun 10 96
        cirdeb.inf           64 bytes    2:00 pm  Fri Sep 16 94
        customer.dbf     13,364 bytes    2:00 pm  Fri Sep 16 94
        customer.mdx      4,096 bytes    2:00 pm  Fri Sep 16 94
        employee.dbf      4,582 bytes    2:00 pm  Fri Sep 16 94
        employee.mdx      4,096 bytes    2:00 pm  Fri Sep 16 94
        orddtail.dbf      1,754 bytes    2:00 pm  Fri Sep 16 94
        orddtail.mdx      4,096 bytes    2:00 pm  Fri Sep 16 94
        orders.dbf        7,885 bytes    2:00 pm  Fri Sep 16 94
        orders.mdx        4,096 bytes    2:00 pm  Fri Sep 16 94
        product.dbf       5,729 bytes    2:00 pm  Fri Sep 16 94
        product.mdx       4,096 bytes    2:00 pm  Fri Sep 16 94
        supplier.dbf      6,621 bytes    2:00 pm  Fri Sep 16 94
        supplier.mdx      4,096 bytes    2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\NODEBUG
        .                   <DIR>       11:02 am  Mon Jun 10 96
        ..                  <DIR>       11:19 am  Mon Jun 10 96
        compobj.dll     108,544 bytes    2:00 pm  Fri Sep 16 94
        compobj.sym       4,500 bytes    2:00 pm  Fri Sep 16 94
        ctl3d.dll        26,768 bytes    1:43 am  Fri Jul 14 95
        ctl3dv2.dll      27,632 bytes    1:43 am  Fri Jul 14 95
        gdi.sym           6,004 bytes    2:00 pm  Fri Sep 16 94
        krnl286.sym       4,308 bytes    2:00 pm  Fri Sep 16 94
        krnl386.sym       4,340 bytes    2:00 pm  Fri Sep 16 94
        mmsystem.sym      5,284 bytes    2:00 pm  Fri Sep 16 94
        ole2.dll        302,592 bytes    2:00 pm  Fri Sep 16 94
        ole2.sym          4,068 bytes    2:00 pm  Fri Sep 16 94
        ole2conv.dll     57,328 bytes    2:00 pm  Fri Sep 16 94
```

```
        ole2conv.sym      7,748 bytes   2:00 pm  Fri Sep 16 94
        ole2disp.dll    164,832 bytes   2:00 pm  Fri Sep 16 94
        ole2disp.sym     44,916 bytes   2:00 pm  Fri Sep 16 94
        ole2nls.dll     150,976 bytes   2:00 pm  Fri Sep 16 94
        ole2nls.sym       4,068 bytes   2:00 pm  Fri Sep 16 94
        ole2prox.dll     51,712 bytes   2:00 pm  Fri Sep 16 94
        ole2prox.sym        164 bytes   2:00 pm  Fri Sep 16 94
        storage.dll     157,696 bytes   2:00 pm  Fri Sep 16 94
        typelib.dll     177,216 bytes   2:00 pm  Fri Sep 16 94
        typelib.sym      43,108 bytes   2:00 pm  Fri Sep 16 94
        user.sym          9,268 bytes   2:00 pm  Fri Sep 16 94
        WFW311              <DIR>       11:02 am  Mon Jun 10 96

I:\VC152\MSVC15\NODEBUG\WFW311
        .                   <DIR>       11:02 am  Mon Jun 10 96
        ..                  <DIR>       11:02 am  Mon Jun 10 96
        gdi.sym           6,004 bytes   2:00 pm  Fri Sep 16 94
        krnl386.sym       4,340 bytes   2:00 pm  Fri Sep 16 94
        user.sym          9,268 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\OLE2
        .                   <DIR>       11:05 am  Mon Jun 10 96
        ..                  <DIR>       11:19 am  Mon Jun 10 96
        DOCS                <DIR>       11:05 am  Mon Jun 10 96
        REG                 <DIR>       11:05 am  Mon Jun 10 96
        SAMPLES             <DIR>       11:11 am  Mon Jun 10 96

I:\VC152\MSVC15\OLE2\DOCS
        .                   <DIR>       11:05 am  Mon Jun 10 96
        ..                  <DIR>       11:05 am  Mon Jun 10 96
        ole2spec.doc 8,731,328 bytes    2:00 pm  Fri Sep 16 94
        olemgmt.doc  1,329,163 bytes    2:00 pm  Fri Sep 16 94
        oleobjec.doc 1,387,069 bytes    2:00 pm  Fri Sep 16 94
        oletech.doc    341,920 bytes    2:00 pm  Fri Sep 16 94
        relnotes.wri    23,552 bytes    2:00 pm  Fri Sep 16 94
        remoting.doc   150,805 bytes    2:00 pm  Fri Sep 16 94
        TECHNOTE            <DIR>       11:05 am  Mon Jun 10 96

I:\VC152\MSVC15\OLE2\DOCS\TECHNOTE
        .                   <DIR>       11:05 am  Mon Jun 10 96
        ..                  <DIR>       11:05 am  Mon Jun 10 96
        cfperf.doc       34,031 bytes   2:00 pm  Fri Sep 16 94
        cshelp.doc       49,139 bytes   2:00 pm  Fri Sep 16 94
        extents.doc       6,378 bytes   2:00 pm  Fri Sep 16 94
        inplace.doc      43,984 bytes   2:00 pm  Fri Sep 16 94
        olelcnvt.doc      5,405 bytes   2:00 pm  Fri Sep 16 94
        ole2fmt.doc      15,891 bytes   2:00 pm  Fri Sep 16 94
        ole2tips.doc      5,384 bytes   2:00 pm  Fri Sep 16 94
        palette.doc       8,449 bytes   2:00 pm  Fri Sep 16 94
        rpcitf.doc       11,573 bytes   2:00 pm  Fri Sep 16 94
        self.doc         21,882 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\OLE2\REG
        .                   <DIR>       11:05 am  Mon Jun 10 96
```

```
..                      <DIR>       11:05 am  Mon Jun 10 96
ole2.reg           24,606 bytes     2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\OLE2\SAMPLES
.                       <DIR>       11:11 am  Mon Jun 10 96
..                      <DIR>       11:05 am  Mon Jun 10 96
BIN                     <DIR>       11:07 am  Mon Jun 10 96
BTTNCUR                 <DIR>       11:07 am  Mon Jun 10 96
DEFO2V                  <DIR>       11:07 am  Mon Jun 10 96
DFVIEW                  <DIR>       11:07 am  Mon Jun 10 96
DISPCALC                <DIR>       11:07 am  Mon Jun 10 96
DISPDEMO                <DIR>       11:07 am  Mon Jun 10 96
DLLSRV                  <DIR>       11:08 am  Mon Jun 10 96
DSPCALC2                <DIR>       11:08 am  Mon Jun 10 96
GIZMOBAR                <DIR>       11:08 am  Mon Jun 10 96
HANDLER                 <DIR>       11:08 am  Mon Jun 10 96
HELLO                   <DIR>       11:08 am  Mon Jun 10 96
ILCKBYTS                <DIR>       11:08 am  Mon Jun 10 96
INCLUDE                 <DIR>       11:08 am  Mon Jun 10 96
LAUNCHER                <DIR>       11:08 am  Mon Jun 10 96
LIB                     <DIR>       11:09 am  Mon Jun 10 96
OLE2UI                  <DIR>       11:10 am  Mon Jun 10 96
OUTLINE                 <DIR>       11:11 am  Mon Jun 10 96
PSSAMP                  <DIR>       11:11 am  Mon Jun 10 96
readme.txt         1,431 bytes      2:00 pm  Fri Sep 16 94
SIMPCNTR                <DIR>       11:11 am  Mon Jun 10 96
SIMPDND                 <DIR>       11:11 am  Mon Jun 10 96
SIMPSVR                 <DIR>       11:11 am  Mon Jun 10 96
SPOLY                   <DIR>       11:11 am  Mon Jun 10 96
SPOLY2                  <DIR>       11:11 am  Mon Jun 10 96
SUMINFO                 <DIR>       11:11 am  Mon Jun 10 96
TIBROWSE                <DIR>       11:11 am  Mon Jun 10 96
XSERVER                 <DIR>       11:11 am  Mon Jun 10 96

I:\VC152\MSVC15\OLE2\SAMPLES\BIN
.                       <DIR>       11:07 am  Mon Jun 10 96
..                      <DIR>       11:11 am  Mon Jun 10 96
bttncur.dll        16,912 bytes     2:00 pm  Fri Sep 16 94
cl2test.exe     1,366,984 bytes     2:00 pm  Fri Sep 16 94
cl2test.reg           313 bytes     2:00 pm  Fri Sep 16 94
cl2test.sym        47,588 bytes     2:00 pm  Fri Sep 16 94
cntroutl.exe      497,212 bytes     2:00 pm  Fri Sep 16 94
ctl3d.dll          26,768 bytes     1:43 am  Fri Jul 14 95
ctl3d.sym           1,892 bytes     2:00 pm  Fri Sep 16 94
ctl3dv2.dll        27,632 bytes     1:43 am  Fri Jul 14 95
ctl3dv2.sym         1,892 bytes     2:00 pm  Fri Sep 16 94
dataview.dll      108,992 bytes     2:00 pm  Fri Sep 16 94
dataview.reg        1,629 bytes     2:00 pm  Fri Sep 16 94
dbghndlr.dll      109,056 bytes     2:00 pm  Fri Sep 16 94
dispcalc.exe        9,680 bytes     2:00 pm  Fri Sep 16 94
dispcalc.reg        1,004 bytes     2:00 pm  Fri Sep 16 94
dispdemo.exe       13,792 bytes     2:00 pm  Fri Sep 16 94
dspcalc2.exe       11,776 bytes     2:00 pm  Fri Sep 16 94
dspcalc2.reg        1,979 bytes     2:00 pm  Fri Sep 16 94
```

```
dspcalc2.tlb      3,329 bytes   2:00 pm  Fri Sep 16 94
dvlaunch.exe     41,904 bytes   2:00 pm  Fri Sep 16 94
dvlaunch.reg        245 bytes   2:00 pm  Fri Sep 16 94
gizmobar.dll     10,592 bytes   2:00 pm  Fri Sep 16 94
hello.exe        10,240 bytes   2:00 pm  Fri Sep 16 94
hello.reg         2,164 bytes   2:00 pm  Fri Sep 16 94
hello.tlb         2,519 bytes   2:00 pm  Fri Sep 16 94
icntrotl.exe    535,696 bytes   2:00 pm  Fri Sep 16 94
ilb.reg             277 bytes   2:00 pm  Fri Sep 16 94
ilbcntr.exe      65,952 bytes   2:00 pm  Fri Sep 16 94
ilbdll.dll       10,112 bytes   2:00 pm  Fri Sep 16 94
ilbsvr.exe       65,984 bytes   2:00 pm  Fri Sep 16 94
installr.exe     33,120 bytes   2:00 pm  Fri Sep 16 94
isvrotl.exe     500,504 bytes   2:00 pm  Fri Sep 16 94
outline.exe     296,580 bytes   2:00 pm  Fri Sep 16 94
outline.reg      11,884 bytes   2:00 pm  Fri Sep 16 94
outlui.dll      146,976 bytes   2:00 pm  Fri Sep 16 94
outlui.sym       13,908 bytes   2:00 pm  Fri Sep 16 94
prgidmon.dll     10,048 bytes   2:00 pm  Fri Sep 16 94
prgidmon.reg        198 bytes   2:00 pm  Fri Sep 16 94
pssamp.dll        8,288 bytes   2:00 pm  Fri Sep 16 94
pssamp.reg          807 bytes   2:00 pm  Fri Sep 16 94
realtime.exe    100,112 bytes   2:00 pm  Fri Sep 16 94
simpcntr.exe     98,832 bytes   2:00 pm  Fri Sep 16 94
simpdnd.exe     102,464 bytes   2:00 pm  Fri Sep 16 94
simpsvr.exe      91,824 bytes   2:00 pm  Fri Sep 16 94
simpsvr.reg       1,464 bytes   2:00 pm  Fri Sep 16 94
spoly.exe        20,688 bytes   2:00 pm  Fri Sep 16 94
spoly.reg           984 bytes   2:00 pm  Fri Sep 16 94
spoly2.exe       19,952 bytes   2:00 pm  Fri Sep 16 94
spoly2.reg          994 bytes   2:00 pm  Fri Sep 16 94
sr2test.exe  1,243,424 bytes   2:00 pm  Fri Sep 16 94
sr2test.reg       5,348 bytes   2:00 pm  Fri Sep 16 94
sr2test.sym      40,884 bytes   2:00 pm  Fri Sep 16 94
suminfo.exe      57,056 bytes   2:00 pm  Fri Sep 16 94
svroutl.exe     469,208 bytes   2:00 pm  Fri Sep 16 94
xserver.dll      68,464 bytes   2:00 pm  Fri Sep 16 94
xserver.reg       4,421 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\OLE2\SAMPLES\BTTNCUR
.                  <DIR>       11:07 am  Mon Jun 10 96
..                 <DIR>       11:11 am  Mon Jun 10 96
bttncur.c        29,845 bytes   2:00 pm  Fri Sep 16 94
bttncur.def         486 bytes   2:00 pm  Fri Sep 16 94
bttncur.h         7,557 bytes   2:00 pm  Fri Sep 16 94
bttncur.mak       2,153 bytes   2:00 pm  Fri Sep 16 94
bttncur.rc        1,487 bytes   2:00 pm  Fri Sep 16 94
bttncur.rcv       1,485 bytes   2:00 pm  Fri Sep 16 94
bttncuri.h        2,337 bytes   2:00 pm  Fri Sep 16 94
cursors.c         4,535 bytes   2:00 pm  Fri Sep 16 94
DEMO               <DIR>       11:07 am  Mon Jun 10 96
makefile          3,254 bytes   2:00 pm  Fri Sep 16 94
RES                <DIR>       11:07 am  Mon Jun 10 96
```

I:\VC152\MSVC15\OLE2\SAMPLES\BTTNCUR\DEMO

.	<DIR>	11:07	am	Mon	Jun	10 96
..	<DIR>	11:07	am	Mon	Jun	10 96
appim120.bmp	670 bytes	2:00	pm	Fri	Sep	16 94
appim72.bmp	294 bytes	2:00	pm	Fri	Sep	16 94
appim96.bmp	358 bytes	2:00	pm	Fri	Sep	16 94
bcdemo.c	6,641 bytes	2:00	pm	Fri	Sep	16 94
bcdemo.def	310 bytes	2:00	pm	Fri	Sep	16 94
bcdemo.h	1,294 bytes	2:00	pm	Fri	Sep	16 94
bcdemo.ico	1,086 bytes	2:00	pm	Fri	Sep	16 94
bcdemo.mak	2,058 bytes	2:00	pm	Fri	Sep	16 94
bcdemo.rc	1,927 bytes	2:00	pm	Fri	Sep	16 94
init.c	3,180 bytes	2:00	pm	Fri	Sep	16 94
makefile	1,943 bytes	2:00	pm	Fri	Sep	16 94

I:\VC152\MSVC15\OLE2\SAMPLES\BTTNCUR\RES

.	<DIR>	11:07	am	Mon	Jun	10 96
..	<DIR>	11:07	am	Mon	Jun	10 96
harrows.cur	326 bytes	2:00	pm	Fri	Sep	16 94
help.cur	326 bytes	2:00	pm	Fri	Sep	16 94
larrows.cur	326 bytes	2:00	pm	Fri	Sep	16 94
magnify.cur	326 bytes	2:00	pm	Fri	Sep	16 94
neswarrs.cur	326 bytes	2:00	pm	Fri	Sep	16 94
nodrop.cur	326 bytes	2:00	pm	Fri	Sep	16 94
nwsearrs.cur	326 bytes	2:00	pm	Fri	Sep	16 94
rarrow.cur	326 bytes	2:00	pm	Fri	Sep	16 94
sarrows.cur	326 bytes	2:00	pm	Fri	Sep	16 94
sizebarh.cur	326 bytes	2:00	pm	Fri	Sep	16 94
sizebarv.cur	326 bytes	2:00	pm	Fri	Sep	16 94
splith.cur	326 bytes	2:00	pm	Fri	Sep	16 94
splitv.cur	326 bytes	2:00	pm	Fri	Sep	16 94
stdim120.bmp	2,602 bytes	2:00	pm	Fri	Sep	16 94
stdim72.bmp	910 bytes	2:00	pm	Fri	Sep	16 94
stdim96.bmp	1,198 bytes	2:00	pm	Fri	Sep	16 94
tabletop.cur	326 bytes	2:00	pm	Fri	Sep	16 94
varrows.cur	326 bytes	2:00	pm	Fri	Sep	16 94

I:\VC152\MSVC15\OLE2\SAMPLES\DEFO2V

.	<DIR>	11:07	am	Mon	Jun	10 96
..	<DIR>	11:11	am	Mon	Jun	10 96
debug.h	1,201 bytes	2:00	pm	Fri	Sep	16 94
defo2v.cpp	859 bytes	2:00	pm	Fri	Sep	16 94
defo2v.def	423 bytes	2:00	pm	Fri	Sep	16 94
defo2v.h	2,154 bytes	2:00	pm	Fri	Sep	16 94
defo2v.mak	2,775 bytes	2:00	pm	Fri	Sep	16 94
defo2v.rc	3,486 bytes	2:00	pm	Fri	Sep	16 94
defo2v.rc2	1,657 bytes	2:00	pm	Fri	Sep	16 94
idataobj.cpp	13,833 bytes	2:00	pm	Fri	Sep	16 94
idataobj.h	1,826 bytes	2:00	pm	Fri	Sep	16 94
idisp.cpp	45,093 bytes	2:00	pm	Fri	Sep	16 94
idisp.h	2,196 bytes	2:00	pm	Fri	Sep	16 94
makefile	2,282 bytes	2:00	pm	Fri	Sep	16 94
precomp.cpp	24 bytes	2:00	pm	Fri	Sep	16 94
precomp.h	192 bytes	2:00	pm	Fri	Sep	16 94

```
        readme.txt        2,879 bytes   2:00 pm  Fri Sep 16 94
        resource.h        1,174 bytes   2:00 pm  Fri Sep 16 94
        tofile.cpp       13,374 bytes   2:00 pm  Fri Sep 16 94
        util.cpp         30,806 bytes   2:00 pm  Fri Sep 16 94
        util.h            1,488 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\OLE2\SAMPLES\DFVIEW
        .                   <DIR>       11:07 am  Mon Jun 10 96
        ..                  <DIR>       11:11 am  Mon Jun 10 96
        bmp_blan.bmp          358 bytes   2:00 pm  Fri Sep 16 94
        bmp_docu.bmp          438 bytes   2:00 pm  Fri Sep 16 94
        dflistb.c          14,198 bytes   2:00 pm  Fri Sep 16 94
        dfview.c           26,781 bytes   2:00 pm  Fri Sep 16 94
        dfview.def            240 bytes   2:00 pm  Fri Sep 16 94
        dfview.h            1,459 bytes   2:00 pm  Fri Sep 16 94
        dfview.ico            766 bytes   2:00 pm  Fri Sep 16 94
        dfview.mak          2,654 bytes   2:00 pm  Fri Sep 16 94
        dfview.rc           4,648 bytes   2:00 pm  Fri Sep 16 94
        dfvopen.ico           766 bytes   2:00 pm  Fri Sep 16 94
        dialogs.c           3,750 bytes   2:00 pm  Fri Sep 16 94
        folders.bmp           438 bytes   2:00 pm  Fri Sep 16 94
        foldlist.c         27,944 bytes   2:00 pm  Fri Sep 16 94
        foldlist.h          3,005 bytes   2:00 pm  Fri Sep 16 94
        globals.h             399 bytes   2:00 pm  Fri Sep 16 94
        makefile            3,816 bytes   2:00 pm  Fri Sep 16 94
        precomp.c              22 bytes   2:00 pm  Fri Sep 16 94
        precomp.h             221 bytes   2:00 pm  Fri Sep 16 94
        readme.txt          1,328 bytes   2:00 pm  Fri Sep 16 94
        resource.h          1,254 bytes   2:00 pm  Fri Sep 16 94
        windows.bmp           358 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\OLE2\SAMPLES\DISPCALC
        .                   <DIR>       11:07 am  Mon Jun 10 96
        ..                  <DIR>       11:11 am  Mon Jun 10 96
        clsid.c               724 bytes   2:00 pm  Fri Sep 16 94
        clsid.h               813 bytes   2:00 pm  Fri Sep 16 94
        dispcalc.cpp       12,981 bytes   2:00 pm  Fri Sep 16 94
        dispcalc.def          215 bytes   2:00 pm  Fri Sep 16 94
        dispcalc.h          5,665 bytes   2:00 pm  Fri Sep 16 94
        dispcalc.ico          766 bytes   2:00 pm  Fri Sep 16 94
        dispcalc.mak        2,429 bytes   2:00 pm  Fri Sep 16 94
        dispcalc.rc         1,216 bytes   2:00 pm  Fri Sep 16 94
        dispcalc.reg        1,004 bytes   2:00 pm  Fri Sep 16 94
        driver.bas          1,025 bytes   2:00 pm  Fri Sep 16 94
        driver.mak            130 bytes   2:00 pm  Fri Sep 16 94
        frmdrive.frm        3,658 bytes   2:00 pm  Fri Sep 16 94
        frmdrive.frx          770 bytes   2:00 pm  Fri Sep 16 94
        hostenv.h           1,072 bytes   2:00 pm  Fri Sep 16 94
        idata.cpp           2,969 bytes   2:00 pm  Fri Sep 16 94
        makefile            2,906 bytes   2:00 pm  Fri Sep 16 94
        readme.txt          2,642 bytes   2:00 pm  Fri Sep 16 94
        resource.h          1,616 bytes   2:00 pm  Fri Sep 16 94
        winmain.cpp         2,566 bytes   2:00 pm  Fri Sep 16 94
```

```
I:\VC152\MSVC15\OLE2\SAMPLES\DISPDEMO
        .                   <DIR>      11:07 am  Mon Jun 10 96
        ..                  <DIR>      11:11 am  Mon Jun 10 96
        clsid.c               758 bytes   2:00 pm  Fri Sep 16 94
        clsid.h             1,014 bytes   2:00 pm  Fri Sep 16 94
        crempoly.cpp       18,691 bytes   2:00 pm  Fri Sep 16 94
        crempoly.h          3,035 bytes   2:00 pm  Fri Sep 16 94
        dispdemo.def          221 bytes   2:00 pm  Fri Sep 16 94
        dispdemo.h            857 bytes   2:00 pm  Fri Sep 16 94
        dispdemo.ico          766 bytes   2:00 pm  Fri Sep 16 94
        dispdemo.mak        2,423 bytes   2:00 pm  Fri Sep 16 94
        dispdemo.rc           815 bytes   2:00 pm  Fri Sep 16 94
        hostenv.h           1,072 bytes   2:00 pm  Fri Sep 16 94
        makefile            2,956 bytes   2:00 pm  Fri Sep 16 94
        misc.cpp            1,336 bytes   2:00 pm  Fri Sep 16 94
        readme.txt            670 bytes   2:00 pm  Fri Sep 16 94
        resource.h          1,231 bytes   2:00 pm  Fri Sep 16 94
        winmain.cpp         3,618 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\OLE2\SAMPLES\DLLSRV
        .                   <DIR>      11:08 am  Mon Jun 10 96
        ..                  <DIR>      11:11 am  Mon Jun 10 96
        DATAVIEW            <DIR>      11:08 am  Mon Jun 10 96
        libmain.cpp           647 bytes   2:00 pm  Fri Sep 16 94
        makefile            1,552 bytes   2:00 pm  Fri Sep 16 94
        prgidmon.cpp       16,574 bytes   2:00 pm  Fri Sep 16 94
        prgidmon.def          232 bytes   2:00 pm  Fri Sep 16 94
        prgidmon.h            614 bytes   2:00 pm  Fri Sep 16 94
        prgidmon.mak        1,912 bytes   2:00 pm  Fri Sep 16 94
        prgidmon.reg          198 bytes   2:00 pm  Fri Sep 16 94
        readme.txt          2,289 bytes   2:00 pm  Fri Sep 16 94
        REALTIME            <DIR>      11:08 am  Mon Jun 10 96

I:\VC152\MSVC15\OLE2\SAMPLES\DLLSRV\DATAVIEW
        .                   <DIR>      11:08 am  Mon Jun 10 96
        ..                  <DIR>      11:08 am  Mon Jun 10 96
        dataview.cpp       45,646 bytes   2:00 pm  Fri Sep 16 94
        dataview.def          185 bytes   2:00 pm  Fri Sep 16 94
        dataview.h         19,331 bytes   2:00 pm  Fri Sep 16 94
        dataview.ico          766 bytes   2:00 pm  Fri Sep 16 94
        dataview.mak        3,074 bytes   2:00 pm  Fri Sep 16 94
        dataview.reg        1,633 bytes   2:00 pm  Fri Sep 16 94
        dvcache.cpp         7,963 bytes   2:00 pm  Fri Sep 16 94
        dvrun.cpp           6,540 bytes   2:00 pm  Fri Sep 16 94
        dvstock.cpp        11,241 bytes   2:00 pm  Fri Sep 16 94
        libmain.cpp         3,754 bytes   2:00 pm  Fri Sep 16 94
        makefile            2,226 bytes   2:00 pm  Fri Sep 16 94
        resource.h            989 bytes   2:00 pm  Fri Sep 16 94
        stockdlg.cpp        4,903 bytes   2:00 pm  Fri Sep 16 94
        stockdlg.h            590 bytes   2:00 pm  Fri Sep 16 94
        stockrc.rc          3,034 bytes   2:00 pm  Fri Sep 16 94
        stockutl.cpp        1,721 bytes   2:00 pm  Fri Sep 16 94
        stockutl.h            370 bytes   2:00 pm  Fri Sep 16 94
```

```
I:\VC152\MSVC15\OLE2\SAMPLES\DLLSRV\REALTIME
        .                   <DIR>       11:08 am  Mon Jun 10 96
        ..                  <DIR>       11:08 am  Mon Jun 10 96
        mainfrm.cpp       1,641 bytes    2:00 pm  Fri Sep 16 94
        mainfrm.h           788 bytes    2:00 pm  Fri Sep 16 94
        makefile          1,925 bytes    2:00 pm  Fri Sep 16 94
        realtdoc.cpp      2,618 bytes    2:00 pm  Fri Sep 16 94
        realtdoc.h        1,032 bytes    2:00 pm  Fri Sep 16 94
        realtime.cpp      8,202 bytes    2:00 pm  Fri Sep 16 94
        realtime.def        352 bytes    2:00 pm  Fri Sep 16 94
        realtime.h        2,261 bytes    2:00 pm  Fri Sep 16 94
        realtime.mak      3,068 bytes    2:00 pm  Fri Sep 16 94
        realtime.rc       6,622 bytes    2:00 pm  Fri Sep 16 94
        realtvw.h         1,003 bytes    2:00 pm  Fri Sep 16 94
        RES                 <DIR>       11:08 am  Mon Jun 10 96
        resource.h          574 bytes    2:00 pm  Fri Sep 16 94
        rtcntr.cpp        5,043 bytes    2:00 pm  Fri Sep 16 94
        rtcntr.h          1,551 bytes    2:00 pm  Fri Sep 16 94
        rtdata.h          1,424 bytes    2:00 pm  Fri Sep 16 94
        rtformvi.cpp      2,077 bytes    2:00 pm  Fri Sep 16 94
        rtformvi.h        1,064 bytes    2:00 pm  Fri Sep 16 94
        rtitem.cpp        6,002 bytes    2:00 pm  Fri Sep 16 94
        rtitem.h          1,867 bytes    2:00 pm  Fri Sep 16 94
        stdafx.cpp          204 bytes    2:00 pm  Fri Sep 16 94
        stdafx.h            526 bytes    2:00 pm  Fri Sep 16 94
        utils.cpp           833 bytes    2:00 pm  Fri Sep 16 94
        utils.h             861 bytes    2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\OLE2\SAMPLES\DLLSRV\REALTIME\RES
        .                   <DIR>       11:08 am  Mon Jun 10 96
        ..                  <DIR>       11:08 am  Mon Jun 10 96
        realtime.ico        766 bytes    2:00 pm  Fri Sep 16 94
        realtime.rc2      1,553 bytes    2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\OLE2\SAMPLES\DSPCALC2
        .                   <DIR>       11:08 am  Mon Jun 10 96
        ..                  <DIR>       11:11 am  Mon Jun 10 96
        calctype.odl      1,927 bytes    2:00 pm  Fri Sep 16 94
        clsid.c             617 bytes    2:00 pm  Fri Sep 16 94
        clsid.h             629 bytes    2:00 pm  Fri Sep 16 94
        dspcalc2.cpp      9,279 bytes    2:00 pm  Fri Sep 16 94
        dspcalc2.def        250 bytes    2:00 pm  Fri Sep 16 94
        dspcalc2.h        2,752 bytes    2:00 pm  Fri Sep 16 94
        dspcalc2.ico        766 bytes    2:00 pm  Fri Sep 16 94
        dspcalc2.rc       1,216 bytes    2:00 pm  Fri Sep 16 94
        dspcalc2.reg      1,979 bytes    2:00 pm  Fri Sep 16 94
        main.cpp          4,201 bytes    2:00 pm  Fri Sep 16 94
        makefile          3,446 bytes    2:00 pm  Fri Sep 16 94
        readme.txt        2,614 bytes    2:00 pm  Fri Sep 16 94
        resource.h          738 bytes    2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\OLE2\SAMPLES\GIZMOBAR
        .                   <DIR>       11:08 am  Mon Jun 10 96
        ..                  <DIR>       11:11 am  Mon Jun 10 96
```

```
        api.c           18,820 bytes   2:00 pm  Fri Sep 16 94
        DEMO              <DIR>       11:08 am  Mon Jun 10 96
        gizmo.c         18,213 bytes   2:00 pm  Fri Sep 16 94
        gizmo.h          3,259 bytes   2:00 pm  Fri Sep 16 94
        gizmobar.c      11,948 bytes   2:00 pm  Fri Sep 16 94
        gizmobar.def       986 bytes   2:00 pm  Fri Sep 16 94
        gizmobar.h       5,211 bytes   2:00 pm  Fri Sep 16 94
        gizmobar.mak     2,538 bytes   2:00 pm  Fri Sep 16 94
        gizmobar.rc        446 bytes   2:00 pm  Fri Sep 16 94
        gizmobar.rcv     1,509 bytes   2:00 pm  Fri Sep 16 94
        gizmoint.h       2,687 bytes   2:00 pm  Fri Sep 16 94
        init.c           5,983 bytes   2:00 pm  Fri Sep 16 94
        makefile         3,343 bytes   2:00 pm  Fri Sep 16 94
        paint.c          3,186 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\OLE2\SAMPLES\GIZMOBAR\DEMO
        .                 <DIR>       11:08 am  Mon Jun 10 96
        ..                <DIR>       11:08 am  Mon Jun 10 96
        gbdemo.c        10,013 bytes   2:00 pm  Fri Sep 16 94
        gbdemo.def         350 bytes   2:00 pm  Fri Sep 16 94
        gbdemo.h         3,776 bytes   2:00 pm  Fri Sep 16 94
        gbdemo.ico       1,086 bytes   2:00 pm  Fri Sep 16 94
        gbdemo.mak       2,175 bytes   2:00 pm  Fri Sep 16 94
        gbdemo.rc        1,707 bytes   2:00 pm  Fri Sep 16 94
        image120.bmp     2,050 bytes   2:00 pm  Fri Sep 16 94
        image72.bmp        734 bytes   2:00 pm  Fri Sep 16 94
        image96.bmp        958 bytes   2:00 pm  Fri Sep 16 94
        init.c           4,614 bytes   2:00 pm  Fri Sep 16 94
        makefile         2,169 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\OLE2\SAMPLES\HANDLER
        .                 <DIR>       11:08 am  Mon Jun 10 96
        ..                <DIR>       11:11 am  Mon Jun 10 96
        cdebug.cpp      28,598 bytes   2:00 pm  Fri Sep 16 94
        cdebug.h         1,753 bytes   2:00 pm  Fri Sep 16 94
        classfct.cpp     3,757 bytes   2:00 pm  Fri Sep 16 94
        classfct.h       1,299 bytes   2:00 pm  Fri Sep 16 94
        dbghndlr.cpp     5,113 bytes   2:00 pm  Fri Sep 16 94
        dbghndlr.def       446 bytes   2:00 pm  Fri Sep 16 94
        dbghndlr.h       1,529 bytes   2:00 pm  Fri Sep 16 94
        dbghndlr.ico       766 bytes   2:00 pm  Fri Sep 16 94
        dbghndlr.mak     2,803 bytes   2:00 pm  Fri Sep 16 94
        dbghndlr.rc      2,303 bytes   2:00 pm  Fri Sep 16 94
        INSTALLR          <DIR>       11:08 am  Mon Jun 10 96
        makefile         2,284 bytes   2:00 pm  Fri Sep 16 94
        object.cpp      52,149 bytes   2:00 pm  Fri Sep 16 94
        object.h        12,580 bytes   2:00 pm  Fri Sep 16 94
        readme.txt         898 bytes   2:00 pm  Fri Sep 16 94
        resource.h         534 bytes   2:00 pm  Fri Sep 16 94
        stdafx.cpp         643 bytes   2:00 pm  Fri Sep 16 94
        stdafx.h           999 bytes   2:00 pm  Fri Sep 16 94
        utils.cpp        1,812 bytes   2:00 pm  Fri Sep 16 94
        utils.h            602 bytes   2:00 pm  Fri Sep 16 94
```

```
I:\VC152\MSVC15\OLE2\SAMPLES\HANDLER\INSTALLR
        .                   <DIR>       11:08 am  Mon Jun 10 96
        ..                  <DIR>       11:08 am  Mon Jun 10 96
        dbghndlr.ico          766 bytes   2:00 pm  Fri Sep 16 94
        installd.cpp        8,804 bytes   2:00 pm  Fri Sep 16 94
        installd.h            958 bytes   2:00 pm  Fri Sep 16 94
        installr.def          200 bytes   2:00 pm  Fri Sep 16 94
        installr.mak        2,214 bytes   2:00 pm  Fri Sep 16 94
        installr.rc         2,404 bytes   2:00 pm  Fri Sep 16 94
        makefile            1,690 bytes   2:00 pm  Fri Sep 16 94
        resource.h            708 bytes   2:00 pm  Fri Sep 16 94
        stdafx.cpp            645 bytes   2:00 pm  Fri Sep 16 94
        stdafx.h              791 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\OLE2\SAMPLES\HELLO
        .                   <DIR>       11:08 am  Mon Jun 10 96
        ..                  <DIR>       11:11 am  Mon Jun 10 96
        clsid.c               397 bytes   2:00 pm  Fri Sep 16 94
        clsid.h             1,495 bytes   2:00 pm  Fri Sep 16 94
        common.h            1,000 bytes   2:00 pm  Fri Sep 16 94
        hello.cpp           5,770 bytes   2:00 pm  Fri Sep 16 94
        hello.def             233 bytes   2:00 pm  Fri Sep 16 94
        hello.frm           3,998 bytes   2:00 pm  Fri Sep 16 94
        hello.h             1,643 bytes   2:00 pm  Fri Sep 16 94
        hello.ico             766 bytes   2:00 pm  Fri Sep 16 94
        hello.odl           1,296 bytes   2:00 pm  Fri Sep 16 94
        hello.rc            1,952 bytes   2:00 pm  Fri Sep 16 94
        hello.reg           2,164 bytes   2:00 pm  Fri Sep 16 94
        hellopro.cpp        1,574 bytes   2:00 pm  Fri Sep 16 94
        hellopro.h            782 bytes   2:00 pm  Fri Sep 16 94
        main.cpp            4,374 bytes   2:00 pm  Fri Sep 16 94
        makefile            3,365 bytes   2:00 pm  Fri Sep 16 94
        readme.txt          1,777 bytes   2:00 pm  Fri Sep 16 94
        resource.h            458 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\OLE2\SAMPLES\ILCKBYTS
        .                   <DIR>       11:08 am  Mon Jun 10 96
        ..                  <DIR>       11:11 am  Mon Jun 10 96
        ilb.reg               279 bytes   2:00 pm  Fri Sep 16 94
        ILBCNTR             <DIR>       11:08 am  Mon Jun 10 96
        ilbdll.def            469 bytes   2:00 pm  Fri Sep 16 94
        ilbdll.mak          1,754 bytes   2:00 pm  Fri Sep 16 94
        ILBSVR              <DIR>       11:08 am  Mon Jun 10 96
        makefile            1,992 bytes   2:00 pm  Fri Sep 16 94
        memstm.cpp         40,098 bytes   2:00 pm  Fri Sep 16 94
        memstm.h            4,928 bytes   2:00 pm  Fri Sep 16 94
        readme.txt          2,163 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\OLE2\SAMPLES\ILCKBYTS\ILBCNTR
        .                   <DIR>       11:08 am  Mon Jun 10 96
        ..                  <DIR>       11:08 am  Mon Jun 10 96
        ilbcndoc.cpp        1,741 bytes   2:00 pm  Fri Sep 16 94
        ilbcndoc.h            948 bytes   2:00 pm  Fri Sep 16 94
        ilbcntr.cpp         7,909 bytes   2:00 pm  Fri Sep 16 94
```

```
        ilbcntr.def         349 bytes   2:00 pm  Fri Sep 16 94
        ilbcntr.h         1,470 bytes   2:00 pm  Fri Sep 16 94
        ilbcntr.mak       2,649 bytes   2:00 pm  Fri Sep 16 94
        ilbcntr.rc        7,040 bytes   2:00 pm  Fri Sep 16 94
        ilbcnvw.cpp       1,687 bytes   2:00 pm  Fri Sep 16 94
        ilbcnvw.h         1,078 bytes   2:00 pm  Fri Sep 16 94
        mainfrm.cpp       1,262 bytes   2:00 pm  Fri Sep 16 94
        mainfrm.h           825 bytes   2:00 pm  Fri Sep 16 94
        makefile          2,313 bytes   2:00 pm  Fri Sep 16 94
        RES                 <DIR>      11:08 am  Mon Jun 10 96
        resource.h          906 bytes   2:00 pm  Fri Sep 16 94
        stdafx.cpp          204 bytes   2:00 pm  Fri Sep 16 94
        stdafx.h            299 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\OLE2\SAMPLES\ILCKBYTS\ILBCNTR\RES
        .                   <DIR>      11:08 am  Mon Jun 10 96
        ..                  <DIR>      11:08 am  Mon Jun 10 96
        ilbcntr.ico         768 bytes   2:00 pm  Fri Sep 16 94
        ilbcntr.rc2       1,548 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\OLE2\SAMPLES\ILCKBYTS\ILBSVR
        .                   <DIR>      11:08 am  Mon Jun 10 96
        ..                  <DIR>      11:08 am  Mon Jun 10 96
        ilbsvdoc.cpp      1,723 bytes   2:00 pm  Fri Sep 16 94
        ilbsvdoc.h          942 bytes   2:00 pm  Fri Sep 16 94
        ilbsvr.cpp        3,799 bytes   2:00 pm  Fri Sep 16 94
        ilbsvr.def          346 bytes   2:00 pm  Fri Sep 16 94
        ilbsvr.h            878 bytes   2:00 pm  Fri Sep 16 94
        ilbsvr.mak        2,712 bytes   2:00 pm  Fri Sep 16 94
        ilbsvr.rc         6,463 bytes   2:00 pm  Fri Sep 16 94
        ilbsvvw.cpp       1,665 bytes   2:00 pm  Fri Sep 16 94
        ilbsvvw.h         1,068 bytes   2:00 pm  Fri Sep 16 94
        mainfrm.cpp       1,261 bytes   2:00 pm  Fri Sep 16 94
        mainfrm.h           825 bytes   2:00 pm  Fri Sep 16 94
        makefile          2,310 bytes   2:00 pm  Fri Sep 16 94
        RES                 <DIR>      11:08 am  Mon Jun 10 96
        resource.h          360 bytes   2:00 pm  Fri Sep 16 94
        stdafx.cpp          204 bytes   2:00 pm  Fri Sep 16 94
        stdafx.h            299 bytes   2:00 pm  Fri Sep 16 94
        test.cpp          3,066 bytes   2:00 pm  Fri Sep 16 94
        test.h              974 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\OLE2\SAMPLES\ILCKBYTS\ILBSVR\RES
        .                   <DIR>      11:08 am  Mon Jun 10 96
        ..                  <DIR>      11:08 am  Mon Jun 10 96
        ilbsvr.ico          768 bytes   2:00 pm  Fri Sep 16 94
        ilbsvr.rc2        1,543 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\OLE2\SAMPLES\INCLUDE
        .                   <DIR>      11:08 am  Mon Jun 10 96
        ..                  <DIR>      11:11 am  Mon Jun 10 96
        bttncur.h         7,557 bytes   2:00 pm  Fri Sep 16 94
        ctl3d.h           2,673 bytes   5:30 am  Mon Feb 27 95
        enumfetc.h          384 bytes   2:00 pm  Fri Sep 16 94
```

```
        gizmobar.h        5,211 bytes   2:00 pm  Fri Sep 16 94
        msgfiltr.h        1,966 bytes   2:00 pm  Fri Sep 16 94
        suminfo.h        10,344 bytes   2:00 pm  Fri Sep 16 94
        win1632.h         2,770 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\OLE2\SAMPLES\LAUNCHER
        .                   <DIR>       11:08 am  Mon Jun 10 96
        ..                  <DIR>       11:11 am  Mon Jun 10 96
        classfct.cpp      7,779 bytes   2:00 pm  Fri Sep 16 94
        classfct.h        2,109 bytes   2:00 pm  Fri Sep 16 94
        dvlaunch.mak      2,371 bytes   2:00 pm  Fri Sep 16 94
        dvlaunch.reg        245 bytes   2:00 pm  Fri Sep 16 94
        launcher.cpp      3,527 bytes   2:00 pm  Fri Sep 16 94
        launcher.def        217 bytes   2:00 pm  Fri Sep 16 94
        launcher.h        1,242 bytes   2:00 pm  Fri Sep 16 94
        launcher.ico        766 bytes   2:00 pm  Fri Sep 16 94
        launcher.rc       1,637 bytes   2:00 pm  Fri Sep 16 94
        makefile          2,764 bytes   2:00 pm  Fri Sep 16 94
        readme.txt        1,413 bytes   2:00 pm  Fri Sep 16 94
        resource.h          442 bytes   2:00 pm  Fri Sep 16 94
        stdafx.cpp          495 bytes   2:00 pm  Fri Sep 16 94
        stdafx.h            460 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\OLE2\SAMPLES\LIB
        .                   <DIR>       11:09 am  Mon Jun 10 96
        ..                  <DIR>       11:11 am  Mon Jun 10 96
        bttncur.lib       1,536 bytes   2:00 pm  Fri Sep 16 94
        ctl3d.lib         3,072 bytes  12:57 am  Fri Jul 14 95
        ctl3dv2.lib       3,072 bytes  12:56 am  Fri Jul 14 95
        gizmobar.lib      2,560 bytes   2:00 pm  Fri Sep 16 94
        oleuic.lib      144,153 bytes   2:00 pm  Fri Sep 16 94
        oleuicd.lib     467,237 bytes   2:00 pm  Fri Sep 16 94
        oleuid.lib      151,321 bytes   2:00 pm  Fri Sep 16 94
        oleuidd.lib     484,645 bytes   2:00 pm  Fri Sep 16 94
        moleuic.lib     143,641 bytes   2:00 pm  Fri Sep 16 94
        moleuicd.lib    466,725 bytes   2:00 pm  Fri Sep 16 94
        moleuid.lib     150,809 bytes   2:00 pm  Fri Sep 16 94
        moleuidd.lib    483,109 bytes   2:00 pm  Fri Sep 16 94
        outlui.lib       23,552 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\OLE2\SAMPLES\OLE2UI
        .                   <DIR>       11:10 am  Mon Jun 10 96
        ..                  <DIR>       11:11 am  Mon Jun 10 96
        busy.c           15,655 bytes   2:00 pm  Fri Sep 16 94
        busy.h            1,420 bytes   2:00 pm  Fri Sep 16 94
        common.c         12,212 bytes   2:00 pm  Fri Sep 16 94
        common.h          5,355 bytes   2:00 pm  Fri Sep 16 94
        convert.c        49,824 bytes   2:00 pm  Fri Sep 16 94
        convert.h         2,176 bytes   2:00 pm  Fri Sep 16 94
        dballoc.cpp      17,174 bytes   2:00 pm  Fri Sep 16 94
        dballoc.h           838 bytes   2:00 pm  Fri Sep 16 94
        dbgutil.c         9,511 bytes   2:00 pm  Fri Sep 16 94
        debug.h           1,407 bytes   2:00 pm  Fri Sep 16 94
        depend            2,001 bytes   2:00 pm  Fri Sep 16 94
```

```
dllfuncs.c        5,402 bytes   2:00 pm  Fri Sep 16 94
drawicon.c       18,696 bytes   2:00 pm  Fri Sep 16 94
edlinks.h         5,963 bytes   2:00 pm  Fri Sep 16 94
enumfetc.c        8,149 bytes   2:00 pm  Fri Sep 16 94
enumfetc.h          384 bytes   2:00 pm  Fri Sep 16 94
enumstat.c        9,018 bytes   2:00 pm  Fri Sep 16 94
geticon.c        30,182 bytes   2:00 pm  Fri Sep 16 94
geticon.h           708 bytes   2:00 pm  Fri Sep 16 94
hatch.c           8,600 bytes   2:00 pm  Fri Sep 16 94
icon.c           23,756 bytes   2:00 pm  Fri Sep 16 94
icon.h            1,931 bytes   2:00 pm  Fri Sep 16 94
iconbox.c         5,690 bytes   2:00 pm  Fri Sep 16 94
iconbox.h           904 bytes   2:00 pm  Fri Sep 16 94
insobj.c         49,154 bytes   2:00 pm  Fri Sep 16 94
insobj.h          1,797 bytes   2:00 pm  Fri Sep 16 94
links.c          56,473 bytes   2:00 pm  Fri Sep 16 94
makedll           9,726 bytes   2:00 pm  Fri Sep 16 94
makelib           9,258 bytes   2:00 pm  Fri Sep 16 94
mfcoleui.def      9,115 bytes   2:00 pm  Fri Sep 16 94
mfcoleui.mak      5,747 bytes   2:00 pm  Fri Sep 16 94
msgfiltr.c       25,420 bytes   2:00 pm  Fri Sep 16 94
msgfiltr.h        1,966 bytes   2:00 pm  Fri Sep 16 94
objfdbk.c         6,672 bytes   2:00 pm  Fri Sep 16 94
ole2ui.c         30,578 bytes   2:00 pm  Fri Sep 16 94
ole2ui.def        9,029 bytes   2:00 pm  Fri Sep 16 94
ole2ui.h         35,007 bytes   2:00 pm  Fri Sep 16 94
ole2ui.rc           904 bytes   2:00 pm  Fri Sep 16 94
olestd.c         82,793 bytes   2:00 pm  Fri Sep 16 94
olestd.h         29,566 bytes   2:00 pm  Fri Sep 16 94
oleutl.c         20,877 bytes   2:00 pm  Fri Sep 16 94
outlui.def        9,113 bytes   2:00 pm  Fri Sep 16 94
outlui.mak        5,731 bytes   2:00 pm  Fri Sep 16 94
pastespl.c       53,757 bytes   2:00 pm  Fri Sep 16 94
pastespl.h        5,399 bytes   2:00 pm  Fri Sep 16 94
precomp.c           773 bytes   2:00 pm  Fri Sep 16 94
readme.txt          972 bytes   2:00 pm  Fri Sep 16 94
regdb.c           9,496 bytes   2:00 pm  Fri Sep 16 94
regdb.h             189 bytes   2:00 pm  Fri Sep 16 94
RES                 <DIR>      11:10 am  Mon Jun 10 96
resimage.c       10,290 bytes   2:00 pm  Fri Sep 16 94
resimage.h        1,747 bytes   2:00 pm  Fri Sep 16 94
stdpal.c          2,575 bytes   2:00 pm  Fri Sep 16 94
stdpal.h          9,534 bytes   2:00 pm  Fri Sep 16 94
suminfo.cpp      45,945 bytes   2:00 pm  Fri Sep 16 94
suminfo.h        10,344 bytes   2:00 pm  Fri Sep 16 94
targtdev.c        8,274 bytes   2:00 pm  Fri Sep 16 94
template.c        5,977 bytes   2:00 pm  Fri Sep 16 94
template.h        3,355 bytes   2:00 pm  Fri Sep 16 94
utility.c        26,331 bytes   2:00 pm  Fri Sep 16 94
utility.h         1,266 bytes   2:00 pm  Fri Sep 16 94
wn_dos.h          5,444 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\OLE2\SAMPLES\OLE2UI\RES
.                   <DIR>      11:10 am  Mon Jun 10 96
```

```
        ..                  <DIR>       11:10 am  Mon Jun 10 96
        DUT                 <DIR>       11:10 am  Mon Jun 10 96
        FRN                 <DIR>       11:10 am  Mon Jun 10 96
        GER                 <DIR>       11:10 am  Mon Jun 10 96
        ITA                 <DIR>       11:10 am  Mon Jun 10 96
        IUSA                <DIR>       11:10 am  Mon Jun 10 96
        localole.h       37,370 bytes    2:00 pm  Fri Sep 16 94
        ole2ui.rcv        2,774 bytes    2:00 pm  Fri Sep 16 94
        SPA                 <DIR>       11:10 am  Mon Jun 10 96
        STATIC              <DIR>       11:10 am  Mon Jun 10 96
        SWE                 <DIR>       11:10 am  Mon Jun 10 96
        USA                 <DIR>       11:10 am  Mon Jun 10 96

I:\VC152\MSVC15\OLE2\SAMPLES\OLE2UI\RES\DUT
        .                   <DIR>       11:10 am  Mon Jun 10 96
        ..                  <DIR>       11:10 am  Mon Jun 10 96
        busy.dlg            640 bytes    2:00 pm  Fri Sep 16 94
        convert.dlg       1,463 bytes    2:00 pm  Fri Sep 16 94
        fileopen.dlg      1,456 bytes    2:00 pm  Fri Sep 16 94
        icon.dlg          1,429 bytes    2:00 pm  Fri Sep 16 94
        insobj.dlg        1,787 bytes    2:00 pm  Fri Sep 16 94
        links.dlg         1,515 bytes    2:00 pm  Fri Sep 16 94
        pastespl.dlg      1,738 bytes    2:00 pm  Fri Sep 16 94
        prompt.dlg        3,266 bytes    2:00 pm  Fri Sep 16 94
        strings.rc        6,310 bytes    2:00 pm  Fri Sep 16 94
        verlocal.h        1,862 bytes    2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\OLE2\SAMPLES\OLE2UI\RES\FRN
        .                   <DIR>       11:10 am  Mon Jun 10 96
        ..                  <DIR>       11:10 am  Mon Jun 10 96
        busy.dlg            626 bytes    2:00 pm  Fri Sep 16 94
        convert.dlg       1,472 bytes    2:00 pm  Fri Sep 16 94
        fileopen.dlg      1,458 bytes    2:00 pm  Fri Sep 16 94
        icon.dlg          1,441 bytes    2:00 pm  Fri Sep 16 94
        insobj.dlg        1,801 bytes    2:00 pm  Fri Sep 16 94
        links.dlg         1,528 bytes    2:00 pm  Fri Sep 16 94
        pastespl.dlg      1,748 bytes    2:00 pm  Fri Sep 16 94
        prompt.dlg        3,329 bytes    2:00 pm  Fri Sep 16 94
        strings.rc        6,191 bytes    2:00 pm  Fri Sep 16 94
        verlocal.h        1,868 bytes    2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\OLE2\SAMPLES\OLE2UI\RES\GER
        .                   <DIR>       11:10 am  Mon Jun 10 96
        ..                  <DIR>       11:10 am  Mon Jun 10 96
        busy.dlg            657 bytes    2:00 pm  Fri Sep 16 94
        convert.dlg       1,469 bytes    2:00 pm  Fri Sep 16 94
        fileopen.dlg      1,457 bytes    2:00 pm  Fri Sep 16 94
        icon.dlg          1,435 bytes    2:00 pm  Fri Sep 16 94
        insobj.dlg        1,796 bytes    2:00 pm  Fri Sep 16 94
        links.dlg         1,545 bytes    2:00 pm  Fri Sep 16 94
        pastespl.dlg      1,745 bytes    2:00 pm  Fri Sep 16 94
        prompt.dlg        3,374 bytes    2:00 pm  Fri Sep 16 94
        strings.rc        6,350 bytes    2:00 pm  Fri Sep 16 94
        verlocal.h        1,869 bytes    2:00 pm  Fri Sep 16 94
```

```
I:\VC152\MSVC15\OLE2\SAMPLES\OLE2UI\RES\ITA
        .                   <DIR>      11:10 am  Mon Jun 10 96
        ..                  <DIR>      11:10 am  Mon Jun 10 96
        busy.dlg              605 bytes   2:00 pm  Fri Sep 16 94
        convert.dlg         1,458 bytes   2:00 pm  Fri Sep 16 94
        fileopen.dlg        1,436 bytes   2:00 pm  Fri Sep 16 94
        icon.dlg            1,421 bytes   2:00 pm  Fri Sep 16 94
        insobj.dlg          1,781 bytes   2:00 pm  Fri Sep 16 94
        links.dlg           1,524 bytes   2:00 pm  Fri Sep 16 94
        pastespl.dlg        1,734 bytes   2:00 pm  Fri Sep 16 94
        prompt.dlg          3,238 bytes   2:00 pm  Fri Sep 16 94
        strings.rc          6,136 bytes   2:00 pm  Fri Sep 16 94
        verlocal.h          1,856 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\OLE2\SAMPLES\OLE2UI\RES\IUSA
        .                   <DIR>      11:10 am  Mon Jun 10 96
        ..                  <DIR>      11:10 am  Mon Jun 10 96
        busy.dlg              616 bytes   2:00 pm  Fri Sep 16 94
        convert.dlg         1,495 bytes   2:00 pm  Fri Sep 16 94
        fileopen.dlg        1,603 bytes   2:00 pm  Fri Sep 16 94
        icon.dlg            1,505 bytes   2:00 pm  Fri Sep 16 94
        insobj.dlg          1,846 bytes   2:00 pm  Fri Sep 16 94
        links.dlg           1,560 bytes   2:00 pm  Fri Sep 16 94
        pastespl.dlg        1,837 bytes   2:00 pm  Fri Sep 16 94
        prompt.dlg          3,216 bytes   2:00 pm  Fri Sep 16 94
        strings.rc          5,892 bytes   2:00 pm  Fri Sep 16 94
        verlocal.h          1,859 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\OLE2\SAMPLES\OLE2UI\RES\SPA
        .                   <DIR>      11:10 am  Mon Jun 10 96
        ..                  <DIR>      11:10 am  Mon Jun 10 96
        busy.dlg              631 bytes   2:00 pm  Fri Sep 16 94
        convert.dlg         1,481 bytes   2:00 pm  Fri Sep 16 94
        fileopen.dlg        1,465 bytes   2:00 pm  Fri Sep 16 94
        icon.dlg            1,441 bytes   2:00 pm  Fri Sep 16 94
        insobj.dlg          1,810 bytes   2:00 pm  Fri Sep 16 94
        links.dlg           1,518 bytes   2:00 pm  Fri Sep 16 94
        pastespl.dlg        1,738 bytes   2:00 pm  Fri Sep 16 94
        prompt.dlg          3,382 bytes   2:00 pm  Fri Sep 16 94
        strings.rc          6,058 bytes   2:00 pm  Fri Sep 16 94
        verlocal.h          1,894 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\OLE2\SAMPLES\OLE2UI\RES\STATIC
        .                   <DIR>      11:10 am  Mon Jun 10 96
        ..                  <DIR>      11:10 am  Mon Jun 10 96
        bang.ico            1,846 bytes   2:00 pm  Fri Sep 16 94
        default.ico           766 bytes   2:00 pm  Fri Sep 16 94
        egares.bmp          6,838 bytes   2:00 pm  Fri Sep 16 94
        hivgares.bmp       20,326 bytes   2:00 pm  Fri Sep 16 94
        vgares.bmp          9,078 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\OLE2\SAMPLES\OLE2UI\RES\SWE
        .                   <DIR>      11:10 am  Mon Jun 10 96
```

```
..                  <DIR>       11:10 am  Mon Jun 10 96
busy.dlg             612 bytes   2:00 pm  Fri Sep 16 94
convert.dlg        1,470 bytes   2:00 pm  Fri Sep 16 94
fileopen.dlg       1,432 bytes   2:00 pm  Fri Sep 16 94
icon.dlg           1,418 bytes   2:00 pm  Fri Sep 16 94
insobj.dlg         1,770 bytes   2:00 pm  Fri Sep 16 94
links.dlg          1,495 bytes   2:00 pm  Fri Sep 16 94
pastespl.dlg       1,724 bytes   2:00 pm  Fri Sep 16 94
prompt.dlg         3,245 bytes   2:00 pm  Fri Sep 16 94
strings.rc         5,788 bytes   2:00 pm  Fri Sep 16 94
verlocal.h         1,843 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\OLE2\SAMPLES\OLE2UI\RES\USA
.                   <DIR>       11:10 am  Mon Jun 10 96
..                  <DIR>       11:10 am  Mon Jun 10 96
busy.dlg             597 bytes   2:00 pm  Fri Sep 16 94
convert.dlg        1,477 bytes   2:00 pm  Fri Sep 16 94
fileopen.dlg       1,451 bytes   2:00 pm  Fri Sep 16 94
icon.dlg           1,430 bytes   2:00 pm  Fri Sep 16 94
insobj.dlg         1,786 bytes   2:00 pm  Fri Sep 16 94
links.dlg          1,498 bytes   2:00 pm  Fri Sep 16 94
pastespl.dlg       1,731 bytes   2:00 pm  Fri Sep 16 94
prompt.dlg         3,157 bytes   2:00 pm  Fri Sep 16 94
strings.rc         5,753 bytes   2:00 pm  Fri Sep 16 94
verlocal.h         1,860 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\OLE2\SAMPLES\OUTLINE
.                   <DIR>       11:11 am  Mon Jun 10 96
..                  <DIR>       11:11 am  Mon Jun 10 96
classfac.c         7,045 bytes   2:00 pm  Fri Sep 16 94
clipbrd.c        104,334 bytes   2:00 pm  Fri Sep 16 94
cntrbase.c        60,786 bytes   2:00 pm  Fri Sep 16 94
cntrinpl.c        63,439 bytes   2:00 pm  Fri Sep 16 94
cntrline.c       111,980 bytes   2:00 pm  Fri Sep 16 94
CNTROUTL            <DIR>       11:11 am  Mon Jun 10 96
cntroutl.def         550 bytes   2:00 pm  Fri Sep 16 94
cntroutl.h        31,762 bytes   2:00 pm  Fri Sep 16 94
cntroutl.ico         766 bytes   2:00 pm  Fri Sep 16 94
cntroutl.mak       5,227 bytes   2:00 pm  Fri Sep 16 94
cntroutl.rc        5,650 bytes   2:00 pm  Fri Sep 16 94
cntrrc.h             846 bytes   2:00 pm  Fri Sep 16 94
debug.c            2,246 bytes   2:00 pm  Fri Sep 16 94
debug.rc           7,090 bytes   2:00 pm  Fri Sep 16 94
debug2.c           8,274 bytes   2:00 pm  Fri Sep 16 94
defguid.h          2,074 bytes   2:00 pm  Fri Sep 16 94
dialogs.c         17,203 bytes   2:00 pm  Fri Sep 16 94
dialogs.dlg        3,488 bytes   2:00 pm  Fri Sep 16 94
dragcopy.cur         326 bytes   2:00 pm  Fri Sep 16 94
dragdrop.c        20,532 bytes   2:00 pm  Fri Sep 16 94
draglink.cur         326 bytes   2:00 pm  Fri Sep 16 94
dragmove.cur         326 bytes   2:00 pm  Fri Sep 16 94
dragnone.cur         326 bytes   2:00 pm  Fri Sep 16 94
frametls.c        28,690 bytes   2:00 pm  Fri Sep 16 94
frametls.h         3,721 bytes   2:00 pm  Fri Sep 16 94
```

```
heading.c        10,591 bytes   2:00 pm  Fri Sep 16 94
heading.h         2,104 bytes   2:00 pm  Fri Sep 16 94
ICNTROTL            <DIR>      11:11 am  Mon Jun 10 96
icntrotl.def        567 bytes   2:00 pm  Fri Sep 16 94
icntrotl.ico        766 bytes   2:00 pm  Fri Sep 16 94
icntrotl.mak      5,401 bytes   2:00 pm  Fri Sep 16 94
icntrotl.rc       6,857 bytes   2:00 pm  Fri Sep 16 94
image120.bmp      1,498 bytes   2:00 pm  Fri Sep 16 94
image72.bmp         558 bytes   2:00 pm  Fri Sep 16 94
image96.bmp         758 bytes   2:00 pm  Fri Sep 16 94
ISVROTL             <DIR>      11:11 am  Mon Jun 10 96
isvrotl.def         558 bytes   2:00 pm  Fri Sep 16 94
isvrotl.ico         766 bytes   2:00 pm  Fri Sep 16 94
isvrotl.mak       5,378 bytes   2:00 pm  Fri Sep 16 94
isvrotl.rc        6,580 bytes   2:00 pm  Fri Sep 16 94
linking.c        56,567 bytes   2:00 pm  Fri Sep 16 94
main.c           73,025 bytes   2:00 pm  Fri Sep 16 94
makefile          7,966 bytes   2:00 pm  Fri Sep 16 94
memmgr.c            655 bytes   2:00 pm  Fri Sep 16 94
message.h         4,624 bytes   2:00 pm  Fri Sep 16 94
ole2.bmp         14,862 bytes   2:00 pm  Fri Sep 16 94
oleapp.c        104,371 bytes   2:00 pm  Fri Sep 16 94
oledoc.c         38,708 bytes   2:00 pm  Fri Sep 16 94
oleoutl.h        27,671 bytes   2:00 pm  Fri Sep 16 94
outlapp.c        41,590 bytes   2:00 pm  Fri Sep 16 94
outldoc.c        86,053 bytes   2:00 pm  Fri Sep 16 94
OUTLINE             <DIR>      11:11 am  Mon Jun 10 96
outline.def         541 bytes   2:00 pm  Fri Sep 16 94
outline.h        32,194 bytes   2:00 pm  Fri Sep 16 94
outline.ico         766 bytes   2:00 pm  Fri Sep 16 94
outline.mak       4,112 bytes   2:00 pm  Fri Sep 16 94
outline.rc        5,418 bytes   2:00 pm  Fri Sep 16 94
outline.reg      11,854 bytes   2:00 pm  Fri Sep 16 94
outlline.c       16,438 bytes   2:00 pm  Fri Sep 16 94
outllist.c       28,340 bytes   2:00 pm  Fri Sep 16 94
outlname.c        2,634 bytes   2:00 pm  Fri Sep 16 94
outlntbl.c       11,536 bytes   2:00 pm  Fri Sep 16 94
outlrc.h          6,394 bytes   2:00 pm  Fri Sep 16 94
outltxtl.c        9,261 bytes   2:00 pm  Fri Sep 16 94
precomp.c           374 bytes   2:00 pm  Fri Sep 16 94
readme.txt        7,825 bytes   2:00 pm  Fri Sep 16 94
selcross.cur        326 bytes   2:00 pm  Fri Sep 16 94
status.c          8,429 bytes   2:00 pm  Fri Sep 16 94
status.h          1,415 bytes   2:00 pm  Fri Sep 16 94
stripper.exe     37,756 bytes   2:00 pm  Fri Sep 16 94
svrbase.c        59,813 bytes   2:00 pm  Fri Sep 16 94
svrinpl.c        43,622 bytes   2:00 pm  Fri Sep 16 94
SVROUTL             <DIR>      11:11 am  Mon Jun 10 96
svroutl.def         549 bytes   2:00 pm  Fri Sep 16 94
svroutl.h        32,421 bytes   2:00 pm  Fri Sep 16 94
svroutl.ico         766 bytes   2:00 pm  Fri Sep 16 94
svroutl.mak       5,213 bytes   2:00 pm  Fri Sep 16 94
svroutl.rc        5,964 bytes   2:00 pm  Fri Sep 16 94
svrpsobj.c       42,976 bytes   2:00 pm  Fri Sep 16 94
```

```
I:\VC152\MSVC15\OLE2\SAMPLES\OUTLINE\CNTROUTL
        .                   <DIR>      11:11 am  Mon Jun 10 96
        ..                  <DIR>      11:11 am  Mon Jun 10 96
        mksrc.bat         1,272 bytes   2:00 pm  Fri Sep 16 94
        stripole.dfn        190 bytes   2:00 pm  Fri Sep 16 94
        stripper.dfn        190 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\OLE2\SAMPLES\OUTLINE\ICNTROTL
        .                   <DIR>      11:11 am  Mon Jun 10 96
        ..                  <DIR>      11:11 am  Mon Jun 10 96
        mksrc.bat         1,327 bytes   2:00 pm  Fri Sep 16 94
        stripole.dfn        190 bytes   2:00 pm  Fri Sep 16 94
        stripper.dfn        190 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\OLE2\SAMPLES\OUTLINE\ISVROTL
        .                   <DIR>      11:11 am  Mon Jun 10 96
        ..                  <DIR>      11:11 am  Mon Jun 10 96
        mksrc.bat         1,323 bytes   2:00 pm  Fri Sep 16 94
        stripole.dfn        190 bytes   2:00 pm  Fri Sep 16 94
        stripper.dfn        190 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\OLE2\SAMPLES\OUTLINE\OUTLINE
        .                   <DIR>      11:11 am  Mon Jun 10 96
        ..                  <DIR>      11:11 am  Mon Jun 10 96
        mksrc.bat           844 bytes   2:00 pm  Fri Sep 16 94
        stripper.dfn        202 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\OLE2\SAMPLES\OUTLINE\SVROUTL
        .                   <DIR>      11:11 am  Mon Jun 10 96
        ..                  <DIR>      11:11 am  Mon Jun 10 96
        mksrc.bat         1,272 bytes   2:00 pm  Fri Sep 16 94
        stripole.dfn        190 bytes   2:00 pm  Fri Sep 16 94
        stripper.dfn        190 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\OLE2\SAMPLES\PSSAMP
        .                   <DIR>      11:11 am  Mon Jun 10 96
        ..                  <DIR>      11:11 am  Mon Jun 10 96
        makefile            993 bytes   2:00 pm  Fri Sep 16 94
        odcsamp.h           712 bytes   2:00 pm  Fri Sep 16 94
        pssamp.cpp       12,962 bytes   2:00 pm  Fri Sep 16 94
        pssamp.def          242 bytes   2:00 pm  Fri Sep 16 94
        pssamp.h          2,593 bytes   2:00 pm  Fri Sep 16 94
        pssamp.mak        1,728 bytes   2:00 pm  Fri Sep 16 94
        pssamp.reg          807 bytes   2:00 pm  Fri Sep 16 94
        readme.txt        1,218 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\OLE2\SAMPLES\SIMPCNTR
        .                   <DIR>      11:11 am  Mon Jun 10 96
        ..                  <DIR>      11:11 am  Mon Jun 10 96
        app.cpp          25,093 bytes   2:00 pm  Fri Sep 16 94
        app.h             2,147 bytes   2:00 pm  Fri Sep 16 94
        doc.cpp          14,845 bytes   2:00 pm  Fri Sep 16 94
        doc.h             1,212 bytes   2:00 pm  Fri Sep 16 94
```

```
        ias.cpp          7,631 bytes   2:00 pm  Fri Sep 16 94
        ias.h            1,159 bytes   2:00 pm  Fri Sep 16 94
        iocs.cpp         8,403 bytes   2:00 pm  Fri Sep 16 94
        iocs.h           1,196 bytes   2:00 pm  Fri Sep 16 94
        ioipf.cpp       15,876 bytes   2:00 pm  Fri Sep 16 94
        ioipf.h          1,696 bytes   2:00 pm  Fri Sep 16 94
        ioips.cpp       15,348 bytes   2:00 pm  Fri Sep 16 94
        ioips.h          1,592 bytes   2:00 pm  Fri Sep 16 94
        makefile         2,742 bytes   2:00 pm  Fri Sep 16 94
        pre.cpp            304 bytes   2:00 pm  Fri Sep 16 94
        pre.h              511 bytes   2:00 pm  Fri Sep 16 94
        readme.txt         328 bytes   2:00 pm  Fri Sep 16 94
        resource.h         935 bytes   2:00 pm  Fri Sep 16 94
        simpcntr.cpp    11,048 bytes   2:00 pm  Fri Sep 16 94
        simpcntr.def       642 bytes   2:00 pm  Fri Sep 16 94
        simpcntr.h         778 bytes   2:00 pm  Fri Sep 16 94
        simpcntr.ico       766 bytes   2:00 pm  Fri Sep 16 94
        simpcntr.mak     3,079 bytes   2:00 pm  Fri Sep 16 94
        simpcntr.rc      2,605 bytes   2:00 pm  Fri Sep 16 94
        site.cpp        15,969 bytes   2:00 pm  Fri Sep 16 94
        site.h           1,299 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\OLE2\SAMPLES\SIMPDND
        .                  <DIR>       11:11 am  Mon Jun 10 96
        ..                 <DIR>       11:11 am  Mon Jun 10 96
        app.cpp         17,457 bytes   2:00 pm  Fri Sep 16 94
        app.h            2,006 bytes   2:00 pm  Fri Sep 16 94
        doc.cpp         18,382 bytes   2:00 pm  Fri Sep 16 94
        doc.h            2,090 bytes   2:00 pm  Fri Sep 16 94
        dxferobj.cpp    15,188 bytes   2:00 pm  Fri Sep 16 94
        dxferobj.h       1,934 bytes   2:00 pm  Fri Sep 16 94
        ias.cpp          7,591 bytes   2:00 pm  Fri Sep 16 94
        ias.h            1,159 bytes   2:00 pm  Fri Sep 16 94
        ids.cpp          8,315 bytes   2:00 pm  Fri Sep 16 94
        ids.h              995 bytes   2:00 pm  Fri Sep 16 94
        idt.cpp         23,246 bytes   2:00 pm  Fri Sep 16 94
        idt.h            2,333 bytes   2:00 pm  Fri Sep 16 94
        iocs.cpp         8,363 bytes   2:00 pm  Fri Sep 16 94
        iocs.h           1,196 bytes   2:00 pm  Fri Sep 16 94
        makefile         2,751 bytes   2:00 pm  Fri Sep 16 94
        pre.cpp            304 bytes   2:00 pm  Fri Sep 16 94
        pre.h              510 bytes   2:00 pm  Fri Sep 16 94
        readme.txt         194 bytes   2:00 pm  Fri Sep 16 94
        resource.h         979 bytes   2:00 pm  Fri Sep 16 94
        simpdnd.cpp     11,443 bytes   2:00 pm  Fri Sep 16 94
        simpdnd.def        640 bytes   2:00 pm  Fri Sep 16 94
        simpdnd.h          779 bytes   2:00 pm  Fri Sep 16 94
        simpdnd.ico        766 bytes   2:00 pm  Fri Sep 16 94
        simpdnd.mak      3,135 bytes   2:00 pm  Fri Sep 16 94
        simpdnd.rc       2,646 bytes   2:00 pm  Fri Sep 16 94
        site.cpp        15,099 bytes   2:00 pm  Fri Sep 16 94
        site.h           1,147 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\OLE2\SAMPLES\SIMPSVR
```

```
.                   <DIR>       11:11 am  Mon Jun 10 96
..                  <DIR>       11:11 am  Mon Jun 10 96
app.cpp          19,622 bytes    2:00 pm  Fri Sep 16 94
app.h             2,650 bytes    2:00 pm  Fri Sep 16 94
doc.cpp          16,105 bytes    2:00 pm  Fri Sep 16 94
doc.h             1,630 bytes    2:00 pm  Fri Sep 16 94
icf.cpp           5,028 bytes    2:00 pm  Fri Sep 16 94
icf.h             1,094 bytes    2:00 pm  Fri Sep 16 94
ido.cpp          13,857 bytes    2:00 pm  Fri Sep 16 94
ido.h             1,553 bytes    2:00 pm  Fri Sep 16 94
iec.cpp           4,768 bytes    2:00 pm  Fri Sep 16 94
iec.h             1,210 bytes    2:00 pm  Fri Sep 16 94
ioipao.cpp        9,757 bytes    2:00 pm  Fri Sep 16 94
ioipao.h          1,441 bytes    2:00 pm  Fri Sep 16 94
ioipo.cpp        10,338 bytes    2:00 pm  Fri Sep 16 94
ioipo.h           1,162 bytes    2:00 pm  Fri Sep 16 94
ioo.cpp          26,750 bytes    2:00 pm  Fri Sep 16 94
ioo.h             2,377 bytes    2:00 pm  Fri Sep 16 94
ips.cpp          14,834 bytes    2:00 pm  Fri Sep 16 94
ips.h             1,382 bytes    2:00 pm  Fri Sep 16 94
makefile          2,760 bytes    2:00 pm  Fri Sep 16 94
obj.cpp          26,379 bytes    2:00 pm  Fri Sep 16 94
obj.h             5,341 bytes    2:00 pm  Fri Sep 16 94
pre.cpp             304 bytes    2:00 pm  Fri Sep 16 94
pre.h               510 bytes    2:00 pm  Fri Sep 16 94
readme.txt          260 bytes    2:00 pm  Fri Sep 16 94
resource.h        1,114 bytes    2:00 pm  Fri Sep 16 94
simpsvr.cpp       9,802 bytes    2:00 pm  Fri Sep 16 94
simpsvr.def         641 bytes    2:00 pm  Fri Sep 16 94
simpsvr.h           774 bytes    2:00 pm  Fri Sep 16 94
simpsvr.ico         766 bytes    2:00 pm  Fri Sep 16 94
simpsvr.mak       3,367 bytes    2:00 pm  Fri Sep 16 94
simpsvr.rc        2,700 bytes    2:00 pm  Fri Sep 16 94
simpsvr.reg       1,464 bytes    2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\OLE2\SAMPLES\SPOLY
.                   <DIR>       11:11 am  Mon Jun 10 96
..                  <DIR>       11:11 am  Mon Jun 10 96
cenumpt.cpp       5,433 bytes    2:00 pm  Fri Sep 16 94
cenumpt.h         1,002 bytes    2:00 pm  Fri Sep 16 94
clsid.c             724 bytes    2:00 pm  Fri Sep 16 94
clsid.h             893 bytes    2:00 pm  Fri Sep 16 94
cpoint.cpp        8,780 bytes    2:00 pm  Fri Sep 16 94
cpoint.h          2,691 bytes    2:00 pm  Fri Sep 16 94
cpoly.cpp        21,792 bytes    2:00 pm  Fri Sep 16 94
cpoly.h           4,684 bytes    2:00 pm  Fri Sep 16 94
hostenv.h         1,072 bytes    2:00 pm  Fri Sep 16 94
makefile          2,480 bytes    2:00 pm  Fri Sep 16 94
misc.cpp          7,817 bytes    2:00 pm  Fri Sep 16 94
readme.txt        1,794 bytes    2:00 pm  Fri Sep 16 94
resource.h        1,248 bytes    2:00 pm  Fri Sep 16 94
spoly.def           221 bytes    2:00 pm  Fri Sep 16 94
spoly.h           1,496 bytes    2:00 pm  Fri Sep 16 94
spoly.ico           766 bytes    2:00 pm  Fri Sep 16 94
```

```
        spoly.mak        2,785 bytes   2:00 pm  Fri Sep 16 94
        spoly.rc           250 bytes   2:00 pm  Fri Sep 16 94
        spoly.reg          984 bytes   2:00 pm  Fri Sep 16 94
        statbar.cpp      7,390 bytes   2:00 pm  Fri Sep 16 94
        statbar.h        2,593 bytes   2:00 pm  Fri Sep 16 94
        winmain.cpp      4,665 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\OLE2\SAMPLES\SPOLY2
        .                  <DIR>      11:11 am  Mon Jun 10 96
        ..                 <DIR>      11:11 am  Mon Jun 10 96
        cenumpt.cpp      5,427 bytes   2:00 pm  Fri Sep 16 94
        cenumpt.h        1,002 bytes   2:00 pm  Fri Sep 16 94
        clsid.c            724 bytes   2:00 pm  Fri Sep 16 94
        clsid.h            897 bytes   2:00 pm  Fri Sep 16 94
        cpoint.cpp       7,885 bytes   2:00 pm  Fri Sep 16 94
        cpoint.h         3,159 bytes   2:00 pm  Fri Sep 16 94
        cpoly.cpp       17,800 bytes   2:00 pm  Fri Sep 16 94
        cpoly.h          5,614 bytes   2:00 pm  Fri Sep 16 94
        hostenv.h        1,072 bytes   2:00 pm  Fri Sep 16 94
        makefile         2,581 bytes   2:00 pm  Fri Sep 16 94
        misc.cpp         5,814 bytes   2:00 pm  Fri Sep 16 94
        readme.txt       1,897 bytes   2:00 pm  Fri Sep 16 94
        resource.h       1,219 bytes   2:00 pm  Fri Sep 16 94
        spoly.h          1,404 bytes   2:00 pm  Fri Sep 16 94
        spoly2.def         225 bytes   2:00 pm  Fri Sep 16 94
        spoly2.ico         766 bytes   2:00 pm  Fri Sep 16 94
        spoly2.mak       2,934 bytes   2:00 pm  Fri Sep 16 94
        spoly2.rc          251 bytes   2:00 pm  Fri Sep 16 94
        spoly2.reg         994 bytes   2:00 pm  Fri Sep 16 94
        statbar.cpp      7,439 bytes   2:00 pm  Fri Sep 16 94
        statbar.h        2,593 bytes   2:00 pm  Fri Sep 16 94
        tdata.cpp        5,707 bytes   2:00 pm  Fri Sep 16 94
        winmain.cpp      4,669 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\OLE2\SAMPLES\SUMINFO
        .                  <DIR>      11:11 am  Mon Jun 10 96
        ..                 <DIR>      11:11 am  Mon Jun 10 96
        makefile         1,494 bytes   2:00 pm  Fri Sep 16 94
        readme.txt         195 bytes   2:00 pm  Fri Sep 16 94
        suminfo.def        199 bytes   2:00 pm  Fri Sep 16 94
        suminfo.mak      1,824 bytes   2:00 pm  Fri Sep 16 94
        testsumi.cpp    11,450 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\OLE2\SAMPLES\TIBROWSE
        .                  <DIR>      11:11 am  Mon Jun 10 96
        ..                 <DIR>      11:11 am  Mon Jun 10 96
        makefile         2,715 bytes   2:00 pm  Fri Sep 16 94
        readme.txt         834 bytes   2:00 pm  Fri Sep 16 94
        resource.h       1,405 bytes   2:00 pm  Fri Sep 16 94
        tibrowse.cpp    23,011 bytes   2:00 pm  Fri Sep 16 94
        tibrowse.def       347 bytes   2:00 pm  Fri Sep 16 94
        tibrowse.h       2,041 bytes   2:00 pm  Fri Sep 16 94
        tibrowse.ico       766 bytes   2:00 pm  Fri Sep 16 94
        tibrowse.mak     2,065 bytes   2:00 pm  Fri Sep 16 94
```

```
        tibrowse.rc       1,401 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\OLE2\SAMPLES\XSERVER
        .                   <DIR>       11:11 am  Mon Jun 10 96
        ..                  <DIR>       11:11 am  Mon Jun 10 96
        cdebug.cpp       29,649 bytes   2:00 pm  Fri Sep 16 94
        cdebug.h          1,880 bytes   2:00 pm  Fri Sep 16 94
        classfct.cpp      5,459 bytes   2:00 pm  Fri Sep 16 94
        classfct.h        1,366 bytes   2:00 pm  Fri Sep 16 94
        dispatch.cpp     10,003 bytes   2:00 pm  Fri Sep 16 94
        dispatch.h        1,419 bytes   2:00 pm  Fri Sep 16 94
        enumstat.cpp      6,569 bytes   2:00 pm  Fri Sep 16 94
        enumstat.h        1,295 bytes   2:00 pm  Fri Sep 16 94
        intrface.cpp     52,226 bytes   2:00 pm  Fri Sep 16 94
        makefile          2,320 bytes   2:00 pm  Fri Sep 16 94
        MANIP               <DIR>       11:11 am  Mon Jun 10 96
        readme.txt        3,098 bytes   2:00 pm  Fri Sep 16 94
        resource.h          396 bytes   2:00 pm  Fri Sep 16 94
        utils.cpp         8,045 bytes   2:00 pm  Fri Sep 16 94
        utils.h           1,473 bytes   2:00 pm  Fri Sep 16 94
        xobject.cpp      26,652 bytes   2:00 pm  Fri Sep 16 94
        xobject.h        14,609 bytes   2:00 pm  Fri Sep 16 94
        xserver.cpp       1,755 bytes   2:00 pm  Fri Sep 16 94
        xserver.def         471 bytes   2:00 pm  Fri Sep 16 94
        xserver.h         1,067 bytes   2:00 pm  Fri Sep 16 94
        xserver.ico         766 bytes   2:00 pm  Fri Sep 16 94
        xserver.mak       3,029 bytes   2:00 pm  Fri Sep 16 94
        xserver.rc        1,312 bytes   2:00 pm  Fri Sep 16 94
        xserver.reg       4,421 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\OLE2\SAMPLES\XSERVER\MANIP
        .                   <DIR>       11:11 am  Mon Jun 10 96
        ..                  <DIR>       11:11 am  Mon Jun 10 96
        manip.frm         5,434 bytes   2:00 pm  Fri Sep 16 94
        manip.mak           121 bytes   2:00 pm  Fri Sep 16 94
        readme.txt        2,564 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\PEN
        .                   <DIR>       11:12 am  Mon Jun 10 96
        ..                  <DIR>       11:19 am  Mon Jun 10 96
        mars.dll        146,240 bytes   2:00 pm  Fri Sep 16 94
        mars.mob        139,600 bytes   2:00 pm  Fri Sep 16 94
        msmouse.drv       6,752 bytes   2:00 pm  Fri Sep 16 94
        penwin.dll      130,816 bytes   2:00 pm  Fri Sep 16 94
        penwin.ini          392 bytes   2:00 pm  Fri Sep 16 94
        readme.txt       16,209 bytes   2:00 pm  Fri Sep 16 94
        vgap.drv         72,320 bytes   2:00 pm  Fri Sep 16 94
        yesmouse.drv        416 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\PHARLAP
        .                   <DIR>       11:12 am  Mon Jun 10 96
        ..                  <DIR>       11:19 am  Mon Jun 10 96
        BIN                 <DIR>       11:12 am  Mon Jun 10 96
        DOC                 <DIR>       11:12 am  Mon Jun 10 96
```

```
        EXAMPLES            <DIR>      11:12 am  Mon Jun 10 96
        INCLUDE             <DIR>      11:12 am  Mon Jun 10 96
        MSVC                <DIR>      11:12 am  Mon Jun 10 96
        read.me           6,912 bytes   2:00 pm  Fri Sep 16 94
        tntlite.ico         766 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\PHARLAP\BIN
        .                   <DIR>      11:12 am  Mon Jun 10 96
        ..                  <DIR>      11:12 am  Mon Jun 10 96
        bind286.exe      92,514 bytes   2:00 pm  Fri Sep 16 94
        cfig286.exe      43,315 bytes   2:00 pm  Fri Sep 16 94
        doscalls.dll      9,861 bytes   2:00 pm  Fri Sep 16 94
        gdi.dll           3,438 bytes   2:00 pm  Fri Sep 16 94
        gorun286.exe     11,113 bytes   2:00 pm  Fri Sep 16 94
        int33.dll         3,161 bytes   2:00 pm  Fri Sep 16 94
        kbdcalls.dll      3,107 bytes   2:00 pm  Fri Sep 16 94
        kernel.dll        3,438 bytes   2:00 pm  Fri Sep 16 94
        keyboard.dll      3,438 bytes   2:00 pm  Fri Sep 16 94
        lite286.exe     214,844 bytes   2:00 pm  Fri Sep 16 94
        moucalls.dll      6,264 bytes   2:00 pm  Fri Sep 16 94
        pharlap.386       9,343 bytes   2:00 pm  Fri Sep 16 94
        tellme.exe       55,641 bytes   2:00 pm  Fri Sep 16 94
        tellprot.exe    235,903 bytes   2:00 pm  Fri Sep 16 94
        user.dll          3,438 bytes   2:00 pm  Fri Sep 16 94
        viocalls.dll      5,647 bytes   2:00 pm  Fri Sep 16 94
        win87em.dll       3,438 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\PHARLAP\DOC
        .                   <DIR>      11:12 am  Mon Jun 10 96
        ..                  <DIR>      11:12 am  Mon Jun 10 96
        intl.dlr          1,939 bytes   2:00 pm  Fri Sep 16 94
        license.doc       5,798 bytes   2:00 pm  Fri Sep 16 94
        lite286.doc      68,642 bytes   2:00 pm  Fri Sep 16 94
        lite286.hlp      51,761 bytes   2:00 pm  Fri Sep 16 94
        liteinfo          3,894 bytes   2:00 pm  Fri Sep 16 94
        order.frm         1,308 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\PHARLAP\EXAMPLES
        .                   <DIR>      11:12 am  Mon Jun 10 96
        ..                  <DIR>      11:12 am  Mon Jun 10 96
        big.c               727 bytes   2:00 pm  Fri Sep 16 94
        big.exe          30,764 bytes   2:00 pm  Fri Sep 16 94
        dosmem.c            502 bytes   2:00 pm  Fri Sep 16 94
        dosmem.exe       30,252 bytes   2:00 pm  Fri Sep 16 94
        examples.def        250 bytes   2:00 pm  Fri Sep 16 94
        fullscrn.c        4,834 bytes   2:00 pm  Fri Sep 16 94
        fullscrn.h        1,606 bytes   2:00 pm  Fri Sep 16 94
        hello.c             238 bytes   2:00 pm  Fri Sep 16 94
        hello.exe        29,692 bytes   2:00 pm  Fri Sep 16 94
        keys.c            2,020 bytes   2:00 pm  Fri Sep 16 94
        keys.exe         31,326 bytes   2:00 pm  Fri Sep 16 94
        keys2.c           1,740 bytes   2:00 pm  Fri Sep 16 94
        makefile            846 bytes   2:00 pm  Fri Sep 16 94
        memtest.c           689 bytes   2:00 pm  Fri Sep 16 94
```

```
        memtest2.cpp      1,180 bytes   2:00 pm  Fri Sep 16 94
        vidtest.c           660 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\PHARLAP\INCLUDE
        .                   <DIR>       11:12 am  Mon Jun 10 96
        ..                  <DIR>       11:12 am  Mon Jun 10 96
        phapi.h           9,300 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\PHARLAP\MSVC
        .                   <DIR>       11:12 am  Mon Jun 10 96
        ..                  <DIR>       11:12 am  Mon Jun 10 96
        LIB                 <DIR>       11:12 am  Mon Jun 10 96

I:\VC152\MSVC15\PHARLAP\MSVC\LIB
        .                   <DIR>       11:12 am  Mon Jun 10 96
        ..                  <DIR>       11:12 am  Mon Jun 10 96
        apisim.lib        2,048 bytes   2:00 pm  Fri Sep 16 94
        example.def         957 bytes   2:00 pm  Fri Sep 16 94
        extmods.lbc         367 bytes   2:00 pm  Fri Sep 16 94
        fp7.lbc              27 bytes   2:00 pm  Fri Sep 16 94
        fp7.obj             486 bytes   2:00 pm  Fri Sep 16 94
        fpa.lbc              12 bytes   2:00 pm  Fri Sep 16 94
        fpa.obj             105 bytes   2:00 pm  Fri Sep 16 94
        fpe.lbc              27 bytes   2:00 pm  Fri Sep 16 94
        fpe.obj             535 bytes   2:00 pm  Fri Sep 16 94
        graphp.obj          196 bytes   2:00 pm  Fri Sep 16 94
        insmods.lbc         488 bytes   2:00 pm  Fri Sep 16 94
        mkall.bat           549 bytes   2:00 pm  Fri Sep 16 94
        mklib.bat         2,557 bytes   2:00 pm  Fri Sep 16 94
        msc8c.lib        19,143 bytes   2:00 pm  Fri Sep 16 94
        msc8l.lib        19,143 bytes   2:00 pm  Fri Sep 16 94
        msc8m.lib        18,631 bytes   2:00 pm  Fri Sep 16 94
        msc8s.lib        18,119 bytes   2:00 pm  Fri Sep 16 94
        phapi.lib        22,171 bytes   2:00 pm  Fri Sep 16 94
        remobjs.bat         451 bytes   2:00 pm  Fri Sep 16 94
        wilsys.lib        1,536 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\PSS-SAMP
        .                   <DIR>       11:12 am  Mon Jun 10 96
        ..                  <DIR>       11:19 am  Mon Jun 10 96
        DYNSET              <DIR>       11:12 am  Mon Jun 10 96

I:\VC152\MSVC15\PSS-SAMP\DYNSET
        .                   <DIR>       11:12 am  Mon Jun 10 96
        ..                  <DIR>       11:12 am  Mon Jun 10 96
        dyna.h            3,711 bytes    5:24 am  Sat Mar 11 95
        dynacore.cpp     39,330 bytes    5:24 am  Sat Mar 11 95
        dynarfx.cpp      11,875 bytes    6:03 pm  Fri Feb  3 95
        dynaview.cpp      2,585 bytes   11:29 pm  Wed Dec 21 94
        ENROLL              <DIR>       11:12 am  Mon Jun 10 96
        readme.txt        7,212 bytes    5:31 am  Sat Mar 11 95

I:\VC152\MSVC15\PSS-SAMP\DYNSET\ENROLL
        .                   <DIR>       11:12 am  Mon Jun 10 96
```

```
..                      <DIR>       11:12 am  Mon Jun 10 96
STEP3                   <DIR>       11:12 am  Mon Jun 10 96

I:\VC152\MSVC15\PSS-SAMP\DYNSET\ENROLL\STEP3
.                       <DIR>       11:12 am  Mon Jun 10 96
..                      <DIR>       11:12 am  Mon Jun 10 96
coursset.cpp          1,052 bytes    4:47 am  Thu Dec 29 94
coursset.h              689 bytes    4:37 am  Thu Dec 29 94
dyna.h                3,711 bytes    5:24 am  Sat Mar 11 95
dynacore.cpp         39,330 bytes    5:24 am  Sat Mar 11 95
dynarfx.cpp          11,875 bytes    6:03 pm  Fri Feb  3 95
dynaview.cpp          2,585 bytes   11:29 pm  Wed Dec 21 94
enroldoc.cpp          1,492 bytes    9:00 pm  Fri Sep 16 94
enroldoc.h              923 bytes    9:00 pm  Fri Sep 16 94
enroll.clw            3,130 bytes    9:00 pm  Fri Sep 16 94
enroll.cpp            3,348 bytes    9:00 pm  Fri Sep 16 94
enroll.def              346 bytes    9:00 pm  Fri Sep 16 94
enroll.h                819 bytes    9:00 pm  Fri Sep 16 94
enroll.mak            3,363 bytes   12:18 am  Tue Jun 13 95
enroll.rc             9,096 bytes    9:00 pm  Fri Sep 16 94
mainfrm.cpp           2,582 bytes    9:00 pm  Fri Sep 16 94
mainfrm.h               982 bytes    9:00 pm  Fri Sep 16 94
readme.txt            4,188 bytes    9:00 pm  Fri Sep 16 94
RES                     <DIR>       11:12 am  Mon Jun 10 96
resource.h              983 bytes    9:00 pm  Fri Sep 16 94
sectform.cpp          6,728 bytes    4:47 am  Thu Dec 29 94
sectform.h            1,756 bytes    4:22 am  Thu Dec 29 94
sectset.cpp           1,382 bytes    4:30 am  Thu Dec 29 94
sectset.h               791 bytes    4:37 am  Thu Dec 29 94
stdafx.cpp              204 bytes    9:00 pm  Fri Sep 16 94
stdafx.h                369 bytes    4:28 am  Thu Dec 29 94

I:\VC152\MSVC15\PSS-SAMP\DYNSET\ENROLL\STEP3\RES
.                       <DIR>       11:12 am  Mon Jun 10 96
..                      <DIR>       11:12 am  Mon Jun 10 96
enroll.ico              768 bytes    9:00 pm  Fri Sep 16 94
enroll.rc2            1,543 bytes    9:00 pm  Fri Sep 16 94
toolbar.bmp           1,558 bytes    9:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\REDIST
.                       <DIR>       11:14 am  Mon Jun 10 96
..                      <DIR>       11:19 am  Mon Jun 10 96
12500852.cpx          2,320 bytes    2:10 pm  Fri Jan 13 95
12510866.cpx          2,318 bytes    2:10 pm  Fri Jan 13 95
12520437.cpx          2,151 bytes    2:10 pm  Fri Jan 13 95
12520850.cpx          2,233 bytes    2:10 pm  Fri Jan 13 95
12520860.cpx          2,167 bytes    2:10 pm  Fri Jan 13 95
12520861.cpx          2,162 bytes    2:10 pm  Fri Jan 13 95
12520863.cpx          2,173 bytes    2:10 pm  Fri Jan 13 95
12520865.cpx          2,147 bytes    2:10 pm  Fri Jan 13 95
btrv200.dll         111,616 bytes    2:10 pm  Fri Jan 13 95
commdlg.dan          89,072 bytes    2:00 pm  Fri Sep 16 94
commdlg.dll          89,248 bytes    2:00 pm  Fri Sep 16 94
commdlg.dut          89,408 bytes    2:00 pm  Fri Sep 16 94
```

```
commdlg.fin      89,152 bytes   2:00 pm  Fri Sep 16 94
commdlg.frn      89,920 bytes   2:00 pm  Fri Sep 16 94
commdlg.ger      90,144 bytes   2:00 pm  Fri Sep 16 94
commdlg.itn      89,488 bytes   2:00 pm  Fri Sep 16 94
commdlg.nor      89,008 bytes   2:00 pm  Fri Sep 16 94
commdlg.por      89,872 bytes   2:00 pm  Fri Sep 16 94
commdlg.spa      90,320 bytes   2:00 pm  Fri Sep 16 94
commdlg.swe      89,104 bytes   2:00 pm  Fri Sep 16 94
compobj.dll     108,544 bytes   2:10 pm  Fri Jan 13 95
cpn16ut.dll       3,264 bytes   2:10 pm  Fri Jan 13 95
ctl3d.dll        26,768 bytes   1:43 am  Fri Jul 14 95
ctl3dv2.dll      27,632 bytes   1:43 am  Fri Jul 14 95
dbnmp3.dll       10,944 bytes   2:10 pm  Fri Jan 13 95
ddeml.dll        36,864 bytes   2:00 pm  Fri Sep 16 94
dib.drv          29,536 bytes   2:00 pm  Fri Sep 16 94
drvora7.hlp      47,470 bytes   2:10 pm  Fri Jan 13 95
drvoracl.hlp     45,610 bytes   2:10 pm  Fri Jan 13 95
drvssrvr.hlp    105,964 bytes   2:10 pm  Fri Jan 13 95
expand.exe       15,285 bytes   2:00 pm  Fri Sep 16 94
instcat.sql      91,480 bytes   2:10 pm  Fri Jan 13 95
lzexpand.dll      9,936 bytes   2:00 pm  Fri Sep 16 94
mcipionr.drv     17,408 bytes   2:00 pm  Fri Sep 16 94
mfc250.dll      322,384 bytes  11:15 pm  Wed Apr 26 95
mfc250.sym       96,260 bytes  11:22 pm  Wed Apr 26 95
mfcd250.dll      51,936 bytes  11:21 pm  Wed Apr 26 95
mfcd250.sym       8,516 bytes  11:22 pm  Wed Apr 26 95
mfcn250.dll      11,088 bytes  11:22 pm  Wed Apr 26 95
mfcn250.sym       5,012 bytes  11:22 pm  Wed Apr 26 95
mfco250.dll     125,856 bytes  11:20 pm  Wed Apr 26 95
mfco250.sym      51,732 bytes  11:22 pm  Wed Apr 26 95
mfcoleui.dll    146,976 bytes  10:33 pm  Wed Apr 26 95
mfcoleui.sym     13,908 bytes  10:33 pm  Wed Apr 26 95
msajt200.dll    995,056 bytes   2:10 pm  Fri Jan 13 95
mscpxlt.dll      10,304 bytes   2:10 pm  Fri Jan 13 95
msjetdsp.dll     85,792 bytes   2:10 pm  Fri Jan 13 95
msjeterr.dll     11,232 bytes   2:10 pm  Fri Jan 13 95
msjetint.dll     15,936 bytes   2:10 pm  Fri Jan 13 95
mstx2016.dll    102,080 bytes   2:10 pm  Fri Jan 13 95
msxl2016.dll    149,344 bytes   2:10 pm  Fri Jan 13 95
odbc.dll         56,240 bytes   2:10 pm  Fri Jan 13 95
odbc.inf         16,584 bytes   2:10 pm  Fri Jan 13 95
odbc16ut.dll      5,792 bytes   2:10 pm  Fri Jan 13 95
odbc32.dll       12,800 bytes   2:10 pm  Fri Jan 13 95
odbc3216.dll     12,288 bytes   2:10 pm  Fri Jan 13 95
odbcadm.exe       6,464 bytes   2:10 pm  Fri Jan 13 95
odbccp32.dll      5,632 bytes   2:10 pm  Fri Jan 13 95
odbccurs.dll     88,896 bytes   2:10 pm  Fri Jan 13 95
odbcinst.dll     92,576 bytes   2:10 pm  Fri Jan 13 95
odbcinst.hlp     17,412 bytes   2:10 pm  Fri Jan 13 95
odbcjet.hlp     113,064 bytes   2:10 pm  Fri Jan 13 95
odbcjt16.dll    246,928 bytes   2:10 pm  Fri Jan 13 95
odbcjtnw.hlp     83,833 bytes   2:10 pm  Fri Jan 13 95
odbcstp.exe      72,896 bytes   2:10 pm  Fri Jan 13 95
odbctl16.dll     64,080 bytes   2:10 pm  Fri Jan 13 95
```

```
odbtrv16.dll      4,096 bytes   2:10 pm  Fri Jan 13 95
oddbse16.dll      4,080 bytes   2:10 pm  Fri Jan 13 95
odexl16.dll       4,080 bytes   2:10 pm  Fri Jan 13 95
odfox16.dll       4,096 bytes   2:10 pm  Fri Jan 13 95
odpdx16.dll       4,096 bytes   2:10 pm  Fri Jan 13 95
odtext16.dll      4,096 bytes   2:10 pm  Fri Jan 13 95
oemsetup.inf        351 bytes   2:00 pm  Fri Sep 16 94
ole2.dll        302,592 bytes   2:10 pm  Fri Jan 13 95
ole2.reg         27,026 bytes   2:10 pm  Fri Jan 13 95
ole2conv.dll     57,328 bytes   2:10 pm  Fri Jan 13 95
ole2disp.dll    164,832 bytes   2:10 pm  Fri Jan 13 95
ole2nls.dll     150,976 bytes   2:10 pm  Fri Jan 13 95
ole2prox.dll     51,712 bytes   2:10 pm  Fri Jan 13 95
olecli.dll       83,456 bytes   2:00 pm  Fri Sep 16 94
olesvr.dll       24,064 bytes   2:00 pm  Fri Sep 16 94
ora6win.dll     285,573 bytes   2:10 pm  Fri Jan 13 95
ora7note.wri     51,072 bytes   2:10 pm  Fri Jan 13 95
oracle.txt       13,914 bytes   2:10 pm  Fri Jan 13 95
pdx200.dll      233,328 bytes   2:10 pm  Fri Jan 13 95
penwin.dll      130,816 bytes   2:00 pm  Fri Sep 16 94
redistrb.wri     12,800 bytes   2:10 pm  Fri Jan 13 95
regload.exe      16,752 bytes   2:00 pm  Fri Sep 16 94
setup.exe        24,624 bytes   2:10 pm  Fri Jan 13 95
setup.lst           303 bytes   2:10 pm  Fri Jan 13 95
shell.dll        41,600 bytes   2:00 pm  Fri Sep 16 94
smallb.fon       22,016 bytes   2:00 pm  Fri Sep 16 94
smalle.fon       26,112 bytes   2:00 pm  Fri Sep 16 94
smallf.fon       21,504 bytes   2:00 pm  Fri Sep 16 94
sqlsrvr.dll     161,392 bytes   2:10 pm  Fri Jan 13 95
sqora.dll       144,096 bytes   2:10 pm  Fri Jan 13 95
sqora7.dll      181,520 bytes   2:10 pm  Fri Jan 13 95
sqorast7.dll      9,552 bytes   2:10 pm  Fri Jan 13 95
sqorastp.dll      9,632 bytes   2:10 pm  Fri Jan 13 95
stdole.tlb        4,304 bytes   2:10 pm  Fri Jan 13 95
storage.dll     157,696 bytes   2:10 pm  Fri Jan 13 95
stress.dll       50,400 bytes   2:00 pm  Fri Sep 16 94
toolhelp.dll     14,128 bytes   2:00 pm  Fri Sep 16 94
typelib.dll     177,216 bytes   2:10 pm  Fri Jan 13 95
vaen2.olb        41,124 bytes   2:10 pm  Fri Jan 13 95
vbajet.dll        1,984 bytes   2:10 pm  Fri Jan 13 95
vbar2.dll       298,880 bytes   2:10 pm  Fri Jan 13 95
ver.dll           9,008 bytes   2:00 pm  Fri Sep 16 94
vtd.386           6,816 bytes   2:00 pm  Fri Sep 16 94
winhelp.exe     256,192 bytes   2:00 pm  Fri Sep 16 94
winhelp.hlp      31,385 bytes   2:00 pm  Fri Sep 16 94
winmem32.dll      4,320 bytes   2:00 pm  Fri Sep 16 94
xbs200.dll      296,832 bytes   2:10 pm  Fri Jan 13 95
_mssetup.exe      9,813 bytes   2:10 pm  Fri Jan 13 95

I:\VC152\MSVC15\SAMPLES
.                 <DIR>         11:19 am  Mon Jun 10 96
..                <DIR>         11:19 am  Mon Jun 10 96
BITMAP            <DIR>         11:14 am  Mon Jun 10 96
BLANDMDI          <DIR>         11:15 am  Mon Jun 10 96
```

```
BOUNCER         <DIR>      11:15 am  Mon Jun 10 96
CDDEMO          <DIR>      11:15 am  Mon Jun 10 96
CLIDEMO         <DIR>      11:15 am  Mon Jun 10 96
CLIPTEXT        <DIR>      11:15 am  Mon Jun 10 96
COMMDLG         <DIR>      11:15 am  Mon Jun 10 96
CROPDIB         <DIR>      11:15 am  Mon Jun 10 96
CURSOR          <DIR>      11:15 am  Mon Jun 10 96
DBWIN           <DIR>      11:15 am  Mon Jun 10 96
DDE             <DIR>      11:15 am  Mon Jun 10 96
DDEML           <DIR>      11:15 am  Mon Jun 10 96
DEFPROCS        <DIR>      11:15 am  Mon Jun 10 96
DIBIT           <DIR>      11:15 am  Mon Jun 10 96
DIBVIEW         <DIR>      11:16 am  Mon Jun 10 96
DRAGDROP        <DIR>      11:16 am  Mon Jun 10 96
EDITCNTL        <DIR>      11:16 am  Mon Jun 10 96
EDITFILE        <DIR>      11:16 am  Mon Jun 10 96
EDITMENU        <DIR>      11:16 am  Mon Jun 10 96
EXPENSE         <DIR>      11:16 am  Mon Jun 10 96
FILEOPEN        <DIR>      11:16 am  Mon Jun 10 96
FONTEST         <DIR>      11:16 am  Mon Jun 10 96
GDOSMEM         <DIR>      11:16 am  Mon Jun 10 96
GENERIC         <DIR>      11:16 am  Mon Jun 10 96
GWAPI           <DIR>      11:16 am  Mon Jun 10 96
HANDLER         <DIR>      11:16 am  Mon Jun 10 96
HELPEX          <DIR>      11:16 am  Mon Jun 10 96
HELPHOOK        <DIR>      11:16 am  Mon Jun 10 96
HFORM           <DIR>      11:16 am  Mon Jun 10 96
HOOKS           <DIR>      11:16 am  Mon Jun 10 96
ICON            <DIR>      11:16 am  Mon Jun 10 96
INPUT           <DIR>      11:16 am  Mon Jun 10 96
INSTVER         <DIR>      11:17 am  Mon Jun 10 96
LISTHORZ        <DIR>      11:17 am  Mon Jun 10 96
LOWPASS         <DIR>      11:17 am  Mon Jun 10 96
MACROHLP        <DIR>      11:17 am  Mon Jun 10 96
MAKEAPP         <DIR>      11:17 am  Mon Jun 10 96
MCITEST         <DIR>      11:17 am  Mon Jun 10 96
MEMORY          <DIR>      11:17 am  Mon Jun 10 96
MENU            <DIR>      11:17 am  Mon Jun 10 96
MIDIMON         <DIR>      11:17 am  Mon Jun 10 96
MULTIPAD        <DIR>      11:17 am  Mon Jun 10 96
MUSCROLL        <DIR>      11:17 am  Mon Jun 10 96
MYPAL           <DIR>      11:17 am  Mon Jun 10 96
OUTPUT          <DIR>      11:17 am  Mon Jun 10 96
OWNCOMBO        <DIR>      11:17 am  Mon Jun 10 96
OWNERB          <DIR>      11:17 am  Mon Jun 10 96
PALETTE         <DIR>      11:18 am  Mon Jun 10 96
PENCNTL         <DIR>      11:18 am  Mon Jun 10 96
PENPAD          <DIR>      11:18 am  Mon Jun 10 96
PRNTFILE        <DIR>      11:18 am  Mon Jun 10 96
PROFILER        <DIR>      11:18 am  Mon Jun 10 96
QWGDEMO         <DIR>      11:18 am  Mon Jun 10 96
REVERSE         <DIR>      11:18 am  Mon Jun 10 96
ROTARY          <DIR>      11:18 am  Mon Jun 10 96
SELECT          <DIR>      11:18 am  Mon Jun 10 96
```

```
        SHOWDIB             <DIR>      11:18 am  Mon Jun 10 96
        SHOWGDI             <DIR>      11:18 am  Mon Jun 10 96
        SNOOP               <DIR>      11:18 am  Mon Jun 10 96
        SORTDEMO            <DIR>      11:18 am  Mon Jun 10 96
        SREC                <DIR>      11:18 am  Mon Jun 10 96
        SRVRDEMO            <DIR>      11:18 am  Mon Jun 10 96
        TDOSMEM             <DIR>      11:18 am  Mon Jun 10 96
        TIMERS              <DIR>      11:18 am  Mon Jun 10 96
        TOOLHELP            <DIR>      11:18 am  Mon Jun 10 96
        TTY                 <DIR>      11:18 am  Mon Jun 10 96
        VER                 <DIR>      11:18 am  Mon Jun 10 96
        VERSTAMP            <DIR>      11:19 am  Mon Jun 10 96
        WINMEM32            <DIR>      11:19 am  Mon Jun 10 96
        WMFDCODE            <DIR>      11:19 am  Mon Jun 10 96
        XTENSION            <DIR>      11:19 am  Mon Jun 10 96

I:\VC152\MSVC15\SAMPLES\BITMAP
        .                   <DIR>      11:14 am  Mon Jun 10 96
        ..                  <DIR>      11:19 am  Mon Jun 10 96
        bitmap.c         17,319 bytes   2:00 pm  Fri Sep 16 94
        bitmap.def          293 bytes   2:00 pm  Fri Sep 16 94
        bitmap.h            343 bytes   2:00 pm  Fri Sep 16 94
        bitmap.mak        2,148 bytes   2:00 pm  Fri Sep 16 94
        bitmap.rc         2,609 bytes   2:00 pm  Fri Sep 16 94
        cat.bmp             662 bytes   2:00 pm  Fri Sep 16 94
        dog.bmp             662 bytes   2:00 pm  Fri Sep 16 94
        resource.h          764 bytes   2:00 pm  Fri Sep 16 94
        select.c          4,557 bytes   2:00 pm  Fri Sep 16 94
        select.dll        5,018 bytes   2:00 pm  Fri Sep 16 94
        select.h            778 bytes   2:00 pm  Fri Sep 16 94
        select.lib        1,536 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\SAMPLES\BLANDMDI
        .                   <DIR>      11:15 am  Mon Jun 10 96
        ..                  <DIR>      11:19 am  Mon Jun 10 96
        blandini.c        4,744 bytes   2:00 pm  Fri Sep 16 94
        blandmdi.c        8,018 bytes   2:00 pm  Fri Sep 16 94
        blandmdi.def        447 bytes   2:00 pm  Fri Sep 16 94
        blandmdi.h        1,106 bytes   2:00 pm  Fri Sep 16 94
        blandmdi.mak      2,283 bytes   2:00 pm  Fri Sep 16 94
        blandmdi.rc       2,199 bytes   2:00 pm  Fri Sep 16 94
        mdichild.ico        766 bytes   2:00 pm  Fri Sep 16 94
        mdiframe.ico        766 bytes   2:00 pm  Fri Sep 16 94
        resource.h          725 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\SAMPLES\BOUNCER
        .                   <DIR>      11:15 am  Mon Jun 10 96
        ..                  <DIR>      11:19 am  Mon Jun 10 96
        bouncer.c        15,361 bytes   2:00 pm  Fri Sep 16 94
        bouncer.def         544 bytes   2:00 pm  Fri Sep 16 94
        bouncer.h           535 bytes   2:00 pm  Fri Sep 16 94
        bouncer.ico         766 bytes   2:00 pm  Fri Sep 16 94
        bouncer.mak       2,107 bytes   2:00 pm  Fri Sep 16 94
        bouncer.rc        6,014 bytes   2:00 pm  Fri Sep 16 94
```

```
        frog.bmp          3,118 bytes   2:00 pm  Fri Sep 16 94
        resource.h        1,479 bytes   2:00 pm  Fri Sep 16 94
        ribbit.wav        5,772 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\SAMPLES\CDDEMO
        .                   <DIR>       11:15 am  Mon Jun 10 96
        ..                  <DIR>       11:19 am  Mon Jun 10 96
        cddemo.c         56,199 bytes   2:00 pm  Fri Sep 16 94
        cddemo.def          864 bytes   2:00 pm  Fri Sep 16 94
        cddemo.h          5,929 bytes   2:00 pm  Fri Sep 16 94
        cddemo.ico          766 bytes   2:00 pm  Fri Sep 16 94
        cddemo.mak        2,254 bytes   2:00 pm  Fri Sep 16 94
        cddemo.rc         5,767 bytes   2:00 pm  Fri Sep 16 94
        dlgbrush.bmp        150 bytes   2:00 pm  Fri Sep 16 94
        error.c           7,200 bytes   2:00 pm  Fri Sep 16 94
        init.c            2,044 bytes   2:00 pm  Fri Sep 16 94
        resource.h        3,126 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\SAMPLES\CLIDEMO
        .                   <DIR>       11:15 am  Mon Jun 10 96
        ..                  <DIR>       11:19 am  Mon Jun 10 96
        clidemo.c        38,233 bytes   2:00 pm  Fri Sep 16 94
        clidemo.def         705 bytes   2:00 pm  Fri Sep 16 94
        clidemo.h         2,052 bytes   2:00 pm  Fri Sep 16 94
        clidemo.ico         766 bytes   2:00 pm  Fri Sep 16 94
        clidemo.mak       3,203 bytes   2:00 pm  Fri Sep 16 94
        clidemo.rc       12,699 bytes   2:00 pm  Fri Sep 16 94
        cliver.rcv          278 bytes   2:00 pm  Fri Sep 16 94
        demorc.h          7,078 bytes   2:00 pm  Fri Sep 16 94
        dialog.c         39,268 bytes   2:00 pm  Fri Sep 16 94
        dialog.h          2,502 bytes   2:00 pm  Fri Sep 16 94
        global.h          4,571 bytes   2:00 pm  Fri Sep 16 94
        object.c         44,167 bytes   2:00 pm  Fri Sep 16 94
        object.h          1,346 bytes   2:00 pm  Fri Sep 16 94
        register.c        7,456 bytes   2:00 pm  Fri Sep 16 94
        register.h          502 bytes   2:00 pm  Fri Sep 16 94
        stream.c         14,607 bytes   2:00 pm  Fri Sep 16 94
        stream.h            923 bytes   2:00 pm  Fri Sep 16 94
        utility.c        17,640 bytes   2:00 pm  Fri Sep 16 94
        utility.h         1,246 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\SAMPLES\CLIPTEXT
        .                   <DIR>       11:15 am  Mon Jun 10 96
        ..                  <DIR>       11:19 am  Mon Jun 10 96
        cliptext.c       13,776 bytes   2:00 pm  Fri Sep 16 94
        cliptext.def        285 bytes   2:00 pm  Fri Sep 16 94
        cliptext.h          291 bytes   2:00 pm  Fri Sep 16 94
        cliptext.mak      1,988 bytes   2:00 pm  Fri Sep 16 94
        cliptext.rc       3,228 bytes   2:00 pm  Fri Sep 16 94
        resource.h          811 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\SAMPLES\COMMDLG
        .                   <DIR>       11:15 am  Mon Jun 10 96
        ..                  <DIR>       11:19 am  Mon Jun 10 96
```

```
        cddrv.bmp          246 bytes   2:00 pm  Fri Sep 16 94
        color.dlg        2,028 bytes   2:00 pm  Fri Sep 16 94
        fileopen.dlg     3,892 bytes   2:00 pm  Fri Sep 16 94
        findtext.dlg     3,001 bytes   2:00 pm  Fri Sep 16 94
        font.bmp           598 bytes   2:00 pm  Fri Sep 16 94
        font.dlg         1,717 bytes   2:00 pm  Fri Sep 16 94
        landscap.ico     1,846 bytes   2:00 pm  Fri Sep 16 94
        portrait.ico     1,846 bytes   2:00 pm  Fri Sep 16 94
        prnsetup.dlg     2,872 bytes   2:00 pm  Fri Sep 16 94
        smflder.bmp        246 bytes   2:00 pm  Fri Sep 16 94
        smfldr1.bmp        246 bytes   2:00 pm  Fri Sep 16 94
        smfldr2.bmp        246 bytes   2:00 pm  Fri Sep 16 94
        smfldr3.bmp        246 bytes   2:00 pm  Fri Sep 16 94
        smfldr4.bmp        246 bytes   2:00 pm  Fri Sep 16 94
        smfldr5.bmp        246 bytes   2:00 pm  Fri Sep 16 94
        smflopy.bmp        246 bytes   2:00 pm  Fri Sep 16 94
        smhard.bmp         246 bytes   2:00 pm  Fri Sep 16 94
        smnet.bmp          246 bytes   2:00 pm  Fri Sep 16 94
        smram.bmp          246 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\SAMPLES\CROPDIB
        .                    <DIR>     11:15 am  Mon Jun 10 96
        ..                   <DIR>     11:19 am  Mon Jun 10 96
        cropdib.c       17,774 bytes   2:00 pm  Fri Sep 16 94
        cropdib.def        279 bytes   2:00 pm  Fri Sep 16 94
        cropdib.h          469 bytes   2:00 pm  Fri Sep 16 94
        cropdib.ico        766 bytes   2:00 pm  Fri Sep 16 94
        cropdib.mak      2,170 bytes   2:00 pm  Fri Sep 16 94
        cropdib.rc       4,198 bytes   2:00 pm  Fri Sep 16 94
        dib.c           39,794 bytes   2:00 pm  Fri Sep 16 94
        dib.h            2,242 bytes   2:00 pm  Fri Sep 16 94
        resource.h       1,120 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\SAMPLES\CURSOR
        .                    <DIR>     11:15 am  Mon Jun 10 96
        ..                   <DIR>     11:19 am  Mon Jun 10 96
        bullseye.cur       326 bytes   2:00 pm  Fri Sep 16 94
        cursor.c        15,246 bytes   2:00 pm  Fri Sep 16 94
        cursor.def         317 bytes   2:00 pm  Fri Sep 16 94
        cursor.h           284 bytes   2:00 pm  Fri Sep 16 94
        cursor.mak       2,030 bytes   2:00 pm  Fri Sep 16 94
        cursor.rc        2,103 bytes   2:00 pm  Fri Sep 16 94
        resource.h         314 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\SAMPLES\DBWIN
        .                    <DIR>     11:15 am  Mon Jun 10 96
        ..                   <DIR>     11:19 am  Mon Jun 10 96
        allocbrk.c       4,806 bytes   2:00 pm  Fri Sep 16 94
        comout.c         2,945 bytes   2:00 pm  Fri Sep 16 94
        dbapi.txt        9,368 bytes   2:00 pm  Fri Sep 16 94
        dbwin.c         14,774 bytes   2:00 pm  Fri Sep 16 94
        dbwin.def          336 bytes   2:00 pm  Fri Sep 16 94
        dbwin.h          1,304 bytes   2:00 pm  Fri Sep 16 94
        dbwin.ico          766 bytes   2:00 pm  Fri Sep 16 94
```

```
        dbwin.mak         2,435 bytes   2:00 pm  Fri Sep 16 94
        dbwin.rc          7,693 bytes   2:00 pm  Fri Sep 16 94
        dbwin.txt        11,407 bytes   2:00 pm  Fri Sep 16 94
        dbwin.ver         2,022 bytes   2:00 pm  Fri Sep 16 94
        dbwindll.c        9,027 bytes   2:00 pm  Fri Sep 16 94
        dbwindll.def        238 bytes   2:00 pm  Fri Sep 16 94
        dbwindll.h        1,049 bytes   2:00 pm  Fri Sep 16 94
        dbwindll.mak      2,667 bytes   2:00 pm  Fri Sep 16 94
        dbwindll.rc          98 bytes   2:00 pm  Fri Sep 16 94
        dbwindll.ver      2,047 bytes   2:00 pm  Fri Sep 16 94
        dbwindlp.h        1,358 bytes   2:00 pm  Fri Sep 16 94
        dlgdefs.h         1,491 bytes   2:00 pm  Fri Sep 16 94
        monoout.c         5,976 bytes   2:00 pm  Fri Sep 16 94
        nfyfmt.c          9,477 bytes   2:00 pm  Fri Sep 16 94
        resource.h        2,515 bytes   2:00 pm  Fri Sep 16 94
        setdb.c           6,793 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\SAMPLES\DDE
        .                   <DIR>       11:15 am  Mon Jun 10 96
        ..                  <DIR>       11:19 am  Mon Jun 10 96
        clidata.c        21,612 bytes   2:00 pm  Fri Sep 16 94
        clidde.c         20,806 bytes   2:00 pm  Fri Sep 16 94
        client.c         24,986 bytes   2:00 pm  Fri Sep 16 94
        client.def          520 bytes   2:00 pm  Fri Sep 16 94
        client.h          2,062 bytes   2:00 pm  Fri Sep 16 94
        client.mak        2,279 bytes   2:00 pm  Fri Sep 16 94
        client.rc         6,661 bytes   2:00 pm  Fri Sep 16 94
        clires.h          1,120 bytes   2:00 pm  Fri Sep 16 94
        servdata.c       16,590 bytes   2:00 pm  Fri Sep 16 94
        servdde.c        17,171 bytes   2:00 pm  Fri Sep 16 94
        server.c         10,237 bytes   2:00 pm  Fri Sep 16 94
        server.def          288 bytes   2:00 pm  Fri Sep 16 94
        server.h          1,676 bytes   2:00 pm  Fri Sep 16 94
        server.mak        2,295 bytes   2:00 pm  Fri Sep 16 94
        server.rc         2,324 bytes   2:00 pm  Fri Sep 16 94
        servres.h           449 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\SAMPLES\DDEML
        .                   <DIR>       11:15 am  Mon Jun 10 96
        ..                  <DIR>       11:19 am  Mon Jun 10 96
        CLIENT              <DIR>       11:15 am  Mon Jun 10 96
        CLOCK               <DIR>       11:15 am  Mon Jun 10 96
        SERVER              <DIR>       11:15 am  Mon Jun 10 96

I:\VC152\MSVC15\SAMPLES\DDEML\CLIENT
        .                   <DIR>       11:15 am  Mon Jun 10 96
        ..                  <DIR>       11:15 am  Mon Jun 10 96
        client.mak        3,177 bytes   2:00 pm  Fri Sep 16 94
        clinit.c          5,893 bytes   2:00 pm  Fri Sep 16 94
        conv.ico            766 bytes   2:00 pm  Fri Sep 16 94
        dde.c            24,665 bytes   2:00 pm  Fri Sep 16 94
        ddemlcl.c        45,805 bytes   2:00 pm  Fri Sep 16 94
        ddemlcl.def         880 bytes   2:00 pm  Fri Sep 16 94
        ddemlcl.h         6,767 bytes   2:00 pm  Fri Sep 16 94
```

```
        ddemlcl.ico          766 bytes   2:00 pm  Fri Sep 16 94
        ddemlcl.rc        10,875 bytes   2:00 pm  Fri Sep 16 94
        dialog.c          30,314 bytes   2:00 pm  Fri Sep 16 94
        huge.c             7,198 bytes   2:00 pm  Fri Sep 16 94
        huge.h               227 bytes   2:00 pm  Fri Sep 16 94
        infoctrl.c        23,576 bytes   2:00 pm  Fri Sep 16 94
        infoctrl.h         2,122 bytes   2:00 pm  Fri Sep 16 94
        list.ico             766 bytes   2:00 pm  Fri Sep 16 94
        mem.c              2,937 bytes   2:00 pm  Fri Sep 16 94
        resource.h         3,968 bytes   2:00 pm  Fri Sep 16 94
        track.c            8,862 bytes   2:00 pm  Fri Sep 16 94
        track.h              548 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\SAMPLES\DDEML\CLOCK
        .                    <DIR>      11:15 am  Mon Jun 10 96
        ..                   <DIR>      11:15 am  Mon Jun 10 96
        clockdat.asm       1,604 bytes   2:00 pm  Fri Sep 16 94
        clockdat.bin         240 bytes   2:00 pm  Fri Sep 16 94
        ddemlclk.c        33,051 bytes   2:00 pm  Fri Sep 16 94
        ddemlclk.def         289 bytes   2:00 pm  Fri Sep 16 94
        ddemlclk.h         2,935 bytes   2:00 pm  Fri Sep 16 94
        ddemlclk.ico         766 bytes   2:00 pm  Fri Sep 16 94
        ddemlclk.mak       2,434 bytes   2:00 pm  Fri Sep 16 94
        ddemlclk.rc        2,440 bytes   2:00 pm  Fri Sep 16 94
        gettime.c          1,093 bytes   2:00 pm  Fri Sep 16 94
        resource.h           489 bytes   2:00 pm  Fri Sep 16 94
        wrapper.c         17,018 bytes   2:00 pm  Fri Sep 16 94
        wrapper.h          2,431 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\SAMPLES\DDEML\SERVER
        .                    <DIR>      11:15 am  Mon Jun 10 96
        ..                   <DIR>      11:15 am  Mon Jun 10 96
        dde.c             20,842 bytes   2:00 pm  Fri Sep 16 94
        ddemlsv.c         22,444 bytes   2:00 pm  Fri Sep 16 94
        ddemlsv.def          469 bytes   2:00 pm  Fri Sep 16 94
        ddemlsv.h          4,068 bytes   2:00 pm  Fri Sep 16 94
        ddemlsv.ico          766 bytes   2:00 pm  Fri Sep 16 94
        ddemlsv.rc         4,973 bytes   2:00 pm  Fri Sep 16 94
        dialog.c           8,368 bytes   2:00 pm  Fri Sep 16 94
        huge.c             7,219 bytes   2:00 pm  Fri Sep 16 94
        huge.h               227 bytes   2:00 pm  Fri Sep 16 94
        resource.h         1,348 bytes   2:00 pm  Fri Sep 16 94
        server.mak         2,446 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\SAMPLES\DEFPROCS
        .                    <DIR>      11:15 am  Mon Jun 10 96
        ..                   <DIR>      11:19 am  Mon Jun 10 96
        defdlg.c           9,442 bytes   2:00 pm  Fri Sep 16 94
        defwnd.c          12,317 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\SAMPLES\DIBIT
        .                    <DIR>      11:15 am  Mon Jun 10 96
        ..                   <DIR>      11:19 am  Mon Jun 10 96
        dibit.c           26,838 bytes   2:00 pm  Fri Sep 16 94
```

```
        dibit.def           756 bytes   2:00 pm  Fri Sep 16 94
        dibit.h             690 bytes   2:00 pm  Fri Sep 16 94
        dibit.mak         1,975 bytes   2:00 pm  Fri Sep 16 94
        dibit.rc          2,472 bytes   2:00 pm  Fri Sep 16 94
        resource.h          853 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\SAMPLES\DIBVIEW
        .                   <DIR>      11:16 am  Mon Jun 10 96
        ..                  <DIR>      11:19 am  Mon Jun 10 96
        about.c           1,685 bytes   2:00 pm  Fri Sep 16 94
        about.h             330 bytes   2:00 pm  Fri Sep 16 94
        capture.c        13,150 bytes   2:00 pm  Fri Sep 16 94
        capture.h           527 bytes   2:00 pm  Fri Sep 16 94
        child.c          64,722 bytes   2:00 pm  Fri Sep 16 94
        child.h           2,955 bytes   2:00 pm  Fri Sep 16 94
        clipbrd.c        11,397 bytes   2:00 pm  Fri Sep 16 94
        clipbrd.h           683 bytes   2:00 pm  Fri Sep 16 94
        dib.c            15,467 bytes   2:00 pm  Fri Sep 16 94
        dib.h             1,138 bytes   2:00 pm  Fri Sep 16 94
        dibdrag.cur         326 bytes   2:00 pm  Fri Sep 16 94
        dibview.c         2,712 bytes   2:00 pm  Fri Sep 16 94
        dibview.def         647 bytes   2:00 pm  Fri Sep 16 94
        dibview.h         3,087 bytes   2:00 pm  Fri Sep 16 94
        dibview.ico         766 bytes   2:00 pm  Fri Sep 16 94
        dibview.mak       6,325 bytes   2:00 pm  Fri Sep 16 94
        dibview.rc       14,474 bytes   2:00 pm  Fri Sep 16 94
        errors.c          2,505 bytes   2:00 pm  Fri Sep 16 94
        errors.h          2,249 bytes   2:00 pm  Fri Sep 16 94
        file.c           36,158 bytes   2:00 pm  Fri Sep 16 94
        file.h            3,652 bytes   2:00 pm  Fri Sep 16 94
        frame.c          22,910 bytes   2:00 pm  Fri Sep 16 94
        frame.h             308 bytes   2:00 pm  Fri Sep 16 94
        init.c            5,412 bytes   2:00 pm  Fri Sep 16 94
        init.h              233 bytes   2:00 pm  Fri Sep 16 94
        master.h            701 bytes   2:00 pm  Fri Sep 16 94
        mydib.ico           766 bytes   2:00 pm  Fri Sep 16 94
        options.c        12,449 bytes   2:00 pm  Fri Sep 16 94
        options.h         2,991 bytes   2:00 pm  Fri Sep 16 94
        paint.c          10,317 bytes   2:00 pm  Fri Sep 16 94
        paint.h             533 bytes   2:00 pm  Fri Sep 16 94
        palette.c        35,600 bytes   2:00 pm  Fri Sep 16 94
        palette.h         3,040 bytes   2:00 pm  Fri Sep 16 94
        print.c          39,379 bytes   2:00 pm  Fri Sep 16 94
        print.h           3,988 bytes   2:00 pm  Fri Sep 16 94
        resource.h          270 bytes   2:00 pm  Fri Sep 16 94
        select.cur          326 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\SAMPLES\DRAGDROP
        .                   <DIR>      11:16 am  Mon Jun 10 96
        ..                  <DIR>      11:19 am  Mon Jun 10 96
        dragdrop.c       10,895 bytes   2:00 pm  Fri Sep 16 94
        dragdrop.def        344 bytes   2:00 pm  Fri Sep 16 94
        dragdrop.h          537 bytes   2:00 pm  Fri Sep 16 94
        dragdrop.ico        766 bytes   2:00 pm  Fri Sep 16 94
```

```
        dragdrop.mak     2,082 bytes   2:00 pm  Fri Sep 16 94
        dragdrop.rc      3,078 bytes   2:00 pm  Fri Sep 16 94
        resource.h         674 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\SAMPLES\EDITCNTL
        .                    <DIR>     11:16 am  Mon Jun 10 96
        ..                   <DIR>     11:19 am  Mon Jun 10 96
        editcntl.c       7,606 bytes   2:00 pm  Fri Sep 16 94
        editcntl.def       273 bytes   2:00 pm  Fri Sep 16 94
        editcntl.h         266 bytes   2:00 pm  Fri Sep 16 94
        editcntl.mak     2,022 bytes   2:00 pm  Fri Sep 16 94
        editcntl.rc      3,188 bytes   2:00 pm  Fri Sep 16 94
        resource.h         856 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\SAMPLES\EDITFILE
        .                    <DIR>     11:16 am  Mon Jun 10 96
        ..                   <DIR>     11:19 am  Mon Jun 10 96
        editfile.c      19,691 bytes   2:00 pm  Fri Sep 16 94
        editfile.def       264 bytes   2:00 pm  Fri Sep 16 94
        editfile.h         586 bytes   2:00 pm  Fri Sep 16 94
        editfile.mak     2,014 bytes   2:00 pm  Fri Sep 16 94
        editfile.rc      3,188 bytes   2:00 pm  Fri Sep 16 94
        resource.h         856 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\SAMPLES\EDITMENU
        .                    <DIR>     11:16 am  Mon Jun 10 96
        ..                   <DIR>     11:19 am  Mon Jun 10 96
        editmenu.c       6,176 bytes   2:00 pm  Fri Sep 16 94
        editmenu.def       261 bytes   2:00 pm  Fri Sep 16 94
        editmenu.h         266 bytes   2:00 pm  Fri Sep 16 94
        editmenu.mak     1,988 bytes   2:00 pm  Fri Sep 16 94
        editmenu.rc      3,183 bytes   2:00 pm  Fri Sep 16 94
        resource.h         811 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\SAMPLES\EXPENSE
        .                    <DIR>     11:16 am  Mon Jun 10 96
        ..                   <DIR>     11:19 am  Mon Jun 10 96
        custdict.c       4,602 bytes   2:00 pm  Fri Sep 16 94
        custdict.def       280 bytes   2:00 pm  Fri Sep 16 94
        custdict.h         632 bytes   2:00 pm  Fri Sep 16 94
        custdict.mak     1,782 bytes   2:00 pm  Fri Sep 16 94
        deptname.dic       180 bytes   2:00 pm  Fri Sep 16 94
        expense.c       26,767 bytes   2:00 pm  Fri Sep 16 94
        expense.def        299 bytes   2:00 pm  Fri Sep 16 94
        expense.h        2,687 bytes   2:00 pm  Fri Sep 16 94
        expense.ico        766 bytes   2:00 pm  Fri Sep 16 94
        expense.mak      2,088 bytes   2:00 pm  Fri Sep 16 94
        expense.rc       2,633 bytes   2:00 pm  Fri Sep 16 94
        expres.h           360 bytes   2:00 pm  Fri Sep 16 94
        names.dic          128 bytes   2:00 pm  Fri Sep 16 94
        resource.h         628 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\SAMPLES\FILEOPEN
        .                    <DIR>     11:16 am  Mon Jun 10 96
```

```
        ..                  <DIR>       11:19 am  Mon Jun 10 96
        fileopen.c       10,056 bytes   2:00 pm  Fri Sep 16 94
        fileopen.def        266 bytes   2:00 pm  Fri Sep 16 94
        fileopen.h          416 bytes   2:00 pm  Fri Sep 16 94
        fileopen.mak      2,051 bytes   2:00 pm  Fri Sep 16 94
        fileopen.rc       3,188 bytes   2:00 pm  Fri Sep 16 94
        resource.h          856 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\SAMPLES\FONTEST
        .                   <DIR>       11:16 am  Mon Jun 10 96
        ..                  <DIR>       11:19 am  Mon Jun 10 96
        font.ico          1,086 bytes   2:00 pm  Fri Sep 16 94
        fontest.c        17,777 bytes   2:00 pm  Fri Sep 16 94
        fontest.def         331 bytes   2:00 pm  Fri Sep 16 94
        fontest.h         1,665 bytes   2:00 pm  Fri Sep 16 94
        fontest.mak       2,147 bytes   2:00 pm  Fri Sep 16 94
        fontest.rc        5,019 bytes   2:00 pm  Fri Sep 16 94
        log.h               574 bytes   2:00 pm  Fri Sep 16 94
        print.c           3,874 bytes   2:00 pm  Fri Sep 16 94
        print.h             224 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\SAMPLES\GDOSMEM
        .                   <DIR>       11:16 am  Mon Jun 10 96
        ..                  <DIR>       11:19 am  Mon Jun 10 96
        gdosmem.c        17,400 bytes   2:00 pm  Fri Sep 16 94
        gdosmem.def         769 bytes   2:00 pm  Fri Sep 16 94
        gdosmem.h           433 bytes   2:00 pm  Fri Sep 16 94
        gdosmem.rc        2,101 bytes   2:00 pm  Fri Sep 16 94
        gtsr.asm          5,982 bytes   2:00 pm  Fri Sep 16 94
        gtsr.obj            372 bytes   2:00 pm  Fri Sep 16 94
        makefile          1,521 bytes   2:00 pm  Fri Sep 16 94
        read.me              70 bytes   2:00 pm  Fri Sep 16 94
        resource.h          450 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\SAMPLES\GENERIC
        .                   <DIR>       11:16 am  Mon Jun 10 96
        ..                  <DIR>       11:19 am  Mon Jun 10 96
        generic.c        11,463 bytes   2:00 pm  Fri Sep 16 94
        generic.def         755 bytes   2:00 pm  Fri Sep 16 94
        generic.h           262 bytes   2:00 pm  Fri Sep 16 94
        generic.mak       1,979 bytes   2:00 pm  Fri Sep 16 94
        generic.rc        1,939 bytes   2:00 pm  Fri Sep 16 94
        resource.h          315 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\SAMPLES\GWAPI
        .                   <DIR>       11:16 am  Mon Jun 10 96
        ..                  <DIR>       11:19 am  Mon Jun 10 96
        gwapi.c           6,643 bytes   2:00 pm  Fri Sep 16 94
        gwapi.def           271 bytes   2:00 pm  Fri Sep 16 94
        gwapi.mak         1,748 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\SAMPLES\HANDLER
        .                   <DIR>       11:16 am  Mon Jun 10 96
        ..                  <DIR>       11:19 am  Mon Jun 10 96
```

```
        handler.asm       7,227 bytes   2:00 pm  Fri Sep 16 94
        handler.def         242 bytes   2:00 pm  Fri Sep 16 94
        handler.h           272 bytes   2:00 pm  Fri Sep 16 94
        handler.inc          79 bytes   2:00 pm  Fri Sep 16 94
        handler.obj       5,681 bytes   2:00 pm  Fri Sep 16 94
        handtest.c        5,335 bytes   2:00 pm  Fri Sep 16 94
        handtest.def        330 bytes   2:00 pm  Fri Sep 16 94
        handtest.h          226 bytes   2:00 pm  Fri Sep 16 94
        handtest.rc       1,898 bytes   2:00 pm  Fri Sep 16 94
        makefile          2,170 bytes   2:00 pm  Fri Sep 16 94
        resource.h          451 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\SAMPLES\HELPEX
        .                   <DIR>       11:16 am  Mon Jun 10 96
        ..                  <DIR>       11:19 am  Mon Jun 10 96
        bullet.bmp          342 bytes   2:00 pm  Fri Sep 16 94
        chkboff.bmp         122 bytes   2:00 pm  Fri Sep 16 94
        chkbon.bmp          122 bytes   2:00 pm  Fri Sep 16 94
        codec.bmp         1,014 bytes   2:00 pm  Fri Sep 16 94
        continue.bmp        822 bytes   2:00 pm  Fri Sep 16 94
        done.bmp            822 bytes   2:00 pm  Fri Sep 16 94
        help.cur            326 bytes   2:00 pm  Fri Sep 16 94
        helpex.c         13,219 bytes   2:00 pm  Fri Sep 16 94
        helpex.def          255 bytes   2:00 pm  Fri Sep 16 94
        helpex.h            300 bytes   2:00 pm  Fri Sep 16 94
        helpex.hpj          706 bytes   2:00 pm  Fri Sep 16 94
        helpex.rc         3,550 bytes   2:00 pm  Fri Sep 16 94
        helpex.rtf       35,554 bytes   2:00 pm  Fri Sep 16 94
        helpicon.bmp        178 bytes   2:00 pm  Fri Sep 16 94
        helpids.h           595 bytes   2:00 pm  Fri Sep 16 94
        keys.rtf         14,030 bytes   2:00 pm  Fri Sep 16 94
        makefile          1,466 bytes   2:00 pm  Fri Sep 16 94
        max2icon.bmp        146 bytes   2:00 pm  Fri Sep 16 94
        mouse.bmp           194 bytes   2:00 pm  Fri Sep 16 94
        resource.h          827 bytes   2:00 pm  Fri Sep 16 94
        terms.rtf         1,981 bytes   2:00 pm  Fri Sep 16 94
        winword.bmp         438 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\SAMPLES\HELPHOOK
        .                   <DIR>       11:16 am  Mon Jun 10 96
        ..                  <DIR>       11:19 am  Mon Jun 10 96
        helphook.c       16,138 bytes   2:00 pm  Fri Sep 16 94
        helphook.def        833 bytes   2:00 pm  Fri Sep 16 94
        helphook.h          426 bytes   2:00 pm  Fri Sep 16 94
        helphook.ico      1,086 bytes   2:00 pm  Fri Sep 16 94
        helphook.rc      11,353 bytes   2:00 pm  Fri Sep 16 94
        makefile          1,436 bytes   2:00 pm  Fri Sep 16 94
        resource.h          856 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\SAMPLES\HFORM
        .                   <DIR>       11:16 am  Mon Jun 10 96
        ..                  <DIR>       11:19 am  Mon Jun 10 96
        hform.c          22,527 bytes   2:00 pm  Fri Sep 16 94
        hform.def           286 bytes   2:00 pm  Fri Sep 16 94
```

```
        hform.h          1,506 bytes   2:00 pm  Fri Sep 16 94
        hform.ico          766 bytes   2:00 pm  Fri Sep 16 94
        hform.mak        2,013 bytes   2:00 pm  Fri Sep 16 94
        hform.rc         2,634 bytes   2:00 pm  Fri Sep 16 94
        hfres.h            291 bytes   2:00 pm  Fri Sep 16 94
        resource.h         804 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\SAMPLES\HOOKS
        .                    <DIR>     11:16 am  Mon Jun 10 96
        ..                   <DIR>     11:19 am  Mon Jun 10 96
        dllres.h           271 bytes   2:00 pm  Fri Sep 16 94
        dllstuff.c       1,561 bytes   2:00 pm  Fri Sep 16 94
        hooks.c          9,700 bytes   2:00 pm  Fri Sep 16 94
        hooks.def          475 bytes   2:00 pm  Fri Sep 16 94
        hooks.h            845 bytes   2:00 pm  Fri Sep 16 94
        hooks.ico          766 bytes   2:00 pm  Fri Sep 16 94
        hooks.mak        2,034 bytes   2:00 pm  Fri Sep 16 94
        hooks.rc         2,991 bytes   2:00 pm  Fri Sep 16 94
        hooksdll.c      26,096 bytes   2:00 pm  Fri Sep 16 94
        hooksdll.def       465 bytes   2:00 pm  Fri Sep 16 94
        hooksdll.mak     2,043 bytes   2:00 pm  Fri Sep 16 94
        hooksdll.rc        802 bytes   2:00 pm  Fri Sep 16 94
        messages.rc      5,368 bytes   2:00 pm  Fri Sep 16 94
        resource.h         718 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\SAMPLES\ICON
        .                    <DIR>     11:16 am  Mon Jun 10 96
        ..                   <DIR>     11:19 am  Mon Jun 10 96
        icon.c           4,889 bytes   2:00 pm  Fri Sep 16 94
        icon.def           269 bytes   2:00 pm  Fri Sep 16 94
        icon.h             291 bytes   2:00 pm  Fri Sep 16 94
        icon.mak         1,986 bytes   2:00 pm  Fri Sep 16 94
        icon.rc          2,188 bytes   2:00 pm  Fri Sep 16 94
        myicon.ico       1,718 bytes   2:00 pm  Fri Sep 16 94
        resource.h         295 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\SAMPLES\INPUT
        .                    <DIR>     11:16 am  Mon Jun 10 96
        ..                   <DIR>     11:19 am  Mon Jun 10 96
        input.c         11,617 bytes   2:00 pm  Fri Sep 16 94
        input.def          258 bytes   2:00 pm  Fri Sep 16 94
        input.h            266 bytes   2:00 pm  Fri Sep 16 94
        input.mak        1,965 bytes   2:00 pm  Fri Sep 16 94
        input.rc         1,945 bytes   2:00 pm  Fri Sep 16 94
        resource.h         292 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\SAMPLES\INSTVER
        .                    <DIR>     11:17 am  Mon Jun 10 96
        ..                   <DIR>     11:19 am  Mon Jun 10 96
        dllentry.asm     2,337 bytes   2:00 pm  Fri Sep 16 94
        dllentry.obj       469 bytes   2:00 pm  Fri Sep 16 94
        dos.asm          8,369 bytes   2:00 pm  Fri Sep 16 94
        dos.obj          1,225 bytes   2:00 pm  Fri Sep 16 94
        instpriv.h       3,371 bytes   2:00 pm  Fri Sep 16 94
```

```
        instver.c       15,406 bytes   2:00 pm  Fri Sep 16 94
        instver.def        253 bytes   2:00 pm  Fri Sep 16 94
        instver.h        1,276 bytes   2:00 pm  Fri Sep 16 94
        makefile         2,484 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\SAMPLES\LISTHORZ
        .                  <DIR>       11:17 am  Mon Jun 10 96
        ..                 <DIR>       11:19 am  Mon Jun 10 96
        listhapi.h         392 bytes   2:00 pm  Fri Sep 16 94
        listhorz.c       5,561 bytes   2:00 pm  Fri Sep 16 94
        listhorz.def       291 bytes   2:00 pm  Fri Sep 16 94
        listhorz.h         419 bytes   2:00 pm  Fri Sep 16 94
        listhorz.ico     1,846 bytes   2:00 pm  Fri Sep 16 94
        listhorz.mak     2,081 bytes   2:00 pm  Fri Sep 16 94
        listhorz.rc      2,076 bytes   2:00 pm  Fri Sep 16 94
        listhscr.c      14,618 bytes   2:00 pm  Fri Sep 16 94
        listhscr.def       533 bytes   2:00 pm  Fri Sep 16 94
        listhscr.h         415 bytes   2:00 pm  Fri Sep 16 94
        listhscr.mak     1,882 bytes   2:00 pm  Fri Sep 16 94
        resource.h         394 bytes   2:00 pm  Fri Sep 16 94
        wep.c            1,076 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\SAMPLES\LOWPASS
        .                  <DIR>       11:17 am  Mon Jun 10 96
        ..                 <DIR>       11:19 am  Mon Jun 10 96
        lowpass.c       12,986 bytes   2:00 pm  Fri Sep 16 94
        lowpass.def        279 bytes   2:00 pm  Fri Sep 16 94
        lowpass.h          436 bytes   2:00 pm  Fri Sep 16 94
        lowpass.ico        766 bytes   2:00 pm  Fri Sep 16 94
        lowpass.mak      2,079 bytes   2:00 pm  Fri Sep 16 94
        lowpass.rc       2,690 bytes   2:00 pm  Fri Sep 16 94
        resource.h         494 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\SAMPLES\MACROHLP
        .                  <DIR>       11:17 am  Mon Jun 10 96
        ..                 <DIR>       11:19 am  Mon Jun 10 96
        abc.rtf         18,817 bytes   2:00 pm  Fri Sep 16 94
        defghi.rtf      23,568 bytes   2:00 pm  Fri Sep 16 94
        jnprs.rtf       17,706 bytes   2:00 pm  Fri Sep 16 94
        macrohlp.hpj     1,209 bytes   2:00 pm  Fri Sep 16 94
        macrohlp.rtf    12,244 bytes   2:00 pm  Fri Sep 16 94
        windows.shg        464 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\SAMPLES\MAKEAPP
        .                  <DIR>       11:17 am  Mon Jun 10 96
        ..                 <DIR>       11:19 am  Mon Jun 10 96
        app.c            7,697 bytes   2:00 pm  Fri Sep 16 94
        app.h            1,467 bytes   2:00 pm  Fri Sep 16 94
        client.c         4,743 bytes   2:00 pm  Fri Sep 16 94
        client.h         1,572 bytes   2:00 pm  Fri Sep 16 94
        dlg.c            3,988 bytes   2:00 pm  Fri Sep 16 94
        dlg.h              707 bytes   2:00 pm  Fri Sep 16 94
        frame.c          6,380 bytes   2:00 pm  Fri Sep 16 94
        frame.h          2,021 bytes   2:00 pm  Fri Sep 16 94
```

```
        makeapp.bat       1,900 bytes   2:00 pm  Fri Sep 16 94
        makeapp.def         371 bytes   2:00 pm  Fri Sep 16 94
        makeapp.h           259 bytes   2:00 pm  Fri Sep 16 94
        makeapp.ico         766 bytes   2:00 pm  Fri Sep 16 94
        makeapp.mak       2,627 bytes   2:00 pm  Fri Sep 16 94
        makeapp.rc        5,088 bytes   2:00 pm  Fri Sep 16 94
        makeapp.txt       7,700 bytes   2:00 pm  Fri Sep 16 94
        makeapp.ver       2,011 bytes   2:00 pm  Fri Sep 16 94
        makeapp2.bat        114 bytes   2:00 pm  Fri Sep 16 94
        makewc.bat        1,053 bytes   2:00 pm  Fri Sep 16 94
        makewc.c          2,896 bytes   2:00 pm  Fri Sep 16 94
        makewc.h          1,296 bytes   2:00 pm  Fri Sep 16 94
        makewc2.bat         261 bytes   2:00 pm  Fri Sep 16 94
        rep.exe          48,966 bytes   2:00 pm  Fri Sep 16 94
        resource.h        1,886 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\SAMPLES\MCITEST
        .                   <DIR>       11:17 am  Mon Jun 10 96
        ..                  <DIR>       11:19 am  Mon Jun 10 96
        dlgopen.c        10,587 bytes   2:00 pm  Fri Sep 16 94
        dlgopen.h         1,203 bytes   2:00 pm  Fri Sep 16 94
        edit.c            7,827 bytes   2:00 pm  Fri Sep 16 94
        edit.h              361 bytes   2:00 pm  Fri Sep 16 94
        gmem.h            2,695 bytes   2:00 pm  Fri Sep 16 94
        mcitest.c        25,159 bytes   2:00 pm  Fri Sep 16 94
        mcitest.def         324 bytes   2:00 pm  Fri Sep 16 94
        mcitest.h           779 bytes   2:00 pm  Fri Sep 16 94
        mcitest.ico         766 bytes   2:00 pm  Fri Sep 16 94
        mcitest.mak       2,394 bytes   2:00 pm  Fri Sep 16 94
        mcitest.rc        6,335 bytes   2:00 pm  Fri Sep 16 94
        resource.h        1,779 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\SAMPLES\MEMORY
        .                   <DIR>       11:17 am  Mon Jun 10 96
        ..                  <DIR>       11:19 am  Mon Jun 10 96
        mem.h               342 bytes   2:00 pm  Fri Sep 16 94
        memory.def          448 bytes   2:00 pm  Fri Sep 16 94
        memory.mak        2,463 bytes   2:00 pm  Fri Sep 16 94
        memory.rc         1,950 bytes   2:00 pm  Fri Sep 16 94
        memory1.c           822 bytes   2:00 pm  Fri Sep 16 94
        memory2.c         1,671 bytes   2:00 pm  Fri Sep 16 94
        memory3.c         1,073 bytes   2:00 pm  Fri Sep 16 94
        memory4.c           806 bytes   2:00 pm  Fri Sep 16 94
        resource.h          314 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\SAMPLES\MENU
        .                   <DIR>       11:17 am  Mon Jun 10 96
        ..                  <DIR>       11:19 am  Mon Jun 10 96
        checkoff.bmp        544 bytes   2:00 pm  Fri Sep 16 94
        checkon.bmp         544 bytes   2:00 pm  Fri Sep 16 94
        menu.c           21,251 bytes   2:00 pm  Fri Sep 16 94
        menu.def            206 bytes   2:00 pm  Fri Sep 16 94
        menu.h              621 bytes   2:00 pm  Fri Sep 16 94
        menu.ico            766 bytes   2:00 pm  Fri Sep 16 94
```

```
        menu.mak          1,912 bytes   2:00 pm  Fri Sep 16 94
        menu.rc           4,779 bytes   2:00 pm  Fri Sep 16 94
        resource.h        1,226 bytes   2:00 pm  Fri Sep 16 94

 I:\VC152\MSVC15\SAMPLES\MIDIMON
        .                   <DIR>       11:17 am  Mon Jun 10 96
        ..                  <DIR>       11:19 am  Mon Jun 10 96
        about.c           1,269 bytes   2:00 pm  Fri Sep 16 94
        about.h             172 bytes   2:00 pm  Fri Sep 16 94
        callback.c        4,048 bytes   2:00 pm  Fri Sep 16 94
        callback.def        382 bytes   2:00 pm  Fri Sep 16 94
        callback.h          197 bytes   2:00 pm  Fri Sep 16 94
        callback.mak      1,884 bytes   2:00 pm  Fri Sep 16 94
        circbuf.c         4,446 bytes   2:00 pm  Fri Sep 16 94
        circbuf.h           938 bytes   2:00 pm  Fri Sep 16 94
        display.c         9,202 bytes   2:00 pm  Fri Sep 16 94
        display.h         1,868 bytes   2:00 pm  Fri Sep 16 94
        filter.c          2,902 bytes   2:00 pm  Fri Sep 16 94
        filter.h            694 bytes   2:00 pm  Fri Sep 16 94
        instdata.c        1,964 bytes   2:00 pm  Fri Sep 16 94
        instdata.h          592 bytes   2:00 pm  Fri Sep 16 94
        midimon.c        30,234 bytes   2:00 pm  Fri Sep 16 94
        midimon.def         269 bytes   2:00 pm  Fri Sep 16 94
        midimon.h         1,252 bytes   2:00 pm  Fri Sep 16 94
        midimon.ico         766 bytes   2:00 pm  Fri Sep 16 94
        midimon.mak       3,224 bytes   2:00 pm  Fri Sep 16 94
        midimon.rc        5,014 bytes   2:00 pm  Fri Sep 16 94
        prefer.c          2,357 bytes   2:00 pm  Fri Sep 16 94
        prefer.h            936 bytes   2:00 pm  Fri Sep 16 94
        resource.h        1,978 bytes   2:00 pm  Fri Sep 16 94

 I:\VC152\MSVC15\SAMPLES\MULTIPAD
        .                   <DIR>       11:17 am  Mon Jun 10 96
        ..                  <DIR>       11:19 am  Mon Jun 10 96
        mp300.ico           766 bytes   2:00 pm  Fri Sep 16 94
        mpfile.c         12,531 bytes   2:00 pm  Fri Sep 16 94
        mpfind.c          7,856 bytes   2:00 pm  Fri Sep 16 94
        mpinit.c          4,477 bytes   2:00 pm  Fri Sep 16 94
        mpprint.c        10,901 bytes   2:00 pm  Fri Sep 16 94
        multipad.c       26,414 bytes   2:00 pm  Fri Sep 16 94
        multipad.def        558 bytes   2:00 pm  Fri Sep 16 94
        multipad.h        3,518 bytes   2:00 pm  Fri Sep 16 94
        multipad.mak      2,681 bytes   2:00 pm  Fri Sep 16 94
        multipad.rc       7,366 bytes   2:00 pm  Fri Sep 16 94
        note300.ico         766 bytes   2:00 pm  Fri Sep 16 94
        resource.h        2,977 bytes   2:00 pm  Fri Sep 16 94

 I:\VC152\MSVC15\SAMPLES\MUSCROLL
        .                   <DIR>       11:17 am  Mon Jun 10 96
        ..                  <DIR>       11:19 am  Mon Jun 10 96
        ctldlg.c         12,082 bytes   2:00 pm  Fri Sep 16 94
        init.c           10,441 bytes   2:00 pm  Fri Sep 16 94
        msapi.c          11,055 bytes   2:00 pm  Fri Sep 16 94
        muscrdll.h        4,107 bytes   2:00 pm  Fri Sep 16 94
```

```
        muscroll.c      13,563 bytes   2:00 pm  Fri Sep 16 94
        muscroll.def     1,359 bytes   2:00 pm  Fri Sep 16 94
        muscroll.h       2,281 bytes   2:00 pm  Fri Sep 16 94
        muscroll.mak     2,847 bytes   2:00 pm  Fri Sep 16 94
        muscroll.rc      3,247 bytes   2:00 pm  Fri Sep 16 94
        muscroll.txt    12,663 bytes   2:00 pm  Fri Sep 16 94
        muscrres.h       1,147 bytes   2:00 pm  Fri Sep 16 94
        mustest.c        8,549 bytes   2:00 pm  Fri Sep 16 94
        mustest.def        287 bytes   2:00 pm  Fri Sep 16 94
        mustest.h          807 bytes   2:00 pm  Fri Sep 16 94
        mustest.ico        766 bytes   2:00 pm  Fri Sep 16 94
        mustest.rc       1,136 bytes   2:00 pm  Fri Sep 16 94
        mustres.h          315 bytes   2:00 pm  Fri Sep 16 94
        mutest.mak       2,102 bytes   2:00 pm  Fri Sep 16 94
        paint.c         11,410 bytes   2:00 pm  Fri Sep 16 94
        wep.c              717 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\SAMPLES\MYPAL
        .                  <DIR>      11:17 am  Mon Jun 10 96
        ..                 <DIR>      11:19 am  Mon Jun 10 96
        mypal.c         17,614 bytes   2:00 pm  Fri Sep 16 94
        mypal.def          191 bytes   2:00 pm  Fri Sep 16 94
        mypal.h            274 bytes   2:00 pm  Fri Sep 16 94
        mypal.ico          766 bytes   2:00 pm  Fri Sep 16 94
        mypal.mak        2,006 bytes   2:00 pm  Fri Sep 16 94
        mypal.rc         1,466 bytes   2:00 pm  Fri Sep 16 94
        resource.h         312 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\SAMPLES\OUTPUT
        .                  <DIR>      11:17 am  Mon Jun 10 96
        ..                 <DIR>      11:19 am  Mon Jun 10 96
        output.c        13,591 bytes   2:00 pm  Fri Sep 16 94
        output.def         271 bytes   2:00 pm  Fri Sep 16 94
        output.h           266 bytes   2:00 pm  Fri Sep 16 94
        output.mak       1,950 bytes   2:00 pm  Fri Sep 16 94
        output.rc        1,936 bytes   2:00 pm  Fri Sep 16 94
        resource.h         314 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\SAMPLES\OWNCOMBO
        .                  <DIR>      11:17 am  Mon Jun 10 96
        ..                 <DIR>      11:19 am  Mon Jun 10 96
        owncombo.c      25,991 bytes   2:00 pm  Fri Sep 16 94
        owncombo.def       296 bytes   2:00 pm  Fri Sep 16 94
        owncombo.h         778 bytes   2:00 pm  Fri Sep 16 94
        owncombo.ico       766 bytes   2:00 pm  Fri Sep 16 94
        owncombo.mak     1,997 bytes   2:00 pm  Fri Sep 16 94
        owncombo.rc      5,752 bytes   2:00 pm  Fri Sep 16 94
        readme.txt       1,891 bytes   2:00 pm  Fri Sep 16 94
        resource.h       1,506 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\SAMPLES\OWNERB
        .                  <DIR>      11:17 am  Mon Jun 10 96
        ..                 <DIR>      11:19 am  Mon Jun 10 96
        ownerb.c         5,150 bytes   2:00 pm  Fri Sep 16 94
```

```
        ownerb.def          268 bytes   2:00 pm  Fri Sep 16 94
        ownerb.h            359 bytes   2:00 pm  Fri Sep 16 94
        ownerb.ico          766 bytes   2:00 pm  Fri Sep 16 94
        ownerb.mak        1,922 bytes   2:00 pm  Fri Sep 16 94
        ownerb.rc         1,631 bytes   2:00 pm  Fri Sep 16 94
        play.bmp            518 bytes   2:00 pm  Fri Sep 16 94
        plays.bmp           518 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\SAMPLES\PALETTE
        .                   <DIR>       11:18 am  Mon Jun 10 96
        ..                  <DIR>       11:19 am  Mon Jun 10 96
        init.c            1,515 bytes   2:00 pm  Fri Sep 16 94
        palette.c         7,502 bytes   2:00 pm  Fri Sep 16 94
        palette.cur         326 bytes   2:00 pm  Fri Sep 16 94
        palette.def         313 bytes   2:00 pm  Fri Sep 16 94
        palette.h           633 bytes   2:00 pm  Fri Sep 16 94
        palette.ico         766 bytes   2:00 pm  Fri Sep 16 94
        palette.mak       2,140 bytes   2:00 pm  Fri Sep 16 94
        palette.rc        1,766 bytes   2:00 pm  Fri Sep 16 94
        resource.h          313 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\SAMPLES\PENCNTL
        .                   <DIR>       11:18 am  Mon Jun 10 96
        ..                  <DIR>       11:19 am  Mon Jun 10 96
        beditde.dll       8,608 bytes   2:00 pm  Fri Sep 16 94
        heditde.dll       8,800 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\SAMPLES\PENPAD
        .                   <DIR>       11:18 am  Mon Jun 10 96
        ..                  <DIR>       11:19 am  Mon Jun 10 96
        note300.ico         766 bytes   2:00 pm  Fri Sep 16 94
        penpad.c         39,907 bytes   2:00 pm  Fri Sep 16 94
        penpad.def          663 bytes   2:00 pm  Fri Sep 16 94
        penpad.h          3,797 bytes   2:00 pm  Fri Sep 16 94
        penpad.mak        2,801 bytes   2:00 pm  Fri Sep 16 94
        penpad.rc         9,029 bytes   2:00 pm  Fri Sep 16 94
        pp300.ico           766 bytes   2:00 pm  Fri Sep 16 94
        ppfile.c         17,064 bytes   2:00 pm  Fri Sep 16 94
        ppfind.c          9,396 bytes   2:00 pm  Fri Sep 16 94
        ppinit.c          4,069 bytes   2:00 pm  Fri Sep 16 94
        ppopen.c         10,160 bytes   2:00 pm  Fri Sep 16 94
        ppprint.c        14,031 bytes   2:00 pm  Fri Sep 16 94
        resource.h        2,975 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\SAMPLES\PRNTFILE
        .                   <DIR>       11:18 am  Mon Jun 10 96
        ..                  <DIR>       11:19 am  Mon Jun 10 96
        prntfile.c       29,501 bytes   2:00 pm  Fri Sep 16 94
        prntfile.def        432 bytes   2:00 pm  Fri Sep 16 94
        prntfile.h          722 bytes   2:00 pm  Fri Sep 16 94
        prntfile.mak      2,048 bytes   2:00 pm  Fri Sep 16 94
        prntfile.rc       3,586 bytes   2:00 pm  Fri Sep 16 94
        resource.h          946 bytes   2:00 pm  Fri Sep 16 94
```

```
I:\VC152\MSVC15\SAMPLES\PROFILER
        .                   <DIR>       11:18 am  Mon Jun 10 96
        ..                  <DIR>       11:19 am  Mon Jun 10 96
        DOS                 <DIR>       11:18 am  Mon Jun 10 96
        TAB                 <DIR>       11:18 am  Mon Jun 10 96
        WIN                 <DIR>       11:18 am  Mon Jun 10 96

I:\VC152\MSVC15\SAMPLES\PROFILER\DOS
        .                   <DIR>       11:18 am  Mon Jun 10 96
        ..                  <DIR>       11:18 am  Mon Jun 10 96
        jabber.txt        1,025 bytes    2:00 pm  Fri Sep 16 94
        qsort.c           7,095 bytes    2:00 pm  Fri Sep 16 94
        qsort.mak         1,511 bytes    2:00 pm  Fri Sep 16 94
        qsort.pcf           486 bytes    2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\SAMPLES\PROFILER\TAB
        .                   <DIR>       11:18 am  Mon Jun 10 96
        ..                  <DIR>       11:18 am  Mon Jun 10 96
        DOS                 <DIR>       11:18 am  Mon Jun 10 96
        profiler.xlm     10,604 bytes    2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\SAMPLES\PROFILER\TAB\DOS
        .                   <DIR>       11:18 am  Mon Jun 10 96
        ..                  <DIR>       11:18 am  Mon Jun 10 96
        parse.c          12,818 bytes    2:00 pm  Fri Sep 16 94
        parse.exe        38,451 bytes    2:00 pm  Fri Sep 16 94
        parse.mak         1,495 bytes    2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\SAMPLES\PROFILER\WIN
        .                   <DIR>       11:18 am  Mon Jun 10 96
        ..                  <DIR>       11:18 am  Mon Jun 10 96
        jabber.txt        1,025 bytes    2:00 pm  Fri Sep 16 94
        qsortw.c         10,351 bytes    2:00 pm  Fri Sep 16 94
        qsortw.def          216 bytes    2:00 pm  Fri Sep 16 94
        qsortw.h          1,305 bytes    2:00 pm  Fri Sep 16 94
        qsortw.mak        1,900 bytes    2:00 pm  Fri Sep 16 94
        qsortw.pcf          198 bytes    2:00 pm  Fri Sep 16 94
        qsortw.rc         1,452 bytes    2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\SAMPLES\QWGDEMO
        .                   <DIR>       11:18 am  Mon Jun 10 96
        ..                  <DIR>       11:19 am  Mon Jun 10 96
        qwgdemo.cpp       2,775 bytes    2:00 pm  Fri Sep 16 94
        qwgdemo.mak       1,858 bytes    2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\SAMPLES\REVERSE
        .                   <DIR>       11:18 am  Mon Jun 10 96
        ..                  <DIR>       11:19 am  Mon Jun 10 96
        resource.h          270 bytes    2:00 pm  Fri Sep 16 94
        reverse.c        18,825 bytes    2:00 pm  Fri Sep 16 94
        reverse.def         214 bytes    2:00 pm  Fri Sep 16 94
        reverse.h         1,086 bytes    2:00 pm  Fri Sep 16 94
        reverse.ico         766 bytes    2:00 pm  Fri Sep 16 94
        reverse.mak       2,065 bytes    2:00 pm  Fri Sep 16 94
```

```
        reverse.rc        2,159 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\SAMPLES\ROTARY
        .                   <DIR>       11:18 am  Mon Jun 10 96
        ..                  <DIR>       11:19 am  Mon Jun 10 96
        resource.h            314 bytes   2:00 pm  Fri Sep 16 94
        rotary.bmp          2,638 bytes   2:00 pm  Fri Sep 16 94
        rotary.c           17,051 bytes   2:00 pm  Fri Sep 16 94
        rotary.def            289 bytes   2:00 pm  Fri Sep 16 94
        rotary.h            1,571 bytes   2:00 pm  Fri Sep 16 94
        rotary.ico            766 bytes   2:00 pm  Fri Sep 16 94
        rotary.mak          1,984 bytes   2:00 pm  Fri Sep 16 94
        rotary.rc           1,590 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\SAMPLES\SELECT
        .                   <DIR>       11:18 am  Mon Jun 10 96
        ..                  <DIR>       11:19 am  Mon Jun 10 96
        demo.c              7,411 bytes   2:00 pm  Fri Sep 16 94
        demo.def              283 bytes   2:00 pm  Fri Sep 16 94
        demo.h                226 bytes   2:00 pm  Fri Sep 16 94
        demo.mak            2,002 bytes   2:00 pm  Fri Sep 16 94
        demo.rc             2,182 bytes   2:00 pm  Fri Sep 16 94
        resource.h            447 bytes   2:00 pm  Fri Sep 16 94
        select.c            6,799 bytes   2:00 pm  Fri Sep 16 94
        select.def            283 bytes   2:00 pm  Fri Sep 16 94
        select.h              857 bytes   2:00 pm  Fri Sep 16 94
        select.mak          1,791 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\SAMPLES\SHOWDIB
        .                   <DIR>       11:18 am  Mon Jun 10 96
        ..                  <DIR>       11:19 am  Mon Jun 10 96
        dib.c              34,457 bytes   2:00 pm  Fri Sep 16 94
        dlgopen.c          18,885 bytes   2:00 pm  Fri Sep 16 94
        dlgopena.asm        2,547 bytes   2:00 pm  Fri Sep 16 94
        dlgopena.obj          188 bytes   2:00 pm  Fri Sep 16 94
        drawdib.c          38,506 bytes   2:00 pm  Fri Sep 16 94
        makefile            1,583 bytes   2:00 pm  Fri Sep 16 94
        print.c             8,205 bytes   2:00 pm  Fri Sep 16 94
        resource.h          1,725 bytes   2:00 pm  Fri Sep 16 94
        showdib.c          42,470 bytes   2:00 pm  Fri Sep 16 94
        showdib.def           344 bytes   2:00 pm  Fri Sep 16 94
        showdib.h           8,767 bytes   2:00 pm  Fri Sep 16 94
        showdib.ico           766 bytes   2:00 pm  Fri Sep 16 94
        showdib.rc          7,130 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\SAMPLES\SHOWGDI
        .                   <DIR>       11:18 am  Mon Jun 10 96
        ..                  <DIR>       11:19 am  Mon Jun 10 96
        dc.c               10,175 bytes   2:00 pm  Fri Sep 16 94
        dc.h                1,795 bytes   2:00 pm  Fri Sep 16 94
        dialogs.c          61,824 bytes   2:00 pm  Fri Sep 16 94
        dialogs.h           1,955 bytes   2:00 pm  Fri Sep 16 94
        dib.c              36,541 bytes   2:00 pm  Fri Sep 16 94
        dib.h               3,296 bytes   2:00 pm  Fri Sep 16 94
```

```
        draw.c              7,067 bytes   2:00 pm  Fri Sep 16 94
        draw.h              2,244 bytes   2:00 pm  Fri Sep 16 94
        main.c              5,233 bytes   2:00 pm  Fri Sep 16 94
        main.h              1,383 bytes   2:00 pm  Fri Sep 16 94
        menu.h              2,768 bytes   2:00 pm  Fri Sep 16 94
        resource.h         11,331 bytes   2:00 pm  Fri Sep 16 94
        showgdi.def           516 bytes   2:00 pm  Fri Sep 16 94
        showgdi.ico         1,086 bytes   2:00 pm  Fri Sep 16 94
        showgdi.mak         3,140 bytes   2:00 pm  Fri Sep 16 94
        showgdi.rc         63,145 bytes   2:00 pm  Fri Sep 16 94
        util.c              1,361 bytes   2:00 pm  Fri Sep 16 94
        util.h              2,870 bytes   2:00 pm  Fri Sep 16 94
        view.c             12,115 bytes   2:00 pm  Fri Sep 16 94
        view.h              1,372 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\SAMPLES\SNOOP
        .                     <DIR>       11:18 am  Mon Jun 10 96
        ..                    <DIR>       11:19 am  Mon Jun 10 96
        resource.h            403 bytes   2:00 pm  Fri Sep 16 94
        snoop.c            19,201 bytes   2:00 pm  Fri Sep 16 94
        snoop.def             257 bytes   2:00 pm  Fri Sep 16 94
        snoop.h               933 bytes   2:00 pm  Fri Sep 16 94
        snoop.ico             766 bytes   2:00 pm  Fri Sep 16 94
        snoop.mak           2,029 bytes   2:00 pm  Fri Sep 16 94
        snoop.rc            1,713 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\SAMPLES\SORTDEMO
        .                     <DIR>       11:18 am  Mon Jun 10 96
        ..                    <DIR>       11:19 am  Mon Jun 10 96
        resource.h          1,663 bytes   2:00 pm  Fri Sep 16 94
        sort.ico              766 bytes   2:00 pm  Fri Sep 16 94
        sortdemo.c         36,646 bytes   2:00 pm  Fri Sep 16 94
        sortdemo.def          301 bytes   2:00 pm  Fri Sep 16 94
        sortdemo.h          1,779 bytes   2:00 pm  Fri Sep 16 94
        sortdemo.mak        2,102 bytes   2:00 pm  Fri Sep 16 94
        sortdemo.rc         4,740 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\SAMPLES\SREC
        .                     <DIR>       11:18 am  Mon Jun 10 96
        ..                    <DIR>       11:19 am  Mon Jun 10 96
        main.h                837 bytes   2:00 pm  Fri Sep 16 94
        penapp.c           31,267 bytes   2:00 pm  Fri Sep 16 94
        penapp.def            447 bytes   2:00 pm  Fri Sep 16 94
        penapp.h            2,990 bytes   2:00 pm  Fri Sep 16 94
        penapp.ico            766 bytes   2:00 pm  Fri Sep 16 94
        penapp.mak          2,012 bytes   2:00 pm  Fri Sep 16 94
        penapp.rc           1,968 bytes   2:00 pm  Fri Sep 16 94
        penres.h              580 bytes   2:00 pm  Fri Sep 16 94
        srec.c             15,909 bytes   2:00 pm  Fri Sep 16 94
        srec.def              405 bytes   2:00 pm  Fri Sep 16 94
        srec.mak            1,737 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\SAMPLES\SRVRDEMO
        .                     <DIR>       11:18 am  Mon Jun 10 96
```

```
        ..                  <DIR>       11:19 am  Mon Jun 10 96
        doc.c            17,026 bytes    2:00 pm  Fri Sep 16 94
        file.c           13,891 bytes    2:00 pm  Fri Sep 16 94
        ini.reg             420 bytes    2:00 pm  Fri Sep 16 94
        obj.c            29,394 bytes    2:00 pm  Fri Sep 16 94
        resource.h        1,306 bytes    2:00 pm  Fri Sep 16 94
        server.c         16,969 bytes    2:00 pm  Fri Sep 16 94
        srvrdemo.c       35,704 bytes    2:00 pm  Fri Sep 16 94
        srvrdemo.def      1,238 bytes    2:00 pm  Fri Sep 16 94
        srvrdemo.h       10,000 bytes    2:00 pm  Fri Sep 16 94
        srvrdemo.ico        766 bytes    2:00 pm  Fri Sep 16 94
        srvrdemo.mak      2,642 bytes    2:00 pm  Fri Sep 16 94
        srvrdemo.rc       4,169 bytes    2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\SAMPLES\TDOSMEM
        .                   <DIR>       11:18 am  Mon Jun 10 96
        ..                  <DIR>       11:19 am  Mon Jun 10 96
        makefile          1,534 bytes    2:00 pm  Fri Sep 16 94
        read.me              70 bytes    2:00 pm  Fri Sep 16 94
        resource.h          360 bytes    2:00 pm  Fri Sep 16 94
        tdosmem.c        17,683 bytes    2:00 pm  Fri Sep 16 94
        tdosmem.def         487 bytes    2:00 pm  Fri Sep 16 94
        tdosmem.h           528 bytes    2:00 pm  Fri Sep 16 94
        tdosmem.rc        2,048 bytes    2:00 pm  Fri Sep 16 94
        ttsr.asm          5,896 bytes    2:00 pm  Fri Sep 16 94
        ttsr.obj            361 bytes    2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\SAMPLES\TIMERS
        .                   <DIR>       11:18 am  Mon Jun 10 96
        ..                  <DIR>       11:19 am  Mon Jun 10 96
        timers.c          8,998 bytes    2:00 pm  Fri Sep 16 94
        timers.def          450 bytes    2:00 pm  Fri Sep 16 94
        timers.h            551 bytes    2:00 pm  Fri Sep 16 94
        timers.mak        1,952 bytes    2:00 pm  Fri Sep 16 94
        timers.rc         1,351 bytes    2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\SAMPLES\TOOLHELP
        .                   <DIR>       11:18 am  Mon Jun 10 96
        ..                  <DIR>       11:19 am  Mon Jun 10 96
        makefile          1,473 bytes    2:00 pm  Fri Sep 16 94
        resource.h        1,865 bytes    2:00 pm  Fri Sep 16 94
        thasm.asm         6,803 bytes    2:00 pm  Fri Sep 16 94
        thasm.obj           442 bytes    2:00 pm  Fri Sep 16 94
        thsample.c       49,514 bytes    2:00 pm  Fri Sep 16 94
        thsample.def        347 bytes    2:00 pm  Fri Sep 16 94
        thsample.h          380 bytes    2:00 pm  Fri Sep 16 94
        thsample.ico        766 bytes    2:00 pm  Fri Sep 16 94
        thsample.rc       4,428 bytes    2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\SAMPLES\TTY
        .                   <DIR>       11:18 am  Mon Jun 10 96
        ..                  <DIR>       11:19 am  Mon Jun 10 96
        resource.h        3,113 bytes    2:00 pm  Fri Sep 16 94
        tty.bmp           2,166 bytes    2:00 pm  Fri Sep 16 94
```

```
        tty.c           55,527 bytes   2:00 pm  Fri Sep 16 94
        tty.def            343 bytes   2:00 pm  Fri Sep 16 94
        tty.h            6,000 bytes   2:00 pm  Fri Sep 16 94
        tty.ico            766 bytes   2:00 pm  Fri Sep 16 94
        tty.mak          1,993 bytes   2:00 pm  Fri Sep 16 94
        tty.rc           7,500 bytes   2:00 pm  Fri Sep 16 94
        version.h           64 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\SAMPLES\VER
        .                  <DIR>       11:18 am  Mon Jun 10 96
        ..                 <DIR>       11:19 am  Mon Jun 10 96
        can_adf.rcv      6,067 bytes   2:00 pm  Fri Sep 16 94
        dm309.rcv        6,068 bytes   2:00 pm  Fri Sep 16 94
        finstall.rcv     6,792 bytes   2:00 pm  Fri Sep 16 94
        gen.rcv          6,057 bytes   2:00 pm  Fri Sep 16 94
        hppcl5a.rcv      6,782 bytes   2:00 pm  Fri Sep 16 94
        ibm4019.rcv      6,071 bytes   2:00 pm  Fri Sep 16 94
        lbpii.rcv        6,072 bytes   2:00 pm  Fri Sep 16 94
        lbpiii.rcv       6,075 bytes   2:00 pm  Fri Sep 16 94
        nwpopup.rcv      6,507 bytes   2:00 pm  Fri Sep 16 94
        pg306.rcv        6,068 bytes   2:00 pm  Fri Sep 16 94
        recorder.rcv     5,996 bytes   2:00 pm  Fri Sep 16 94
        sf4019.rcv       6,063 bytes   2:00 pm  Fri Sep 16 94
        sfinst.rcv       6,058 bytes   2:00 pm  Fri Sep 16 94
        terminal.rcv     5,997 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\SAMPLES\VERSTAMP
        .                  <DIR>       11:19 am  Mon Jun 10 96
        ..                 <DIR>       11:19 am  Mon Jun 10 96
        resource.h       1,801 bytes   2:00 pm  Fri Sep 16 94
        verstamp.c      38,767 bytes   2:00 pm  Fri Sep 16 94
        verstamp.def       875 bytes   2:00 pm  Fri Sep 16 94
        verstamp.h       2,732 bytes   2:00 pm  Fri Sep 16 94
        verstamp.ico       766 bytes   2:00 pm  Fri Sep 16 94
        verstamp.mak     2,100 bytes   2:00 pm  Fri Sep 16 94
        verstamp.rc      6,582 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\SAMPLES\WINMEM32
        .                  <DIR>       11:19 am  Mon Jun 10 96
        ..                 <DIR>       11:19 am  Mon Jun 10 96
        winmem32.txt    18,275 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\SAMPLES\WMFDCODE
        .                  <DIR>       11:19 am  Mon Jun 10 96
        ..                 <DIR>       11:19 am  Mon Jun 10 96
        ballon.clp      12,341 bytes   2:00 pm  Fri Sep 16 94
        bird.wmf        15,774 bytes   2:00 pm  Fri Sep 16 94
        cmndlg.c         8,133 bytes   2:00 pm  Fri Sep 16 94
        dlgproc.c       28,000 bytes   2:00 pm  Fri Sep 16 94
        resource.h       3,177 bytes   2:00 pm  Fri Sep 16 94
        wmf.ico            766 bytes   2:00 pm  Fri Sep 16 94
        wmfdcode.c      16,638 bytes   2:00 pm  Fri Sep 16 94
        wmfdcode.def     1,197 bytes   2:00 pm  Fri Sep 16 94
        wmfdcode.h      10,164 bytes   2:00 pm  Fri Sep 16 94
```

```
        wmfdcode.mak      2,709 bytes   2:00 pm  Fri Sep 16 94
        wmfdcode.rc       8,427 bytes   2:00 pm  Fri Sep 16 94
        wmfmeta.c        39,126 bytes   2:00 pm  Fri Sep 16 94
        wmfprint.c        8,312 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\SAMPLES\XTENSION
        .                   <DIR>       11:19 am  Mon Jun 10 96
        ..                  <DIR>       11:19 am  Mon Jun 10 96
        makefile          1,499 bytes   2:00 pm  Fri Sep 16 94
        resource.h          938 bytes   2:00 pm  Fri Sep 16 94
        xtension.c       24,284 bytes   2:00 pm  Fri Sep 16 94
        xtension.def        387 bytes   2:00 pm  Fri Sep 16 94
        xtension.h        1,430 bytes   2:00 pm  Fri Sep 16 94
        xtension.ico      1,086 bytes   2:00 pm  Fri Sep 16 94
        xtension.rc       5,290 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\SOURCE
        .                   <DIR>       11:19 am  Mon Jun 10 96
        ..                  <DIR>       11:19 am  Mon Jun 10 96
        MOVE                <DIR>       11:19 am  Mon Jun 10 96
        STARTUP             <DIR>       11:19 am  Mon Jun 10 96

I:\VC152\MSVC15\SOURCE\MOVE
        .                   <DIR>       11:19 am  Mon Jun 10 96
        ..                  <DIR>       11:19 am  Mon Jun 10 96
        moveapi.h         1,999 bytes   2:00 pm  Fri Sep 16 94
        moveinit.c       13,546 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\SOURCE\STARTUP
        .                   <DIR>       11:19 am  Mon Jun 10 96
        ..                  <DIR>       11:19 am  Mon Jun 10 96
        chkstk.asm        3,793 bytes   2:00 pm  Fri Sep 16 94
        chksum.asm        3,257 bytes   2:00 pm  Fri Sep 16 94
        cmacros.inc      12,039 bytes   2:00 pm  Fri Sep 16 94
        cmsgs.inc         3,754 bytes   2:00 pm  Fri Sep 16 94
        crt0fp.asm        2,007 bytes   2:00 pm  Fri Sep 16 94
        cstartup.bat      3,840 bytes   2:00 pm  Fri Sep 16 94
        csub.bat          1,322 bytes   2:00 pm  Fri Sep 16 94
        defsegs.inc       6,058 bytes   2:00 pm  Fri Sep 16 94
        DOS                 <DIR>       11:19 am  Mon Jun 10 96
        emoem.asm        12,304 bytes   2:00 pm  Fri Sep 16 94
        fcntl.inc         1,562 bytes   2:00 pm  Fri Sep 16 94
        file.asm          2,846 bytes   2:00 pm  Fri Sep 16 94
        file2.h           1,921 bytes   2:00 pm  Fri Sep 16 94
        fmsghdr.asm       2,100 bytes   2:00 pm  Fri Sep 16 94
        heap.inc          3,612 bytes   2:00 pm  Fri Sep 16 94
        internal.c        2,086 bytes   2:00 pm  Fri Sep 16 94
        internal.h        3,749 bytes   2:00 pm  Fri Sep 16 94
        makefile.dos      4,537 bytes   2:00 pm  Fri Sep 16 94
        makefile.win      5,176 bytes   2:00 pm  Fri Sep 16 94
        msdos.h           6,064 bytes   2:00 pm  Fri Sep 16 94
        msdos.inc         7,690 bytes   2:00 pm  Fri Sep 16 94
        nulbody.c            30 bytes   2:00 pm  Fri Sep 16 94
        rchkstk.asm       3,572 bytes   2:00 pm  Fri Sep 16 94
```

```
        readme.txt        6,257 bytes   2:00 pm  Fri Sep 16 94
        register.h          396 bytes   2:00 pm  Fri Sep 16 94
        rterr.inc         4,757 bytes   2:00 pm  Fri Sep 16 94
        setargv.asm         971 bytes   2:00 pm  Fri Sep 16 94
        stdio.inc         1,581 bytes   2:00 pm  Fri Sep 16 94
        stdlib.inc        1,054 bytes   2:00 pm  Fri Sep 16 94
        version.inc         575 bytes   2:00 pm  Fri Sep 16 94
        wild.c            6,574 bytes   2:00 pm  Fri Sep 16 94
        WIN                 <DIR>       11:19 am  Mon Jun 10 96
        windll.mkf          826 bytes   2:00 pm  Fri Sep 16 94
        winexe.mkf          806 bytes   2:00 pm  Fri Sep 16 94
        _file.c           1,307 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\SOURCE\STARTUP\DOS
        .                   <DIR>       11:19 am  Mon Jun 10 96
        ..                  <DIR>       11:19 am  Mon Jun 10 96
        crt0.asm         14,294 bytes   2:00 pm  Fri Sep 16 94
        crt0dat.asm      22,320 bytes   2:00 pm  Fri Sep 16 94
        crt0msg.asm       3,205 bytes   2:00 pm  Fri Sep 16 94
        execmsg.asm       1,171 bytes   2:00 pm  Fri Sep 16 94
        nmsghdr.asm       4,775 bytes   2:00 pm  Fri Sep 16 94
        nulbody.lnk         101 bytes   2:00 pm  Fri Sep 16 94
        stdalloc.asm      2,634 bytes   2:00 pm  Fri Sep 16 94
        stdargv.asm      16,944 bytes   2:00 pm  Fri Sep 16 94
        stdenvp.asm       8,198 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\SOURCE\STARTUP\WIN
        .                   <DIR>       11:19 am  Mon Jun 10 96
        ..                  <DIR>       11:19 am  Mon Jun 10 96
        crt0.asm         16,129 bytes   2:00 pm  Fri Sep 16 94
        fatal.asm         3,560 bytes   2:00 pm  Fri Sep 16 94
        noqwin.asm        3,230 bytes   2:00 pm  Fri Sep 16 94
        nulbody.def         175 bytes   2:00 pm  Fri Sep 16 94
        nulbody.lnk         123 bytes   2:00 pm  Fri Sep 16 94
        qwcinit.asm       5,211 bytes   2:00 pm  Fri Sep 16 94
        stubmain.asm      2,597 bytes   2:00 pm  Fri Sep 16 94
        stubwep.asm       1,369 bytes   2:00 pm  Fri Sep 16 94
        wchkstk.asm       2,850 bytes   2:00 pm  Fri Sep 16 94
        wep.asm           3,986 bytes   2:00 pm  Fri Sep 16 94
        wfile.asm         1,629 bytes   2:00 pm  Fri Sep 16 94
        windgrp.asm       1,167 bytes   2:00 pm  Fri Sep 16 94
        windgrpx.c          519 bytes   2:00 pm  Fri Sep 16 94
        wnull.asm           822 bytes   2:00 pm  Fri Sep 16 94

I:\VC152\MSVC15\TCPIP
        .                   <DIR>       11:19 am  Mon Jun 10 96
        ..                  <DIR>       11:19 am  Mon Jun 10 96
        arp.exe          58,445 bytes   2:10 pm  Fri Jan 13 95
        ftp.exe          81,840 bytes   2:10 pm  Fri Jan 13 95
        hosts.sam           755 bytes   2:10 pm  Fri Jan 13 95
        install.txt      10,680 bytes   2:10 pm  Fri Jan 13 95
        ipconfig.exe     40,997 bytes   2:10 pm  Fri Jan 13 95
        license.txt       2,765 bytes   2:10 pm  Fri Jan 13 95
        lmhosts.sam       3,066 bytes   2:10 pm  Fri Jan 13 95
```

```
        mstcp32.def         632 bytes   2:10 pm  Fri Jan 13 95
        mtcpip32.hlp    228,780 bytes   2:10 pm  Fri Jan 13 95
        nbtstat.exe      33,227 bytes   2:10 pm  Fri Jan 13 95
        netstat.exe      67,595 bytes   2:10 pm  Fri Jan 13 95
        networks            402 bytes   2:10 pm  Fri Jan 13 95
        oemsetup.inf      1,091 bytes   2:10 pm  Fri Jan 13 95
        ping.exe         56,907 bytes   2:10 pm  Fri Jan 13 95
        protocol            794 bytes   2:10 pm  Fri Jan 13 95
        readme.txt       15,440 bytes   2:10 pm  Fri Jan 13 95
        route.exe        55,325 bytes   2:10 pm  Fri Jan 13 95
        services          6,003 bytes   2:10 pm  Fri Jan 13 95
        tcp32ui.dll      63,904 bytes   2:10 pm  Fri Jan 13 95
        telnet.exe       57,152 bytes   2:10 pm  Fri Jan 13 95
        telnet.hlp       21,778 bytes   2:10 pm  Fri Jan 13 95
        tracert.exe      53,935 bytes   2:10 pm  Fri Jan 13 95
        vdhcp.386        20,565 bytes   2:10 pm  Fri Jan 13 95
        vip.386          56,444 bytes   2:10 pm  Fri Jan 13 95
        vnbt.386         89,545 bytes   2:10 pm  Fri Jan 13 95
        vtcp.386         46,782 bytes   2:10 pm  Fri Jan 13 95
        vtdi.386          5,321 bytes   2:10 pm  Fri Jan 13 95
        vudp.386         15,838 bytes   2:10 pm  Fri Jan 13 95
        winsock.dll      39,771 bytes   2:10 pm  Fri Jan 13 95
        wsasrv.exe        6,505 bytes   2:10 pm  Fri Jan 13 95
        wsock.386         9,604 bytes   2:10 pm  Fri Jan 13 95
        wstcp.386        18,295 bytes   2:10 pm  Fri Jan 13 95

I:\VC152\MSVCCDK
        .                   <DIR>       11:23 am  Mon Jun 10 96
        ..                  <DIR>       11:19 am  Mon Jun 10 96
        afxctl.h         59,506 bytes    7:00 am  Wed Aug  9 95
        afxctl.inl        3,118 bytes    7:00 am  Wed Aug  9 95
        bldtyplb.bat        395 bytes    7:00 am  Wed Aug  9 95
        btnctl.bmp          238 bytes    7:00 am  Wed Aug  9 95
        btnctl.cpp        6,888 bytes    7:00 am  Wed Aug  9 95
        btnctl.h          2,027 bytes    7:00 am  Wed Aug  9 95
        btnppg.cpp        2,512 bytes    7:00 am  Wed Aug  9 95
        btnppg.h          1,147 bytes    7:00 am  Wed Aug  9 95
        button.clw        1,244 bytes    7:00 am  Wed Aug  9 95
        button.cpp        2,201 bytes    7:00 am  Wed Aug  9 95
        button.def          921 bytes    7:00 am  Wed Aug  9 95
        button.h            950 bytes    7:00 am  Wed Aug  9 95
        button.ico          768 bytes    7:00 am  Wed Aug  9 95
        button.mak        2,611 bytes    7:00 am  Wed Aug  9 95
        button.odl        2,154 bytes    7:00 am  Wed Aug  9 95
        button.r1         2,514 bytes    7:00 am  Wed Aug  9 95
        button.rc         4,068 bytes    7:00 am  Wed Aug  9 95
        circ1.clw           864 bytes    7:00 am  Wed Aug  9 95
        circ1.cpp         2,310 bytes    7:00 am  Wed Aug  9 95
        circ1.def           918 bytes    7:00 am  Wed Aug  9 95
        circ1.h             945 bytes    7:00 am  Wed Aug  9 95
        circ1.ico           766 bytes    7:00 am  Wed Aug  9 95
        circ1.mak         2,621 bytes    7:00 am  Wed Aug  9 95
        circ1.odl         2,218 bytes    7:00 am  Wed Aug  9 95
        circ1.r3          2,508 bytes    7:00 am  Wed Aug  9 95
```

```
circ1.rc          3,168 bytes   7:00 am  Wed Aug  9 95
circ1ctl.bmp        238 bytes   7:00 am  Wed Aug  9 95
circ1ctl.cpp      5,842 bytes   7:00 am  Wed Aug  9 95
circ1ctl.h        2,238 bytes   7:00 am  Wed Aug  9 95
circ1ppg.cpp      2,758 bytes   7:00 am  Wed Aug  9 95
circ1ppg.h        1,382 bytes   7:00 am  Wed Aug  9 95
circ2.clw           887 bytes   7:00 am  Wed Aug  9 95
circ2.cpp         2,310 bytes   7:00 am  Wed Aug  9 95
circ2.def           918 bytes   7:00 am  Wed Aug  9 95
circ2.h             945 bytes   7:00 am  Wed Aug  9 95
circ2.ico           766 bytes   7:00 am  Wed Aug  9 95
circ2.mak         2,621 bytes   7:00 am  Wed Aug  9 95
circ2.odl         2,481 bytes   7:00 am  Wed Aug  9 95
circ2.r4          2,508 bytes   7:00 am  Wed Aug  9 95
circ2.rc          3,168 bytes   7:00 am  Wed Aug  9 95
circ2ctl.bmp        238 bytes   7:00 am  Wed Aug  9 95
circ2ctl.cpp      9,723 bytes   7:00 am  Wed Aug  9 95
circ2ctl.h        2,611 bytes   7:00 am  Wed Aug  9 95
circ2ppg.cpp      2,758 bytes   7:00 am  Wed Aug  9 95
circ2ppg.h        1,382 bytes   7:00 am  Wed Aug  9 95
circ3.clw         1,194 bytes   7:00 am  Wed Aug  9 95
circ3.cpp         2,310 bytes   7:00 am  Wed Aug  9 95
circ3.def           918 bytes   7:00 am  Wed Aug  9 95
circ3.h             945 bytes   7:00 am  Wed Aug  9 95
circ3.ico           766 bytes   7:00 am  Wed Aug  9 95
circ3.mak         2,621 bytes   7:00 am  Wed Aug  9 95
circ3.odl         2,696 bytes   7:00 am  Wed Aug  9 95
circ3.r5          2,508 bytes   7:00 am  Wed Aug  9 95
circ3.rc          3,618 bytes   7:00 am  Wed Aug  9 95
circ3ctl.bmp        238 bytes   7:00 am  Wed Aug  9 95
circ3ctl.cpp     11,273 bytes   7:00 am  Wed Aug  9 95
circ3ctl.h        2,720 bytes   7:00 am  Wed Aug  9 95
circ3ppg.cpp      3,012 bytes   7:00 am  Wed Aug  9 95
circ3ppg.h        1,349 bytes   7:00 am  Wed Aug  9 95
clockpic.bmp        382 bytes   7:00 am  Wed Aug  9 95
common.x16       82,768 bytes   7:00 am  Wed Aug  9 95
ctlbind.cpp       2,226 bytes   7:00 am  Wed Aug  9 95
ctlcache.cpp      3,394 bytes   7:00 am  Wed Aug  9 95
ctlconn.cpp      12,302 bytes   7:00 am  Wed Aug  9 95
ctlconn2.cpp      3,328 bytes   7:00 am  Wed Aug  9 95
ctlcore.cpp      54,751 bytes   7:00 am  Wed Aug  9 95
ctldata.cpp       9,539 bytes   7:00 am  Wed Aug  9 95
ctldisp.cpp       8,160 bytes   7:00 am  Wed Aug  9 95
ctlevent.cpp     15,868 bytes   7:00 am  Wed Aug  9 95
ctlexcep.cpp      1,754 bytes   7:00 am  Wed Aug  9 95
ctlfact.cpp       7,011 bytes   7:00 am  Wed Aug  9 95
ctlfont.cpp       6,093 bytes   7:00 am  Wed Aug  9 95
ctlframe.cpp      1,966 bytes   7:00 am  Wed Aug  9 95
ctlimpl.h        16,803 bytes   7:00 am  Wed Aug  9 95
ctlinl.cpp          732 bytes   7:00 am  Wed Aug  9 95
ctlinplc.cpp      7,627 bytes   7:00 am  Wed Aug  9 95
ctllic.cpp        2,052 bytes   7:00 am  Wed Aug  9 95
ctlmodul.cpp     15,342 bytes   7:00 am  Wed Aug  9 95
ctlobj.cpp       11,665 bytes   7:00 am  Wed Aug  9 95
```

```
ctlpict.cpp      4,836 bytes   7:00 am  Wed Aug  9 95
ctlppg.cpp      47,602 bytes   7:00 am  Wed Aug  9 95
ctlprop.cpp     21,372 bytes   7:00 am  Wed Aug  9 95
ctlpropx.cpp    23,936 bytes   7:00 am  Wed Aug  9 95
ctlpset.cpp     23,746 bytes   7:00 am  Wed Aug  9 95
ctlpstg.cpp      6,243 bytes   7:00 am  Wed Aug  9 95
ctlpstm.cpp      3,164 bytes   7:00 am  Wed Aug  9 95
ctlrefl.cpp      2,237 bytes   7:00 am  Wed Aug  9 95
ctlreg.cpp      16,183 bytes   7:00 am  Wed Aug  9 95
ctltrack.cpp     7,915 bytes   7:00 am  Wed Aug  9 95
ctltype.cpp      2,875 bytes   7:00 am  Wed Aug  9 95
ctlverb.cpp      5,938 bytes   7:00 am  Wed Aug  9 95
ctlview.cpp      6,340 bytes   7:00 am  Wed Aug  9 95
ctlwzlib.dll    16,288 bytes   7:00 am  Wed Aug  9 95
db.clw           1,281 bytes   7:00 am  Wed Aug  9 95
db.cpp           2,285 bytes   7:00 am  Wed Aug  9 95
db.def             902 bytes   7:00 am  Wed Aug  9 95
db.h               923 bytes   7:00 am  Wed Aug  9 95
db.ico             768 bytes   7:00 am  Wed Aug  9 95
db.mak           2,550 bytes   7:00 am  Wed Aug  9 95
db.odl           2,363 bytes   7:00 am  Wed Aug  9 95
db.rc            4,093 bytes   7:00 am  Wed Aug  9 95
db.rd            2,180 bytes   7:00 am  Wed Aug  9 95
dbctl.bmp          240 bytes   7:00 am  Wed Aug  9 95
dbctl.cpp       10,190 bytes   7:00 am  Wed Aug  9 95
dbctl.h          3,038 bytes   7:00 am  Wed Aug  9 95
dbppg.cpp       11,782 bytes   7:00 am  Wed Aug  9 95
dbppg.h          1,304 bytes   7:00 am  Wed Aug  9 95
licenctl.bmp       238 bytes   7:00 am  Wed Aug  9 95
licenctl.cpp     7,075 bytes   7:00 am  Wed Aug  9 95
licenctl.h       1,844 bytes   7:00 am  Wed Aug  9 95
licenppg.cpp     2,439 bytes   7:00 am  Wed Aug  9 95
licenppg.h       1,148 bytes   7:00 am  Wed Aug  9 95
licensed.clw       900 bytes   7:00 am  Wed Aug  9 95
licensed.cpp     2,334 bytes   7:00 am  Wed Aug  9 95
licensed.def       927 bytes   7:00 am  Wed Aug  9 95
licensed.h         960 bytes   7:00 am  Wed Aug  9 95
licensed.ico       768 bytes   7:00 am  Wed Aug  9 95
licensed.lic       418 bytes   7:00 am  Wed Aug  9 95
licensed.mak     2,675 bytes   7:00 am  Wed Aug  9 95
licensed.odl     2,283 bytes   7:00 am  Wed Aug  9 95
licensed.ra      2,526 bytes   7:00 am  Wed Aug  9 95
licensed.rc      3,202 bytes   7:00 am  Wed Aug  9 95
localctl.bmp       240 bytes   7:00 am  Wed Aug  9 95
localctl.cpp    10,905 bytes   7:00 am  Wed Aug  9 95
localctl.h       2,182 bytes   7:00 am  Wed Aug  9 95
localide.odl     2,466 bytes   7:00 am  Wed Aug  9 95
localifr.odl     2,465 bytes   7:00 am  Wed Aug  9 95
localize.clw       998 bytes   7:00 am  Wed Aug  9 95
localize.cpp     2,488 bytes   7:00 am  Wed Aug  9 95
localize.def       927 bytes   7:00 am  Wed Aug  9 95
localize.h         960 bytes   7:00 am  Wed Aug  9 95
localize.ico       768 bytes   7:00 am  Wed Aug  9 95
localize.mak     2,675 bytes   7:00 am  Wed Aug  9 95
```

localize.odl	2,402	bytes	7:00	am	Wed	Aug	9 95
localize.rb	2,526	bytes	7:00	am	Wed	Aug	9 95
localize.rc	3,372	bytes	7:00	am	Wed	Aug	9 95
localppg.cpp	5,058	bytes	7:00	am	Wed	Aug	9 95
localppg.h	1,447	bytes	7:00	am	Wed	Aug	9 95
locresde.cpp	549	bytes	7:00	am	Wed	Aug	9 95
locresde.def	890	bytes	7:00	am	Wed	Aug	9 95
locresde.mak	2,197	bytes	7:00	am	Wed	Aug	9 95
locresde.rc	2,109	bytes	7:00	am	Wed	Aug	9 95
locresfr.cpp	549	bytes	7:00	am	Wed	Aug	9 95
locresfr.def	890	bytes	7:00	am	Wed	Aug	9 95
locresfr.mak	2,197	bytes	7:00	am	Wed	Aug	9 95
locresfr.rc	2,109	bytes	7:00	am	Wed	Aug	9 95
makefile.0	1,879	bytes	7:00	am	Wed	Aug	9 95
makefile.1	1,880	bytes	7:00	am	Wed	Aug	9 95
makefile.3	1,879	bytes	7:00	am	Wed	Aug	9 95
makefile.4	1,879	bytes	7:00	am	Wed	Aug	9 95
makefile.5	1,875	bytes	7:00	am	Wed	Aug	9 95
makefile.6	1,869	bytes	7:00	am	Wed	Aug	9 95
makefile.7	1,874	bytes	7:00	am	Wed	Aug	9 95
makefile.8	1,886	bytes	7:00	am	Wed	Aug	9 95
makefile.9	1,874	bytes	7:00	am	Wed	Aug	9 95
makefile.a	1,882	bytes	7:00	am	Wed	Aug	9 95
makefile.b	1,886	bytes	7:00	am	Wed	Aug	9 95
makefile.d	1,166	bytes	7:00	am	Wed	Aug	9 95
mfcclswz.dll	494,512	bytes	7:00	am	Wed	Aug	9 95
mfcctlwz.exe	232,224	bytes	7:00	am	Wed	Aug	9 95
mfcctlwz.hlp	39,414	bytes	7:00	am	Wed	Aug	9 95
mscomstf.dl_	41,320	bytes	7:00	am	Wed	Aug	9 95
mscuistf.dl_	14,847	bytes	7:00	am	Wed	Aug	9 95
msdetstf.dl_	14,949	bytes	7:00	am	Wed	Aug	9 95
msinsstf.dl_	40,805	bytes	7:00	am	Wed	Aug	9 95
msshlstf.dl_	9,226	bytes	7:00	am	Wed	Aug	9 95
mstest.dl_	29,000	bytes	7:00	am	Wed	Aug	9 95
mstview.dl_	3,406	bytes	7:00	am	Wed	Aug	9 95
msuilstf.dl_	3,970	bytes	7:00	am	Wed	Aug	9 95
msvc.hlp	471,834	bytes	7:00	am	Wed	Aug	9 95
msvchelp.idx	132,794	bytes	7:00	am	Wed	Aug	9 95
oc25.dll	536,048	bytes	7:00	am	Wed	Aug	9 95
oc25.lib	400,384	bytes	7:00	am	Wed	Aug	9 95
oc25d.dll	3,099,656	bytes	7:00	am	Wed	Aug	9 95
oc25d.lib	571,392	bytes	7:00	am	Wed	Aug	9 95
ocd25.dll	449,292	bytes	7:00	am	Wed	Aug	9 95
ocd25.lib	23,040	bytes	7:00	am	Wed	Aug	9 95
ocd25d.dll	513,068	bytes	7:00	am	Wed	Aug	9 95
ocd25d.lib	27,648	bytes	7:00	am	Wed	Aug	9 95
ocs25.lib	8,192	bytes	7:00	am	Wed	Aug	9 95
ocs25d.lib	338,944	bytes	7:00	am	Wed	Aug	9 95
ocxpg.hlp	848,993	bytes	7:00	am	Wed	Aug	9 95
ocxpg.ind	225,280	bytes	7:00	am	Wed	Aug	9 95
ocxtut.hlp	1,424,497	bytes	7:00	am	Wed	Aug	9 95
ocxtut.ind	593,920	bytes	7:00	am	Wed	Aug	9 95
ole2ui.cpp	16,861	bytes	7:00	am	Wed	Aug	9 95
olectl.h	39,425	bytes	7:00	am	Wed	Aug	9 95

```
olectlid.h        6,211 bytes   7:00 am  Wed Aug  9 95
pal.clw             965 bytes   7:00 am  Wed Aug  9 95
pal.cpp           2,293 bytes   7:00 am  Wed Aug  9 95
pal.def             912 bytes   7:00 am  Wed Aug  9 95
pal.h               935 bytes   7:00 am  Wed Aug  9 95
pal.ico             768 bytes   7:00 am  Wed Aug  9 95
pal.mak           2,557 bytes   7:00 am  Wed Aug  9 95
pal.odl           2,389 bytes   7:00 am  Wed Aug  9 95
pal.r6            2,496 bytes   7:00 am  Wed Aug  9 95
pal.rc            3,013 bytes   7:00 am  Wed Aug  9 95
palctl.bmp          238 bytes   7:00 am  Wed Aug  9 95
palctl.cpp       14,990 bytes   7:00 am  Wed Aug  9 95
palctl.h          2,444 bytes   7:00 am  Wed Aug  9 95
palppg.cpp        2,471 bytes   7:00 am  Wed Aug  9 95
palppg.h          1,120 bytes   7:00 am  Wed Aug  9 95
propset.cpp      43,444 bytes   7:00 am  Wed Aug  9 95
push.clw          1,069 bytes   7:00 am  Wed Aug  9 95
push.cpp          2,301 bytes   7:00 am  Wed Aug  9 95
push.def            915 bytes   7:00 am  Wed Aug  9 95
push.h              940 bytes   7:00 am  Wed Aug  9 95
push.ico            768 bytes   7:00 am  Wed Aug  9 95
push.mak          2,589 bytes   7:00 am  Wed Aug  9 95
push.odl          2,462 bytes   7:00 am  Wed Aug  9 95
push.r7           2,502 bytes   7:00 am  Wed Aug  9 95
push.rc           3,309 bytes   7:00 am  Wed Aug  9 95
pushcd.bmp          566 bytes   7:00 am  Wed Aug  9 95
pushctl.bmp         238 bytes   7:00 am  Wed Aug  9 95
pushctl.cpp       8,246 bytes   7:00 am  Wed Aug  9 95
pushctl.h         2,176 bytes   7:00 am  Wed Aug  9 95
pushcu.bmp          566 bytes   7:00 am  Wed Aug  9 95
pushppg.cpp       2,727 bytes   7:00 am  Wed Aug  9 95
pushppg.h         1,172 bytes   7:00 am  Wed Aug  9 95
rbheap.dl_        6,300 bytes   7:00 am  Wed Aug  9 95
readme.t3_        4,385 bytes   7:00 am  Wed Aug  9 95
readme.t4         4,385 bytes   7:00 am  Wed Aug  9 95
readme.t5         4,385 bytes   7:00 am  Wed Aug  9 95
readme.wri       26,240 bytes   7:00 am  Wed Aug  9 95
regsvr.exe        7,216 bytes   7:00 am  Wed Aug  9 95
resource.h0       1,215 bytes   7:00 am  Wed Aug  9 95
resource.h1       1,404 bytes   7:00 am  Wed Aug  9 95
resource.h3         949 bytes   7:00 am  Wed Aug  9 95
resource.h4         949 bytes   7:00 am  Wed Aug  9 95
resource.h5       1,305 bytes   7:00 am  Wed Aug  9 95
resource.h6       1,213 bytes   7:00 am  Wed Aug  9 95
resource.h7       1,349 bytes   7:00 am  Wed Aug  9 95
resource.h8       1,173 bytes   7:00 am  Wed Aug  9 95
resource.h9       1,259 bytes   7:00 am  Wed Aug  9 95
resource.ha         976 bytes   7:00 am  Wed Aug  9 95
resource.hb       1,218 bytes   7:00 am  Wed Aug  9 95
resource.hd       1,489 bytes   7:00 am  Wed Aug  9 95
setup.exe        24,624 bytes   7:00 am  Wed Aug  9 95
setup.in_         5,176 bytes   4:26 pm  Wed Aug  9 95
setup.lst           608 bytes   7:00 am  Wed Aug  9 95
setupcdk.pc_     19,515 bytes   7:00 am  Wed Aug  9 95
```

```
shipctrl.wri      7,424 bytes   7:00 am  Wed Aug  9 95
spinctl.bmp         238 bytes   7:00 am  Wed Aug  9 95
spinctl.cpp       8,191 bytes   7:00 am  Wed Aug  9 95
spinctl.h         2,093 bytes   7:00 am  Wed Aug  9 95
spindial.clw      1,011 bytes   7:00 am  Wed Aug  9 95
spindial.cpp      2,333 bytes   7:00 am  Wed Aug  9 95
spindial.def        927 bytes   7:00 am  Wed Aug  9 95
spindial.h          960 bytes   7:00 am  Wed Aug  9 95
spindial.ico        768 bytes   7:00 am  Wed Aug  9 95
spindial.mak      2,661 bytes   7:00 am  Wed Aug  9 95
spindial.odl      2,340 bytes   7:00 am  Wed Aug  9 95
spindial.r8       2,526 bytes   7:00 am  Wed Aug  9 95
spindial.rc       2,985 bytes   7:00 am  Wed Aug  9 95
spinppg.cpp       2,635 bytes   7:00 am  Wed Aug  9 95
spinppg.h         1,174 bytes   7:00 am  Wed Aug  9 95
stdafx.c0           628 bytes   7:00 am  Wed Aug  9 95
stdafx.c1           628 bytes   7:00 am  Wed Aug  9 95
stdafx.c3           628 bytes   7:00 am  Wed Aug  9 95
stdafx.c4           628 bytes   7:00 am  Wed Aug  9 95
stdafx.c5           628 bytes   7:00 am  Wed Aug  9 95
stdafx.c6           628 bytes   7:00 am  Wed Aug  9 95
stdafx.c7           628 bytes   7:00 am  Wed Aug  9 95
stdafx.c8           628 bytes   7:00 am  Wed Aug  9 95
stdafx.c9           628 bytes   7:00 am  Wed Aug  9 95
stdafx.ca           628 bytes   7:00 am  Wed Aug  9 95
stdafx.cb           628 bytes   7:00 am  Wed Aug  9 95
stdafx.cd           626 bytes   7:00 am  Wed Aug  9 95
stdafx.h0           664 bytes   7:00 am  Wed Aug  9 95
stdafx.h1           664 bytes   7:00 am  Wed Aug  9 95
stdafx.h3           664 bytes   7:00 am  Wed Aug  9 95
stdafx.h4           664 bytes   7:00 am  Wed Aug  9 95
stdafx.h5           664 bytes   7:00 am  Wed Aug  9 95
stdafx.h6           664 bytes   7:00 am  Wed Aug  9 95
stdafx.h7           664 bytes   7:00 am  Wed Aug  9 95
stdafx.h8           664 bytes   7:00 am  Wed Aug  9 95
stdafx.h9           664 bytes   7:00 am  Wed Aug  9 95
stdafx.ha           664 bytes   7:00 am  Wed Aug  9 95
stdafx.hb           664 bytes   7:00 am  Wed Aug  9 95
stdafx.hd           668 bytes   7:00 am  Wed Aug  9 95
stdafx.hq         2,342 bytes   7:00 am  Wed Aug  9 95
stdctl.h            545 bytes   7:00 am  Wed Aug  9 95
stdtype.odl       7,127 bytes   7:00 am  Wed Aug  9 95
testcon.hlp      51,805 bytes   7:00 am  Wed Aug  9 95
testexit.ex_      3,266 bytes   7:00 am  Wed Aug  9 95
time.clw          1,030 bytes   7:00 am  Wed Aug  9 95
time.cpp          2,301 bytes   7:00 am  Wed Aug  9 95
time.def            915 bytes   7:00 am  Wed Aug  9 95
time.h              940 bytes   7:00 am  Wed Aug  9 95
time.ico            768 bytes   7:00 am  Wed Aug  9 95
time.mak          2,589 bytes   7:00 am  Wed Aug  9 95
time.odl          2,316 bytes   7:00 am  Wed Aug  9 95
time.r9           2,502 bytes   7:00 am  Wed Aug  9 95
time.rc           3,117 bytes   7:00 am  Wed Aug  9 95
timectl.bmp         238 bytes   7:00 am  Wed Aug  9 95
```

```
timectl.cpp       8,679 bytes   7:00 am  Wed Aug  9 95
timectl.h         2,317 bytes   7:00 am  Wed Aug  9 95
timeppg.cpp       2,643 bytes   7:00 am  Wed Aug  9 95
timeppg.h         1,153 bytes   7:00 am  Wed Aug  9 95
tstcon16.exe    343,248 bytes   7:00 am  Wed Aug  9 95
wbexec20.dl_     58,704 bytes   7:00 am  Wed Aug  9 95
wbrun20.ex_      11,200 bytes   7:00 am  Wed Aug  9 95
winbas20.dl_     68,488 bytes   7:00 am  Wed Aug  9 95
xlist.clw           961 bytes   7:00 am  Wed Aug  9 95
xlist.cpp         2,310 bytes   7:00 am  Wed Aug  9 95
xlist.def           918 bytes   7:00 am  Wed Aug  9 95
xlist.h             945 bytes   7:00 am  Wed Aug  9 95
xlist.ico           768 bytes   7:00 am  Wed Aug  9 95
xlist.mak         2,621 bytes   7:00 am  Wed Aug  9 95
xlist.odl         3,181 bytes   7:00 am  Wed Aug  9 95
xlist.r0          2,508 bytes   7:00 am  Wed Aug  9 95
xlist.rc          2,973 bytes   7:00 am  Wed Aug  9 95
xlistctl.bmp        238 bytes   7:00 am  Wed Aug  9 95
xlistctl.cpp     34,907 bytes   7:00 am  Wed Aug  9 95
xlistctl.h        5,021 bytes   7:00 am  Wed Aug  9 95
xlistppg.cpp      2,507 bytes   7:00 am  Wed Aug  9 95
xlistppg.h        1,146 bytes   7:00 am  Wed Aug  9 95
```

8.1 Description of PROM IC options

The Atmel 27C080-12PC IC used in the computational block contains 1-Mbyte of once programmable read-only memory (PROM). Since the boot disk only contains 128-kbytes, another option would be to use smaller PROM ICs. PROMs compatible with the Ampro 3SXi computer are available in the following sizes:

- 27C010 – 128-kbytes
- 27C020 – 256-kbytes
- 27C040 – 512-kbytes
- 27C080 – 1024-kbytes = 1-Mbyte ← Used.

The Atmel 27C080 PROM ICs come in two styles, once programmable and UV erasable. Two types of memory ICs from Atmel have been used:

- 27C080-12 PC — Once programmable ← Used
- 27C080-12 DC — UV erasable.

The UV erasable version is useful during program development. The once programmable version is used in an information barrier system to reduce the capability to change the system. The PC suffix indicates a plastic DIP package with a commercial temperature range. The DC suffix indicates a cerdip (ceramic) package with a commercial temperature range.

A side-by-side photograph and x-ray of these two PROM devices were provided in section 2.4.1 and 2.4.2 respectively. The data sheet for the Atmel 27C080 PROM was provided in section 2.4.3. This data sheet was printed from the Atmel Internet site at:

http://www.atmel.com/atmel/acrobat/doc0360.pdf.

8.2 Atmel, 27C080-12 PC PROM – Hardware

This section includes:

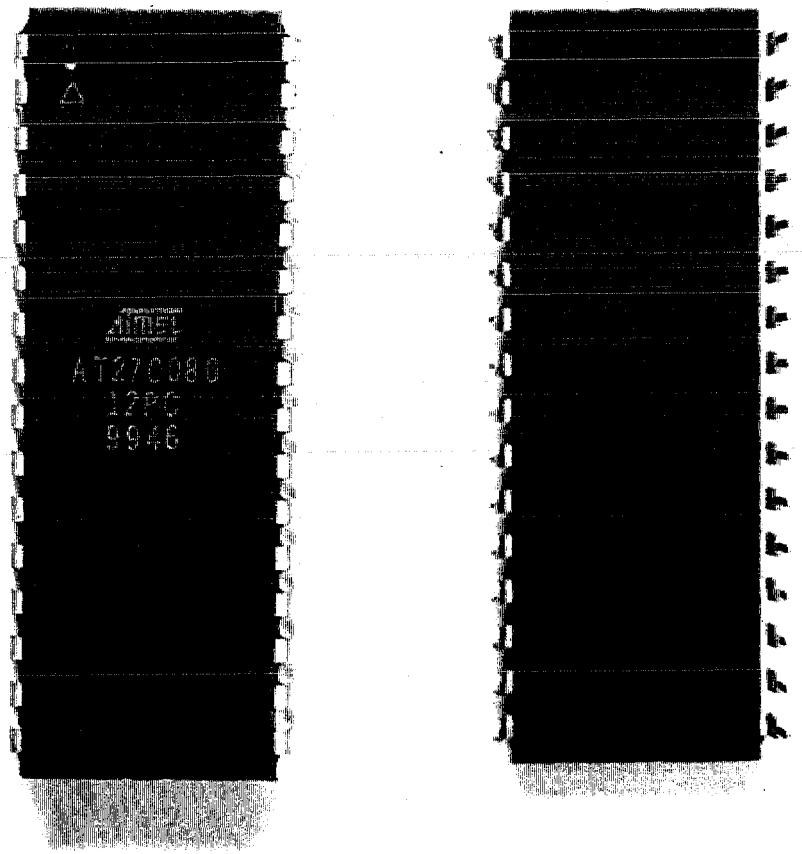
- Photograph of the top and bottom view of the Atmel 27C080-12 PC PROM IC
- Three x-ray images of the Atmel 27C080-12 PC PROM IC at three brightness levels.

Various levels of detail can be seen in the x-ray of the plastic-cased PROM. The metal legs are easily seen in all the views. The central region appears to be supported by a central metal trace. This might be a ground plane or UV light shield.

Once-Programmable PROM Integrated Circuits

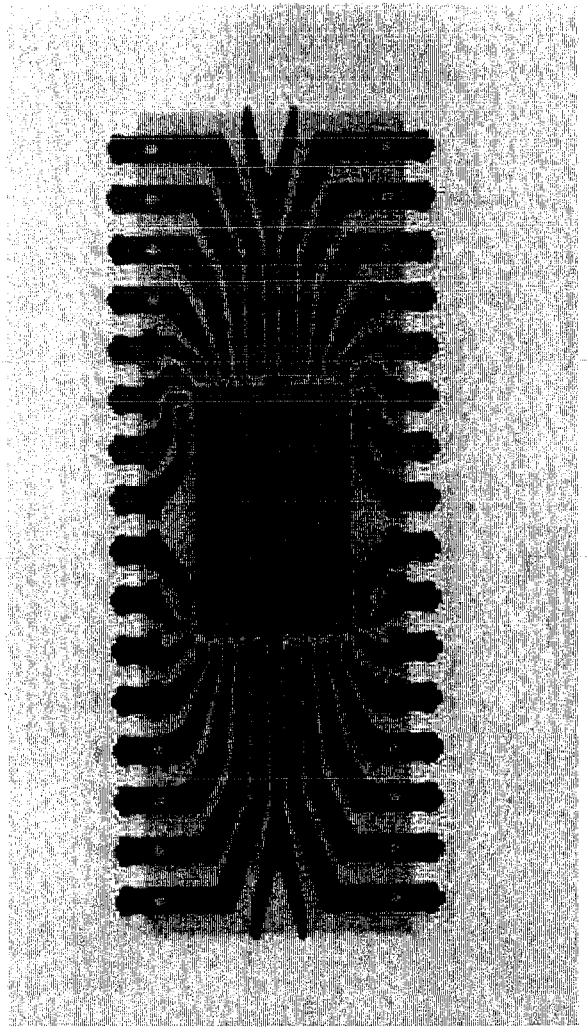
Top View

Bottom View

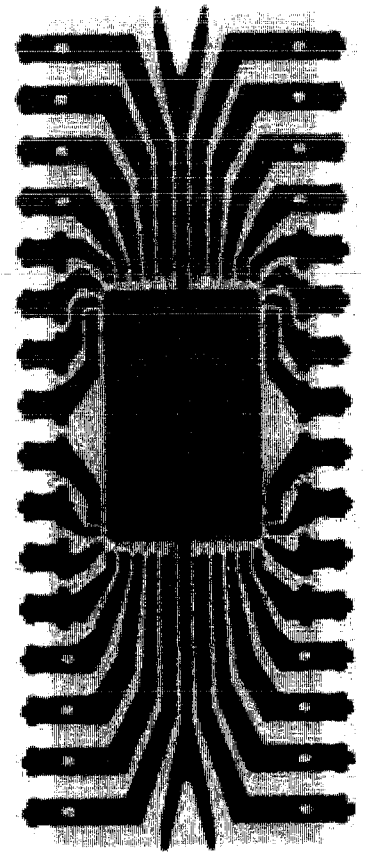


Once-Programmable PROM
AT27C080-12PC

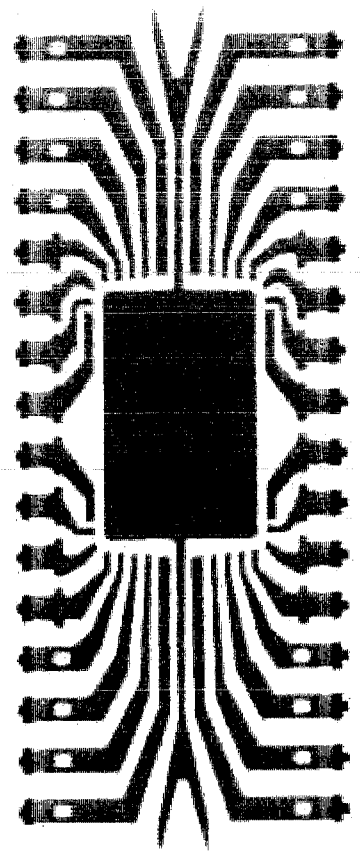
Once-Programmable PROM IC – X-Ray



Darkened X-Ray



Raw X-Ray



Brightened X-Ray

8.3 Atmel, 27C080-12 PC PROM – Software

The PROM software includes textual files (e.g., CONFIG.SYS and AUTOEXEC.BAT), operating system executable files (e.g., IBMDOS.COM and COMMAND.COM), and an application executable file (e.g., DATA_ATT.EXE.). In addition, the two basic ROM-DOS operating system executable files (e.g., IBMDOS.COM and IBMBIO.COM) are hidden files following the normal DOS convention, which prevents accidental erasure of these system files from a boot disk.

Directory of BOOT Disk – Floppy disk matching of the PROM disk

```
 Volume in drive A has no label
 Volume Serial Number is 2D69-14D7
 Directory of A:\

IBMBIO   COM        54,542  07-22-99  6:22a IBMBIO.COM -- Hidden
IBMDOS   COM            74  07-22-99  6:22a IBMDOS.COM -- Hidden
COMMAND  COM        34,099  07-22-99  6:22a COMMAND.COM
AUTOEXEC BAT            73  11-04-99  9:07a AUTOEXEC.BAT
HIMEM    SYS         5,832  07-22-99  6:22a HIMEM.SYS
DATA_ATT EXE        87,608  01-31-00  1:05p DATA_ATT.EXE
CONFIG   SYS            85  02-08-00 10:16a CONFIG.SYS
         5 file(s)          127,697 bytes
         0 dir(s)           832,512 bytes free
```

Directory of OUTFILE Disk – Floppy disk with image of the PROM contents OUTFILE.SO is the exact image of the 1-Mbyte PROM as programmed

```
 Volume in drive A has no label
 Directory of A:\

OUTFILE  SO       1,048,576  02-08-00 10:18a OUTFILE.SO
         1 file(s)      1,048,576 bytes
         0 dir(s)         409,088 bytes free
```

Internet References

A large amount of documentation for the Computational Block hardware is found on the Internet at vendor web sites. The following is a comprehensive listing of Internet address references related to the AMS computational block hardware and software.

Ampro 3SXi – Processor Board

The Ampro base Internet address is http://www.ampro.com
Ampro Hardware Information is located at the following sites
- Product list: http://www.ampro.com/products/index.htm
- CoreModule 3SXi: http://www.ampro.com/products/coremod/cm-3sxi.htm
- 3SXi product data sheet: http://www.ampro.com/products/coremod/cm3sxi2.pdf
- 3SXi technical manual: http://www.ampro.com/techman/coremodule/5001131e.pdf

Ampro BIOS Information:
- Features w/ hooks: http://www.ampro.com/products/software/sw-bios.htm
- Hook #1: http://www.ampro.com/products/software/sw-bios1.htm
- Hook #2: http://www.ampro.com/products/software/sw-bios2.htm
- Hook #3: http://www.ampro.com/products/software/sw-bios3.htm

Ampro Common Utilities:
- 1-page summary of the Ampro BIOS & Utilities:
  - http://www.ampro.com/products/software/sw-bios.pdf
- Technical manual: http://www.ampro.com/techman/5000931b.pdf

Ampro Development Platform:
- This is hardware to develop PC104 systems on containing:
  - Power supply, floppy disk, hard disk, PC104 socket, two ISA sockets
- Info: http://www.ampro.com/products/coremod/cm-ids.htm
- Data Sheet: http://www.ampro.com/products/coremod/cm-ids.pdf
- Manual: Purchased in Quick Start Kit

Support info: http://www.ampro.com/support/index.htm
Timing info: http://www.ampro.com/university/wpapers/ISAtiming.pdf
Applications & White Papers: http://www.ampro.com/university/index.htm

Components of the 3SXi Processor Board – Memory ICs

SO = Byte-wide socket = 8-bit wide memory (14-bit address 16,384 bytes)
ATMEL AT27C080 = 1-Mbyte EPROM memory = http://www.atmel.com
Data Sheet = http://www.atmel.com/atmel/acrobat/doc0360.pdf
Part Number Codes: = http://www.atmel.com/atmel/acrobat/doc0538.pdf
Alternate choices: 27C040 = 0.5-Mbtye memory -- EPROM
Info:http://www.fairchildsemi.com/pf/NM/NM27C040.html
Data Sheet: http://www.fairchildsemi.com/ds/NM/NM27C040.pdf
Application: http://www.fairchildsemi.com/an/AN/AN-794.pdf

Main CPU Memory can be at U1 or (U2&U4) or (U3&U5) → Only U3 & U5 is installed

U1 = MEM32 = 32-bit DRAM memory (13-bit address 16,384 words) = missing

U2 = 256Kx16 = 256-kwords of 16-bit DRAM memory (9-bit address 512-words) = missing
U4 = 256Kx16 = 256-kwords of 16-bit DRAM memory (9-bit address 512-words) = missing

U3 = 1MX16 = 1-Mwords of 16-bit DRAM memory (10-bit address 1-kwords) = Toshiba
U5 = 1MX16 = 1-Mwords of 16-bit DRAM memory (10-bit address 1-kwords) = Toshiba
2-Mbyte RAM with IC labeled: Toshiba A24419 | 9817KBD | TC5118160CFT-60
Toshiba TC5118160CFT-60 is a 60-ns 2-Mbyte DRAM organized as 1Mx16 (1-M of 16-bits).
Toshiba no longer lists this IC on the internet and perhaps no longer makes it.
HITACHI makes an equivalent chip is HM5118160JP-8
First info: http://www.hitachi-eu.com/hel/ecg/sitemap/search_memory.htm
Short info: http://www.hitachi-eu.com/hel/ecg/sitemap/search_memory.htm
Data Sheet: http://www.hitachi-eu.com/hel/ecg/products/memory/pdf/5116160.pdf
SAMSUNG makes an equivalent chip is KM416C1200AJ-8:
First Info: http://www.usa.samsungsemi.com/admin/production/globalsearch.asp?key=KM416C1200
Short Info: http://www.usa.samsungsemi.com/products/summary/asyncdram/KM416C1200C.htm
Data Sheet:
http://www.usa.samsungsemi.com/Memory/DRAM/DRAM_Components/Asynchronous_DRAM/16Mb/KM416C1200C/KM416C1200C.PDF

Flash Memory containing BIOS and extra space can be U6 or U7 → U6 is installed
U6 = 28F008SA = 1-Mbyte of 8-bit wide data (20-bit address 1Mbyte) = INTEL Flash memory
INTEL [1-Mbyte Flash 64-kbyte BIOS rest OEM]
Data Sheet: ftp://download.intel.com/design/flcomp/datashts/29042908.pdf
Prod Codes: http://apps.intel.com/product_selector/chart1.asp
U7 = 28F010 = 128-kbyte of 8-bit wide data (18-bit address 262-kbytes) = missing
INTEL [128-kbyte Flash 64-kbyte BIOS 64-kbyte OEM]
Data Sheet: ftp://download.intel.com/design/flcomp/datashts/29066301.pdf

Note: Not all memory components for the Ampro 3SXi computer board can be simultaneously installed.

Components of the 3SXi Processor Board – Non-Memory ICs

U8 = RTC = Real-time clock = Benchmarq BQ3285S
IC label = BENCHMARQ | bq3285S | 9930-SB2 (S=24-pin SOIC package)
Benchmarq is now a component or division of Texas Instruments.
General info: http://www.benchmarq.com/pressrel/pr.11.html
Products: http://www-s.ti.com/cgi-bin/sc/generic2.cgi?keyword=REAL-TIME+CLOCKS&family=REAL-TIME+CLOCKS&tech=All
Summary: http://www.ti.com/sc/docs/products/analog/bq3285.html
Data Sheet: http://www-s.ti.com/sc/psheets/slus028/slus028.pdf
Datasheet: http://www-s.ti.com/sc/psheets/slua052/slua052.pdf
Applications: http://www-s.ti.com/sc/psheets/slua094/slua094.pdf
Oscillators: http://www-s.ti.com/sc/psheets/slua051/slua051.pdf

U9 = 16V8 = Programmable Electrically Erasable Logic (PEEL) IC
IC paper label = 7104 | 145A
ICT Summary Info: http://www.nuhorizons.com/linecard/ict/16v8.htm
Integrated Circuit Technology (ICT) Home Page: http://www.ictpld.com/
Info: http://www.ictpld.com/products/products.htm
Datasheet: http://www.ictpld.com/products/16cv8.htm

U10 = ALi M6117C = 386SX CPU
IC label = ALi M6117C A1 | 9937 TS05 | XBM24391000D
Acer Laboratories Inc home page: http://www.acerlabs.com/
The ALi M6117 is an embedded microcontroller which contains a 386SX, a coprocessor interface, a memory controller, peripheral interface, ISA interface, built-in real-time clock, keyboard controller, PMU interface, watchdog timer, IDE & interface
First Info from SSV
Functional Description ALI M6117: http://www.ssv-embedded.de/ssv/pc104/p24.htm
BIOS setup: http://www.award.com.tw/DOCS/Internet/docs/alim6117.htm
ALi M6117 Info: http://www.ali.com.tw/eng/product/embed/m6117c.htm
ALi M6117 Manual: E:\FMTT\6117Cds093 (not on the network)

U11 = MAX809 = Power-On-Reset IC = J1JA
Info: http://www.nuhorizons.com/new/MAX4.htm
MAXIM data sheet list: http://www.iit.edu/~branjov/link/MAXIM.HTM
MAXIM products: http://www.maxim-ic.com/MaximProducts/products.htm
MAX809 data sheet: http://209.1.238.250/arpdf/1194.pdf

U12 = 37C665 = I/O controller
IC label = SMSC | FDC37C665GT | B9912-D347 | 6H14211-5
SMSC = Standard Microsystems Corporation
Summary info: http://www.smsc.com/main/catalog/fdc37c665gt.html
Data Sheet 152-pages: http://www.smsc.com/main/datasheets/37c665gt.pdf

U13 = SXCVR = RS-232 transceiver
IC label = SIPEX | SP211CA | 9924
SP211E data sheet: http://www.sipex.com/dsheets/SP207E.pdf
Note SP211, SP211B, SP211E, SP211EH, SP211H were the choices

U14 = SXCVR = RS-232 transceiver: SIPEX | SP211CA | 9924
See U14

U15 = 74ACT02 = NORs = F P95SB | 74ACT02 = Quad 2-input NOR gate
Fairchild: 74ACT02
General: http://www.fairchildsemi.com/pf/74/74ACT02.html#General Description

U16 = NVCTRL = MXD1210 | CSA | 940 = Nonvolatile RAM controller
MAXIM info: http://www.maxim-ic.com/
MAXIM data sheet: http://209.1.238.250/arpdf/1194.pdf

U17 = 74HCT174 = 6-bit register = HCT174 | H9845
Harris chip is on the board but the following are equivalents with data available
Phillips info: http://www-us2.semiconductors.philips.com/pip/74HC174D
Motorola info: http://scgproducts.motorola.com/ProdSum.asp?base=MC74HC174A
Data Sheet : http://scgproducts.motorola.com/onsemi/Collateral/DataSheet/mc74hc174arev6.pdf

U18 = MXLT62 = labeled as LT923 |1262 = 8-pin chip provides PROG and VPP signals
Linear Technology LTC1262 --12V, 30mA Flash Memory Programming Supply
Summary: http://www.linear.com/cgi-bin/database?function=elementinhtml&filename=DataSheet.html&name=DataSheet&num=111
Data Sheet: http://www.linear-tech.com/pdf/1262fa.pdf

U19 = 9356 = Motorola 93LC56X | /SN = 2-kbit EEPROM
Fairchild Info: http://www.fairchildsemi.com/pf/NM/NM93C56.html
Fairchild Data Sheet: http://www.fairchildsemi.com/ds/NM/NM93C56.pdf
Application Note AAN-8805 says National NMC9346 (or equivalent) is a serial 1024-bit EEPROM organized as 64 16-bit words.

U20 = FG54-16 = Clock chip = ICS AV9154-16 | A9934 | AN500490
Base: http://www.icst.com/
Data Sheet: http://www.icst.com/pdf/9154-16.pdf

U21 = 74HCT245 = 8-bit register = F P85SG | MM74HCT | 245WM
Fairchild info: http://www.fairchildsemi.com/pf/MM/MM74HCT245.html
Datasheet: http://www.fairchildsemi.com/ds/MM/MM74HCT245.pdf

U22 = 74HCT245 = 8-bit register = F P85SG | MM74HCT | 245WM – See U21

U23 = 74HCT245 = 8-bit register = F P85SG | MM74HCT | 245WM – See U21

U24 = 74HCT244 = 8-bit register = Motorola HCT244A | CPBC9924
Fairchild
Short Info: http://www.fairchildsemi.com/pf/MM/MM74HCT244.html
Data sheet: http://www.fairchildsemi.com/ds/MM/MM74HCT244.pdf
Motorola Search: http://search.motorola.com/semiconductors/query.html?qt=HCT244A
Short Info: http://scgproducts.motorola.com/ProdSum.asp?base=MC74HCT244A
Data Sheet: http://scgproducts.motorola.com/onsemi/Collateral/DataSheet/mc74hct244arev7.pdf

U25 = 75176 = LTC485 = RS486 line transceiver = labeled LT930 | 485
DL159 long paper on RS232 & RS486
Linear tecnology LTC485 Low Power RS485 Interface Trasceiver
Summary: http://www.linear.com/cgi-bin/database?function=elementinhtml&filename=DataSheet.html&name=DataSheet&num=347
Data Sheet: http://www.linear-tech.com/pdf/lt0485.pdf

U35 = 74HCT245 = 8-bit register = F P85SG | MM74HCT | 245WM
See U21

Components of the 3SXi Processor Board -- Missing ICs

Keyboard controller chips are missing from the 3SXi board an presumably this is handled internally to the ALi M6117C

U55 = 82C42 = QFP-44 = keyboard something = missing

U56 = 74ACT05 = inverters = keyboard something = missing
Short Info: http://scgproducts.motorola.com/ProdSum.asp?base=MC74ACT05
Sheet: http://scgproducts.motorola.com/onsemi/Collateral/DataSheet/mc74ac05rev3.pdf

Components of the 3SXi Processor Board -- Crystals for various clocks

Y1 = Crystal: 32768 = 32.768-kHz quartz clock crystal for RTC in ALi M6117C
This is missing from the board because the M6117C internal RTC is not used

Y2 = Crystal: E14.318 = 14.318-MHz clock crystal for the ICS FG54-16 clock IC in U20

Y3 = Crystal: 32768 = 32.768-kHz quartz clock crystal for the Benchmarq RTC in U8
Benchmarq recommends a Diawa DT-26 or equivalent as the 32.768-kHz crystal
Package shape matches KDS America DT-26 or Epson America C-002RX
DT-26 Data sheet: http://www.kdsj.co.jp/english.html
Note select English at the KDSJ home page site at http://www.kdsj.co.jp

Ampro SVG-II – Video Board

The SVG-II video board is not on the Ampro Web site because this product is subject to continuing availability. The Ampro Internet address is http://www.ampro.com

Components of the SVG-II Video Board

U1 = CIRRUS LOGIC CL-GD5429 = Memory-Mapped I/O VGA GUI Accelerator

Info: http://www.cirrus.com/servlet/HTMLEngine/dashboard/srframe.html?SearchString=CL-GD5429&AttributeBlockCount=0&ReplaceBlockID=106&BlockID=106&LastID=0&ContentID=1316

Specifications: http://www.cirrus.com/products/overviews/gd5429.html

Data Book: http://216.35.18.143:80/servlet/SiteDriver/Content/655/gd542xdb.pdf

Technical Manual: http://216.35.18.143:80/servlet/SiteDriver/Content/656/gd542xtrm.pdf

U4 & U5 = MT 4C16257 = 256kx16 -- 512-kbyte DRAM memory

IC may be from Micron ?

Info is from Fairchild 27C256 - MEMORY ON SVG-2

Info: http://www.fairchildsemi.com/pf/NM/NM27C256.html

Data sheet: http://www.fairchildsemi.com/ds/NM/NM27C256.pdf

U7 = Motorola 74HCT244A = 8-bit Register

Short Info: http://scgproducts.motorola.com/ProdSum.asp?base=MC74HCT244A

Data Sheet: http://scgproducts.motorola.com/onsemi/Collateral/DataSheet/mc74hct244arev7.pdf

U11 = LM334 on SVG Constant Source and Temperature Sensor

Linear Technology L334-3A data sheet

Info: http://www.linear.com/cgi-bin/database?function=elementinhtml&filename=DataSheet.html&name=DataSheet&num=156

Y1 = E14.318 = 14.318-MHz Clock crystal for CL-GD5429

Diamond Systems Emerald-MM-DIO – Extra I/O Board

The Diamond Systems Home Page: http://www.diamondsystems.com/
Emerald-MM-DIO product sheet: http://www.diamondsystems.com/emmdio.htm
Emerald-MM-DIO manual: http://www.diamondsystems.com/files/emmdio.pdf

Components of the Emerald-MM-DIO Extra I/O Board

U1-U4 = SIPEX = RS-232 Line Driver/Receiver = SIPEX | SP211CA | 9924
SP211E data sheet: http://www.sipex.com/dsheets/SP207E.pdf

U5 = XILINX 17128EPI | 925884 = 128 kbit one-time programmable memory for FPGA
Info: http://www.xilinx.com/support/programr/files/1700ex.pdf

U6 = XILINX XC5204 = FPGA – 480 Cells
One major component of the DIO card is the XILINX = XC5204 FPGA – 480 Cells
Data Sheet: http://www.xilinx.com/partinfo/5200.pdf

U7 = Texas Instruments TL 16C554 = Quad UART Chip
One major component of the DIO card is the Texas Instruments TL16C554 IC
Information from the Diamond Systems site:
TL16C554 Quad UART IC: http://www.diamondsystems.com/files/TL16C554.pdf
The Texas Instruments Home Page is at: http://www-s.ti.com
TL16C554 info: http://www.ti.com/sc/docs/products/analog/tl16c554.html
TL16C554 data sheet: http://www-s.ti.com/sc/psheets/slls165d/slls165d.pdf#xml=http://www-search.ti.com/search97cgi/s97r_cgi?action=View&VdkVgwKey=http%3A%2F%2Fwww%2Ds%2Eti%2Ecom%2Fsc%2Fpsheets%2Fslls165d%2Fslls165d%2Epdf&doctype=xml&Collection=TechDocs&QueryZip=tl16c554&

Y1 = PLE SRMP49 = 1.8432-MHz quartz clock crystal

Support Hardware – Other Parts

Line Filter Corcom3EC1: http://www.cor.com/
Data Sheet: http://www.cor.com/catalog/filters/ec/default.htm

Power supply Condor GSC20-5 5V 3.8A switching power supply
Data sheet Info: http://www.powerbox.com.au/gsc20.htm
Info: http://www.digikey.com/MKT/New975/Condor/Commercial.html

Fuse assembly = Telfuse 250 VAC max

Push-Button On/Off Switch =

Metal Case =

Computational Block Supporting Information – PC104

PC-104 Specifications from Diamond Systems
PC-104: http://www.diamondsystems.com/files/pc104-23.pdf
PC-104-Plus: http://www.diamondsystems.com/files/pc104p10.pdf

Non-Computational Block Hardware Information – Counter Card

Symmetry counter card:
Base address: http://www.computerboards.com/specs.html
Data sheet: http://www.computerboards.com/pdfs/pc104-ctr10hd.pdf
AM9513 chip: http://www.computerboards.com/PDFmanuals\9513A.pdf

Non-Computational Block Hardware Information – SSD Card

Ampro Drivers for SSD/DOS:
http://www.ampro.com/products/software/sw-ssd.htm
http://www.ampro.com/products/software/sw-ssd.pdf

Computational Block Software Information – ROM-DOS

ROM-DOS by Datalight Inc.
Datalight home page: http://www.datalight.com
Datalight ROM-DOS: http://www.datalight.com/products.htm#rom-dos
ROM-DOS info: http://www.datalight.com/rom-dos.htm
User's Guide Manual: http://www.datalight.com/eval/rd-ug.pdf
Hidden files: http://www.datalight.com/superboot-wp.htm
Dynamic configuration & Vdisk: http://www.datalight.com/dyndrv-wp.htm
Supper Boot: http://www.datalight.com/tb-superboot.htm
Download manuals: http://www.datalight.com/files.htm

Computational Block Software Information – BIOS

BIOS: AWARD BIOS | c1998 | ISA 386 | 161995858 @U10 CPU chip
Award Home Page: http://www.award.com
Phoenix acquired Award – Phoenix home page: http://www.phoenix.com
Award BIOS info: http://www.phoenix.com/platform/awardbios.html
PC user support: http://www.phoenix.com/pcuser/
PhoenixBIOS 4.0 Release 6 User's Manual: http://www.phoenix.com/pcuser/userman.pdf
Setup info: http://www.phoenix.com/pcuser/bios-award-setup.html
Index of setup fields: http://www.phoenix.com/pcuser/bios-award-fields.html
Setup Award BIOS for M6117: http://www.phoenix.com/pcuser/bios-award-m6117.pdf
Manual Download Menu: http://www.phoenix.com/pcuser/index.html
BIOS upgrade info: http://www.unicore.com/
Note no BIOS ID string appeared at bottom of screen during boot up
Good BIOS search engine: http://www.ping.be/bios/

Computational Block Software Information – COMM LIBRARY

Willies Computer Software Company, WCSC home page: http://www.wcscnet.com/
COMM-DRV/LIB info: http://www.wcscnet.com/CdrvLBro.htm

Microsoft Visual C

Data Sheet Version 6.0: http://msdn.microsoft.com/visualc/prodinfo/datasheet/default.asp
Previous Versions to 4.0: http://msdn.microsoft.com/visualc/prodinfo/previous/default.asp
Alternate C/C++ compilers that generate DOS code
C-Compilers: http://www.grey-matter.com/gmWEB/nodes/NODE05B8.HTM
Borland C++ version 5.0 http://www.grey-matter.com/gmWEB/nodes/NODE05B8.HTM
Borland features http://www.grey-matter.com/Blobs/pdf/00002026.pdf
Borland features: http://www.Borland.com/bcppbuilder/freecompiler/feaben.html
Borland Turbo C for DOS: http://www.borland.com/borlandcpp/turbosuite/turbo3.html
Watcom C++: http://www.grey-matter.com/Blobs/pdf/00001800.pdf
Symantec C++: http://www.symantec.com/scpp/fs_scpp72_95.html
Free C: http://www.delorie.com/

Other Supporting Information

RS232 Standards from Motorola:
http://www.mot.com/SPS/MCTG/MDAD/pdf/DL159/AN781A.pdf

DL159—LonTalk 234-pages with Interconnection standards from Motorola
http://www.mot.com/SPS/MCTG/MDAD/pdf/DL159/DL159.Rev5vol3.pdf

Memory Guide:
http://www.kingston.com/tools/umg/umg-en.pdf

Logic Chips On Web
http://www.zettweb.com/CDROMs/cdrom006/families/logic.htm#CD54/74 AC/ACT
Commercial Logic Data Sheets

List of IC manufactures links
http://www.mitronet.com/chipdir/c/a.htm
http://www.SemiResources.com/

Data Book Shelf links
http://www.crhc.uiuc.edu/~dburke/databookshelf.html

Emac home page for other computers in the AMS
http://www.emacinc.com/

Books:
BIOS: http://www.advicepress.com/products/books/Companion/BIOS/bios.html