User's Manual
from Emerson Network Power™
Embedded Computing

PmPPC: PowerPC™-Based PMC Module

October 2007

EMERSON
Network Power

The information in this manual has been checked and is believed to be accurate and reliable. HOWEVER, NO RESPONSIBILITY IS ASSUMED BY EMERSON NETWORK POWER, EMBEDDED COMPUTING FOR ITS USE OR FOR ANY INACCURACIES. Specifications are subject to change without notice. EMERSON DOES NOT ASSUME ANY LIABILITY ARISING OUT OF USE OR OTHER APPLICATION OF ANY PRODUCT, CIRCUIT, OR PROGRAM DESCRIBED HEREIN. This document does not convey any license under Emerson patents or the rights of others.

Emerson. Consider It Solved is a trademark, and Business-Critical Continuity, Emerson Network Power, and the Emerson Network Power logo are trademarks and service marks of Emerson Electric Co. © 2007 Emerson Electric Co.

Revision Level:	Principal Changes:	Date:
0002M634-00	First release	March 2000
0002M634-01	Update reset section and monitor	August 2000
0002M634-02	Microprocessor to 458MHz, L2Cache to 183MHz, add 256MB SDRAM, ECR00367, ECR00374, updated board configuration register and power requirements	April 2001
0002M634-03	Add errata, ECR00573, monitor version 1.8	March 2005
0002M634-04	Artwork stitch	August 2005
0002M634-05	New format	June 2006
0002M634-06	Emerson update	October 2007

Copyright © 2007 Emerson Electric Co. All rights reserved.

Regulatory Agency Warnings & Notices

The Emerson PmPPC meets the requirements set forth by the Federal Communications Commission (FCC) in Title 47 of the Code of Federal Regulations. The following information is provided as required by this agency.

This device complies with part 15 of the FCC Rules. Operation is subject to the following two conditions: (1) This device may not cause harmful interference, and (2) this device must accept any interference received, including interference that may cause undesired operation.

FCC RULES AND REGULATIONS – PART 15

This equipment has been tested and found to comply with the limits for a Class B digital device, pursuant to part 15 of the FCC Rules. These limits are designed to provide reasonable protection against harmful interference in a residential installation. This equipment generates, uses and can radiate radio frequency energy and, if not installed and used in accordance with the instructions, may cause harmful interference to radio communications. However, there is no guarantee that interference will not occur in a particular installation. If this equipment does cause harmful interference to radio or television reception, which can be determined by turning the equipment off and on, the user is encouraged to try to correct the interference by one or more of the following measures:

- Reorient or relocate the receiving antenna
- Increase the separation between the equipment and receiver
- Connect the equipment into an outlet on a circuit different from that to which the receiver is connected
- Consult the dealer or an experienced radio/TV technician for help

Caution: **Making changes or modifications to the PmPPC hardware without the explicit consent of Emerson Network Power could invalidate the user's authority to operate this equipment.**

EMC COMPLIANCE

The electromagnetic compatibility (EMC) tests used a PmPPC model that includes a front panel assembly from Emerson Network Power.

Caution: **For applications where the PmPPC is provided without a front panel, or where the front panel has been removed, your system chassis/enclosure must provide the required electromagnetic interference (EMI) shielding to maintain EMC compliance.**

EC Declaration of Conformity

According to EN 45014:1998

Manufacturer's Name: Emerson Network Power
Embedded Computing

Manufacturer's Address: 8310 Excelsior Drive
Madison, Wisconsin 53717

Declares that the following product, in accordance with the requirements of 2004/108/EEC, EMC Directive and 1999/5/EC, RTTE Directive and their amending directives,

Product: PrPMC Module

Model Name/Number: PmPPC/10000168-xx

has been designed and manufactured to the following specifications:

EN55022:1998 Information Technology Equipment, Radio disturbance characteristics, Limits and methods of measurement

EN55024:1998 Information Technology Equipment, Immunity characteristics, Limits and methods of measurement

As manufacturer we hereby declare that the product named above has been designed to comply with the relevant sections of the above referenced specifications. This product complies with the essential health and safety requirements of the EMC Directive and RTTE Directive. We have an internal production control system that ensures compliance between the manufactured products and the technical documentation.

Bill Fleury
Compliance Engineer

Issue date: October 29, 2007

CE

(blank page)

Overview

The PmPPC module is a Processor PCI Mezzanine Card (PrPMC). It is based on an IBM PowerPC™ central processor unit and provides additional processing power for the baseboard, which must be compatible with the PrPMC architecture. The PmPPC module supports various memory configurations, programmable user flash memory, a PCI bridge/controller, an Ethernet interface, and two serial ports, as well as a year-2000-compatible real-time clock and EEPROM.

COMPONENTS AND FEATURES

The following is a brief summary of the PmPPC hardware components and features:

CPU: The CPU is an IBM PowerPC microprocessor running internally at up to 458 MHz. The PPC750 has 32-kilobyte Level 1 data and instruction caches, three instructions per clock cycle, and a 32/64-bit data bus mode.

L2 Cache: Two Synchronous Random Access Memory (SRAM) devices running at up to 183 MHz provide a 1-megabyte Level 2 cache. The cache design is two-way, set-associative, with 4096 tags each way. It features a pipeline burst access mode.

SDRAM: The PmPPC may be populated with 32, 64, 128, or 256 megabytes of Error-correcting Code (ECC) 72-bit wide Synchronous Dynamic Random Access Memory (SDRAM). The IBM CPC700 memory controller provides DRAM refresh and other control functions.

Boot EPROM/Flash: The PmPPC has a 32-pin Plastic-Leaded Chip Carrier (PLCC) socket on the front of the board to accommodate an 8-bit, 512-kilobyte EPROM or flash device (factory-installed option). A jumper selects whether the PmPPC boots the monitor from the socketed memory or from flash. The IBM CPC700 memory controller provides the ROM access and control functions.

User Flash: The PmPPC allows for up to 128 megabytes of 64-bit user flash. The IBM CPC700 memory controller supports this memory.

PCI Bridge/Controller: The IBM CPC700 is a versatile, single-chip solution that provides a PowerPC interface, memory controller, Peripheral Component Interconnect (PCI) interface, interrupt controller, arbiter, timers, two Universal Asynchronous Receiver/Transmitter (UART) devices, and two Inter-Integrated Circuit (I^2C) ports. The 32-bit PCI interface can operate at 33 or 66 MHz.

Ethernet: The PmPPC utilizes an IEEE 802.3-compliant CS8900A Crystal LAN™ ISA Ethernet Controller chip from Crystal Semiconductor. This device features an 802.3 Media Access Control (MAC) engine, integrated buffer memory, serial EEPROM interface, and a complete analog front end with 10BASE-T. The Ethernet signals are accessible via the front panel RJ45 connector or the PMC connector at P14 (optional).

Serial and I²C Ports: The IBM CPC700 provides two, full-duplex, 16550-compatible serial ports with programmable baud rates. Both serial ports are accessible at the P2 front panel connector (9-pin, Nano D) and the P14 PMC connector. In addition to the serial ports, the CPC700 provides two I²C ports. One of these ports supports a serial EEPROM; the other is accessible to target applications via PMC connector P14.

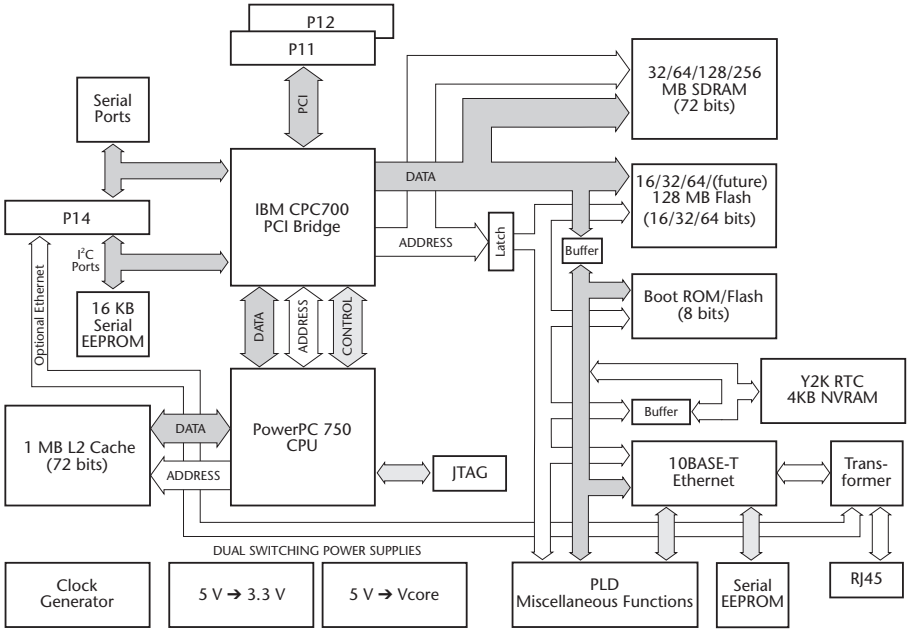
PLD: The PmPPC uses a programmable logic device (PLD) to implement various memory-mapped registers and to control access to the Ethernet chip, the year-2000-compatible real-time clock, and non-volatile RAM.

RTC: The real-time clock is a Dallas Semiconductor DS17485 chip that also provides four kilobytes of non-volatile RAM. This year-2000-compatible clock also has power control circuitry. The DS17485 utilizes an external 32.768 KHz crystal and is powered from the +3.3 V rail during normal operation. A supercap on the PmPPC board provides at least 12 hours of backup operation.

FUNCTIONAL OVERVIEW

The following block diagram provides a functional overview for the PmPPC:

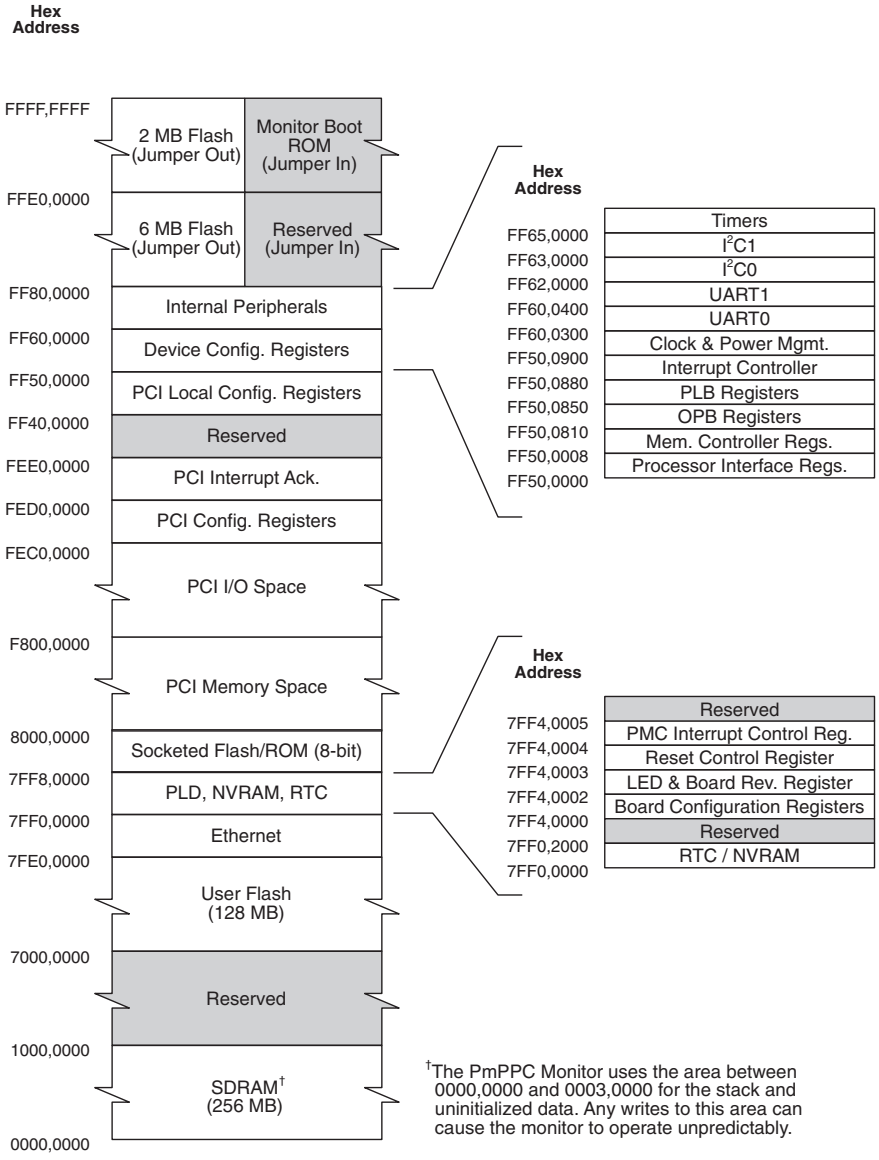
Figure 1-1: *General System Block Diagram*



PHYSICAL MEMORY MAP

Information on particular portions of the memory map can be found in later sections of this manual, see Table 1-1.

Figure 1-2: *General System Block Diagram*



†The PmPPC Monitor uses the area between 0000,0000 and 0003,0000 for the stack and uninitialized data. Any writes to this area can cause the monitor to operate unpredictably.

The following table summarizes the physical addresses for the PmPPC and provides a reference to more detailed information.

Table 1-1: *Address Summary*

Physical Address (hex):	Access Mode:	Description:	See Page:
FFE0,0000	R	2 MB flash memory (jumper JP1 out); Boot ROM (jumper JP1 in)	4-2, 4-2
FF80,0000	R-R/W	6 MB flash memory (jumper JP1 out)	4-2
FF65,0000	R-R/W	Timers	8-3
FF63,0000	R/W	I^2C1	8-2
FF62,0000	R/W	I^2C0	8-2
FF60,0400	R-R/W	UART1	7-1
FF60,0300	R-R/W	UART0	7-1
FF50,0900	R-R/W	Clock and Power Management	8-4
FF50,0880	R-W-R/W	Interrupt Controller	8-2
FF50,0850	R-W-R/W	PLB registers	8-1
FF50,0810	R-W	OPB registers	8-1
FF50,0008	R/W	Memory Controller registers	4-1
FF50,0000	R-W-R/W	Processor Interface registers	3-3
FF40,0000	R-R/W	PCI Local Configuration registers	5-2
FEE0,0000	–	reserved	—
FED0,0000	R	PCI Interrupt Acknowledge	5-1
FEC0,0000	R/W	PCI Configuration registers	5-2
FC00,0000	–	reserved	—
F880,0000	R/W	PCI I/O space	5-1
F801,0000	–	reserved	—
F800,0000	R/W	PCI I/O space	5-1
8000,0000	R/W	PCI memory space	5-1
7FF8,0000	R	Socketed flash/ROM (8-bit)	4-2
7FF5,0000	–	reserved	—
7FF4,0004	R/W	PMC Interrupt Control register	5-5
7FF4,0003	R/W	Reset Control register	3-1
7FF4,0002	R/W	LEDs	2-5
7FF4,0000	R	Board Configuration registers	2-7
7FF0,2000	–	reserved	—
7FF0,0000	R/W	NVRAM, Real-time clock	9-1
7FE0,0000	R-W-R/W	Ethernet	6-1
7000,0000	R-R/W	User flash memory (up to 128 MB)	4-2
1000,0000	–	reserved	—
0000,0000	R/W	SDRAM (up to 256 MB)	4-4

ADDITIONAL INFORMATION

This section lists the PmPPC hardware's regulatory certifications and briefly discusses the terminology and notation conventions used in this manual. It also lists general technical references.

Mean time between failures (MTBF) has been calculated at 890,701 hours using the Bellcore Issue 6 Reliability Method I case 3.

Product Certification

The PmPPC hardware has been tested to comply with various safety, immunity, and emissions requirements as specified by the Federal Communications Commission (FCC), Underwriters Laboratories (UL), and others. The following table summarizes this compliance:

Table 1-2: *Regulatory Agency Compliance*

Type:	Specification:
Safety	UL60950-1, CSA C22.2 No. 60950-1-03, 1st Edition — Safety of Information Technology Equipment, including Electrical Business Equipment (BI-National)
EMC	FCC Part 15, ClassB — Title 47, Code of Federal Regulations, Radio Frequency Devices ICES 003 — Radiated and Conducted Emissions, Canada EN55022 — Information Technology Equipment, Radio Disturbance Characteristics, Limits and Methods of Measurement EN55024 — Information Technology Equipment, Immunity Characteristics, Limits and Methods of Measurement
Telecom	CE Mark — Verifies UL 1950 safety, ETSI CTR 12 and CTR 13, CENELEC EN 55022 and EN 50082-1 and ITU-T K.20 and K.21

Vibration testing was successfully completed using the MIL-STD 202F, Method 214.

Emerson maintains test reports that provide specific information regarding the methods and equipment used in compliance testing. Unshielded external I/O cables, loose screws, or a poorly grounded chassis may adversely affect the PmPPC hardware's ability to comply with any of the stated specifications.

The UL web site at ul.com has a list of Emerson's UL certifications. To find the list, search in the online certifications directory using Emerson's UL file number, E190079. There is a list for products distributed in the United States, as well as a list for products shipped to Canada. To find the PmPPC, search in the list for 10000168-*xx*, where *xx* changes with each revision of the printed circuit board.

Terminology and Notation

Active low signals: An active low signal is indicated with an asterisk * after the signal name.

Byte, word: Throughout this manual *byte* refers to 8 bits, *word* refers to 16 bits, and *long word* refers to 32 bits, *double long word* refers to 64 bits.

PLD: This manual uses the acronym, *PLD*, as a generic term for programmable logic device (also known as FPGA, CPLD, EPLD, etc.).

Radix 2 and 16: Hexadecimal numbers end with a subscript *16*. Binary numbers are shown with a subscript *2*.

Technical References

Further information on basic operation and programming of the PmPPC components can be found in the following documents:

Table 1-3: *Technical References*

Device / Interface:	Document: [1]
CPU	*PowerPC™ 750 RISC Microprocessor User's Manual* (IBM number GK21-0263-00) *IBM PowerPC™ 750GL RISC Microprocessor Datasheet* (Revision Level DD1.x, Version: 1.2, March 13, 2006) *PowerPC™ Microprocessor Family: The Programming Environments* (IBM number G522-0290-00) http://www.chips.ibm.com
Memory controller / PCI bridge	Memory Controller and PCI Bridge: *CPC700 User's Manual, Version 1, Preliminary* (IBM Microelectronics Division, 1999) http://www.chips.ibm.com
Ethernet	*CS8900A Crystal LAN™ ISA Ethernet Controller Data Sheet, DS271PP3* (Cirrus Logic, March 1999) http://www.crystal.com
Flash	*3 Volt Intel®StrataFlash™ Memory Product Preview Datasheet* (Intel, August 1999) Intel® Embedded Flash™ Memory (J3 v. D) Datasheet (Intel, September 2005) http://www.intel.com
Real-time Clock	DS17485/DS17487 3 Volt / 5 Volt Real Time Clock Data Sheet (Dallas Semiconductor, April 1999) http://www.dalsemi.com
PCI	*PCI Local Bus Specification* (PCI Special Interest Group, Revision 2.2, 1998) http://www.pcisig.com

Device / Interface:	Document: [1] (continued)
PMC	Draft Standard for a Common Mezzanine Card Family: CMC P1386 / Draft 2.4 January 12, 2001 (IEEE: New York, NY) *Draft Standard for Physical and Environmental Layers for PCI Mezzanine Cards: PMC P1386.1 / Draft 2.4 January 12, 2001* (IEEE: New York, NY) http://www.ieee.org
PrPMC	*Processor PMC Standard for Processor PCI Mezzanine Cards: Draft 0.41 / September 8, 2000* (VITA: Scottsdale, AZ) http://www.vita.com

1. Frequently, the most current information regarding addenda/errata for specific documents may be found on the corresponding web site.

Setup

This chapter describes the physical layout of the boards, the setup process, and how to check for proper operation once the board has been installed. This chapter also includes troubleshooting, service, and warranty information.

ELECTROSTATIC DISCHARGE

Before you begin the setup process, remember that electrostatic discharge (ESD) can easily damage the components on the PmPPC hardware. Electronic devices, especially those with programmable parts, are susceptible to ESD, which can result in operational failure. Unless you ground yourself properly, static charges can accumulate in your body and cause ESD damage when you touch the board.

Caution: **Use proper static protection and handle PmPPC boards only when absolutely necessary. Always wear a wriststrap to ground your body before touching a board. Keep your body grounded while handling the board. Hold the board by its edges—do not touch any components or circuits. When the board is not in an enclosure, store it in a static-shielding bag.**

To ground yourself, wear a grounding wriststrap. Simply placing the board on top of a static-shielding bag does not provide any protection—place it on a grounded dissipative mat. Do not place the board on metal or other conductive surfaces.

PMPPC CIRCUIT BOARD

The PmPPC circuit board is PrPMC module assembly. It uses a 12-layer printed circuit board with the following dimensions:

Table 2-1: *Circuit Board Dimensions*

Width:	Depth:	Height, top side:	Height, Bottom side:
2.913 in. (74 mm)	5.866 in. (149 mm)	0.323 in. (8.2 mm) 0.524 in. (13.5 mm) I/O area	0.007 in. (1.9 mm)

The following figures show the component maps and jumper locations for the PmPPC circuit board.

Figure 2-1: *Component Map, Top (Rev. 04)*

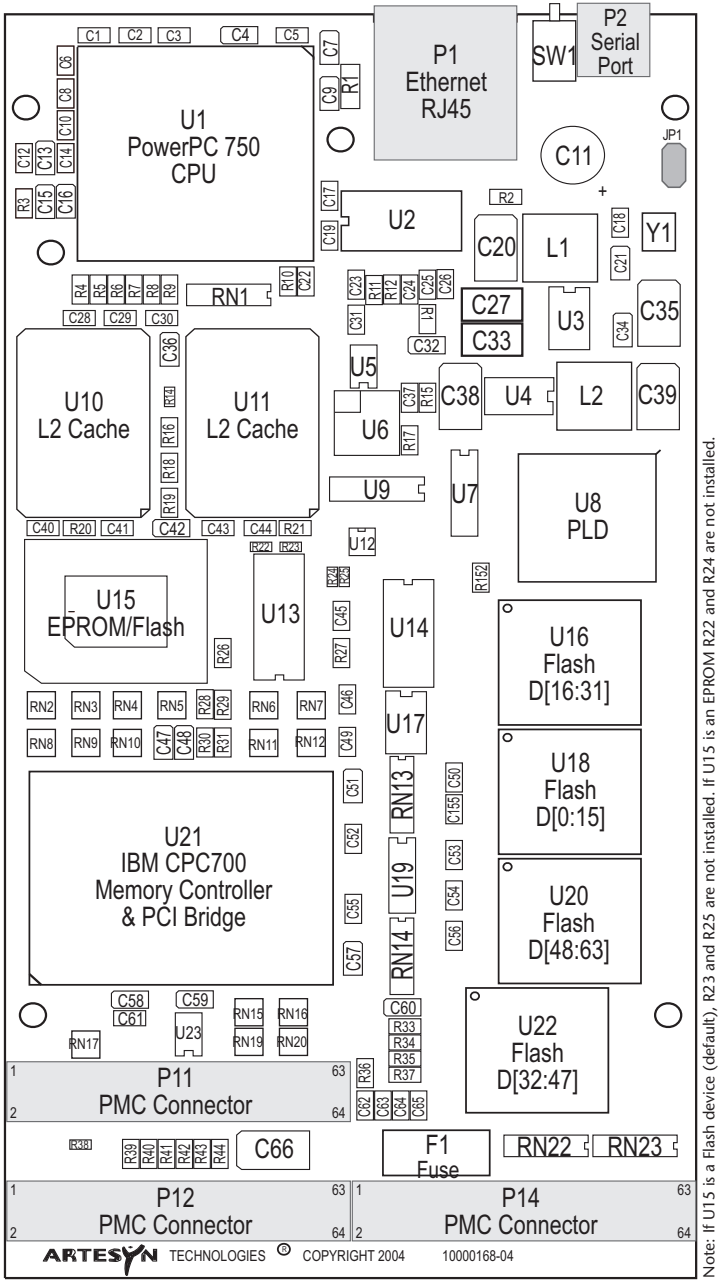


Figure 2-2: *Component Map, Bottom (Rev. 04)*

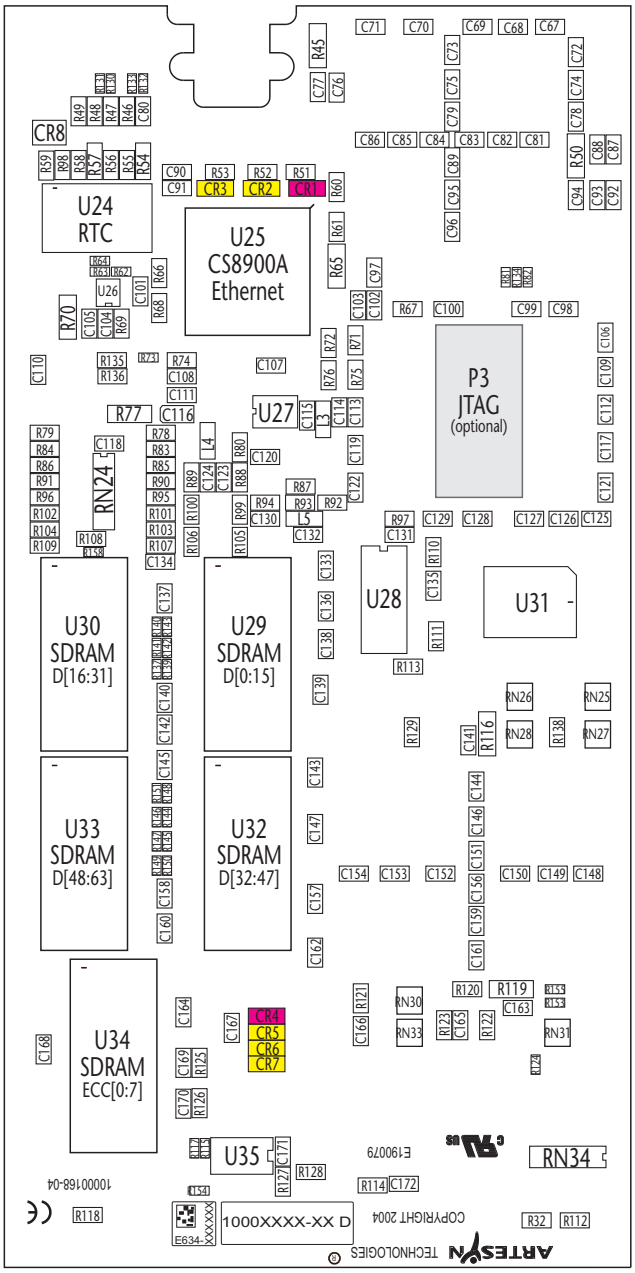
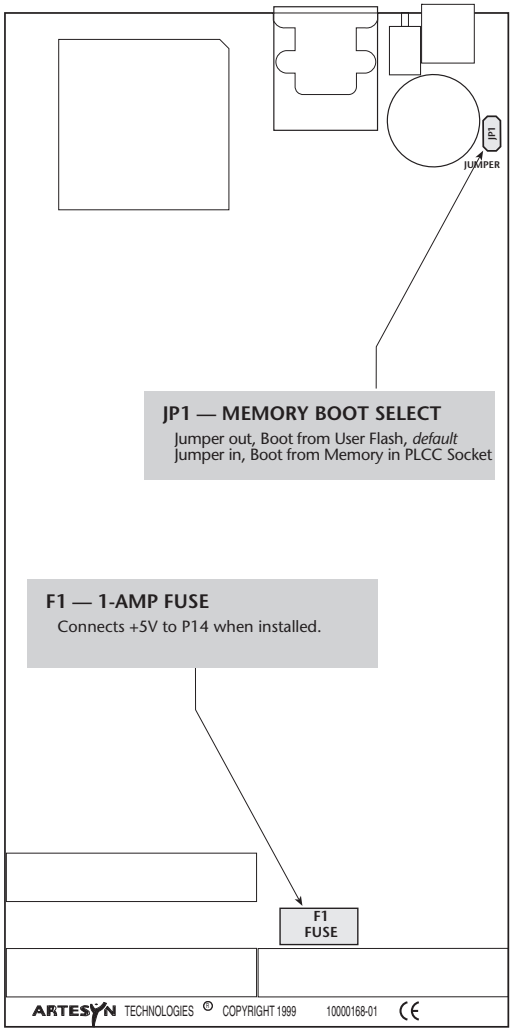


Figure 2-3: *Jumper and Fuse Locations, Top*



Connectors

The PmPPC circuit board has various connectors (see the figures beginning on page 2-2), summarized as follows:

P1: This RJ45 connector is the front panel Ethernet port. Refer to Table 6-4 for the pin assignments.

P2: This miniature 9-pin (Nano D) connector is for the front panel serial port. Refer to Table 7-3 for the pin assignments. These signals also are available at the P14 connector.

Caution: **It is strongly recommended that a weight bearing attachment point to the serial cable, within three inches of the PmPPC front panel, be provided by the system. Such a mechanism is needed to provide an adequate strain relief for the serial I/O cable at the P2 (Nano DB9) connector.**

P3: This is an optional 16-pin JTAG/COP header on the back of the board that allows for boundary scan testing of the PowerPC CPU and the PmPPC. See Table 3-5 for pin assignments. Additional JTAG signals at the P12 connector allow for access to the CPC700 memory controller/PCI bridge.

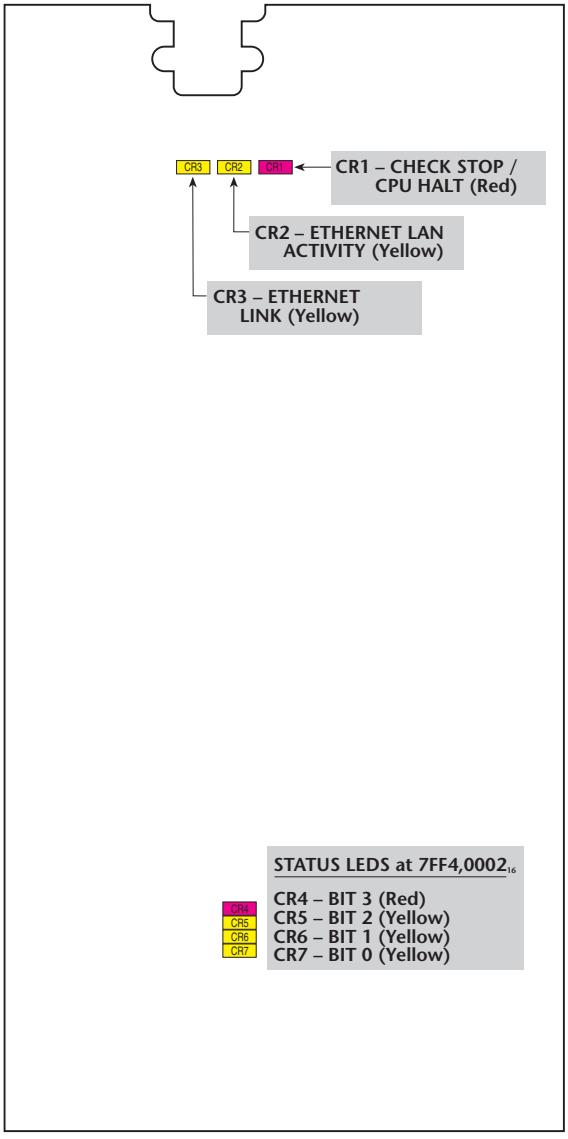
P11, P12: These are the 64-pin connectors that provide the standard 32-bit PCI interface between the PmPPC and the PMC host. See Table 5-4 and Table 5-5 for pin assignements.

P14: This 64-pin connector conforms to the PCI specification as user-defined. An optional configuration allows for the Ethernet signals to be available at P14. See Table 5-6 for pin assignments.

LEDs

The PmPPC has seven light-emitting diodes (LEDs) on the back side of the board (see Fig. 2-4). These include a red CPU halt indicator, two yellow Ethernet monitoring LEDs, and four software-driven status LEDs accessible via the register at location $7FF4,0002_{16}$ (bits 0–3). For the status LEDs: 1=on and 0=off, with one red LED corresponding to the most significant bit and three yellow LEDs corresponding to the least significant bits.

Figure 2-4: *LED Locations, Bottom*



Board Configuration Registers

A programmable logic device (PLD) on the PmPPC contains two, byte-wide, read-only Board Configuration registers starting at location $7FF4,0000_{16}$. These registers allow the monitor software to easily determine specific hardware configurations. The BCR0 allows the monitor software to easily determine specific hardware configurations.

Register 2-1: *Board Configuration 0 (BCR0), 0x7ff4,0001*

7	6	5	4	3	2	1	0
BDEV	ECC	MEM_SIZE		L2_CACHE		BUS_FREQ	

BDEV: Monitor boot device indicato. (Jumper JP1 selects the boot device)
0 ROM
1 Flash

ECC: Error-correcting Code option
0 ECC
1 No ECC option

MEM_SIZE: SDRAM size configuration
00 32 MB
01 64 MB
10 128 MB
11 256 MB

L2_CACHE: L2 cache configuration
00 150 MHz (72-bit)
01 166 MHz (72-bit)
10 200 MHz (72-bit)
11 No cache

BUS_FREQ: Local bus clock configuration
00 1660 MHz
01 83 MHz
10 100 MHz
11 reserved

Register 2-2: *Board Configuration 1 (BCR1), 0x7ff4,0000*

7	6	5	4	3	2	1	0
MON	reserved	ETH	RTC	F_SIZE		F_WIDTH	

MON: Monarch functionality (Read from PCI connector P12, pin 64)
0 Monarch
1 Non-Monarch

ETH: Ethernet interfacen
0 Not present
1 Present

RTC: Real-time Clock device configuration
0 DS17485
1 No RTC

F_SIZE: Flash memory, number of bytes per flash device on the board
00 32 MB
01 16 MB
10 8 MB
11 4 MB

F_WIDTH: Flash memory width
00 64 bits
01 32 bits
10 16 bits
11 No flash

PMPPC SETUP

You need the following items to set up and check the operation of the Emerson PmPPC:

- ❒ Compatible PrPMC baseboard, such as the BajaPPC
- ❒ Chassis and power supply
- ❒ Optional serial cable, Emerson item number C0007658-00
- ❒ Computer terminal

Save the antistatic bag and box for future shipping or storage.

Caution: **Do not install the board in a rack or remove the board from a rack while power is applied, at risk of damage to the board.**

Power Requirements

The Emerson PmPPC circuit board is capable of operating in either a 3.3 V or 5 V PCI signaling environment.

The exact power requirements for the PmPPC circuit board depend upon the specific configuration of the board, including the CPU frequency and amount of memory installed on the board. Contact Emerson Technical Support at 1-800-327-1251 if you have specific questions regarding the board's power requirements.

Table 2-2: *Power Requirements*

Voltage:	Range:	Current[1] (amps):	Current[2] (amps):	Watts[2]:	Configuration:
+5	+/-5%	1.46	1.74[3]	8.7	333 MHz, 64 MB, 16FL
+5	+/-5%	1.66	2.07[3]	10.35	458 MHz, 64 MB, 16FL
+5	+/-5%	1.67	2.21[3]	11.05	458 MHz, 128 MB, 16FL
+5	+/-5%	1.67	2.22[3]	11.1	458 MHz, 256 MB, 64FL

1. Measured at monitor prompt.
2. Measured while running on-card memory test.
3. Actual current draw depends on software application.

Environmental Considerations

As with any printed circuit board, be sure that air flow to the board is adequate. Chassis constraints and other factors greatly affect the air flow rate. The environmental requirements are as follows:

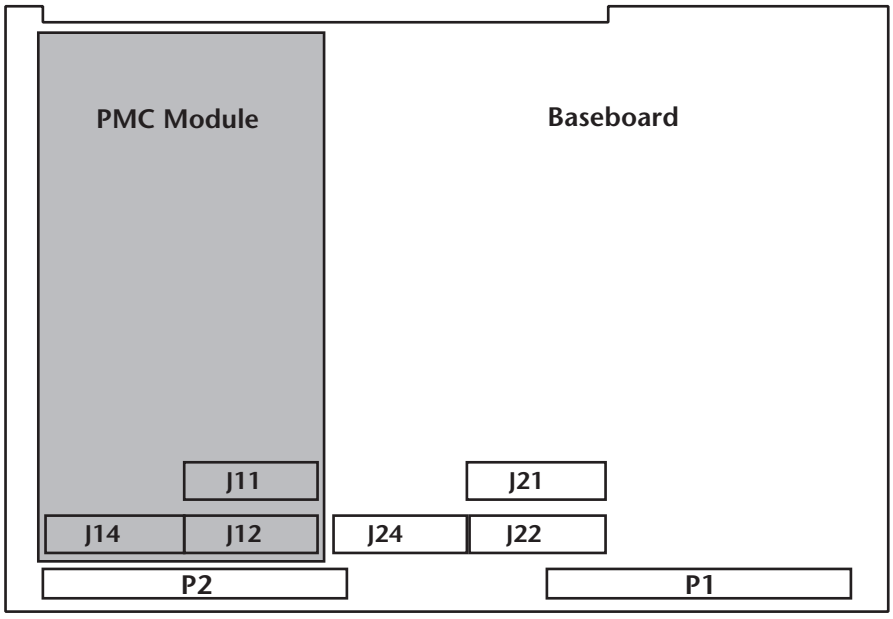
Table 2-3: *Environmental Requirements*

Environment:	Range:	Relative Humidity:
Operating Temperature	0° to +55° Centigrade, ambient (at board)	Not to exceed 95% (non-condensing)
Storage Temperature	–40° to 705° Centigrade	Not to exceed 95% (non-condensing)
Air Flow	100 linear feet/minute	

Installation

Most PMC-compatible baseboards have two sets of three connectors (J11-J14 and J21-J23), as defined by the PMC specification. Fig. 2-5 shows the location of these connectors on the baseboard and the location of the PmPPC module.

Figure 2-5: *Module Location on Baseboard*



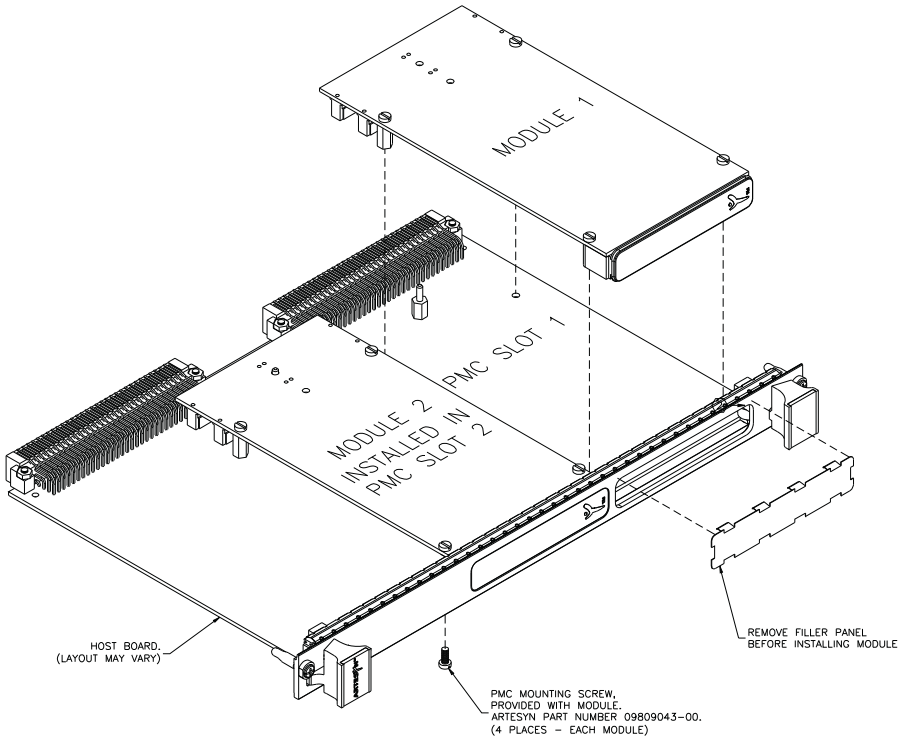
Use the following procedure to attach the PmPPC module to your baseboard:

1. Remove the screws from the standoffs on the PMC module.
2. Hold the module at an angle and gently slide the faceplate into the opening on the baseboard.
3. Align the P11 and P12 connectors and gently press the module into place until firmly mated.

Caution: **To avoid damaging the module and/or baseboard, do not force the module onto the baseboard.**

4. Using four M2.5x6 mm flathead screws (item number 09809043-00), secure the PmPPC module from the bottom of the baseboard. First, insert and tighten the screws closest to the P11, P12, and P14 connectors. Next, insert and tighten the screws nearest to the front panel.

Figure 2-6: *Installing the Module on the Baseboard*



TROUBLESHOOTING

In case of difficulty, use this checklist:

- ❒ Be sure the PmPPC circuit board is seated firmly in the carrier.
- ❒ Verify the boot jumper setting (see Fig. 2-3)
- ❒ Be sure the system is not overheating.
- ❒ Check the cables and connectors to be certain they are secure.
- ❒ Check your power supply for proper DC voltages. If possible, use an oscilloscope to look for excessive power supply ripple or noise (over 50 mV_{pp} below 10 MHz).
- ❒ **(Emerson PMC hosts only.)** In order for the monitor to map any PCI devices found at power-up time, the NVRAM field *DoModConfig* should be set to true. After the power-up diagnostics have completed, the monitor will display a list of all PMC devices it found. The list includes the vendor and device IDs and the base addresses of each device found.

Technical Support

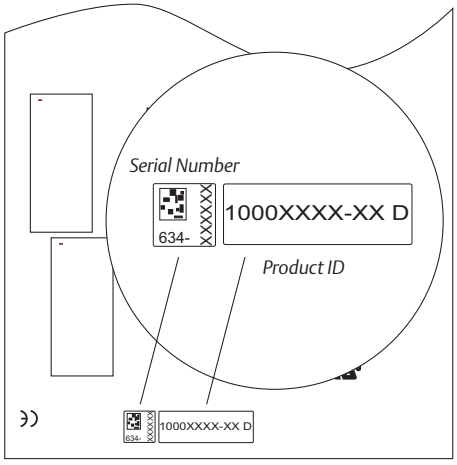
If you need help resolving a problem with your PmPPC, visit http://www.emersonembeddedcomputing.com/contact/postsalessupport.html on the Internet or send e-mail to support@artesyncp.com. If you do not have internet access, call Emerson for further assistance:

(800) 327-1251 or (608) 826-8006 (US)
44-131-475-7070 (UK)

Have the following information available when contacting support:

- PmPPC serial number and product identification (see Fig. 2-7)
- monitor version (see Fig. 9-1 start-up display)
- baseboard model number and monitor version (if applicable)
- version and part number of the operating system (if applicable). This information is labeled on the master media supplied by Emerson or another vendor
- whether your board has been customized for options such as a higher processor speed or additional memory
- license agreements (if applicable)

Figure 2-7: *Serial Number and Product ID on Bottom Side*



Product Repair

If you plan to return the board to Emerson Network Power for service, visit http://www.emersonembeddedcomputing.com/contact/productrepair.html on the internet or send e-mail to serviceinfo@artesyncp.com to obtain a Return Merchandise Authorization (RMA) number. We will ask you to list which items you are returning and the board serial number, plus your purchase order number and billing information if your PmPPC hardware is out of warranty. Contact our Test and Repair Services Department for any warranty questions. If you return the board, be sure to enclose it in an antistatic bag, such as the one in which it was originally shipped. Send it prepaid to:

Emerson Network Power, Embedded Computing
Test and Repair Services Department
8310 Excelsior Drive
Madison, WI 53717

RMA #_____

Put the RMA number on the outside of the package so we can handle your problem efficiently. Our service department cannot accept material received without an RMA number.

CPU

This chapter is an overview of the processor logic on the PmPPC. It includes information on the CPU, exception handling, and cache memory. The PmPPC utilizes the IBM PPC750 PowerPC microprocessor, running at an internal clock speed of up to 458 MHz.

The following table outlines some of the key features for the PPC750 CPU:

Table 3-1: *CPU Features*

Category:	PPC750 Key Features:
Instruction Set	32-bit
CPU Speed (internal)	up to 458 MHz
Data Bus	32/64-bit modes
Address Bus	32-bit
Instructions per Clock	3 (2 + Branch)
Cache(s)	32 K Instruction, 32 K Data
Execution Units	2 Integer, Float, Branch, Load/Store, System
Voltages	internal, 2.05 V; input/output, 3.3 V

PROCESSOR RESET

Circuitry on the PmPPC module resets the processor and the board. It activates the RESET_OUT* signal on pin 60 of the P12 connector if the module voltages fall out of tolerance or if the optional on-board reset switch is activated.

In addition, the Reset Control register allows the host to reset the PmPPC processor via the PCI interface. To reset the CPU, write a series of two 8-bit values to the register located at $7FF4,0003_{16}$.

Register 3-1: *Reset Control*

7	6	5	4	3	2	1	0
RST7	RST6	RST5	RST4	RST3	RST2	RST1	RST0

First, enable a reset by writing the hex value AC_{16} to the Reset Control register. Once it is enabled, you can reset the CPU by writing the hex value $1D_{16}$ to the register. The reset stays asserted until you write a new value. Since it is a read/write register, you can read the value in the Reset Control register at any time. After a reset, all the bits in this register are set to zero.

PROCESSOR INITIALIZATION

Initially, the PmPPC powers up with specific values stored in the CPU registers. The initial power-up state of the Hardware Implementation Dependent register (HID0) and the Machine State register (MSR) are given in Table 3-2.

Table 3-2: *CPU Internal Register Initialization*

Register:	Default After Initialization (hex):	Notes:
HID0	8020,802C	Hardware Implementation Dependent register 0
MSR	0000,B032	Machine State register

Reference the *PowerPC™ 750 RISC Microprocessor User's Manual* for detailed register information.

INTERRUPT HANDLING

The CPC700 controller handles all PmPPC interrupts other than the PCI interrupts (see "PCI Interrupts" on page 5-5). It drives the 750_INT* interrupt input on the CPU. The interrupt handler sends a command for the interrupting device to acknowledge the interrupt and deassert 750_INT*. (Note that software should clear the CPC700 ERRDET1 register after a memory select error.) The following table shows the PmPPC interrupt mapping:

Table 3-3: *Interrupt Mapping*

Interrupt Signal:	Source:	IRQ Pin on CPC700:	CPC700 UIC Bits:
BPMC_INTA*	PMC	IRQ0	20
BPMC_INTB*	PMC	IRQ1	21
BPMC_INTC*	PMC	IRQ2	22
BPMC_INTD*	PMC	IRQ3	23
ETH_INT (active high)	Ethernet	IRQ4	24
NV_INT*	RTC/NVRAM	IRQ5	25
IRQ6*–IRQ11*	reserved	IRQ6-IRQ11	26-31

As a Monarch module (see "Monarch Functionality" on page 5-5), the PmPPC is typically an interrupt handler, although it could be configured to generate interrupts. As a non-Monarch module, it is typically an interrupt generator. Since the resident monitor software does not enable PMC interrupts, the application software must perform this task. The software can read bit 7 of Board Configuration register 1 (see "PCI Interrupts" on page 5-5) at location $7FF4,0000_{16}$ to determine how the module is configured (0 is Monarch, 1 is non-Monarch).

CACHE MEMORY

The PPC750 processor has separate, on-chip, 32-kilobyte, Level 1, instruction and data caches with eight-way, set-associative translation lookaside buffers (TLBs). The CPU supports the modified/exclusive/invalid (MEI) cache coherency protocol. Each cache has 128 entries and supports demand-paged virtual memory address translation and variable-sized block translation. The PPC750 also employs pseudo-least-recently used (PLRU) replacement algorithms for enhanced performance.

In addition to the on-chip caches, the PPC750 CPU utilizes a 1-megabyte, integrated level-two (L2) cache provided by two synchronous random access memory (SRAM) chips. For the PmPPC, the cache integrates data, tag, host interface, and LRU memory with a cache controller. At speeds up to 183 MHz, it performs with zero wait states (3-1-1-1 pipelined burst mode). The cache design is two-way, set-associative and employs LRU logic.

CPC700 PROCESSOR INTERFACE

The IBM CPC700 memory controller/PCI bridge device interfaces directly with the PPC750 processor bus. Table 3-4 lists the control registers associated with this interface. Refer to the *CPC700 Functional Specification* for complete details on the processor interface core registers.

Table 3-4: *Processor Interface Core Registers*

CPC700 Hex Offset:	Register Name:	Access Mode:	Function:
00	PRIFOPT1	R/W	Processor Interface Options 1
04	ERRDET1	R/W	Error Detection 1
08	ERREN1	R/W	Error Detection Enable 1
0C	CPUERAD	R	Processor Error Address
10	CPUERAT	R	Processor Error Attributes
14	–	–	reserved
18	PLBMIFOPT	R/W	Processor-PLB Master Interface Options
20	PLBMTLSA1	R/W	Processor-PLB Bank 1 Translation Start Address
24	PLBMTLEA1	R/W	Processor-PLB Bank 1 Translation End Address
28	PLBMTLSA2	R/W	Processor-PLB Bank 2 Translation Start Address
2C	PLBMTLEA2	R/W	Processor-PLB Bank 2 Translation End Address
30	PLBMTLSA3	R/W	Processor-PLB Bank 3 Translation Start Address
34	PLBMTLEA3	R/W	Processor-PLB Bank 3 Translation End Address
38	PLBSNSSA0	R/W	No Snoop Starting Address
3C	PLBSNSEA0	R/W	No Snoop Ending Address
40	BESR	R/W	PLB Bus Error Syndrome register
44	BESRSET	W	PLB Bus Error Syndrome register Set (test use only)

CPC700 Hex Offset:	Register Name:	Access Mode:	Function: (continued)
48	–	–	reserved
4C	BEAR	R/W	PLB Bus Master Error Address register
50-54	–	–	reserved
80	PLBSWRINT	R/W	Write Interrupt Region Base Address

The Processor Interface Configuration Address and Data registers at $FF50,0000_{16}$ and $FF50,0004_{16}$ allow access to the CPC700 processor interface registers. To initiate an access, write the hex offset value for the desired register to address $FF50,0000_{16}$. The data for that register then may be read from or written to the data register at $FF50,0004_{16}$. Note that all configuration accesses from the processor must be 4-byte aligned.

JTAG/COP INTERFACE

The JTAG/COP interface provides for boundary-scan testing of the CPU and the PmPPC. This interface is compliant with the IEEE 1149.1 interface standard. The JTAG interface signals route to the optional P3 header on the back of the board (see component map in Fig. 2-2).

Table 3-5: *JTAG/COP Interface Pin Assignments (P3)*

Pin:	Signal:	Pin:	Signal:
1	CPU_TDO	2	no connect
3	CPU_TDI	4	DBG_TRST*
5	no connect	6	+3.3 V
7	CPU_TCK	8	no connect
9	CPU_TMS	10	no connect
11	DBG_SRESET*	12	ground
13	DBG_HRESET*	14	used as a keying pin
15	CKSTP_OUT*	16	ground

CPU_TDO: Test Data Output signal acts as the output port for scan instructions and data.

CPU_TDI: Test Data Input signal acts at the input port for scan instructions and data.

DBG_TRST*: Test Reset input signal resets the test access port.

CPU_TMS: Test Mode Select input signal is the test access port (TAP) controller mode signal.

CPU_TCK: Test Clock Input scan data is latched at the rising edge of this signal.

DBG_SRESET*: Soft Reset input signal may initiate a warm reset. (This signal is not defined in the IEEE 1149.1 standard.)

DBG_HRESET*: Hard Reset input signal is used at power-up to reset the processor. (This signal is not defined in the IEEE 1149.1 standard.)

CKSTP_OUT*: Checkstop Output, when asserted, indicates that the CPU has detected a checkstop condition and has ceased operation. On the PmPPC, this signal connects to the red HALT LED.

On-card Memory Configuration

The PmPPC has a 32-pin, plastic-leaded chip carrier (PLCC) socket to support up to 512 kilobytes of flash memory. The PmPPC also provides up to 128 megabytes of 16, 32, or 64-bit user flash memory. The board supports on-card synchronous SDRAM configurations of up to 256 megabytes.

CPC700 MEMORY INTERFACE

The IBM CPC700 acts as the memory controller for the PmPPC. Table 4-1 lists the control registers associated with the memory interface. Chapter 5 describes the PCI bridge. Refer to the *CPC700 Functional Specification* for complete details on the memory interface registers.

Table 4-1: *Processor Interface Core Registers*

CPC700 Hex Offset:	Register Name:	Access Mode:	Description:
20	MCOPT1	R/W	Memory Controller Options 1
24	MBEN	R/W	Memory Bank Enable
28	MEMTYPE	R/W	Installed Memory Type
2C	RWD	R/W	Bank Active Watchdog Timer
30	RTW	R/W	Refresh Timer register
34	DAM	R/W	DRAM Addressing Mode
38	MBOSA	R/W	Memory Bank 0 Starting Address
3C	MB1SA	R/W	Memory Bank 1 Starting Address
40	MB2SA	R/W	Memory Bank 2 Starting Address
44	MB3SA	R/W	Memory Bank 3 Starting Address
48	MB4SA	R/W	Memory Bank 4 Starting Address
58	MB0EA	R/W	Memory Bank 0 Ending Address
5C	MB1EA	R/W	Memory Bank 1 Ending Address
60	MB2EA	R/W	Memory Bank 2 Ending Address
64	MB3EA	R/W	Memory Bank 3 Ending Address
68	MB4EA	R/W	Memory Bank 4 Ending Address
80	SDTR1	R/W	SDRAM Timing register 1
88	RBW	R/W	ROM Bank Width
90	FWEN	R/W	Flash Write Enable
94	ECCCF	R/W	ECC Configuration
98	ECCERR	R/W	ECC Error
E0	RPB0P	R/W	ROM / Peripheral Bank 0 Parameters
E4	RPB1P	R/W	ROM / Peripheral Bank 1 Parameters
E8	RPB2P	R/W	ROM / Peripheral Bank 2 Parameters
EC	RPB3P	R/W	ROM / Peripheral Bank 3 Parameters

CPC700 Hex Offset:	Register Name:	Access Mode:	Description: (continued)
F0	RPB4P	R/W	ROM / Peripheral Bank 4 Parameters

The Memory Controller Configuration Address and Data registers at $FF50,0008_{16}$ and $FF50,000C_{16}$ allow access to the CPC700 memory interface registers. To initiate an access, write the hex offset value for the desired register to address $FF50,0008_{16}$. The data for that register then may be read from or written to the data register at $FF50,000C_{16}$. See Monitor sections "CPC700 Indirect Reads" on page 9-39 and "CPC700 Indirect Writes" on page 9-39. Note that all configuration accesses from the processor must be 4-byte aligned.

BOOT MEMORY CONFIGURATION

The PmPPC has a 32-pin PLCC socket on the front of the board for a 512-kilobyte flash memory chip. When booting from the socketed memory, it occupies the physical address space beginning at $FFE0,0000_{16}$. Otherwise, it starts at $7FF8,0000_{16}$. The PLCC sockets only support 3.3-volt devices.

Caution: **When removing socketed PLCC devices, always use an extraction tool designed specifically for that task. Otherwise, you risk damaging the PLCC device.**

A jumper on the PmPPC circuit board configures the memory as shown in Table 4-2. The on-board monitor is standard in the last two megabytes of user flash memory space.

Table 4-2: *Memory Configuration Jumpers*

Jumper:	Function:	Options:	Default Configuration:
JP1	Selects monitor boot device	JP1 out, user flash JP1 in, PLCC socket	JP1 out, user flash

The CPC700 controls the access time for ROM. The default power-up timing allows boards of any speed to work with ROMs that have access times faster than 150 nanoseconds. We strongly suggest that you use the default timing because of the inherent risks of optimizing timing for a specific configuration and because the ROM is cached.

USER FLASH

The PmPPC circuit board accommodates up to four Intel StrataFlash™ devices (16 bits wide), allowing for as much as 128 megabytes of 64-bit wide user flash at location $7000,0000_{16}$. Bits 3 and 2 of Board Configuration register 1 (BCR1) at $7FF4,000_{16}$ (see page 2-7) configure the flash device density. Bits 1 and 0 of this same register configure the data path width. The following table shows various configuration options:

Table 4-3: *Flash Memory Configurations*

Device Density:	Data Path Width:	Number of Devices:	Total Memory (megabytes):	BCR1 Bits [3:0]:
256 Mbit[1]	64 bits	4	128	0000
256 Mbit[1]	32 bits	2	64	0001
256 Mbit[1]	16 bits	1	32	0010
128 Mbit	64 bits	4	64	0100
128 Mbit	32 bits	2	32	0101
128 Mbit	16 bits	1	16	0110
64 Mbit	16 bits	1	8	1010

1. Future offers that are not yet available at the writing of this manual

The flash devices interface to the most significant data bits of the PowerPC data bus. For example, if the data path is 64 bits wide, the PowerPC data bus is declared as D[0:63], where D0 is the most significant bit and D63 is the least significant bit. For a 32 bit data path, the flash devices interface to D[0:31]. For a 16 bit path, the data bus is D[0:15]. (Refer to Fig. 2-1 for the individual device mappings.)

If booting from user flash, the CPC700 controller initially maps two megabytes of flash memory (beginning at $FFE0,0000_{16}$) at the top of the address space. Another six megabytes (beginning at $FF80,0000_{16}$) is reserved to yield a combined total of eight megabytes for contiguous user flash at the top of the address space. Only the top eight megabytes of the entire soldered flash array ($7000,0000_{16}$ – $77FF,FFFF_{16}$) appear at the top of the boot address space ($FF80,0000_{16}$ – $FFFF,FFFF_{16}$). When an 8-bit flash device is installed in the PLCC socket, it always appears at $7FF8,0000_{16}$ (and is mirrored at $FFE0,0000_{16}$ when the socket is the boot device).

Since the 64-bit flash memory is soldered, an 8-bit ROM could be used to bootstrap the processor and execute a routine that programs the soldered flash from a serial port, Ethernet, or through the PCI interface.

The PmPPC accommodates several different configurations of user flash as noted in Table 4-3. With the accommodation of using 16, 32, and 64-bit data widths, it is important to access the boards's user flash correctly with the correct data width. For example, to put the device into read array mode, write:

```
64-bit write: command = 00ff 00ff 00ff 00ff
(Command sent to all 4 devices)

32-bit write: command = 00ff 00ff
(Command sent to both devices)

16-bit write: command = 00ff
(Command sent to single device)
```

If you are reading the status register to see if a write command has completed (Ready bit: bit-7 = 1), then you would read:

```
64-bit write: command = 0080 0080 0080 0080
(All 4 devices ready)

32-bit write: command = 0080 0080
(Both devices ready)

16-bit write: command = 0080
(Single device ready)
```

Bits [1:0] of the Board Configuration register 1 at 7FF4,0000 (see page 2-7) provide the data path width:

Table 4-4: *BCR1 Bits [1:0]*

Bits [1:0]:	Data Path Width:
00	64 bits
01	32 bits
10	16 bits

ON-CARD SDRAM

The PmPPC supports 32-, 64-, 128-, and 256-megabyte configurations of 72-bit wide synchronous DRAM (SDRAM). The memory chips are 4-, 8-, 16-, or 32 M x 16, 3.3 V, SDRAM devices organized as one bank of five devices (the fifth part supports the 8-bit ECC option). On-card SDRAM occupies physical addresses from $0000,0000_{16}$ to $0FFF,FFFF_{16}$. The SDRAM is controlled by the CPC700 DRAM controller, which may be programmed for most memory sizes and speeds, various block sizes, and write protection.

In addition to the basic SDRAM control functions the CPC700 chip provides several additional DRAM-related functions and contains the following performance enhancing features:

- Programmable delay insertion for controlling RAS precharge time, RAS low time, CAS setup before RAS time, CAS precharge time, CAS pulse width, CAS access time, and address access time.
- Logic needed to control ECC generation, and checking logic with functions to clear ECC errors.

Bits 5:4 of Board Configuration register 0 (see page 2-7) at $7FF4,0001_{16}$ store the SDRAM bank configuration information. A programmable logic device (PLD) maintains these configuration values, which are determined by whether or not specific configuration resistors are physically installed on the PmPPC circuit board.

NONVOLATILE MEMORY MAP

A portion of ROM is reserved by Emerson for data storage. The following memory map convention allows various operating systems to store their boot parameters without affecting each other.

Table 4-5: *Nonvolatile Memory Map*

Address Range (hex):	Description:
400-7FF	User nonvolatile data storage
300-3FF	Reserved for the operating system
000-2FF	Reserved for the PmPPC monitor

Refer to “NVRAM Commands” on page 9-23 for details on programming the nonvolatile memory.

PMC/PCI Interface

The PmPPC module complies with the PCI Mezzanine Card (PMC) form factor for Peripheral Component Interconnect (PCI) modules and the proposed specification for Processor PCI Mezzanine Cards (PrPMC). The IBM CPC700 memory controller/PCI bridge chip implements the interface.

PCI BRIDGE

The CPC700 supports PCI bus speeds of up to 66 MHz (refer to page 5-6). Additional features include:

- compliance with the PCI 2.1 bus interface specification
- 32-bit PCI address/data bus
- optional internal PCI arbiter
- error tracking/status

PCI Interface Registers

Table 5-1 lists the registers associated with the PCI interface. The PmPPC monitor configures the CPC700 controller so that it provides these 32-bit registers to the PowerPC processor in the correct byte order (assuming the access width is 32 bits). Refer to the *CPC700 Functional Specification* for complete details on the PCI interface registers.

Table 5-1: *PCI Interface Register Summary*

CPC700 Address (hex):	Register Name:	Access Mode:	Description:
FF40,0000	PMM0LA	R/W	PMM 0 Local Address
FF40,0004	PMM0MA	R/W	PMM 0 Mask/Attribute
FF40,0008	PMM0PCILA	R/W	PMM 0 PCI Low Address
FF40,000C	PMM0PCIHA	R/W	PMM 0 PCI High Address
FF40,0010	PMM1LA	R/W	PMM 1 Local Address
FF40,0014	PMM1MA	R/W	PMM 1 Mask/Attribute
FF40,0018	PMM1PCILA	R/W	PMM 1 PCI Low Address
FF40,001C	PMM1PCIHA	R/W	PMM 1 PCI High Address
FF40,0020	PMM2LA	R/W	PMM 2 Local Address
FF40,0024	PMM2MA	R/W	PMM 2 Mask/Attribute
FF40,0028	PMM2PCILA	R/W	PMM 2 PCI Low Address
FF40,002C	PMM2PCIHA	R/W	PMM 2 PCI High Address
FF40,0030	PTM1MS	R/W	PTM 1 Memory Size
FF40,0034	PTM1LA	R/W	PTM 1 Local Address
FF40,0038	PTM2MS	R/W	PTM 2 Memory Size

CPC700 Address (hex):	Register Name:	Access Mode:	Description: (continued)
FF40,003C	PTM2LA	R/W	PTM 2 Local Address
FEC0,0000	PCICFGADR	R/W	PCI Configuration Address register
FEC0,0004	PCICFGDATA	R/W	PCI Configuration Data register

PCI Configuration Registers

The CPC700 has a number of configuration registers that affect the PCI interface. Table 5-2 summarizes these registers. Refer to the ***CPC700 Functional Specification*** for complete details on the individual PCI configuration registers.

Table 5-2: *PCI Interface Register Summary*

CPC700 Address (hex):	Register Name:	Access Mode:	Description:	Hex Default:
01–00	PCIVENDID	R/W	Vendor ID	1223
03–02	PCIDEVID	R	Device ID	000E
05–04	PCICMD	R/W	Command register	0006
07–06	PCISTATUS	R/W	Status register	0200
08	PCIREVID	R	Revision ID	01
09	PCIINTCLS	R/W	Interface Class	00
0A	PCISUBCLS	R/W	Sub-Class Code	00
0B	PCIBASECC	R/W	Base Class Code	06
0C	PCICACHELS	R	Cache Line Size	00
0D	PCILATTIM	R/W	Latency Timer	07
0E	PCIHDTYPE	R	Header Type	00
0F	PCIBIST	R	Built-In Self Test Control	00
10	PCIBAR0	R	BAR 0 not used	–
14	PCIPTM1BAR	R/W	PTM 1 BAR	varies
18	PCIPTM2BAR	R/W	PTM 2 BAR	varies
1C–2B	reserved	–	not used or reserved	–
2C	PCISUBSYSID	R/W	PCI Subsystem ID	00F9
2E	PCISUBSYSVENDID	R/W	PCI Subsystem Vendor ID	1014
30–3B	reserved	–	reserved	–
3C	PCIINTLN	R/W	Interrupt Line	00
3D	PCIINTPN	R	Interrupt Pin	00
3E	PCIMINGNT	R	Minimum Grant	00
3F	PCIMAXLTNCY	R	Maximum Latency	00
40	PCIBUSNUM	R	Bus Number	00
41	PCISUBBUSNUM	R	Subordinate Bus Number	00
42	PCIDSCCNT	R	Disconnect Counter	00

CPC700 Address (hex):	Register Name:	Access Mode:	Description: (continued)	Hex Default:
47–44	PCIARBCNTL	R/W	PCI Arbiter Control	00000000
48	PCIERREN	R/W	Error Enable	6F
49	PCIERRSTS	R/W	Error Status	00
4B–4A	PCIBRDGOPT1	R/W	Bridge Options 1	FF60
4F–4C	PCIPLBSES	R/W	PLB Slave Error Syndrom	00000000
53–50	PCIPLBSEAR0	R	PLB Slave Error Address reg. 0	00000000
57–54	PCIPLBSEAR1	R	PLB Slave Error Address reg. 1	00000000
58–5F	reserved	–	reserved	–
61–60	PCIBRDGOPT2	R/W	Bridge Options 2	0111

The PCI Configuration Address and Data registers at $FEC0,0000_{16}$ and $FEC0,0004_{16}$ allow access to the CPC700 memory interface registers. To initiate an access, write the hex offset value for the desired register to address $FEC0,0000_{16}$. The data for that register then may be read from or written to the data register at $FEC0,0004_{16}$. See Monitor Sections “CPC700 Indirect Reads” on page 9-39 and “CPC700 Indirect Writes” on page 9-39. Note that accesses should be non-line, non-burst PLB transactions of the correct size. Failure to access all the bytes of a particular register or reading reserved bits will produce unpredictable results.

PCI INITIALIZATION

As a PCI bus master, the CPC700 can generate memory, I/O, configuration, interrupt acknowledge, and special cycles. PCI memory cycles occur when the PCI interface detects a cycle within a specific address range, as specified via the CPC700 PMM registers (see Table 5-1). These registers also specify whether the PCI memory cycle is an offset from the PLB address. The PMM registers must be initialized before attempting to generate PCI memory cycles.

As a target on the PCI bus, the CPC700 can respond to memory cycles occurring in certain address ranges, as specified by the BAR1 and BAR2 registers (see Table 5-2). The local CPU must specify the size of the ranges and the resulting PLB address via the PTM registers, which must be initialized before the CPC700 can respond as a PCI memory target.

Figure 5-1: *Example, PCI Address Map*

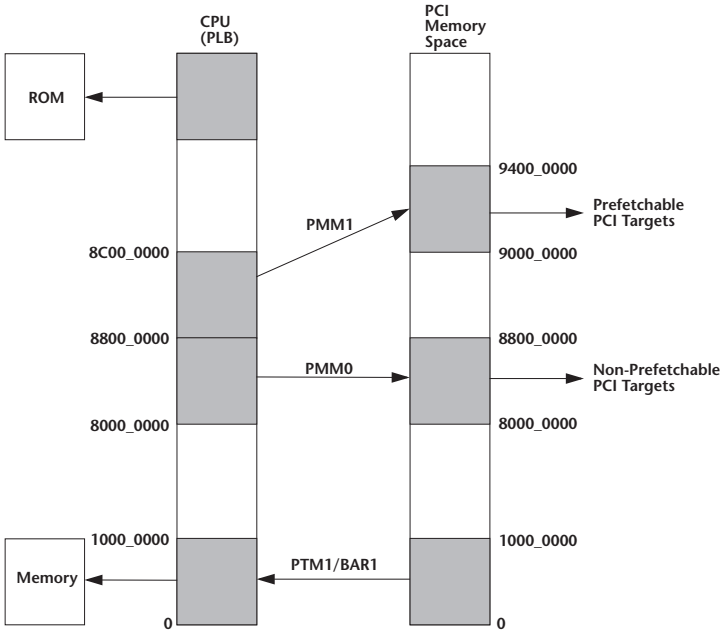


Fig. 5-1 shows a typical PCI address map. In this example, the system memory resides from $0000,0000_{16}$ to $0FFF,FFFF_{16}$ in the CPU/PLB address space. It is accessible in the same address space as defined by PTM1/BAR1. The CPU/PLB master can access PCI memory space in two places. $8000,0000_{16}$ to $87FF,FFFF_{16}$ is mapped to the same address on the PCI bus and is non-prefetchable. $8800,0000_{16}$ to $8BFF,FFFF_{16}$ is translated to address range $9000,0000_{16}$ to $93FF,FFFF_{16}$ of PCI memory space. Table 5-3 lists the register initialization values for this example. Refer to the *CPC700 User's Manual* (Table 1-3) for complete details regarding PCI initialization.

Table 5-3: *PCI Interface Register Summary*

CPC700 Address (hex):	Register Name:	Description:	Default (hex):
FF40,0000	PMM0LA	PMM 0 Local Address	8000,0000
FF40,0004	PMM0MA	PMM 0 Mask/Attribute	F800,0001
FF40,0008	PMM0PCILA	PMM 0 PCI Low Address	8000,0000
FF40,000C	PMM0PCIHA	PMM 0 PCI High Address	0000,0000
FF40,0010	PMM1LA	PMM 1 Local Address	8800,0000
FF40,0014	PMM1MA	PMM 1 Mask/Attribute	FC00,0003
FF40,0018	PMM1PCILA	PMM 1 PCI Low Address	9000,0000
FF40,001C	PMM1PCIHA	PMM 1 PCI High Address	0000,0000

CPC700 Address (hex):	Register Name:	Description:	Default (hex):
FF40,0030	PTM1MS	PTM 1 Memory Size	F000,0001
FF40,0034	PTM1LA	PTM 1 Local Address	0000,0000
14 (offset)	PCIPTM1BAR	PTM 1 BAR	0000,0008

PCI INTERRUPTS

The PmPPC module can generate and receive interrupts using doorbell-type interrupts on the INTA*, INTB*, INTC*, and INTD* lines from the P11 connector. A programmable logic device (PLD) on the PmPPC board provides an 8-bit, read/write register, called the PMC Interrupt Output register (PIOR), at location $7FF4,0004_{16}$ for the interrupt lines.

Register 5-1: *PMC Interrupt Output (PIOR)*

7	6	5	4	3	2	1	0
EREADY		M66EN	reserved	INTD	INTC	INTB	INTA

Note: *Technically, the PCI specification prohibits using the PMC Interrupt Output register to drive the INTA*–INTD* lines. However, the PmPPC provides this capability for applications that do not require strict adherence to the specification.*

A host access from the PCI interface to this register allows the target PCI device to acknowledge the interrupt by turning off the INTA* interrupt. The module's PowerPC processor also can use this register to determine if a pending interrupt has been serviced. Although the interrupts are active low, the register values are active high. For example, a value of one in the INTA field indicates that an interrupt is pending on INTA*. Also, writing a one to this location asserts the INTA* interrupt.

MONARCH FUNCTIONALITY

The PmPPC can be configured to function as either a Monarch or non-Monarch module, as described in the VITA 32 PrPMC draft specification. A Monarch is the main PrPMC device on the local PCI bus. It performs enumeration on that bus after power-up and is often the interrupt handler. A non-Monarch module does not perform enumeration on the local bus after power-up. Bit 7 of Board Configuration register 1 (see page 2-7) at location $7FF4,0000_{16}$ indicates how the module is configured (0=Monarch, 1=non-Monarch), as determined by the signal on pin 64 of connector P12. The software can read the Monarch line status to configure the board, and the hardware is unaffected.

Note: *For a Monarch module, the EREADY signal must be monitored at bit 6 because bit 7 must be off before querying its state (refer to page 5-5).*

The PMC Interrupt Output register (see page 5-5) at location 7FF4,0004$_{16}$ has additional bits to support Monarch functionality. EREADY bits 7 and 6, respectively, drive and monitor the EREADY line. For a non-Monarch, it is presumed that this signal is initially asserted, then removed when the bus is ready for enumeration. Once the CPC700 PCI bridge has been initialized by the on-card software, bit 7 can be driven low to tri-state the EREADY signal, indicating that the PmPPC is ready. When all the other PCI devices have stopped driving this signal low, the Monarch will enumerate the bus. See the draft PrPMC standard (in Table 1-3) for carrier board pullup requirements.

66MHZ BUS OPERATION

In order for the PCI bus to operate at 66 MHz, all devices on the bus must be capable of that speed. When the M66EN signal is high for a particular PCI device, it indicates that the device can operate at 66 MHz. For 33 MHz modules, M66EN is grounded, so the signal will be high only when all devices on the PCI bus are capable of operating at 66 MHz. The software can read bit 5 of the PMC Interrupt Output register at location 7FF4,0004$_{16}$ (see page 5-5) to determine the bus speed. If bit 5 is high, the bus speed is 66 MHz; if it is low, the bus speed is 33 MHz. If any PCI device pulls the wire or M66EN signal low, then the bus speed will be 33 MHz for all of the devices. See the draft PrPMC standard (in Table 1-3) for carrier board pullup requirements.

PCI ARBITRATION

In its standard configuration, the PmPPC module is not the system controller. The following REQ* and GNT* options are available on specific hardware versions. A PmPPC with PLD version 03 will reset the registers, whereas PLD version 04 will not.

The PmPPC can drive the REQ* output to request permission from the baseboard arbiter to become a PCI bus master. The PmPPC also can monitor the GNT* bus grant input, as described by the PMC standard. The IBM CPC700 has an optional internal arbiter (limited to a maximum PCI bus speed of 33 MHz) that can handle six PCI request inputs. The REQ0*–REQ5* and GNT0*–GNT5* signals for this arbiter route to PMC connector P14 for access by the carrier board. See the draft PrPMC standard (in Table 1-3) for carrier board pullup requirements.

PCI RESET

Circuitry on the PmPPC resets the entire module if the voltages fall out of tolerance (due to power-on reset) or if the optional on-board reset switch is activated. The IBM CPC700 control register settings are initialized immediately following this reset to configure the module properly before allowing any external PCI accesses to occur. (The PCI RST* signal does not affect these CPC700 register settings.)

The PmPPC module activates the RESET_OUT* signal on pin 60 of the P12 connector when the voltages fall out of tolerance (due to power-on reset) or when the optional on-board reset switch is activated. Writing to bit 12 of the CPC700 controller's PCIBRDGOPT2 register (see Table 5-2) also activates the RESET_OUT* signal. This is an optional, open-drain signal (see PrPMC draft standard, Table 1-3) that can be driven by either the processor PMC module or the carrier board. The carrier may use this signal to generate a PCI RST* signal for the processor PMC module.

When PCI RST* is asserted or the proper sequence is written to the Reset Control register by an external host (see "Processor Reset" on page 3-1), the PmPPC module resets all the on-board components, except the CPC700 bridge/controller (depending on the PLD version).

PCI BUS CONTROL SIGNALS

The following signals for the PCI interface are available on connectors P11 or P12. Refer to the PCI specification for details on using these signals. All signals are bi-directional unless otherwise stated.

Note: *A sustained tri-state line is driven high for one clock cycle before float.*

AD00–AD31: ADDRESS and DATA bus (bits 0-31) three-state lines are used for both address and data handling. A bus transaction consists of an address phase followed by one or more data phases.

C/BE0*–C/BE3*: BUS COMMAND and BYTE ENABLE three-state lines have different functions depending on the phase of a transaction. During the address phase of a transaction these lines define the bus command. During a data phase the lines are used as byte enables.

CLK: CLOCK input signal to PMC modules provides timing for PCI transactions.

DEVSEL*: DEVICE SELECT sustained three-state signal indicates when a device on the bus has been selected as the target of the current access.

EREADY: READY signal is an input for Monarch modules and an output for non-Monarch modules. It indicates that all modules are initialized and the PCI bus is ready to be enumerated.

FRAME*: CYCLE FRAME sustained three-state line is driven by the current master to indicate the beginning of an access, and continues to be asserted until the transaction reaches its final data phase.

GNT*: GRANT input signal indicates that access to the bus has been granted to a particular master. Each master has its own GNT*.

IDSEL: INITIALIZATION DEVICE SELECT input signal acts as a chip select during configuration read and write transactions.

INTA*, INTB*, INTC*, INTD*:

PMC INTERRUPTS A, B, C, D bidirectional lines are used by the PMC module to interrupt the baseboard, or vice versa.

IRDY*: INITIATOR READY sustained three-state signal indicates that the bus master is ready to complete the data phase of the transaction.

M66EN: ENABLE 66MHZ when grounded, prevents 66 MHz operation of the PCI bus.

MONARCH*: MONARCH when grounded, it indicates that the PrPMC module is a Monarch and must provide PCI bus enumeration and interrupt handling.

PAR: PARITY is even parity across AD00-AD31 and C/BE0-C/BE3*. Parity generation is required by all PCI agents. This three-state signal is stable and valid one clock after the address phase, and one clock after the bus master indicates that it is ready to complete the data phase (either IRDY* or TRDY* is asserted). Once PAR is asserted, it remains valid until one clock after the completion of the current data phase.

PERR*: PARITY ERROR sustained three-state line is used to report parity errors during all PCI transactions.

PRESENT*: PRESENT when grounded, indicates to a carrier that a PMC module is installed.

RESET_OUT*: RESET OUTPUT optional signal may be used to support a reset button or other reset source on the PrPMC module. It is an open drain output from the PrPMC module that becomes an input to the reset logic on the carrier card. To avoid reset loops, do not use RST* to generate RESET_OUT*.

Note: *PmPPC with PLD version 03 will reset the bridge and drive RESET_OUT, PLD version 04 will not.*

REQ64*: This optional output signal is used to tell a 64-bit PCI device whether to use the 64-bit or the 32-bit data width.

REQ*: REQUEST output pin indicates to the arbiter that a particular master wants to use the bus.

RST*: RESET assertion of this input line brings PCI registers, sequencers, and signals to a consistent state. The carrier card generates this system reset signal as an input to all PrPMC modules.

SERR*: SYSTEMS ERROR open-collector output signal is used to report any system error with catastrophic results.

STOP*: STOP sustained three-state signal is used by the current target to request that the bus master stop the current transaction.

TRDY*: TARGET READY sustained three-state signal indicates the target's ability to complete the current data phase of the transaction.

PMC CONNECTOR PINOUTS

P11 and P12 are the 32-bit PCI bus connectors. Each connector has 64 pins (Emerson item number 01899011-00, Molex P/N 53508-0648; see figure on page 5-12).

Table 5-4: *P11 Pin Assignments*

Pin:	Signal:	Pin:	Signal:
1	TCK	2	no connect
3	ground	4	INTA*
5	INTB*	6	INTC*
7	PRESENT*	8	+5 V
9	INTD*	10	no connect
11	ground	12	no connect
13	CLK	14	ground
15	ground	16	GNT*
17	REQ*	18	+5 V
19	V(I/O)	20	AD31
21	AD28	22	AD27
23	AD25	24	ground
25	ground	26	C/BE3*
27	AD22	28	AD21
29	AD19	30	+5 V
31	V(I/O)	32	AD17
33	FRAME*	34	ground
35	ground	36	IRDY*
37	DEVSEL*	38	+5 V
39	ground	40	no connect
41	no connect	42	no connect
43	PAR	44	ground
45	V(I/O)	46	AD15
47	AD12	48	AD11
49	AD9	50	+5 V
51	ground	52	C/BE0*

Pin:	Signal:	Pin:	Signal:
53	AD6	54	AD5
55	AD4	56	ground
57	V(I/O)	58	AD3
59	AD2	60	AD1
61	AD0	62	+5 V
63	ground	64	REQ64*

Table 5-5: *P12 Pin Assignments*

Pin:	Signal:	Pin:	Signal:
1	no connect	2	TRST*
3	TMS	4	TDO
5	TDI	6	ground
7	ground	8	no connect
9	no connect	10	no connect
11	PUP (pull up)	12	no connect
13	RST*	14	PDN (pull down)
15	no connect	16	PDN (pull down)
17	PME*	18	ground
19	AD30	20	AD29
21	ground	22	AD26
23	AD24	24	no connect
25	IDSEL	26	AD23
27	no connect	28	AD20
29	AD18	30	ground
31	AD18	32	C/BE2*
33	ground	34	no connect
35	TRDY*	36	no connect
37	ground	38	STOP*
39	PERR*	40	ground
41	no connect	42	SERR*
43	C/BE1*	44	ground
45	M66EN	46	AD13
47	AD12	48	AD10
49	AD8	50	no connect
51	AD7	52	no connect
53	no connect	54	no connect
55	no connect	56	ground
57	no connect	58	EREADY
59	ground	60	RESETOUT*

Pin:	Signal:	Pin:	Signal:
61	no connect	62	no connect
63	ground	64	MONARCH*

P14 routes the serial port and I^2C signals to the backplane. It is a 64-pin connector (see figure on page 5-12). Optional, factory-installed jumpers allow for Motorola compatibility by connecting the additional signals shown in the shaded table cells below.

Table 5-6: *P14 Pin Assignments*

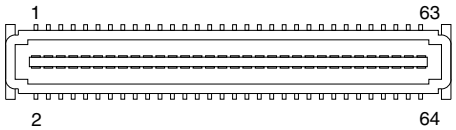
Pin:	Signal[1]:	Pin:	Signal:
1	I^2C1 SDA	2	I^2C1 SCL
3	Console TxData	4	Console RxData
5	Download TxData	6	Download RxData
7	no connect	8	ground
9	ground	10	no connect
11	no connect	12	no connect
13	TD_P (Ethernet)	14	TD_N (Ethernet)
15	RD_P (Ethernet)	16	RD_N (Ethernet)
17	no connect	18	no connect
19	no connect	20	no connect
21	no connect	22	no connect
23	no connect	24	ground
25	ground	26	no connect
27	no connect	28	no connect
29	no connect	30	no connect
31	no connect	32	no connect
33	no connect	34	no connect
35	no connect	36	no connect
37	no connect	38	no connect
39	+5V (with fuse)	40	no connect
41	ground	42	no connect
43	PMC REQ0*	44	PMC GNT0*
45	PMC REQ1*	46	PMC GNT1*
47	PMC REQ2*	48	PMC GNT2*
49	PMC REQ3*	50	PMC GNT3*
51	PMC REQ4*	52	PMC GNT4*
53	PMC REQ5*	54	PMC GNT5*
55	no connect	56	ground
57	ground	58	no connect
59	no connect	60	no connect
61	no connect	62	no connect

Pin:	Signal[1]:	Pin:	Signal:
63	no connect	64	no connect

1. Shaded signal cells are “no connect” for the standard (default) pin out.

Refer to the component map in Fig. 2-1 for the location and orientation of the PMC connectors on the PmPPC circuit board.

Figure 5-2: *PMC Connector*



Ethernet

The PmPPC provides a local area network (LAN) interface using the CS8900A Crystal LAN™ ISA Ethernet Controller chip from Crystal Semiconductor. The CS8900A is a single-chip solution that includes a full-duplex, 10BASE-T port with analog filters. Some additional features include:

- IEEE 802.3 compliant
- On-chip RAM buffers
- Programmable transmit and receive features
- Serial EEPROM support
- LED drivers for link status and LAN activity

CS8900A CONFIGURATION

The CS8900A Ethernet controller has various configuration and status registers, as summarized in Table 6-1. The CS8900A is optimized to operate in either memory mode or I/O mode and does not use direct memory access. In memory mode, it allows internal memory to be mapped anywhere within the CPC700 controller's memory space. In I/O mode, it supports indirect accesses through 16 locations.

The CS8900A automatically reloads its configuration from a direct connection to a 3-wire, serial EEPROM located on the PmPPC circuit board. The EEPROM stores information such as the MAC address and sets the CS8900A to operate in memory mode. During configuration the software may select one of four active-high interrupt pins to be active on the CS8900A. The Ethernet controller's INTRQ0 line is configured as an interrupt output because it drives the active-high ETH_INT line on the CPC700 (see Table 3-2 for interrupt assignments).

For complete details on specific registers and configuration values, consult the CS8900A data sheet (see reference in Table 1-3).

Table 6-1: *Ethernet Controller Memory Map*

64-bit Offset (hex):	CS8900A Offset (hex):	Access Mode:	Description:	Defalt Reset (hex):
7FE0,4000	000	R	Product identification code	0E63,0007
7FE0,4010	004	–	reserved	–
7FE0,4080	020	R/W	I/O base address	300
7FE0,4088	022	R/W	Interrupt number	xxx4
7FE0,4090	024	R/W	DMA channel number	xxx3
7FE0,4098	026	R	DMA start of frame	0000
7FE0,40A0	028	R	DMA frame count (12 bits)	x000
7FE0,40A8	02A	R	RxDMA byte count	0000

64-bit Offset (hex):	CS8900A Offset (hex):	Access Mode:	Description: (continued)	Defalt Reset (hex):
7FE0,40B0	02C	R/W	Memory base address register (low word)	xxx0,0000
7FE0,40B8	02E	R/W	Memory base address register (high word)	xxx0,0000
7FE0,40C0	030	R/W	Boot PROM base address	xxx0,0000
7FE0,40D0	034	R/W	Boot PROM address mask	xxx0,0000
7FE0,40E0	038	–	reserved	–
7FE0,4100	040	R/W	EEPROM command	xxxx
7FE0,4108	042	R/W	EEPROM data	xxxx
7FE0,4110	044	–	reserved	–
7FE0,4140	050	R	Received frame byte counter	xxxx
7FE0,4148	052	–	reserved	–
7FE0,4400	100	R/W	Configuration & control registers (2 bytes each)	from ROM
7FE0,4480	120	R	Status & event registers (2 bytes each)	undefined
7FE0,4500	140	–	reserved	–
7FE0,4510	144	W	TxCMD (transmit command)	undefined
7FE0,4518	146	W	TxLength (transmit length)	undefined
7FE0,4520	148	–	reserved	–
7FE0,4540	150	R/W	Logical address filter (hash table)	0
7FE0,4560	158	R/W	Individual address	from ROM
7FE0,4578	15E	–	reserved	–
7FE0,5000	400	R	RxStatus (receive status)	undefined
7FE0,5008	402	R	RxLength (receive length)	undefined
7FE0,5010	404	R	Receive frame location	undefined
7FE0,6800	A00	W	Transmit frame location	undefined

Note: *Since the CPC700 controller always tries to read a 64-bit quantity, the CS8900A is mapped as a 64-bit device to eliminate inadvertent clearing of adjacent registers by the CPC700. The internal (PacketPage) offsets listed in the CS8900A data sheet must be adjusted for 64-bit alignment.*

In I/O mode, the CS8900A provides indirect access to the I/O ports. (See the CS8900A data sheet for details on using and configuring this mode.) The control register at offset 20_{16} determines the base address (default=300_{16}). When accessing a PacketPage register in I/O mode, the I/O space offsets need to be shifted left by two bits, then logically OR'd with the CS8900A base address. For example, the normal I/O base address is 7FE0,0000_{16} + 300_{16}. This becomes 7FE0,0000_{16} + $C00_{16}$, or 7FE0,$0C00_{16}$. To access the PacketPage Pointer using this example, the address would be 7FE0,$0C00_{16}$ + 28_{16}, or 7FE0,$0C28_{16}$. The following table describes I/O memory locations for this mode:

Table 6-2: *Ethernet I/O Locations (I/O mode)*

64-bit Offset (hex):	Access Mode:	Description:
0000	R/W	Receive/Transmit Data (Port 0)
0010	R/W	Receive/Transmit Data (Port 1)
0010	W	TxCMD (Transmit Command)
0018	W	TxLength (Transmit Length)
0040	R	Interrupt Status Queue
0028	R/W	PacketPage Pointer
0030	R/W	PacketPage Data (Port 0)
0038	R/W	PacketPage Data (Port 1)

In memory mode, the CS8900A supports Standard or Ready Bus cycles without introducing additional wait states. For this mode, the offsets in memory space also need to be shifted left by two bits. Additionally, you must enable and program the memory space before using it. CS8900A memory space should start at $7FE0,4000_{16}$ in the CPC700 address map. For the address decode to work properly, program the value $0008,1000_{16}$ into the Memory Base Address register in PacketPage space. This can be accomplished by using I/O accesses to PacketPage offsets $2C_{16}$ (least significant word) and $2E_{16}$ (most significant word). Once this register is programmed and the MemoryE bit is set (BusCTL register, PacketPage offset 116_{16}), memory mode will be enabled. The following table summarizes the memory mode I/O locations for the PmPPC:

Table 6-3: *Ethernet I/O Locations (memory mode)*

64-bit Offset (hex):	Access Mode:	Name:	Description:
1000	R	RxStatus	Receive Status
0008	R	RxLength	Receive Length
1010	R	RxFrame	Receive Frame
2800	W	TxFrame	Transmit Frame

For both I/O and memory modes, you should read from and write to the following location in the 64-bit word: xxxx,xxxx,DDDD,xxxx where D=good data and x=unused data.

ETHERNET ADDRESS

The Ethernet address for your board is a unique identifier on a network and must not be altered. The address consists of 48 bits divided into two equal parts. The upper 24 bits define a unique identifier that has been assigned to Emerson Network Power, Embedded Computing by IEEE. The lower 24 bits are defined by Emerson for identification of each of our products.

The Ethernet address for the PmPPC is a binary number referenced as 12 hexadecimal digits separated into pairs, with each pair representing eight bits. The address assigned to the PmPPC has the following form:

00 80 F9 52 yy zz

00 80 F9 is Emerson's identifier. **52** is the identifier for the PmPPC product group. The last two pairs of hex numbers correspond to the following formula: *n* – 1000, where *n* is the unique serial number assigned to each board. For example, if the serial number of a PmPPC is 2867, the calculated value is 1867 ($74B_{16}$). Therefore, the board's Ethernet address is 00:80:F9:52:07:4B. The complete Ethernet address is stored at byte offset 20_{16} in the serial ROM attached to the CS8900A.

ETHERNET PORT

The Crystal Semiconductor CS8900A supports 10 Mb/s communications for the Ethernet port, which is available at the P1 RJ45 connector on the front panel. An optional configuration provides Ethernet signals on pins 13–16 of the PMC connector at P14 (see Table 5-6). The P1 pin assignments are given in the following table:

Table 6-4: *Ethernet Pin Assignments, P1*

Pin:	Signal[1]:	Pin:	Signal:
1	Tx+	2	Tx-
3	Rx+	4	reserved (RX_CT)
5	reserved (RX_CT)	6	Rx-
7	reserved (TX_CT)	8	reserved (TX_CT)
9	no connect	10	no connect
11	chassis ground	12	chassis ground

1. The reserved pins are terminated as differential pairs and tied to chassis ground via a resistor and capacitor (see schematics for details).

The PmPPC has two 16550-compatible serial ports internal to the IBM CPC700 Memory Controller and PCI Bridge. These ports consist of serial data in and serial data out lines, translated to EIA-232 levels by a Maxim transceiver. The serial ports are accessible via the front panel 9-pin connector or the P14 PMC connector. However, you should not use both connectors simultaneously for serial port signals. If the EIA-232 serial cable is removed or the UART transmitters are inactive for more than 30 seconds, an automatic low-power shutdown occurs in the transceiver. It wakes up again upon sensing a valid transition on the input.

UART CONTROL

Table 7-1 summarizes the registers associated with the serial ports. Refer to the *CPC700 Functional Specification* for complete details on the UART configuration registers.

Table 7-1: *UART Configuration Registers*

CPC700 Address (hex):	Name:	Access Mode:	Description:
FF60,0300	UART0RBR	R	UART 0 Receiver Buffer register, DLAB=0
	UART0THR	W	UART 0 Transmitter Holding register, DLAB=0
	UART0DLL	R/W	UART 0 Divisor Latch (LSB), DLAB=1
FF60,0401	UART1IER	R/W	UART 0 Interrupt Enable register, DLAB=0
	UART1DLM	R/W	UART 0 Divisor Latch (MSB), DLAB=1
FF60,0402	UART1IIR	R	UART 0 Interrupt Identification register
	UART1FCR	W	UART 0 FIFO Control register
FF60,0403	UART1LCR	R/W	UART 0 Line Control register
FF60,0404	UART1MCR	R/W	UART 0 Modem Control register
FF60,0405	UART1LSR	R/W	UART 0 Line Status register
FF60,0406	UART1MSSR	R/W	UART 0 Modem Status register
FF60,0407	UART1SCR	R/W	UART 0 Scratch register
FF60,0400	UART1RBR	R	UART 1 Receiver Buffer register, DLAB=0
	UART1THR	W	UART 1 Transmitter Holding register, DLAB=0
	UART1DLL	R/W	UART 1 Divisor Latch (LSB), DLAB=1
FF60,0401	UART1IER	R/W	UART 1 Interrupt Enable register, DLAB=0
	UART1DLM	R/W	UART 1 Divisor Latch (MSB), DLAB=1
FF60,0402	UART1IIR	R	UART 1 Interrupt Identification register
	UART1FCR	W	UART 1 FIFO Control register
FF60,0403	UART1LCR	R/W	UART 1 Line Control register
FF60,0404	UART1MCR	R/W	UART 1 Modem Control register
FF60,0405	UART1LSR	R/W	UART 1 Line Status register
FF60,0406	UART1MSSR	R/W	UART 1 Modem Status register

CPC700 Address (hex):	Name:	Access Mode:	Description: (continued)
FF60,0407	UART1SCR	R/W	UART 1 Scratch register

SERIAL PORT REGISTERS

This section describes configuration bits for the PmPPC serial port control registers. See Table 7-1 for the appropriate hex addresses associated with each register. Note that both UARTs have the same registers, except the locations are different. Complete details are available in the *CPC700 Functional Specification*.

Interrupt Enable Register

The Interrupt Enable register (IER) enables specific interrupt sources for the serial ports. It is possible to disable all of the CPC700 serial port interrupts using this register.

Register 7-1: *CPC700 Serial Port Interrupt Enable (IER)*

0	1	2	3	4	5	6	7
reserved					ELSI	ETBEI	ERBFI

ELSI: Enable receiver Line Status Interrupt for error sources Overrun, parity, Framing, and Break
0 Disable
1 Enable

ETBEI: Enable Transmitter holding register Empty Interrupt
0 Disable
1 Enable

ERBFI: Enable Received data available interrupt
0 Disable
1 Enable

Interrupt Identification Register

The Interrupt Identification register (IIR) is a read-only register that allows the host CPU to determine the priority and source of an interrupt on a serial port.

Register 7-2: *CPC700 Serial Port Interrupt Identification (IIR)*

0	1	2	3	4	5	6	7
FIFO_EN		reserved		INT_ID			PEND

FIFO_EN[0:1]: Bits are set when FIFO control register bit [0] is equal to 1. Bits 2 and 3 are always zero in non-FIFO mode.

INT_ID[4:6]: Interrupt priority Identification. Bit 4 is always zero in non-FIFO mode.

011 Receiver line status (highest priority)
010 Received data ready
110 Character time-out indication
001 Transmitter holding register empty (lowest priority)
000 Reserved

PEND: Interrupt priority Identification. Bit 4 is always zero in non-FIFO mode.

0 Pending
1 None pending

Line Control Register

The Line Control register (LCR) controls the format of the serial line.

Register 7-3: *CPC700 Serial Port LIne Control (LCR)*

0	1	2	3	4	5	6	7
DLAB	BREAK	STICK	EPS	PEN	STB	WLS	

DLAB: Divisor Latch Access Bit

0 Allow access to RBR, THR, and IER
1 Allow access to baud rate generator divisor latch

BREAK: Set Break control bit

0 Break is disabled
1 Force TXD to spacing or logic “0” until reset

STICK: Sticky parity bit

0 Sticky parity is disabled
1 With PEN enabled, parity bit is transmitted and detected by receiver in opposite state from EPS bit

EPS: Even Parity Select–with PEN enabled

0 Odd parity
1 Even parity

PEN: Parity Enable

0 Disable
1 Enable

STB: Stop Bits
0 1 stop bit
1 1.5 stop bits for 5-bit words or 2 stop bits for 6- ,7- , and 8-bit words

WLS: Word length select bits
00 5 bits
01 6 bits
10 7 bits
11 8 bit

Line Status Register

The Line Status register (LSR) tracks the status of the serial line.

Register 7-4: *CPC700 Serial Port LIne Status (LSR)*

0	1	2	3	4	5	6	7
FIFO_ER	TEMT	THRE	BI	FE	PE	OE	DR

FIFO_ER: FIFO Error is always zero, except when in FIFO mode
0 Default
1 FIFO error (only when in FIFO mode)

TEMT: Transmitter Empty
0 THR or TSR contains a data character
1 THR and TSR are empty

THRE: Transmitter Holding Register Empty
0 Serial port not ready
1 Serial port ready for transmissionEPS

BI: Break Interrupt is reset when the CPU reads the LSRd
0 Reset to 0 when CPU reads LSR
1 Received data was held at logic “0” for longer than a full word transmission time

FE: Framing Error is reset when read
0 Reset to 0 when CPU reads LSR
1 Framing error detected (no stop bit)

PE: Parity Error is reset when read.
0 Reset to 0 when CPU reads LSR
1 Parity error detected

OE: Overrun Error
- 0 Reset to 0 when CPU reads LSR
- 1 Data in RBR was not read before being overwritten

DR: Data Ready
- 0 All data has been read in RBR or FIFO
- 1 An incoming character has been received and transferred into the RBR or FIFO

FIFO Control Register

The FIFO Control register (FCR) is a write-only register with the same address as the IIR. It is used in FIFO control operations.

Register 7-5: *CPC700 FIFO Control (FCR)*

0	1	2	3	4	5	6	7
RCVR_TRIG		reserved		DMA	XMIT	RCVR_F	FIFO

RCVR_TRIG: Receiver FIFO level Trigger level
- 00 1 byte
- 01 4 bytes
- 10 8 bytes
- 11 14 bytes

DMA: DMA mode select
- 0 If FIFOs are enabled, select DMA mode 0 for RXRDY and TXRDY outputs (single transfer DMA)
- 1 If FIFOs are enabled, select DMA mode 1 for RXRDY and TXRDY outputs (multiple transfer DMA)

XMIT: Transmitter FIFO reset
- 0 The 1 written to this position is self-clearing
- 1 Clear all bytes in transmitter FIFO and reset its counter logic to zero

RCVR_F: Receiver FIFO reset
- 0 The 1 written to this position is self-clearing
- 1 Clear all bytes in receiver FIFO and reset its counter logic to zero

FIFO: FIFO enable
- 0 Reset FIFOs
- 1 Receiver and transmitter FIFOs enabled

PROGRAMMABLE BAUD RATE

The CPC700 has a programmable baud rate generator that is configured by the divisor latch registers. The baud rate generator can divide the clock input by a number from 1 to 65535, which is stored as a 16-bit binary value in the divisor latch registers. The resulting clock output frequency is 16 times the baud rate. The divisor may be calculated as follows:

decimal divisor value = (OPB clock ÷ 4) ÷ (16 × baud rate)

For example, if the processor bus speed is 83.33 MHz, then the CPC700 on-chip peripheral bus (OPB) clock is 41.66 MHz. For a baud rate of 9600 bits/second, you would calculate the divisor as follows:

(41,666,666 ÷ 4) ÷ (16 × 9600) = 68 (rounded) = 44_{16}

The table below lists divisor latch values for some of the more common baud rates when the processor bus is running at 83 MHz.

Note: *Rounding errors can preclude the use of some baud rates at certain clock frequencies.*

Table 7-2: *UART Divisor Values (83 MHz processor bus)*

Bit Rate (bits/second):	Divisor Latch (MSB) Hex Value:	Divisor Latch (LSB) Hex Value:
1200	02	1F
2400	01	0F
4800	00	88
9600	00	44
19200	00	22
28800	00	17
38400	00	11
57600	00	0B

Note: *The EIA-232C specification defines a maximum rate of 20,000 bits per second over a typical 50-foot cable (2,500 picofarads maximum load capacitance). Higher baud rates are possible, but depend specifically upon the application, cable length, and overall signal quality.*

CONNECTORS AND CABLING

The IBM CPC700 provides two standard EIA-232 serial I/O ports. Both ports are available at the PmPPC front panel P2 connector and at the PMC P14 connector. Table 7-3 below shows the pinouts for the front panel connector and optional Y-cable assembly (item number C0007568-00). See Table 5-6 for the P14 connector pinouts

Note: *Do not use P2 and P14 simultaneously for the serial port signals.*

Table 7-3: *Front Panel Serial POrt Pin Assignements*

P2 Pin (Nano D):	Y-Cable Pin (DB9):	Signal Names:
1	A1	chassis ground (connected to bezel)
2	A3	serial IN1 (console)
3	A5	signal ground
4	A2	serial OUT1 (console)
5	–	no connect
6	B3	serial IN2 (download)
7	B5	signal ground
8	B2	serial OUT2 (download)
9	B1	chassis ground (connected to bezel)

Figure 7-1: *Front Panel Serial Port Connector, P2 (nano D)*

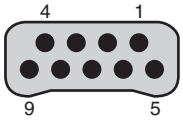
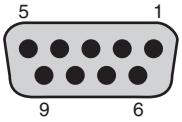
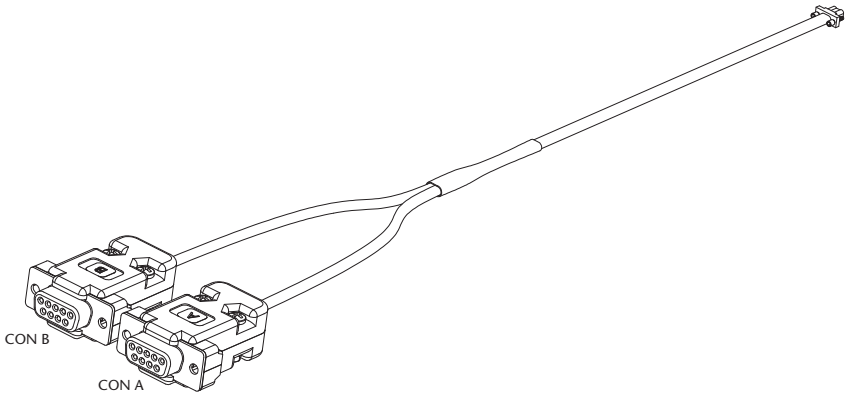


Figure 7-2: *Serial Port Cable Connector, A & B (DB9)*



Caution: **It is strongly recommended that the system provide a weight bearing attachment point to the serial cable within three inches of the PmPPC front panel. Such a mechanism is needed to provide an adequate strain relief for the serial I/O cable at the nano DB9 connector.**

Figure 7-3: *Y-Cable Assembly for Serial Port (C0007568-00)*



Other CPC 700 Functions

The IBM CPC700 memory controller/bridge chip provides various interfaces for the Emerson PmPPC. These include a processor interface (Chapter 3), memory interface (Chapter 4), PCI interface (Chapter 5), and serial I/O interface (Chapter 7). In addition, the CPC700 supports a processor local bus (PLB), an on-chip peripheral bus (OPB), a universal interrupt controller, inter-integrated circuit (I^2C) ports, and general purpose timers. This chapter summarizes these additional functions and their corresponding registers. Refer to the *CPC700 Functional Specification* for complete details.

BUS SUPPORT

The CPC700 can arbitrate up to four bus masters on the PowerPC processor local bus (PLB). The following registers configure the PLB:

Table 8-1: *PLB Macro Registers*

CPC 700 Address (hex):	Name:	Access Mode:	Description:
FF50,0850	PESRRD	R	PLB Error Status (read/clear)
FF50,0854	PESRWR	W	PLB Error Status (set)
FF50,085C	PACR	R/W	PLB Arbiter Control

The CPC700 on-chip peripheral bus (OPB) bridge allows a PLB bus master to control the OPB, which operates at half the clock speed of the PLB. The following registers support the PLB bridge:

Table 8-2: *OPB Macro Registers*

CPC 700 Address (hex):	Name:	Access Mode:	Description:
FF50,0810	GESRRD	R	OPB Bridge Error Status (read/clear)
FF50,0814	GESRWR	W	OPB Bridge Error Status (set)
FF50,0818	GEAR	R	OPB Bridge Error Address

UNIVERSAL INTERRUPT CONTROLLER

The CPC700 has a programmable, universal interrupt controller (UIC) that can handle 29 interrupt sources, including external, timer, UART, I^2C, PCI, and ECC interrupts. The following registers configure the UIC:

Table 8-3: *UIC Configuration Registers*

CPC 700 Address (hex):	Name:	Access Mode:	Description:
FF50,0880	UICSR	R/C	UIC Status (read/clear)

CPC 700 Address (hex):	Name:	Access Mode:	Description: (continued)
FF50,0884	UICSRS	R/S	UIC Status (set)
FF50,0888	UICER	R/W	UIC Enable
FF50,088C	UICCR	R/W	UIC Critical
FF50,0890	UICPR	R/W	UIC Polarity
FF50,0894	UICTR	R/W	UIC Trigger
FF50,0898	UICMSR	R	UIC Masked Status
FF50,089C	UICVR	R	UIC Vector
FF50,08A0	UICVCR	W	UIC Vector Configuration

IIC PORTS

The CPC700 has two inter-integrated circuit (IIC or I^2C) ports. For the PmPPC, port 0 (IIC0) supports an on-board serial EEPROM, while port 1 (IIC1) is accessible to target applications via PMC connector P14. The following table summarizes the I^2C configuration registers:

Table 8-4: *IIC Configuration Registers*

CPC 700 Address (hex):	Name:	Access Mode:	Description:
FF62,0000	UICSR	R/C	UIC Status (read/clear)
FF62,0001	UICSRS	R/S	UIC Status (set)
FF62,0002	UICER	R/W	UIC Enable
FF62,0003	UICCR	R/W	UIC Critical
FF62,0004	UICPR	R/W	UIC Polarity
FF62,0005	UICTR	R/W	UIC Trigger
FF62,0006	IIC0MDBUF	R/W	IIC0 Master Data Buffer
FF62,0007	–	–	reserved
FF62,0008	IIC0SDBUF	R/W	IIC0 Slave Data Buffer
FF62,0009	–	–	reserved
FF62,000A	IIC0LMADR	R/W	IIC0 Low Master Address
FF62,000B	IIC0HMADR	R/W	IIC0 High Master Address
FF62,000C	IIC0CNTL	R/W	IIC0 Control
FF62,000D	IIC0MDCNTL	R/W	IIC0 Mode Control
FF62,000E	IIC0STS	R/W	IIC0 Status
FF62,000F	IIC0EXTSTS	R/W	IIC0 Extended Status
FF62,0010	IIC0LSADR	R/W	IIC0 Low Slave Address
FF63,0000	IIC0HSADR	R/W	IIC0 High Slave Address
FF63,0001	IIC0CLKDIV	R/W	IIC0 Clock Divide
FF63,0002	IIC0INTRMSK	R/W	IIC0 Interrupt Mask
FF63,0003	IIC0XFRCNT	R/W	IIC0 Transfer Count

CPC 700 Address (hex):	Name:	Access Mode:	Description: (continued)
FF63,0004	IIC0XTCNTLSS	R/W	IIC0 Extended Control & Slave Status
FF63,0005	IIC0DIRECTCNTL	R/W	IIC0 Direct Control
FF63,0006	IIC1MDBUF	R/W	IIC1 Master Data Buffer
FF63,0007	–	–	reserved
FF63,0008	IIC1SDBUF	R/W	IIC1 Slave Data Buffer
FF63,0009	–	–	reserved
FF63,000A	IIC1LMADR	R/W	IIC1 Low Master Address
FF63,000B	IIC1HMADR	R/W	IIC1 High Master Address
FF63,000C	IIC1CNTL	R/W	IIC1 Control
FF63,000D	IIC1MDCNTL	R/W	IIC1 Mode Control
FF63,000E	IIC1STS	R/W	IIC1 Status
FF63,000F	IIC1EXTSTS	R/W	IIC1 Extended Status
FF63,0010	IIC1LSADR	R/W	IIC1 Low Slave Address

GENERAL PURPOSE TIMERS

The CPC700 has a 32-bit time-base counter, as well as five capture timers and five compare timers. Memory-mapped registers provide direct control of the general purpose timer functions.

Table 8-5: *General Purpose Timer Registers*

CPC 700 Address (hex):	Name:	Access Mode:	Description:
FF65,0000	GPTTBC	R/W	GPT Time Base Counter
FF65,0004	GPTCE	R/W	Capture Timers Enable
FF65,0008	GPTEC	R/W	Capture Events Edge Detection Control
FF65,000C	GPTSC	R/W	Capture Events Synchronization Control
FF65,0010-0014	–	–	reserved
FF65,0018	GPTIE	R/W	Timers Interrupt Enable
FF65,001C	GPTIS	R/W	Timers Interrupt Status
FF65,0020	GPTIS	R	Timers Interrupt Status (clear upon read)
FF65,0024	GPTIM	R/W	Timers Interrupt Mask
FF65,0028-003C	–	–	reserved
FF65,0040	GPTCAPT0	R	Capture Timer 0
FF65,0044	GPTCAPT1	R	Capture Timer 1
FF65,0048	GPTCAPT2	R	Capture Timer 2
FF65,004C	GPTCAPT3	R	Capture Timer 3
FF65,0050	GPTCAPT4	R	Capture Timer 4
FF65,0054-007C	–	–	reserved

CPC 700 Address (hex):	Name:	Access Mode:	Description: (continued)
FF65,0080	GPTCOMP0	R/W	Compare Timer 0
FF65,0084	GPTCOMP1	R/W	Compare Timer 1
FF65,0088	GPTCOMP2	R/W	Compare Timer 2
FF65,008C	GPTCOMP3	R/W	Compare Timer 3
FF65,0090	GPTCOMP4	R/W	Compare Timer 4
FF65,0094-00BC	–	–	reserved
FF65,00C0	GPTMASK0	R/W	TBC Mask (Compare Timer 0)
FF65,00C4	GPTMASK1	R/W	TBC Mask (Compare Timer 1)
FF65,00C8	GPTMASK2	R/W	TBC Mask (Compare Timer 2)
FF65,00CC	GPTMASK3	R/W	TBC Mask (Compare Timer 3)
FF65,00D0	GPTMASK4	R/W	TBC Mask (Compare Timer 4)
FF65,00D4-FC	–	–	reserved

MISCELLANEOUS CONTROL

The following registers setup clocking, power management, and reset control (CPR) for the CPC700.

Table 8-6: *Gemeral Purpose Timer Registers*

CPC 700 Address (hex):	Name:	Access Mode:	Description:
FF50,0900	CPRPMCTRL	R/W	Peripheral Power Management Control
FF50,0904	CPRRESET	R/W	Peripheral Reset Control
FF50,0908	CPRCAPTEVNT	R/W	GPT Capture Event Generation
FF50,090C	CPRPLLACCESS	R/W	PLL Configuration Access (unlocks CPRPLLTUNE)
FF50,0910	CPRPLLTUNE	R/W	PLL Configuration (resets system upon write)
FF50,0914	CPRSTRAPREAD	R	Strapping Pin Status Read

Monitor

The PmPPC monitor consists of about 150 C language functions. The monitor commands are a subset of these functions that provide easy-to-use tools for configuring the PmPPC at power-up or reset, as well as for communications, downloads, and other common tasks. This chapter describes the monitor's features, basic operation, and configuration sequences. This chapter also serves as a reference for the monitor commands and functions.

MONITOR FEATURES

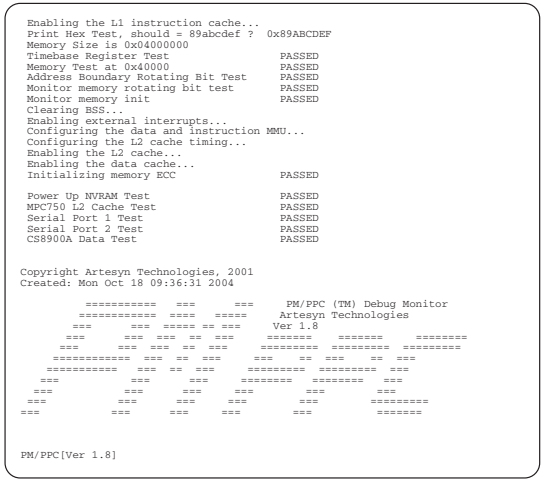
The PmPPC monitor has a command-line editor and can recall previous command lines. This section describes these features, as well as the start-up display.

Start-Up Display

At power-up or after a reset, the monitor runs diagnostics and reports the results in the start-up display. A failed memory test could indicate a hardware malfunction that should be reported to our Test and Repair Services department, visit http://www.emersonembeddedcomputing.com/contact/productrepair.html on the internet or send email to service-info@artesyn.com.

At power-up and reset, the monitor configures the board according to the contents of non-volatile configuration memory. If the configuration indicates that an autoboot device has been selected, the monitor attempts to load an operating system or application program from the specified device. You can prevent the board from booting the operating system or application if any of the power-up tests fail by setting the NVRAM configuration field *Halt-OnFailure* (see Table 9-3 and "NVRAM Commands" on page 9-23).

Figure 9-1: *Monitor Startup Display*



History

The monitor maintains a history of up to 50 command lines for reuse. Press the <ESC> key from the command line to access the history.

k or -: Move backward in the command history to access a previous command.

j or -: Move forward in the command history to access a subsequent command.

Command-Line Editor

The command-line editor uses typical UNIX® *vi* editing commands.

help editor: To access an on-line description of the editor, type `help editor` or `h editor`.

<ESC>: To exit Entry mode and start the editor, press <ESC>. You can use most common *vi* commands, such as **x**, **i**, **a**, **A**, **$**, **w**, **cw**, **dw**, **r**, and **e.**

<cr>: To exit the editor and execute the current command, press Enter or Return.

<DEL>: To discard an entire line and create a new command line, press <DEL> at any time.

a or A: Append text on the command line.

i or I: Insert text on the command line.

x or X: Delete a single character.

r: Replace a single character.

w: Move the cursor to the next word.

c: Change–use additional commands with c to change words or groups of words, as shown below.

cw or CW: Change a word after the cursor (capital W ignores punctuation).

ce or CE: Change text to the end of a word (capital E ignores punctuation).

cb or CB: Change the word before the cursor (capital B ignores punctuation).

c: Change text from the cursor to the end of the line.

d: Delete–use additional commands with **d** to delete words or groups of words, as shown below.

dw or DW: Delete a word after the cursor (capital W ignores punctuation).

de or DE: Delete to the end of a word (capital E ignores punctuation).

db or DB: Delete the word before the cursor (capital B ignores punctuation).

d$: Delete text from the cursor to the end of the line.

PowerPC Debugger

The PowerPC debugger allows the operator to probe memory-mapped devices. It features simple commands that execute without requiring a stack or memory. The debugger starts automatically if the internal diagnostics discover an error. Also, the operator may force the debugger to start by pressing the 'd' key before resetting the board. The debugger may be called from the monitor command line by using the **Debugger** command (although the **q** command is not functional in this case).

The following commands are available in the debugger:

? Display a list of available commands.

r [b l] *address*: Read a byte *[b]* or 32-bit long word *[l]* from an *address*.

w [b l] *address data*: Write a byte *[b]* or 32-bit long word *[l]* to an *address*.

d *address* : Display a 256-byte block of data beginning at *address*. After this command, additional blocks may be displayed by pressing return.

f *data address size*: Fill a block of *size* bytes with the byte *data* starting at *address*.

t *start end*: Perform a memory test from the *start* address to the *end* address. This is a rotating bit test on the block of memory. It writes 0x00000001 to the first 32-bit data value, 0x00000002 to the second, 0x00000004 to the third, and so on, until all addresses are written. It then reads the block of memory to verify that the data was written correctly. Next, it writes a second pattern starting with data values 0x00000002, 0x00000004, etc. and verifies the data

in the same way. In all, the test writes 32 patterns to the block of memory. If errors are detected, only the first 18 are displayed. The pattern test repeats with a rotating zero this time, rather than a one. And finally, the test writes a unique address value to every 32-bit long word to check for address mirrors.

i *[0 1]*: Enable *[1]* or disable *[0]* the L1 instruction cache.

m : Attempt to initialize the stack and monitor data, and start the monitor without executing the powerup/reset diagnostics.

BASIC OPERATION

The PmPPC monitor performs various configuration tasks upon power-up or reset. This section describes the monitor operation as it relates to these specific tasks and the memory initialization. The flowcharts beginning with Fig. 9-2 illustrate the power-up/reset sequence (italic text in flowcharts indicate an NVRAM field).

Power-Up/Reset Sequence

At power-up or board reset, the monitor performs hardware initialization, diagnostic routines, autoboot procedures, free memory initialization, and if necessary, it invokes the command-line editor.

If an unexpected interrupt occurs before the console port is available, the red LED flashes twice, followed by a static “binary” display of the exception number on all four LEDs (where the red LED represents the most-significant bit of the nibble). For example, if a DSI exception occurs (exception 3), the red LED would flash twice, followed by a static display of a binary 0011 (top two LEDs off, bottom two LEDs on). The flash/display sequence repeats indefinitely.

Power-up/reset sequence:

1. Initialize CPC700 Bank 2 parameters and turn off the four LEDs.
2. Read Board Configuration register 1 to determine if the monitor is booting from socketed or soldered flash, and then initialize Bank 0 accordingly.
3. Write 001 to the yellow LEDs. Test the decrementer. If it is not functioning, the red LED flashes three times while 001 remains displayed on the yellow LEDs. The power-up sequence will continue after this error, although the monitor may not function properly.
4. Write 010 to the yellow LEDs. Perform an 8-bit register test on the UART 0 Scratchpad register. If this test fails, the red LED flashes three times while 010 remains displayed on the yellow LEDs. The power-up sequence will continue after this error, but the console port may be unavailable.

5 Write 011 to the yellow LEDs. Initialize the console port to the 9600 baud (default), 8 data bits, 2 stop bits, and no parity. Print “Initializing...”.

6 Write 100 to the yellow LEDs. Perform a 32-bit register test on some internal CPC700 registers. If this test fails, the red LED flashes three times while 100 remains displayed on the yellow LEDs. After the flashing sequence, display a message to the console port, indicating the nature of the failure. (Note: the console port may not be accessible, depending on the specific error.)

7 Write 101 to the yellow LEDs. Initialize the CPC700 Bank 4 parameters to permit access to the SDRAM. This includes enabling read-modify-write mode, if ECC RAM is present on the board (although ECC correction is not enabled until later).

8 Check the console port for a key press. If a “d” key is pressed, start the debugger. If an “s” key is pressed, skip the diagnostics, **nvopen**, and **configboard**, and use the default NVRAM fields. If no key is pressed, or if any other key is pressed, then read NVRAM fields to determine if diagnostics should be executed and if ECC should be enabled.

9 Turn off the LED display and print the monitor version number.

10 Enable the L1 instruction cache.

11 If skipping the diagnostics and ECC is either not present or not enabled, go to step 19. If skipping the diagnostics but ECC is enabled, go to step 18. Otherwise, continue on to diagnostics.

12 Print a test number in hexadecimal. Print the memory size (read from Board Configuration register 0).

13 Check timebase timer function. The counter/timer test flag (Table 9-1) reports failures. If an error occurs, start the debugger.

14 Write and read locations $0004,0000_{16}$ and $0004,0004_{16}$ with the data pattern $0505,0A0A_{16}$ and its complement. The DRAM data test flag (Table 9-1) reports failures. If a failure occurs, the monitor displays the failed address, followed by the incorrect and expected data, and then starts the debugger.

15 Perform a rotating bit test on all address boundaries with ECC still disabled. Then, initialize these address boundaries by writing each long word with its own address. The DRAM data test flag (Table 9-1) reports failures. If a failure occurs, the monitor displays the failed address, followed by the incorrect and expected data, and then starts the debugger.

16 If ECC RAM is installed and ECC is requested (by the NVRAM field), test the ability to detect ECC errors. With ECC enabled, write a value to address $0004,0001_{16}$. Then with ECC turned off, modify the data, leaving the ECC code unchanged. Re-enable ECC and read the value. If there is no ECC error, start the debugger.

17 Perform a rotating bit test on the first 40000_{16} of memory required by the monitor. If a data error occurs, start the debugger. If an ECC error occurs, the test displays the error and continues the test with ECC disabled.

If ECC RAM is installed and ECC is requested, enable ECC checking. Then, initialize the lower 40000_{16} of memory by writing each long word with its own address. Verify the data written. The DRAM data test flag (Table 9-1) reports failures. If a data error occurs, the monitor displays the failed address, followed by the incorrect and expected data, and then starts the debugger. If an ECC error occurs, the test displays the error and continues the test with ECC disabled.

Note: *Even when diagnostics are disabled, this test is performed if ECC is enabled in the NVRAM fields. To avoid erasing memory after a reset, disable both the diagnostics and ECC in the NVRAM fields.*

18 Initialize at system level to set up for running compiled C code. Enable machine checks in the CPC700 and initialize BSS. Relocate the dynamic data section from ROM to its linked address space starting at $0000,2000_{16}$. Initialize the stack pointer to $0001,FFF8_{16}$.

19 If an "s" key was not pressed on the serial port, load NVRAM data into memory and configure board according to the NVRAM fields. If NVRAM is invalid or the monitor detected a pressed key, load the default fields (see Table 9-3) into memory. In either case, the actual NVRAM contents are not changed and may be edited with **nvdisplay**, followed by **nvupdate**.

Finally, configure the serial port with the fields that were loaded into memory.

20 Initialize the RAM-based interrupt vector table. Change the interrupt prefix to point to the RAM-based interrupt table at $0000,0000_{16}$. Initialize the CPC700 error registers, interrupt handler table, and timebase register.

Store the results of the diagnostics at an offset of 78_{16} in NVRAM. To read the PASS/FAIL flags, do four byte reads from the NVRAM at 78_{16}, 79_{16}, $7A_{16}$, and $7B_{16}$. The byte at 78_{16} should contain the magic number $A5_{16}$ indicating that the device is functional and that PASS/FAIL reporting is supported. The values for the long word when a failure occurs are listed in Table 9-1.

Table 9-1: *Power-up Diagnostic PASS/FAIL Flags*

Device:	Read on Failure (hex):	Monitor Test Command:
Serial Ports	A5000001	**serialtest**
Real Time Clock	A5000004	–
Cache	A5000010	**cachetest**
NVRAM	A5000020	**nvramtest**
Ethernet Port	A5000080	**ethertest**
DRAM ECC	A5000200	–
DRAM Data	A5000400	–

21 Initialize the free memory pool.

22 Execute the configboard function if the “s” key was not pressed as follows: First, configure the serial ports according to the NVRAM fields. Next, configure and enable the MMUs and set up the L1/L2 caches, as determined by NVRAM. Then, if diagnostics are enabled in NVRAM or if the board contains ECC RAM (and ECC is enabled in NVRAM), perform an address mirror test on system memory above address 40000_{16}. Next, set up and enable the thermal protect interrupt. Finally, if DoPCIConfig is set in NVRAM and the board is operating in Monarch mode, map the base addresses of PCI devices.

23 Initialize the CPC700 Bank 1 parameters to allow access to the entire StrataFlash in one contiguous block, beginning at address $7000,0000_{16}$. (If booting from StrataFlash, the last eight megabytes of it also are visible at address $FF80,0000_{16}$.)

24 Initialize the CPC700 Bank 3 parameters to allow access to the Ethernet controller.

25 If the ResetDiags flag is set in NVRAM, execute and display the results of the following reset diagnostics on the console: NVRAM, cache, serial ports, and Ethernet controller. The NVRAM flag (Table 9-1) also reports any failures.

26 Branch execution to StartMonitor, which checks the boot device.

27 If a boot device (BootDev) is specified, begin the countdown to autoboot. After the countdown, boot from the selected device. If boot device is “none”, or the user interrupts the countdown by pressing “H”, or any diagnostic tests fail and the HaltOnFailure NVRAM boot field is set, skip the autoboot and start the line editor.

Figure 9-2: *Monitor Startup Flowchart (1 of 4)*

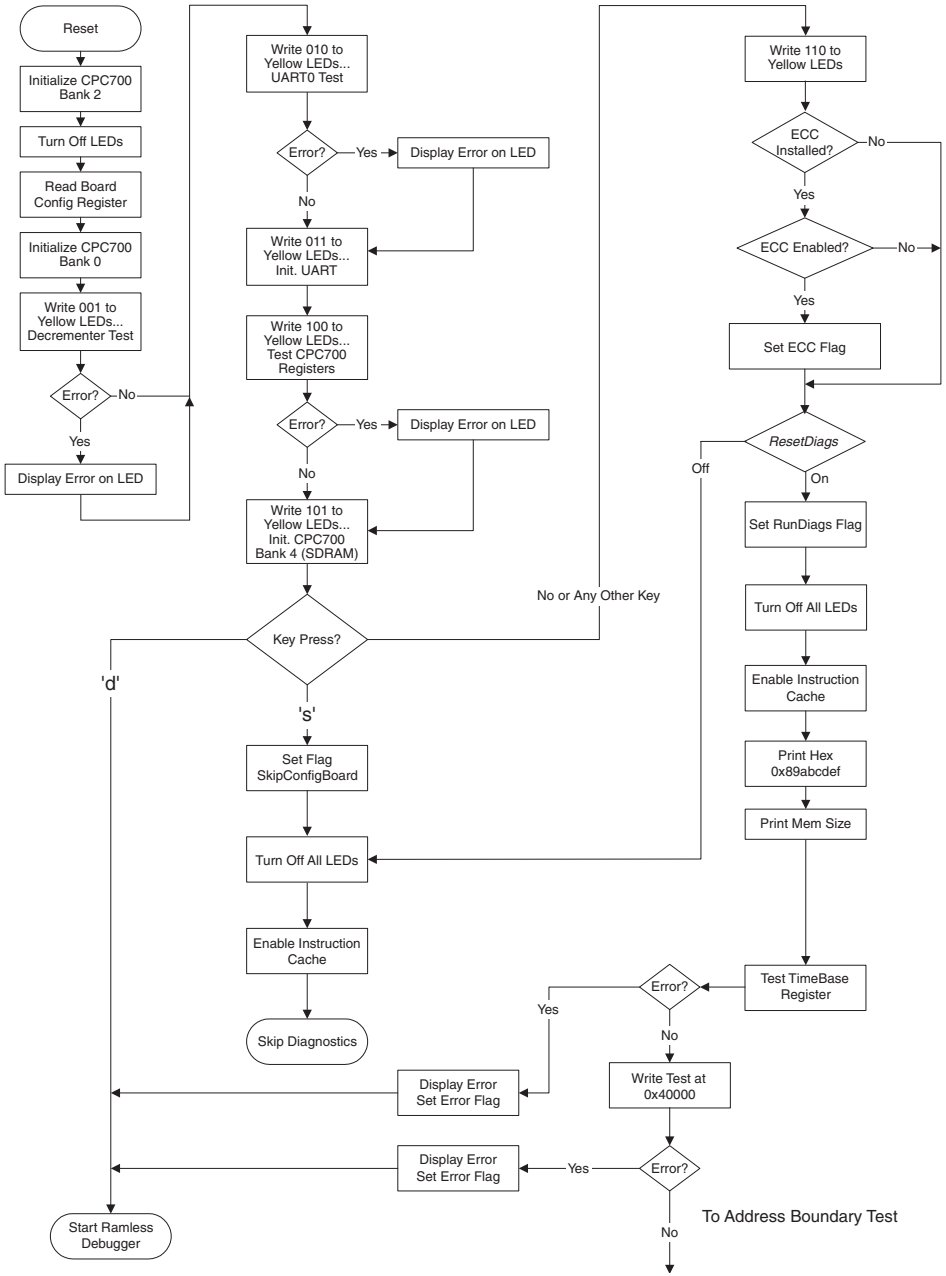


Figure 9-3: *Monitor Startup Flowchart (2 of 4)*

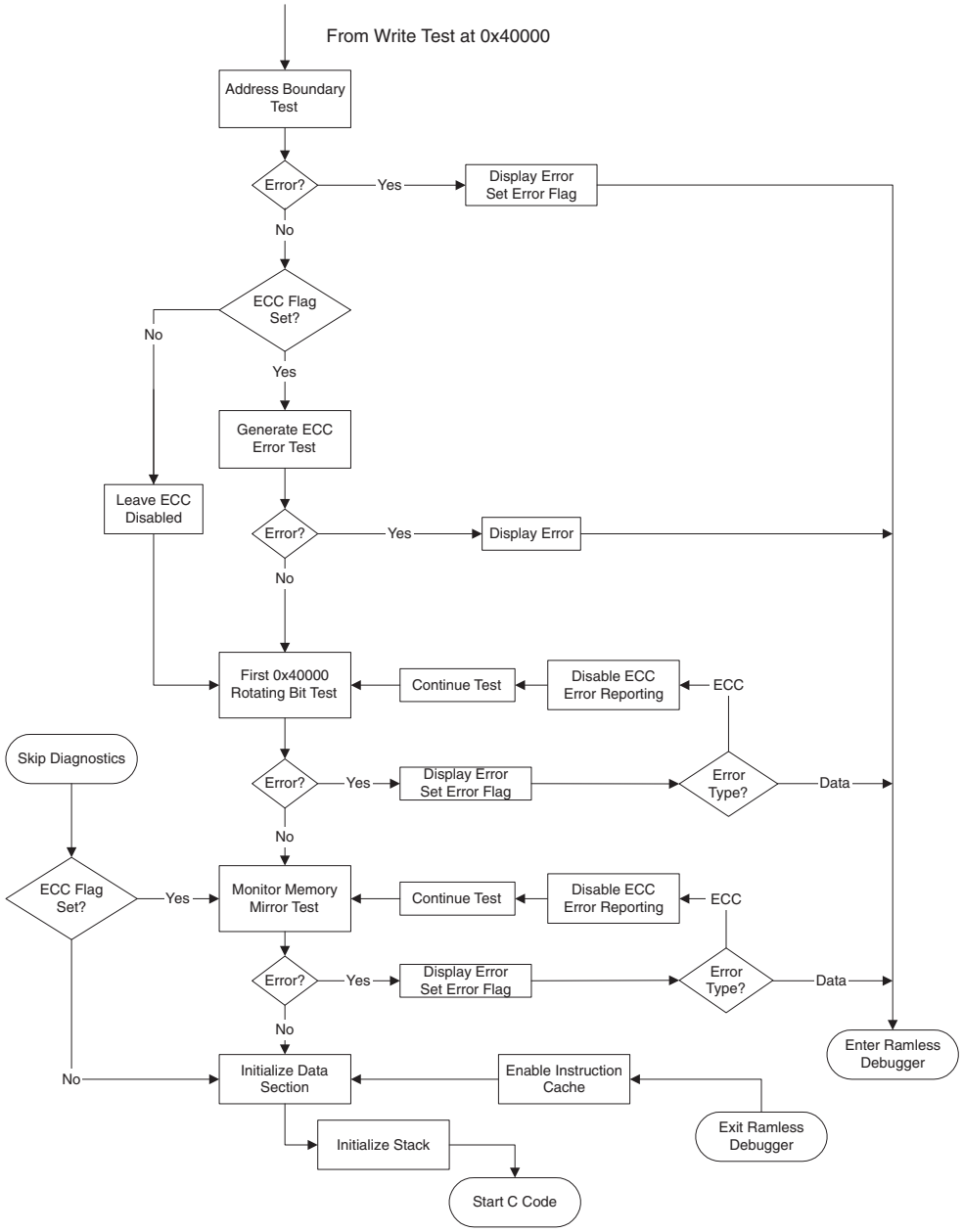


Figure 9-4: *Monitor Startup Flowchart (3 of 4)*

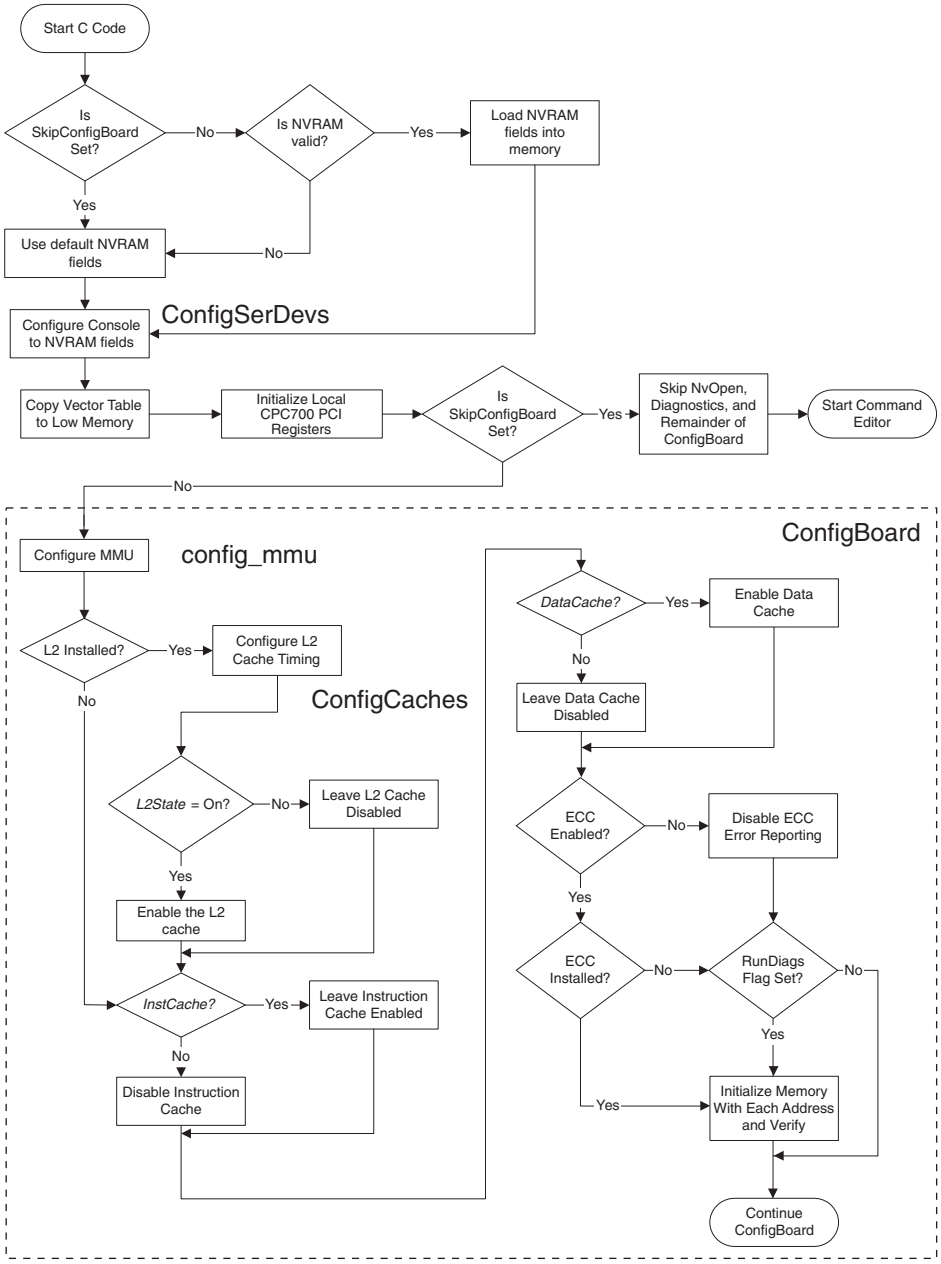
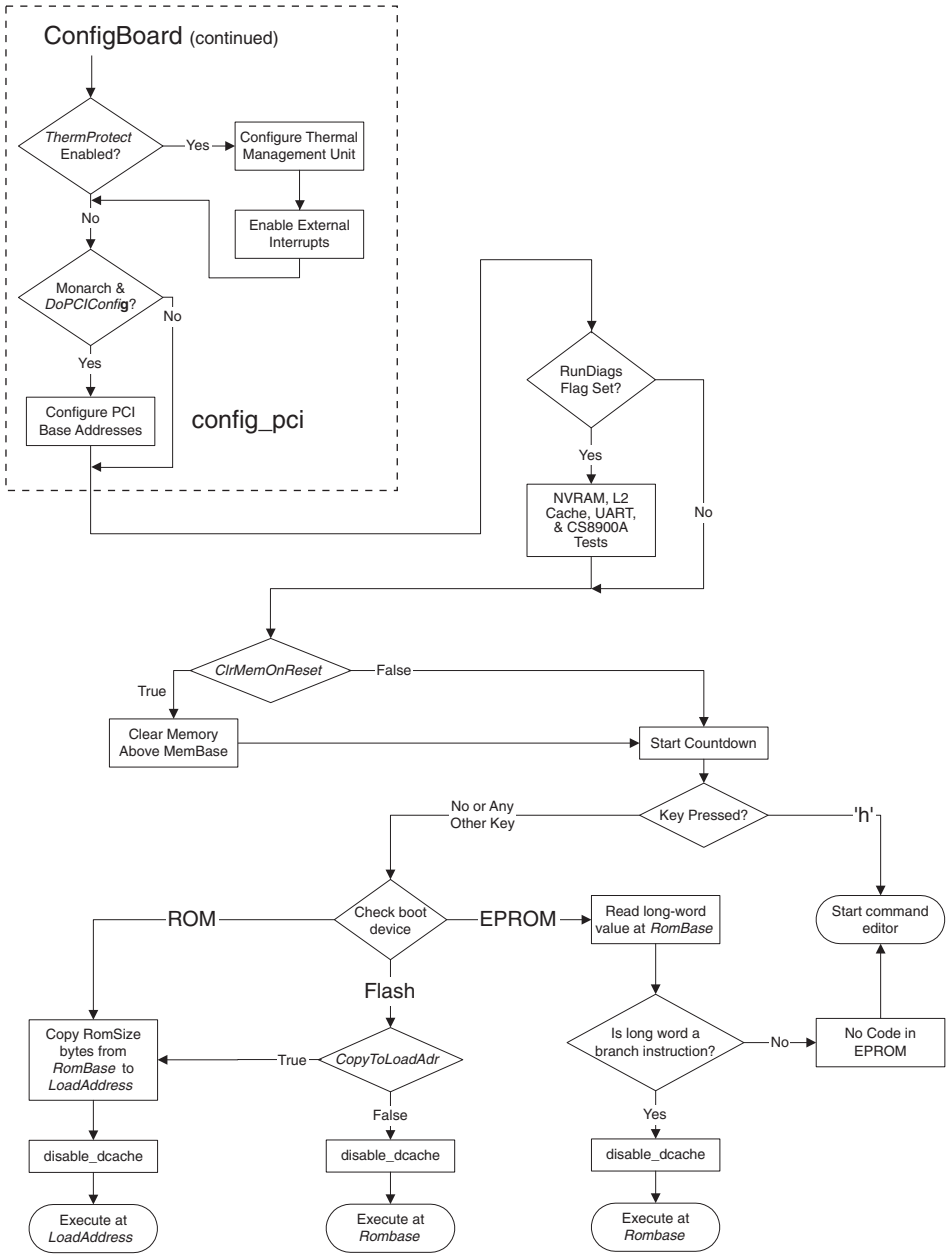


Figure 9-5: *Monitor Startup Flowchart 4 of 4)*



Initializing Memory

The monitor uses the area between $0000{,}0000_{16}$ and $0003{,}0000_{16}$ for stack and uninitialized-data space.

Caution: **Any writes to that area can cause unpredictable operation of the monitor.**

The monitor initializes the on-board memory by writing each long word with its own address to prevent subsequent ECC errors. If the NVRAM field *ClrMemOnReset* is set, the monitor will then clear the memory pool. It is left up to the programmer to initialize any other accessible memory areas, such as off-card or module memory.

MONITOR COMMAND REFERENCE

This section describes the syntax and typographic conventions for the PmPPC monitor commands. Subsequent sections in this chapter describe the individual commands, which fall into the following categories: boot, memory, flash, NVRAM, test, remote host, arithmetic, and other commands.

Note: *The PmPPC monitor performs argument checking for commands, but not for functions. (See page 9-38 for function reference.)*

Command Syntax

Each command may be typed with the shortest number of characters that uniquely identify the command. For example, you can type **nvd** instead of **nvdisplay**. (There is no distinction between uppercase and lowercase.) Note, however, that abbreviated command names cannot be used with on-line help; you must type **help** and the full command name. Press Enter or Return (carriage return <cr>) to execute a command.

- The command line accepts three argument formats: string, numeric, and symbolic. Arguments to commands must be separated by spaces.
- Monitor commands that expect numeric arguments assume a default base for each argument. However, the base can be altered or specified by entering a colon (:) followed by the base. Several examples are provided below.

hexadecimal: `1234ABCD:16`

decimal: `123456789:10`

octal: `1234567:8`

binary: `101010:2`

- The default numeric base for functions is hexadecimal. Some commands use a different default base.

- String arguments must start and end with double quotation marks (“). For example, typing the argument “Foo” would result in a string argument with the value Foo, which is passed to the command.
- A character argument is a single character that begins and ends with a single quotation mark (‘). The argument ‘A’ would result in the character A being passed to the command.
- A flag argument is a single character that begins with a hyphen (-). For example, the flag arguments -b, -w, or -l could be used for a byte, word, or long flag.

There is a symbol entry for every function and command defined in the monitor. Each command must begin with a symbol. Commands are type-checked and argument-validated, but functions are not checked in any way.

Commands that are not symbolic are assumed to be numeric, and the hexadecimal, decimal, octal, and binary value of the number is printed.

Typographic Conventions

Text formatted in `Courier` indicates a command example or a direct value that you enter. Square brackets [] enclose optional arguments, and angled brackets < > enclose required arguments. *Italic* type indicates an environment variable or a field that requires input.

BOOT COMMANDS

The boot commands provide facilities for booting application programs from various devices. They disable the data cache before calling the application.

bootbus

The **bootbus** is an autoboot device that allows you to boot an application program over a bus interface. This command is used for fast downloads to reduce development time.

Definition: `void BootBus (void)`

bootbus the *LoadAddress* field from the nonvolatile memory definitions group ‘BootParams’ (see Table 9-3) as the base address of a shared memory communications structure, described below:

```
struct BusComStruct
{
    unsigned long MagicLoc;
    unsigned long CallAddress;
};
```

The structure consists of two unsigned long locations. The first is used for synchronization, and the second is the entry address of the application.

The sequence of events used for loading an application is described in the following steps:

1. The host board waits for the target (this board) to write the value 0x496d4f6b (character string “ImOk”) to “MagicLoc” to show that the target is initialized and waiting for a download.
2. The host board downloads the application to the target board, writes the start address to “CallAddress,” and then writes 0x596f4f6b (character string “YoOk”) to “MagicLoc” to show that the application is ready for the target.
3. Target writes value 0x42796521 (character string “Bye!”) to “MagicLoc” to show that the application was found. The target then calls the application at “CallAddress.”
4. When the application is called, four fields are passed to the application from the nonvolatile memory boot configuration section. The fields are seen by the application as shown below:

```
Application(unsigned char Device,
            unsigned char Number,
            unsigned long RomSize,
            unsigned long RomBase)
```

These fields allow multiple boards using the same facility to receive configuration information from the monitor.

Also refer to the function **BootUp** in the “Booting”, Section .

booteprom

booteprom is an autoboot device that allows you to boot an application program from EPROM. It starts execution of the application at *RomBase*, read from the non-volatile memory group ‘BootParams.’

Definition: `void BootEPROM(void)`

In order for the monitor to jump to the start of the program, the first long word of the EPROM image must contain a branch link (**bl**) instruction of the form 0100,10xx,xxxx,xxxx,xxxx,xxxx,xxxx,xx01$_2$.

You can avoid jumping to an EPROM, even if a valid one is present, by changing the nonvolatile configuration field *BootDev* to something other than EPROM.

Also refer to the function **BootUp** in the “Booting”, Section .

bootrom

bootrom is an autoboot device that allows you to boot an application program from ROM. It copies code from ROM into RAM and then jumps to the RAM address. The ROM source address *RomBase*, the RAM destination address *LoadAddress*, and the number of bytes to copy *RomSize* are read from the nonvolatile memory group 'BootParams.'

Definition:

```
void BootROM(void)
```

When the application is called, two fields are passed to the application from the nonvolatile memory group 'BootParams.' The fields are seen by the application as shown below:

```
Application(unsigned char Device,
            unsigned char Number)
```

There are no arguments for this command. The nonvolatile configuration is modified with the NVRAM commands **nvdisplay** and **nvupdate**.

Also refer to the function **BootUp** in the "Booting", Section .

bootflash

bootflash is an autoboot device that allows you to boot an application program from the on-board flash.

If the *CopyToLoadAdr* field is true, the *RomSize* (number of bytes to copy) of the application code is copied from the *RomBase* (flash source address) to the *LoadAddress* (RAM destination address). The application is called at *LoadAddress*.

If the *CopyToLoadAdr* field is false, then the application code is called at *RomBase*.

Definition:

```
void BootFlash(void)
```

When the application is called, two fields are passed to the application from the nonvolatile memory group 'BootParams.' The fields are seen by the application as shown below:

```
Application(unsigned char Device,
            unsigned char Number)
```

There are no arguments for this command. The nonvolatile configuration is modified with the NVRAM commands **nvdisplay** and **nvupdate**.

Also refer to the function **BootUp** in the "Booting", Section .

bootserial

bootserial is an autoboot device that allows you to boot an application program from a serial port.

Definition:

```
void BootSerial(void)
```

It determines the format of the download and the entry execution address of the downloaded application from the *LoadAddress* and *DevType* fields in the nonvolatile memory group 'BootParams.' The *DevType* field selects one of the download formats specified in Table 9-2:

Table 9-2: *Device Download Formats*

Device Type:	Download Format:
MOT_EXORMAT 1	Motorola Exormax Format (S0-S3,S7-S9 Records)
HK_BINARY 2	Emerson Binary Format

The nonvolatile configuration is modified with the NVRAM commands **nvdisplay** and **nvupdate**.

When the application is called, three fields are passed to the application from the nonvolatile memory boot configuration section. The fields are seen by the application as shown below:

```
Application(unsigned char Number,
            unsigned long RomSize,
            unsigned lone RomBase)
```

These fields allow multiple boards using the same facility to receive different configuration information from the monitor.

Also refer to the function **BootUp** in the "Booting", Section .

boottftp

boottftp is an autoboot device that allows you to transfer an application to RAM using the TFTP functions, then boot the application.

Definition:

```
void BootTFTP(void)
```

boottftp uses the fields *BoardIPAddr*, *HostIPAddr*, *GatewayIPAddr*, and *GatewayMask*, as defined in the nonvolatile memory group 'Network,' to set up a connection with the application's host. (The application should be in appropriate binary format.) Also, **boottftp** uses *FileName* and *LoadAddress* in the memory group 'BootParams' to determine which application to retrieve, as well as where to load it.

After the successful TFTP transfer to memory, the application is called and receives three prameters from the nonvolatile memory group 'BootParams' as follows:

```
Application(unsigned char Number,
            unsigned long RomSize,
            unsigned long RomBase)
```

These fields allow multiple boards using the same facility to receive different configuration information from the monitor.

Also refer to the function **BootUp** in the “Booting”, Section .

bootverify

bootverify is an autoboot device that allows you to validate a boot image and boot from a backup image, if necessary.

Definition: `void BootVerify(void)`

This command is set using the *BootDev* field from the ‘BootParams’ group in the nonvolatile memory. If performs the following sequence to boot code:

1. Compute the checksum for the image found in flash at the address pointed to by PRIMARY_ADDRESS_OFFSET (0x400 in NVRAM) and compare the result to the checksum at PRIMARY_CHKSUM_OFFSET (0x40C in NVRAM).
2. If the checksum comparison shows that the image is good, branch directly to the address at PRIMARY_ADDRESS_OFFSET. However, if the CopyToLoadAddr field in the nonvolatile memory group ‘BootParams’ is `TRUE`, copy the image to the address pointed to by the LoadAddress field, and then branch to that address.
3. If the checksum comparison fails, compute a checksum for the image found in flash at the address pointed to by SECONDARY_ADDRESS_OFFSET and compare the result to the checksum at SECONDARY_CHKSUM_OFFSET.
4. If the secondary checksum passes, branch directly to the address at SECONDARY_ADDRESS_OFFSET. However, if the CopyToLoadAddr field in the nonvolatile memory group ‘BootParams’ is `TRUE`, copy the image to the address pointed to by the LoadAddress field, and then branch to that address.
5. If the secondary checksum does not pass, display an error message on the screen and drop control to the Monitor prompt.

MEMORY COMMANDS

The memory commands provide facilities for manipulating specific regions of the memory. For some memory commands, the data size is determined by the following flags:

-b: The flag .b is for data in 8-bit bytes.

-w: The flag .w is for data in 16-bit words.

-l: The flag .l is for data in 32-bit long words.

These flags are optional arguments and describe the objects on which the command operates. If you do not specify a flag, memory commands default to 32-bit long words. Numeric arguments are in hexadecimal.

checksummem

checksummem *source bytecount* reads *bytecount* bytes starting at address *source* and computes the checksum for that region of memory. The checksum is the 16-bit sum of the bytes in the memory block.

Definition:

```
int CheckSumMem(unsigned char *Addr,
                unsigned long ByteCount)
```

clearmem

clearmem *destination bytecount* clears *bytecount* bytes starting at address *destination*.

Definition:

```
int ClearMem(unsigned char *Dest,
             unsigned long ByteCount)
```

cmpmem

cmpmem *source destination bytecount* compares *bytecount* bytes at the *source* address with those at the *destination* address. Any differences are displayed.

Definition:

```
int CmpMem(char *Src,
           char *Dest,
           int ByteCount)
```

copymem

copymem *source destination bytecount* copies *bytecount* bytes from the *source* address to the *destination* address.

Definition:

```
int CopyMem(unsigned char *Src,
            unsigned char *Dest,
            unsigned long ByteCount)
```

displaymem

displaymem *startaddr lines* displays memory in 16-byte lines starting at address *startaddr*. The number of lines displayed is determined by *lines*. If the *lines* argument is not specified, sixteen lines of memory are shown. The data is displayed as hex character values on the left and printable ASCII equivalents on the right. Unprintable ASCII characters appear as a dot.

Press any key to interrupt the display. If the previous command was **displaymem**, pressing <cr> displays the next block of memory.

Definition:
```
int DisplayMem(unsigned long Address,
               unsigned long Lines)
```

fillmem

fillmem -[b,w,l] *value startaddr endaddr* fills memory with *value* starting at address *startaddr* to address *endaddr*.

For example, to fill the second megabyte of memory with the data 0x12345678 type:

```
fill -l 12345678 100000 200000
```

Definition:
```
int FillMem(char Flag, unsigned long Value,
            unsigned long StartAddr,
            unsigned long EndAddr)
```

findmem

findmem -[b,w,l] *searchval startaddr endaddr* searches memory for a value from address *startaddr* to address *endaddr* for memory locations specified by the data *searchval*. When enabled with a '1', the optional *invflag* parameter causes the command to return locations that do not match *searchval*, providing the same functionality as **findnotmem** (described below).

Definition:
```
int FindMem(char Flag,
            unsigned long SearchVal,
            unsigned long StartAddr,
            unsigned long EndAddr,
            unsigned long InvFlag)
```

findnotmem

findnotmem -[b,w,l] *searchval startaddr endaddr* searches from address *startaddr* to address *endaddr* for memory locations that are different from the data specified by *searchval*.

Definition:
```
int FindNotMem(char Flag,
               unsigned long SearchVal,
               unsigned long StartAddr,
               unsigned long EndAddr)
```

findstr

findstr *searchstr startaddr endaddr* searches from address *startaddr* to address *endaddr* for a string matching the data string *searchstr*.

Definition:
```
int FindStr(char *SearchStr,
            unsigned long StartAddr,
```

```
                unsigned long EndAddr)
```

readmem

readmem -[b,w,l] *address* reads a memory location specified by *address*. This command displays the data in hexadecimal, decimal, octal, and binary format.

Definition:

```
int ReadMem(char Flag,
            unsigned long Address)
```

setmem

setmem -[b,w,l] *address* allows memory locations to be modified starting at *address*. **setmem** first displays the value that was read. Then you can type new data for the value or leave the data unchanged by entering an empty line. If you press <cr> after the data, the address counts up. If you press <ESC> after the data, the address counts down. To quit this command type any illegal hex character.

Definition:

```
int SetMem(int Flag,
           unsigned long Address)
```

swapmem

swapmem *source destination bytecount* swaps *bytecount* bytes at the *source* address with those at the *destination* address.

Definition:

```
int SwapMem(char *Src,
            char *Dest,
            int ByteCount)
```

testmem

testmem *startaddr endaddr* performs a nondestructive memory test from *startaddr* to *endaddr*. If *endaddr* is zero, the address range is obtained from the functions **MemBase** and **MemTop**. The memory test can be interrupted by pressing any character.

This command can be used to verify memory (DRAM). It prints the progress of the test and summarizes the number of passes and failures.

Also refer to the functions **MemBase** and **MemTop** on page 9-48.

Definition:

```
int TestMem(unsigned long Base,
            unsigned long Top)
```

um

um -[b,w,l] *base_addr top_addr* performs a destructive memory test from *base_addr* to *top_addr*. This is done by first clearing all memory in the range specified, doing a rotating bit test at each location, and finally filling each data location with its own address. If *top_addr* is zero, the address range is obtained from the functions **MemBase** and **MemTop**.

This command prints the progress of the test and summarizes the number of passes and failures. The memory test can be interrupted at the start of the next pass by pressing any character.

Also refer to the functions **MemBase** and **MemTop** on page 9-48.

Definition:

```
int UM(char flag,
       unsigned long l_limit,
       unsigned long u_limit)
```

writemem

writemem -[b,w,l] *address value* writes *value* to a memory location specified by *address*.

Definition:

```
int WriteMem(char Flag,
             unsigned long Address,
             unsigned long Value)
```

writestr

writestr *"string" address* writes the ASCII string specified by *string* to a memory location specified by *address*. The string must be enclosed in double quotes (" ").

Definition:

```
int WriteStr(char *Str,
             unsigned long Address)
```

FLASH COMMANDS

The flash commands affect the StrataFlash™ devices on the PmPPC circuit board. They return zero upon successful completion of the operation, or non-zero upon failure.

The flash commands protect the monitor code after it is copied into the flash memory. If jumper JP1 is not installed, attempts to write above $FFF0,0000_{16}$ return an error. (The monitor resides in this area.) If jumper JP1 is installed, the monitor code still can be installed in flash and addressed at one megabyte below the top of flash (base address $7000,0000_{16}$).

Note: *The location for the top of flash memory will vary according to the flash device size and configuration.*

flashblkwr

flashblkwr *source destination bytecount* writes *bytecount* from the *source* address to the *destination* address (flash memory). **flashblkwr** calls **flasheraseblk** to erase the block(s) it will write to. In the event of a write or erase error, the flash is returned to read array mode.

Definition:

```
int FlashBlkWr(unsigned char *Src,
               unsigned char *Dest,
               int ByteCnt)
```

flashclrstat

flashclrstat *destination* resets the status register of the flash memory that contains the *destination* address.

Definition:

```
void FlashClrStat(unsigned char *FlashAddr)
```

flasheraseblk

flasheraseblk *destination* erases a block of flash memory that contains the *destination* address. The block size varies, depending upon its location within the flash.

Definition:

```
int FlashEraseBlk(unsigned char *FlashAddr)
```

flashsocketerase

flashsocketerase erases the memory on the socketed flash device. It is not necessary to pass an address to this command.

Definition:

```
int FlashSocketErase(void)
```

flashsocketblkwr

flashsocketblkwr *source destination bytecount* writes *bytecount* from the *source* address to the *destination* address (socketed flash memory). **flashsocketblkwr** calls **flashsocketerase** to erase the socketed memory before writing. Any data already in the device will be lost.

Definition:

```
int FlashSocketBlkWr(unsigned char *Src,
                     unsigned char *Dest,
                     int ByteCnt)
```

rewritemonitor

rewritemonitor overwrites the monitor software in the on-board soldered flash device with a new monitor image. It asks you for a source address and the size of the new monitor image. Once invoked, the function gives you two opportunities to exit without modifying the on-board flash.

Caution: **This function cannot recover from a failed write to the flash device. If a failure does occur, it cannot print an error and the monitor may no longer be able to boot the board. To recover, you would have to rewrite the monitor from a socketed flash device using the flashblkwr function.**

Definition: `int ReWriteMonitor(void)`

NVRAM COMMANDS

The monitor uses on-board NVRAM for nonvolatile memory. A memory map is given in the "Physical Memory Map", Section , earlier in this manual. Portions of this nonvolatile memory are reserved for factory configuration/identification information and for the monitor.

The nonvolatile memory support commands deal only with the monitor- and Emerson-defined sections of the nonvolatile memory. The monitor-defined sections are readable and writable and can be modified by the monitor.

Caution: **When clearing NVRAM, interrupt may be held for 2 µS.**

nvdisplay

nvdisplay is used to display the Emerson-defined and monitor-defined nonvolatile sections. The nonvolatile memory configuration information is used to completely configure the PmPPC at reset. The utility command **configboard** can also be used to reconfigure the board after modifications to the nonvolatile memory.

Definition: `void NVDisplay(void)`

The configuration values are displayed in groups. Each group has a number of fields. Each field is displayed as a hexadecimal or decimal number, or as a list of legal values.

To display the next group, press <space> or <cr>.

To edit fields within the displayed group, press E.

To quit the display, press <ESC> or Q.

To save the changes, type the command **nvupdate**.

To quit without saving the changes, type the command **nvopen**.

Table 9-3 shows all the groups and fields you can edit with the **nvdisplay** command.

Table 9-3: *NVRAM Configuration Groups*

Group:	Field:	Purpose:	Emerson Default:	Optional Values:
Console				
	Port	Select communications port	A (Console)	(A, B)

Group:	Field:	Purpose:	Emerson Default:	Optional Values:
	Baud	Select baud rate	9600	(1200, 2400, 4800, 9600, 19200, 38400, 56000, 128000 bps)
	Parity	Select parity type	None	(Even, Odd, None, Force)
	Data	Select the number of data bits for transfer	8-Bits	(5-Bits, 6-Bits, 7-Bits, 8-Bits)
	StopBits	Select the number of stop bits for transfer	1-Bit	(1-Bit, 2-Bits)
Download				
	Port	Select communications port	B (Download)	(A, B)
	Baud	Select baud rate	9600	(1200, 2400, 4800, 9600, 19200, 38400, 56000, 128000 bps)
	Parity	Select parity type	None	(Even, Odd, None, Force)
	Data	Select the number of data bits for transfer	8-Bits	(5-Bits, 6-Bits, 7-Bits, 8-Bits)
	StopBits	Select the number of stop bits for transfer	1-Bit	(1-Bit, 2-Bits)
Cache				
	InstrCache	Turn instruction cache on or off	On	(On, Off)
	DataCache	Turn data cache on or off	On	(On, Off)
	CacheMode	Select the cache mode	Copyback	(Writethru, Copyback)
	L2State	Enable or disable L2 Cache	On	(On, Off)
Misc				
	ClrMemOnReset	Clear memory on reset	False	(True, False)
	ResetDiags	Run diagnostics on reset	Off	(On, Off)
	ECCCorrection	Request memory ECC (no effect if ECC is not installed)	On	(On, Off)
	ThermProtect	Put CPU in sleep mode when its temperature exceeds 100° C	Off	(On, Off)
	CountValue	Choose shortest (0) to longest (7) duration for autoboot countdown	1	(0, 1, 2, 3, 4, 5, 6, 7)
	DoPCIConfig	Configure module (Monarch boards only)	True	(True, False)
Network				
	BoardIPAddr	IP address of board	0.0.0.0	$x.x.x.x$; where $0 \leq x \leq 255$
	HostIPAddr	IP address of host	0.0.0.0	$x.x.x.x$; where $0 \leq x \leq 255$
	GatewayIPAddr	IP address of gateway	0.0.0.0	$x.x.x.x$; where $0 \leq x \leq 255$
	GatewayMask	Gateway mask	0.0.0.0	$x.x.x.x$; where $0 \leq x \leq 255$

Group:	Field:	Purpose:	Emerson Default:	Optional Values:
	DoEtherInit	Initialize network interface upon boot	False	(True, False)
BootParams				
	BootDev	Select boot device	EPROM	(None, Serial, ROM, Bus, Stos, EPROM)
	LoadAddress	Define load address	0x40000	See *User Manual*
	RomBase	Define ROM base	0x7ff80000	Used only when BootDev is defined as ROM or EPROM
	RomSize	Define ROM size	0x80000	Used only when BootDev is defined as ROM
	DevType	Define device type	0	Depends on the application
	DevNumber	Define device number	0	Depends on the application
	FileName	Define file name and path	null	See boottftp function
	ClrMemOnBoot	Clear memory on boot	False	(True, False)
	HaltOnFailure	Halt if a failure occurs	True	(True, False)
	CopyToLoadAddr	Copies from RomBase to LoadAddr (Boot flash only)	False	(True, False)
HardwareConfig, Manufacturing, Service				
	Reserved for use by Emerson Network Power, Embedded Computing manufacturing.			

Example:

1. At the monitor prompt, type:

```
PmPPC{Ver 1.0] => nvdisplay
```

2. Press <cr> until the group you want to modify is displayed. An example for the group "Console" is shown below.

```
Group 'Console'
PortA(A, B)
Baud9600
ParityNone(Even, Odd, None, Force)
Data8-bits(5-Bits, 6-Bits, 7-Bits, 8-Bits)
StopBits2-bits(1-Bit, 2-Bits)
[SP, CR to continue] or [E, e to Edit]
```

3. Press E to edit the group.
4. Press <cr> until the field you want to change is displayed.
5. Type a new value. For most fields, legal options are displayed in parentheses.
6. Press <ESC> or Q to quit the display.

7 Type **nvupdate** to save the new value or **nvopen** to cancel the change by reading the old value.

nvinit

nvinit *sernum "revlev" ecolev writes* is used to initialize the nonvolatile memory to the default state defined by the monitor. First **nvinit** clears the memory and then writes the Emerson and monitor data back to memory. It also saves the changes in NVRAM.

Caution: **nvinit clears any values you have changed from the default. Use nvinit only if the nonvolatile configuration data structures might be in an unknown state and you must return them to a known state.**

sernum	serial number
revlev	revision level
ecolev	standard ECO level
writes	the number of writes to nonvolatile memory

Definition:

```
void NVInit(int SerNum,
            char *RevLev,
            int ECOLev,
            int Writes)
```

nvopen

nvopen reads and checks the monitor and Emerson-defined sections. If the nonvolatile sections are not valid, an error message is displayed.

Definition:

```
int NVOpen(void)
```

nvset

nvset *group field value* is used to modify the Emerson-defined and monitor-defined nonvolatile sections. To modify the list with the **nvset** command, you must specify the group and field to be modified and the new value. The group, field, and value can be abbreviated, as in the following examples:

```
nvset console port A

nvset con dat 6
```

The nonvolatile memory support commands provide the interface to the nonvolatile memory. The nonvolatile commands deal only with the monitor- and Emerson-defined sections of the nonvolatile memory. The monitor-defined sections of nonvolatile memory are readable and writable and can be modified by the monitor. The Emerson-defined section of nonvolatile memory is also readable and writable, but should not be modified.

Definition:

```
void NVSet(char *GroupName,
           char *FieldName,
           char *Value)
```

nvupdate

nvupdate attempts to write the Emerson- and monitor-defined nonvolatile sections back to the NVRAM device. First the data is verified, and then it is written to the device. The write is verified and all errors are reported.

Definition:

```
void NVUpdate(void)
```

Default Boot Device Configuration Example

The default boot device is defined in the nonvolatile memory group 'BootParams,' in the field *BootDev*. When the PmPPC is reset or powered up, the monitor checks this field and attempts to boot from the specified device.

Currently, the monitor supports Serial, ROM, Bus, EPROM, flash, Stos, and TFTP as standard. If you edit the *BootDev* field and define a device that is unsupported on your board, the monitor will display the message:

```
Unknown boot device
```

Defining *BootDev* as: "Serial" calls **bootserial**, "ROM" calls **bootrom**, "Bus" calls **bootbus**, "EPROM" calls **booteprom**, "flash" calls **bootflash**, "Stos" calls **stos_boot**, and "TFTP" calls **boottftp**. See the "Boot Commands", Section for details on these commands.

Example: In this example, **nvdisplay** and **nvupdate** are used to change the default boot device from the bus to the ROM. The changes are made to the 'BootParams' group.

Note: *The fields in the 'BootParams' group have different meanings for each device. For example, "DevType" values are not used for Bus devices, but are used by Serial devices to select the format for downloading.*

1 At the monitor prompt, type:

```
PmPPC{Ver 1.0] => nvdisplay
```

2 Press <cr> until the 'BootParams' group is displayed.

```
Group 'BootParams'
BootDevBus(None,Serial,ROM,Bus,EPROM,Flash,Stos,TFTP)
LoadAddress0x40000
```

```
ROMBase0x7ff80000
ROMSize0x80000
DevType1
DevNumber0
ClrMemOnBootFalse(False, True)
[SP, CR to continue] or [E, e to Edit]
```

3 Press E to edit the group.

4 Press <cr> until the BootDev field is displayed.

5 Type the new value “ROM.”

6 Press <cr> to display the LoadAddress field.

7 Type the address where execution begins.

8 Press <cr> to display the ROMBase field.

9 Type the ROM base address.

10 Press <cr> to display the ROMSize field.

11 Type the ROM size.

12 Press <ESC> or Q to quit the display.

13 Type **nvupdate** to save the new values.

Example: In this example, **nvdisplay** and **nvupdate** are used to change the default boot device from the bus to the serial port. The changes are made to the ‘BootParams’ group.

1 At the monitor prompt, type:

```
PmPPC{Ver 1.0] => nvdisplay
```

2 Press <cr> until the ‘BootParams’ group is displayed.

3 Press E to edit the group.

4 Press <cr> until the DevType field is displayed.

5 Type the new value “Serial.”

6 Press <cr> until the DevType field is displayed.

7 Type the new value for DevType; for example, 2 selects downloads in Emerson binary format.

8 Edit any other fields you want to modify. Whether you use the DevType and DevNumber fields depends on the application.

9 Press <ESC> or Q to quit the display.

10 Type **nvupdate** to save the new values.

Download Port Configuration Example

In this example, the NVRAM command **nvdisplay** changes fields in the 'Download' group, which contains fields for port selection, baud rate, parity, number of data bits, and number of stop bits:

1 At the monitor prompt, type:

```
PmPPC{Ver 1.0] => nvdisplay
```

2 Press <cr> until the 'Download' group is displayed.

3 Press E to edit the group.

4 Press <cr> until the Baud field is displayed.

5 Type a new value.

6 Change other fields in the same way.

7 <cr> over all fields whether you edit them or not, until the monitor prompt reappears.

8 Type **nvupdate** to save the new value.

Note: *A cable reverser might be necessary for the connection.*

TEST COMMANDS

The following on-card functional tests are available to be run any time you desire. The non-volatile configuration memory can be used to enable or disable the execution of these tests on power-up and reset (see the **nvdisplay** monitor command's Misc group in Table 9-3).

The results of the tests are stored at an offset of 0x78 in NVRAM. To read the PASS/FAIL flags, do four byte reads from the ROM at 0x78, 0x79, 0x7A, and 0x7B. The byte at 0x78 should contain the magic number 0xA5 indicating that the device is functional and that PASS/FAIL reporting is supported. The values for the long word when a failure occurs are listed in Table 9-4.

Table 9-4: *Test Command PASS/FAIL Flags*

Test:	Value Red on Failure (hex):	Monitor Command:
Console Serial Port	A5000001	**serialtest**
Cache	A5000010	**cachetest**
NVRAM	A5000020	**nvramtest**
Ethernet Port	A5000080	**ethertest**

ethertest

ethertest checks all the logic necessary to interface the Ethernet controller as follows:

- Performs a data line test to ensure that all data line connectivity is intact.
- Verifies that data can be transferred successfully with the CS8900A in loopback mode.

Definition: `void ethertest(void)`

serialtest

serialtest verifies that data can be transmitted and received by the console and download ports. This test operates in internal loopback mode and simultaneously tests serial interrupts.

Definition: `void serialtest(void)`

nvramtest

nvramtest performs a non-destructive write test at offset $7FF_{16}$ in NVRAM. The NVRAM test command pass/fail flag (Table 9-4) reports any errors.

Definition: `void nvramtest(void)`

cachetest

cachetest verifies the connectivity of address and data lines to the CPU's backside L2 cache. During the test, modified blocks are not cast out to system memory and instructions are not cached.

The test fills the L2 cache with a one-megabyte block of addresses, then performs a rotating bit and address mirror test on all addresses in the test block. After invalidating the contents of the cache, it tests the L2 address tags stored on the PPC750 processor. For unique tag entries, it writes and verifies a series of rotating-one addresses, rotating-zero addresses, and alternating one and zero addresses. The L2 synchronous RAM parity is disabled during the test, and the test restores the L2 cache to the original state when it is finished.

Definition: `int cachetest(void)`

REMOTE HOST COMMANDS

The monitor commands **download** and **call** are used for downloading applications and data in S-record or binary format and then running the application.

S-record is a common format for representing binary object code as ASCII for reliable and manageable file downloads. It sends data in blocks called records, which are ASCII strings. Records may be separated by any ASCII characters except for the start-of-record "S" charac-

ters. In practice, records are usually separated by a convenient number of carriage returns, linefeeds, or nulls to separate the records in a file and make them easily distinguishable by humans.

All records contain fields for the length of the record, the data in the record, and some kind of checksum. Some records also contain an address field. Most software requires the hexadecimal characters that make up a record to be in uppercase only.

download

download -[b,m] *address* provides a serial download from a host computer to the board. **download** uses binary or Motorola S-record format, as specified by the following flags:

-b: binary (*address* not used)

-m: Motorola S-record (load address in memory = *address* + record address)

If no flag is specified, the default format is binary

Refer to “Download Port Configuration Example”, Section for an example of how to configure the download port using NVRAM commands. The “Binary Download Format”, Section and “Motorola S-Record Download Format”, Section describe the download formats in detail.

Definition:

```
int DownLoad(char Flag,
             unsigned long Address)
```

call

call *address arg0 arg1 arg2 arg3 arg4 arg5 arg6 arg7* allows execution of a program after a download from one of the board’s interfaces. This command allows up to eight arguments to be passed to the called *address* from the command line. Arguments can be symbolic, numeric, characters, flags, or strings. The default numeric base is hexadecimal. The function disables and flushes the data cache before branching to the application.

If the application wants to return to the monitor, it should save and restore the processor registers. Also, it is important that special-purpose registers remain unchanged.

Note: *The code at vector table locations 300_{16}, 1100_{16}, and 1200_{16} must remain intact, unless you wish to disable or reconfigure the data MMU for other user-defined purposes.*

Definition:

```
int Call(int (*Funct) (),
         unsigned long Arg0,
         unsigned long Arg1... )
```

Binary Download Format

The binary download format consists of two parts:

- Magic number (which is 0x12345670) + number of sections
- Information for each section including: the load address (unsigned long), the section size (unsigned long), a checksum (unsigned long) that is the long-word sum of the memory bytes of the data section.

If you download from a UNIX host in binary format, be sure to disable the host from mapping carriage return <cr> to carriage return line feed <cr-lf>. The download port is specified in the nonvolatile memory configuration.

Motorola S-Record Download Format

S-records are named for the ASCII character "S," which is used for the first character in each record. After the "S" character is another character that indicates the record type. Valid types are 0, 1, 2, 3, 5, 7, 8, and 9. After the type character is a sequence of characters that represent the length of the record, and possibly the address. The rest of the record is filled out with data and a checksum.

The checksum is the one's complement of the 8-bit sum of the binary representation of all elements of the record except the S and the record type character. In other words, if you sum all the bytes of a record except for the S and the character immediately following it with the checksum itself, you should get FF_{16} for a proper record.

User-Defined (S0)

```
S0nnd1d2d3...dncs
S0indicates the record type.
nnis the count of data and checksum bytes.
d1...dnare the data bytes.
csis the checksum.
```

S0 records are optional, and can contain any user-defined data.

Example:

```
S008763330627567736D
```

In this example, the length of the field is 8, and the data characters are the ASCII representation of "v30bugs." The checksum is $6D_{16}$.

Data Records (S1, S2, S3)

```
S1nnaaaad1d2d3...dncs
S2nnaaaaaad1d2d3...dncs
S3nnaaaaaaaad1d2d3...dncs
```

S1 indicates the record type.

nn is the count of address, data, and checksum bytes.

a...a is a 4-, 6-, or 8-digit address field.

d1...dn are the data bytes.

cs is the checksum.

These are data records. They differ only in that S1-records have 16-bit addresses, S2-records have 24-bit addresses, and S3-records have 32-bit addresses.

Examples:

```
S10801A00030FFDC95B6
```

In this example, the bytes 00_{16}, 30_{16}, FF_{16}, DC_{16}, and 95_{16} are loaded into memory starting at address $01A0_{16}$.

```
S30B30000000FFFF5555AAAAD3
```

In this example, the bytes FF_{16}, FF_{16}, 55_{16}, 55_{16}, AA_{16}, and AA_{16} are loaded into memory starting at address $3000,0000_{16}$. Note that this address requires an S3-record because the address is too big to fit into the address range of an S1-record or S2-record.

Data Count Records (S5)

S5*nnd1d2d3...dncs*

S5 indicates the record type.

nn is the count of data and checksum bytes.

d1...dn are the data bytes.

cs is the checksum.

S5-records are optional. When they are used, there can be only one per file. If an S5-record is included, it is a count of the S1-, S2-, and S3-records in the file. Other types of records are not counted in the S5-record.

Example:

```
S5030343B6
```

In this example, the number of bytes is 3, the checksum is $B6_{16}$, and the count of the S1-records, S2-records, and S3-records in the file is 343_{16}.

Termination and Start Address Records (S7, S8, S9)

S705*aaaaaaaacs*

S804*aaaaaacs*

S903*aaaacs*

S7, S8, or S9 indicate the record type.

05, 04, 03 is the count of address digits and *cs* field.

a...a is a 4-, 6-, or 8-digit address field.

cs is the checksum.

These are trailing records. There can be only one trailing record per file, and it must be the last record in the output file. Included in the data for this record is the initial start address for the downloaded code.

Examples:

```
S903003CC0
```

In this example, the start address is $3C_{16}$.

```
S8048000007B
```

In this example, the start address is 800000_{16}.

Example: Complete S-record File

```
S0097A65726F6A756D707A
S10F00000000100000000084EFAFFFE93
S5030001FB
S9030008F4
```

Here is a line-by-line explanation of the example file:

`S0097A65726F6A756D707A` contains the ASCII representation of the string “zerojump.”

`S10F00000000100000000084EFAFFFE93` loads the following data to the following addresses:

byte 00_{16} to address 00_{16}

byte 00_{16} to address 01_{16}

byte 10_{16} to address 02_{16}

byte 00_{16} to address 03_{16}

byte 00_{16} to address 04_{16}

byte 00_{16} to address 05_{16}

byte 00_{16} to address 06_{16}

byte 08_{16} to address 07_{16}

byte $4E_{16}$ to address 08_{16}

byte FA_{16} to address 09_{16}

byte FF_{16} to address $0A_{16}$

byte FE_{16} to address $0B_{16}$

`S5030001FB` indicates that only one S1-record, S2-record, or S3-record was sent.

`S9030008F4` indicates that the start address is 00000008_{16}.

ARITHMETIC COMMANDS

The commands in this group allow for basic arithmetic functions to be performed at the command line.

add

add *number1 number2* adds two integers in hexadecimal, binary, octal, or decimal (default).

The default numeric base is decimal. Specify hexadecimal by typing ":16" at the end of the value, octal by typing ":8" or binary by typing ":2." The result of the operation is displayed in hex, decimal, octal, and binary.

Definition:

```
int Add(unsigned long Arg1,
        unsigned long Arg2)
```

div

div *number1 number2* divides two integers in hexadecimal, binary, octal, or decimal (default). *number1* is divided by *number2*. The command also checks the operation to avoid dividing by zero.

The default numeric base is decimal. Specify hex by typing ":16" at the end of the value, octal by typing ":8" or binary by typing ":2." The result of the operation is displayed in hex, decimal, octal, and binary.

Definition:

```
int Div(unsigned long Arg1,
        unsigned long Arg2)
```

mul

mul *number1 number2* multiplies two integers in hexadecimal, binary, octal, or decimal (default) from the monitor.

The default numeric base is decimal. Specify hex by typing ":16" at the end of the value, octal by typing ":8" or binary by typing ":2." The result of the operation is displayed in hex, decimal, octal, and binary.

Definition:

```
int Mul(unsigned long Arg1,
        unsigned long Arg2)
```

rand

rand is a linear congruent random number generator that uses a function **Seed** and a field *Value*. The random number returned is an unsigned long.

Definition:

```
unsigned long Rand(void)
```

sub

sub *number1 number2* subtracts two integers in hexadecimal, binary, octal, or decimal (default). *number2* is subtracted from *number1*.

The default numeric base is decimal. Specify hexadecimal by typing ":16" at the end of the value, octal by typing ":8" or binary by typing ":2." The result of the operation is displayed in hex, decimal, octal, and binary.

Definition:

```
int Sub(unsigned long Arg1,
        unsigned long Arg2)
```

OTHER COMMANDS

These commands provide basic configuration and help facilities.

configboard

configboard configures the board to the state specified by the nonvolatile memory configuration. This includes the serial port, processor caches, and PCI (Monarch boards only).

configboard can be used to reconfigure the board's various interfaces after modification of the nonvolatile memory configuration (using **nvdisplay** or **nvset**). This command accepts no parameters.

Definition:

```
void ConfigBoard(void)
```

enumerate_pci (Monarch only)

enumerate_pci determines if any PMC modules or PCI devices are present and, if so, uses software polling to determine their type. The memory and I/O spaces of any installed modules are mapped to default locations—only the base addresses of the PCI devices are initialized. Subsequently, you can use the **pci_show** command to display the PCI devices and their base addresses on the screen.

Definition:

```
void enumerate_pci(void)
```

pci_show

After the PCI bus has been enumerated, **pci_show** displays the PCI devices and their base addresses on the screen.

The following is an example of the output:

```
VEND DEV  REV LOCALE  INUM  BASE0    BASE1    BASE2    BASE3    BASE4    BASE5
---- ---- --- ------- ---- -------- -------- -------- -------- -------- -------
1223 000e 01  00,00,0  00  ........ 80000000 84000000 ........ ........ ........
```

Definition:

```
void pci_show(void)
```

ethernetaddr

ethernetaddr returns the Ethernet address of the board in hexadecimal. For a description of the Ethernet address, refer to the “Ethernet Address”, Section .

Definition:

```
void EthernetAddr(void)
```

getboardconfig

getboardconfig displays the contents of the board configuration register that specifies the memory, L2 cache, and other configuration information. The following is an example of the output:

```
Board Configuration Register = 0x6891
-----------------------------------------------
Boot DeviceSoldered (Wide) Flash
SDRAM size64 MB
ECC MemoryInstalled
Local Bus Speed83 MHz
L2 CacheInstalled (w/ Parity)
Module ModeNon-monarch
EthernetInstalled
Real-time ClockDS17485 (4K NVRAM)
Soldered (Wide) Flash64 MB (64 bits wide)
```

Definition:

```
void GetBoardConfig(void)
```

help

help *name* allows you to view the description of the monitor command specified by *name*. The full name of the command must be given.

For instructions on editing command lines, type `help editor`.

For a list of command-line functions, type `help functions`.

For a detailed memory map, type `help memmap`.

Definition:

```
int Help(char *Name)
```

COMMAND ERRORS AND SCREEN MESSAGES

Most commands return an explanatory message for misspelled or mistyped commands, missing arguments, or invalid values. Table 9-5 lists errors that can be attributed to other causes, especially errors that indicate a problem in the nonvolatile memory configuration.

Table 9-5: *Error and Screen Messages*

Message:	Source and Suggested Solution:
Error while clearing NV memory Error while reading NV memory Error while storing NV memory	NV memory has become corrupted. Use the nvinit command to restore defaults. If the problem persists, contact Technical Support[1].
Hit 'H' to skip auto-boot...	Consult the introduction to this chapter for information about power-up conditions.
Power-up Test FAILED	A failed test could mean a hardware malfunction. Report the error to Test and Repair Services[1].
Unknown boot device.	The boot device is invalid. Use **nvdisplay** to check and edit the 'BootParams' group, *BootDev* field. Save a new value with **nvupd**
Unexpected _____ Exception at _____	There are many possible sources for this error. If the error is displayed during boot, it could mean that autoboot is enabled and invalid fields are being used. If the error is displayed at reset or power-up and autoboot is *not* enabled, report the error to Technical Support[1]. If the error is displayed after a command has been executed, an attempt to perform an operation that causes an exception has probably been made.
Warning NV memory is invalid - using defaults	Consult the introduction to this chapter for information about reset conditions.

1. Visit http://www.emersonembeddedcomputing.com/contact/customersupport.html

MONITOR FUNCTION REFERENCE

The PmPPC monitor functions fall into two groups: PmPPC-specific and standard Emerson monitor functions. For convenience, related functions are combined in groups under a single name. If you cannot find a particular function, refer to the index for the appropriate page number.

Note: *Unlike the monitor commands, no argument checking takes place for functions that are called directly from the command line.*

The functions require spaces between the function name and its arguments. No parentheses or other punctuation is necessary.

Examples:

```
PmPPC{Ver 1.0] => getMSR
PmPPC{Ver 1.0] => ConnectHandler f8 1000
```

PMPPC-SPECIFIC FUNCTIONS

This section describes functions which are specific to the PmPPC monitor implementation.

CPC700 Indirect Reads

Synopsis:

```
unsigned long read_pif_cpc700(unsigned long offset);
unsigned long read_memctrl_cpc700(unsigned long offset);
```

Description: These read functions permit access to the indirect register sets of the CPC700 Processor Interface and Memory Controller sections. *offset* is simply the numeric value indicated in the *CPC700 User's Manual* (see Table 1-1). For example:

```
read_memctrl_cpc700 20
```

reads the contents of the MCOPT1 register in the CPC700 memory controller and prints the value in hexadecimal, decimal, octal, and binary.

CPC700 Indirect Writes

Synopsis:

```
void write_pif_cpc700(unsigned long offset,
                      unsigned long data,
                      unsigned long rsvmask);
void write_memctrl_cpc700(unsigned long offset);
                          unsigned long data,
                          unsigned long rsvmask);
```

Description: These write functions permit access to the indirect register sets of the CPC700 Processor Interface and Memory Controller sections. *offset* is the numeric value indicated in the *CPC700 User's Manual* (see Table 1-1). *data* is the data to be written to the target register. *rsvmask* contains ones in the positions of the reserved bits of the target register (preserves the values of these bits). For example:

```
write_pif_cpc700 4 40000000 01FFFFFF
```

writes a one to the MEM_SEL_ER bit of the ERRDET2 register in the CPC700 processor interface. This clears the error from the register.

Note: *Be careful when writing to any of the processor interface or memory controller registers, as most of them control fundamental system operations. Changing these values may result in undefined behavior or system reset.*

Hardware Implementation Dependent Register

Synopsis:

```
unsigned long getHID0(void)
void setHID0(unsigned long 32-bit)
void clrHID0(unsigned long 32-bit)
```

Description: **GetHID0** returns the value of the CPU's Hardware Implementation Dependent register (HID0). Functions **SetHID0** and **ClrHID0** set and clear the bits in HID0. HID0 is described in the *PowerPC™ 750 RISC Microprocessor User's Manual*.

Miscellaneous

Synopsis:

```
unsigned long getMSR(void)
void setMSR(unsigned long 32-bit)
void clrMSR(unsigned long 32-bit)
unsigned long getTBU(void)
unsigned long getTBL(void)
unsigned long getDEC(void)
void setDEC(unsigned long 32-bit)
unsigned long getSRR0(void)
unsigned long getPVR(void)
unsigned long get_dbatu_entry(int batnum)
unsigned long get_dbatl_entry(int batnum)
```

Description: The functions **getMSR, setMSR**, and **clrMSR** return the value of the Machine State register (MSR). **setMSR** and **clrMSR** either set or clear the bits in the MSR. **getTBU** and **getTBL** return the upper and lower 32-bit Time Base register values, respectively. Functions **getDEC** and **getSRR0** return the value for the appropriate register. **setDEC** writes a value to the decrementer. **getPVR** gets the processor version number. **get_dbatu_entry** and **get_dbatl_entry** return the upper and lower MMU data block address translation register values, as indexed by 0, 1, 2, and 3.

Display Processor Temperature

Synopsis:

```
void DisplayTemp(void)
```

Description: **DisplayTemp** successively approximates the processor junction temperature from the Thermal Management Unit. It displays in real time the temperature in degrees Celsius. The resolution of the thermal sensor is 4° C. If the *ThermProtect* NVRAM field is enabled and MSR[EE] is not disabled, the monitor puts the processor into permanent sleep mode when the junction temperature exceeds 100° C. The application can change this behavior at 100° C by connecting a new interrupt handler to vector 1700_{16}. *ThermProtect* is disabled by default.

STANDARD EMERSON FUNCTIONS

This section describes functions which are part of the standard Emerson monitor implementation.

Conversions

Synopsis:

```
unsigned long atoh(char *p)
```

```
unsigned long atod(char *p)
unsigned long atoo(char *p)
unsigned long atob(char *p)
unsigned long atoX(char *p, int Base)
void BinToHex(unsigned long Val)
void HexToBin(unsigned long Val)
void FindBitSet(unsigned long Number)
```

Description: These functions are a collection of numeric conversion programs used to convert character strings to numeric values, convert hexadecimal to BCD, BCD to hexadecimal, and to search for bit values.

The **atoh** function converts an ASCII string to a hex number. The **atod** function converts an ASCII string to a decimal number. The **atoo** function converts an ASCII string to an octal number. The **atob** function converts an ASCII string to a binary number.

The function **atoX** accepts both the character string *p* and the numeric base *Base* to be used in converting the string. This can be used for numeric bases other than the standard bases 16, 10, 8, and 2.

The **BinToHex** function converts a binary value to packed nibbles (BCD). The **HexToBin** function converts packed nibbles (BCD) to binary. This function accepts the parameter *Val*, which is assumed to contain a single hex number of value 0-99.

The **FindBitSet** function searches the *Number* for the first non-zero bit. The bit position of the least significant non-zero bit is returned.

Booting

Synopsis:

```
BootUp(void)
```

Description: The **BootUp** function is called after the nonvolatile memory device has been opened and the board has been configured according to the nonvolatile configuration. This function also determines if memory is to be cleared according to the nonvolatile configuration.

The monitor provides an autoboot feature that allows an application to be loaded from a variety of devices and executed. This function uses the nonvolatile configuration to determine which device to boot from and calls the appropriate bootstrap program. The monitor supports the ROM, BUS, and SERIAL autoboot devices, which are not hardware-specific. The remainder of the devices may or may not be supported by board-specific functions described elsewhere. Currently, the board-specific devices support Ethernet, Stos, and flash.

See Also: StartMon.c, NvMonDefs.h, NVTable.c, “Boot Commands”, Section .

Cache Control

Synopsis:

```
void enable_icache(void)
void disable_icache(void)
void invalidate_icache(void)
void enable_dcache(void)
void disable_dcache(void)
void flush_dcache(void)
void invalidate_dcache(void)
void flush_L2(void)
void L2_on(void)
void L2_off(void)
```

Description: As the names indicate, these functions enable, disable, invalidate, and flush the instruction and data caches. The **disable_dcache** function calls **flush_dcache** before disabling the data cache. **L2_off** should not be called until **disable_dcache** and **flush_L2** have been executed.

MMU Control

Synopsis:

```
void mmu_inst_enable(void)
void mmu_inst_disable(void)
void mmu_data_enable(void)
void mmu_data_disable(void)
```

Description: The MMU functions enable or disable instruction and data address translation in the MSR register.

Baud Rate

Synopsis:

```
void baud_c(unsigned long baud)
void baud_d(unsigned long baud)
void ConfigSerDevs(void)
```

Description: The **baud_c** and **baud_d** functions set the baud rates for the console and download ports, respectively. The valid baud rate values are 1200, 2400, 4800, 9600, 19200, 38400, 56000, and 128000 bps.

The **ConfigSerDevs** function uses the current definitions in the nonvolatile memory configuration to configure the serial ports. It is important that the configuration be valid when this function is called, or unpredictable behavior may result.

Both serial ports can be configured to use 5 to 8 data bits, 1 or 2 stop bits, and odd, even, or no parity.

Exceptions

Synopsis:

```
vectinit(HANDLER default_handler
         HANDLERPARAM default_param,
         unsigned long vectmask)
void connecthandler(unsigned long Vector, HANDLER handler)
void disconnecthandler(unsigned long Vector)
void Probe(char DirFlag,
           char SizeFlag,
           unsigned long Address,
           unsigned long DataPtr)
```

Description: These processor-specific functions provide interrupt and exception handling support.

The lower 2000_{16} bytes of memory contain routines that the processor executes upon receiving an interrupt. These routines comprise the low memory interrupt table. For example, upon receiving a vector 200_{16} interrupt, the processor branches to address 200_{16} in memory and executes the corresponding interrupt routine.

The function **vectinit** initializes these routines in the interrupt table so that they reference an unexpected-interrupt handler. **vectinit** expects a pointer to the default unexpected-interrupt handler and an optional fixed parameter for the handler. This ensures that the board will not hang upon receiving unexpected interrupts. The unexpected-interrupt handler saves the state of the processor at the point of detection and then calls **IntrErr**, which displays the error and restarts the monitor.

For those applications requiring an interrupt vector to perform only a simple task, **vectinit** has a third parameter. This parameter, *vectmask*, specifies which vectors to initialize and which vectors to leave unmodified in low memory. The parameter is a 32-bit value, where each set bit indicates that the corresponding routine should be replaced. For example, if *vectmask* contains FFFF,$FFFE_{16}$, all 32 vector routines will be overwritten except the routine at address 0_{16}. If *vectmask* contains FFFF,$FFF3_{16}$, all the routines will be overwritten except those at addresses 200_{16} and 300_{16}.

The function **connecthandler** initializes the entry in the vector table to point to the *Handler* address. The argument *Vector* indicates the vector number to be connected and the argument *Handler* is the address of the function that will handle the interrupts. With this structure, assembly language programming for interrupts is avoided.

The function **disconnecthandler** modifies the interrupt table entry associated with *Vector* to use the unexpected interrupt handler.

The function **Probe** accesses memory locations that may or may not result in a watchdog time-out or bus error. This function returns TRUE (non-zero) if the location was accessed and FALSE (zero) if the access resulted in a bus error. The argument *DirFlag* indicates whether a read (0) or a write (1) should be attempted. The argument *SizeFlag* selects either a byte access (1), a word access (2), or a long access (4). The argument *Address* indicates the address to be accessed, and the argument *Data* is a pointer to the read or write data.

Serial I/O

Synopsis:

```
unsigned char get_c (void)
unsigned char get_d (void)
void put_c (unsigned char c)
void put_d (unsigned char c)
int key_c (void)
int key_d (void)
int tx_empty (void)
int tx_empty_d (void)
```

Description: These functions provide low-level input/output support to read, write, and configure the CPC700 UARTs. These functions interface with both the console and download devices.

The **get_c** and **get_d** functions read a character from the console or download port, respectively.

The **put_c** and **put_d** functions write the character *c* to the console or download port, respectively. If the character was sent, they return TRUE (non-zero). If the function times out, they return FALSE (zero).

The **key_c** and **key_d** functions check for a character on the console or download port, respectively. If a character is available, they return TRUE (non-zero). If no character is available, they return FALSE (zero).

The **tx_empty** and **tx_empty_d** functions determine whether the transmitter is available for sending a character on the console or download port, respectively. If the transmitter is available, they return TRUE (non-zero); otherwise, they return FALSE (zero).

Initialize Board

Synopsis:

```
void config_MMU(void)
void configSerDevs(void)
void ConfigCaches(void)
```

Description: These functions provide initialization of the board's interfaces at various points in the monitor. They use the nonvolatile memory configuration to determine how to configure an interface, so the data structures must contain valid data before the functions are called. The **ConfigBoard** command executes all of these functions.

The **config_MMU** function sets the block address translation registers of the processor. **ConfigSerDevs** sets the console and download serial ports as specified by the NVRAM fields. **ConfigCaches** initializes the processor caches to be On or Off as defined by the NVRAM fields.

Initialize FIFO

Synopsis:

```
void InitFifo(struct Fifo *FPtr,
              unsigned char *StartAddr,
              int Length)
void ToFifo(struct Fifo *FPtr,
            unsigned char c)
void FromFifo(struct Fifo *FPtr,
              unsigned char *Ptr)
```

Description: These functions provide the necessary interface to initialize, read, and write a software FIFO. The FIFO can be used for a variety of applications. All three functions accept a pointer *FPtr* as the first argument to a FIFO management structure. This FIFO structure is described briefly below:

```
struct Fifo {
unsigned char *Top;
unsigned char *Bottom;
int Length;
unsigned char *Front;
unsigned char *Rear;
int Count;
} Fifo;
```

The function **InitFifo** initializes the FIFO control structure specified by *FPtr* to use the unsigned character buffer starting at *StartAddr* that is of size *Length*.

The function **ToFifo** writes the byte *c* to the specified FIFO. This function returns TRUE if there is room in the FIFO (before adding *c* to the FIFO), or FALSE if the FIFO is full.

The function **FromFifo** reads a byte from the specified FIFO. If a character is available, it is written to the address specified by the pointer *Ptr* and the function returns TRUE. If no character is available, the function returns FALSE.

Initialize Ethernet Address

Synopsis: `void ConfigEthernet(int serialnum)`

Description: The function **ConfigEthernet** sets the Ethernet address of the board. It combines the decimal serial number with the company code and product ID to form the 6-byte address. The **EthernetAddr** command displays the current address.

Interrupts

Synopsis: `void maskints(void)`

`void unmaskints(void)`

Description: The functions **unmaskints** and **maskints** are used to enable and disable external interrupts at the processor.

Interrupt Error

Synopsis: `void IntrErr(unsigned char Vector)`

Description: When an unexpected interrupt is received, it is necessary to remove the error condition before returning to the monitor. This function is called from the low-level interrupt service routine, which parses the interrupt record for the address and the vector associated with the interrupt. The device is dealt with accordingly, and the monitor is resumed.

Because the interrupt condition might be a program that continually generates exceptions, it is necessary to abort the program and return directly to the monitor level. This is done by calling the function **RestartMon**, which causes the processor to return to the line editor.

Legal Value Check

Synopsis: `void IsLegal(unsigned char Type, char *Str)`

Description: This function is used to determine if the specified character string *Str* contains legal values to allow the string to be parsed as decimal, hex, uppercase, or lowercase. The function **IsLegal** traverses the character string until a NULL is reached. Each character is verified according to the *Type* argument.

The effects of specifying each type are described in the following table:

Table 9-6: *IsLegal Function Types*

Type:	Value:	Legal Characters:
DECIMAL	0x8	0 - 9
HEX	0x4	0 - 9, A - F, a - f
UPPER	0x2	A - Z
LOWER	0x1	a - z

Type:	Value:	Legal Characters: (continued)
ALPHA	0x3	A - Z, a - z

If the character string contains legal characters, this function returns TRUE; otherwise, it returns FALSE. The string equivalent of the character functions isalpha(), isupper(), islower(), and isdigit() can be constructed from this function, which deals with the entire string instead of a single character.

Memory Management

Description: The memory management functions allocate and free memory from a memory pool. The monitor initializes the memory pool to use all on-card memory after the monitor's *bss* section. If any of the autoboot features are used, the memory pool is not initialized and the application program is required to set up the memory pool for these functions.

The functions **Malloc**, **Calloc** and **ReAlloc** allocate memory from the memory pool. Each of these functions returns a pointer to the memory requested if the request can be satisfied and NULL if there is not enough memory to satisfy the request. The function **Malloc** accepts one argument *NumBytes* indicating the number of bytes requested. The function **Calloc** accepts two arguments *NumElements* and *Size* indicating a request for a specified number of elements of the specified size. The function **ReAlloc** reallocates a memory block by either returning the block specified by *Block* to the free pool and allocating a new block of size *NumBytes*, or by determining that the memory block specified by *Block* is big enough and returning the same block to be reused.

The functions **Free** and **CFree** return blocks of memory that were requested by **Malloc**, **Calloc**, or **ReAlloc** to the free memory pool. The address of the block to be returned is specified by the argument *MemLoc*, which must be the same value returned by one of the allocation functions. An attempt to return memory that was not acquired by the allocation functions is a fairly reliable way of blowing up a program and should be avoided.

The function **MemReset** sets the free memory pool to the empty state. This function must be called once for every reset operation and before the memory management facilities can be used. It is also necessary to call this function before every call to **MemAdd**.

The function **MemAdd** initializes the free memory pool to use the memory starting at *MemAddr* of size specified by *MemSize*. This function currently allows for only one contiguous memory pool and must be preceded by a function call to **MemReset**.

The function **MemStats** monitors memory usage. This function outputs a table showing how much memory is available and how much is used and lost as a result of overhead.

See Also: **MemTop**, **MemBase** in the next section.

Miscellaneous

Synopsis:

```
unsigned char *MemTop(void)
unsigned char *MemBase(void)
```

Description: The functions **MemTop** and **MemBase** are used to determine the addresses of the last and first long words in free memory. The size of DRAM is determined by bits 5:4 of Board Configuration register 0 at 7FF4,0001$_{16}$ (see page 2-7). The base of free memory varies, depending on the monitor version. Generally, it begins beyond the end of the monitor's data sections.

Emerson Monitor

Synopsis:

```
int NvHkOffset(void)
int NvMonOffset(void)
int NvMonSize(void)
int NvMonAddr(void)
```

Description: These functions allow the nonvolatile library functions to operate on the nonvolatile memory sections without actually compiling the board configuration files into the library.

The **NvHkOffset** and **NvMonOffset** functions describe where in the nonvolatile memory device the Emerson- and monitor-defined data sections begin. In general, the Emerson-defined data section and the monitor data section reside in the user-writable section of the nonvolatile memory device. The returned value is the offset in bytes from the beginning of the device in which the section is loaded.

The functions **NvMonSize** and **NvMonAddr** return the size and location of the nonvolatile monitor configuration data structure. This again allows other monitor facilities and application programs to get at the monitor configuration structure without having to know too much about the monitor.

Support Functions

Synopsis:

```
void SetNvDefaults(NVGroupPtr Groups, int NumGroups)
void DispGroup(NVGroupPtr Group, unsigned long EditFlag)
int NVOp(unsigned long NVOpCmd,
         unsigned char *Base,
         unsigned long Size,
         unsigned long Offset)
```

Description: The support functions used for displaying, initializing, and modifying the nonvolatile memory data structures can also be used to manage other data structures that may or may not be stored in nonvolatile memory.

The method used to create a display of a data structure is to create a second structure that contains a description of every field of the first structure. This description is done using the NVGroup structure. Each entry in the NVGroup structure describes a field name, pointer to the field, size of the field, indication of how the field is to be displayed, and the initial value of the field.

An example data structure is shown below, as well as the NVGroup data structure necessary to describe the data structure. This example might describe the coordinates and depth of a window structure.

```
struct NVExample {
NV_Internal Internal;
unsigned long XPos, YPos;
unsigned short Mag;
} NVEx;
NVField ExFields[] = {
{ "XPos",   (char *) &NVEx.XPos, sizeof(NVEx.XPos),
NV_TYPE_DECIMAL, 0, 100, NULL},
{ "YPos",   (char *) &NVEx.YPos, sizeof(NVEx.YPos),
NV_TYPE_DECIMAL, 0, 200, NULL},
{ "Depth"   (char *) &NVEx.Mag, sizeof(NVEx.Mag),
NV_TYPE_DECIMAL, 0, 4, NULL}
}
NVGroup ExGroups[] = {
{ "Window", sizeof(ExFields)/sizeof(NVField), ExFields }
};
```

If passed a pointer to the ExGroups structure, the function ***DispGroup*** generates the display shown below. The second parameter *EditFlag* indicates whether to allow changes to the data structure after it is displayed (same as in the **nvdisplay** command).

Window Display Configuration

XPos 100

YPos 200

Magnitude 4

The **SetNvDefaults** function, when called with a pointer to the ExGroup structure, initializes the data structure to those values specified in the NVGroup structure. The second parameter *NumGroups* indicates the number of groups to be initialized.

The **NVOp** function stores and recovers data structures from nonvolatile memory. The only requirement of the data structure to be stored in nonvolatile memory is that the first field of the structure be NVInternal, which is where all the bookkeeping for the nonvolatile memory section is done. The first parameter *NVOpCmd* indicates the command to be performed. A summary of the commands is shown in the following table:

Table 9-7: *NVOP Commands*

Command:	Value:	Description:
NV_OP_FIX	0	Fix nonvolatile section checksum
NV_OP_CLEAR	1	Clear nonvolatile section
NV_OP_CK	2	Check if nonvolatile section is valid
NV_OP_OPEN	3	Open nonvolatile section
NV_OP_SAVE	4	Save nonvolatile section
NV_OP_CMP	5	Compare nonvolatile section data

The second parameter, *Base*, indicates the base address of the data structure to be operated on, and the *Size* parameter indicates the size of the data structure to be operated on. The *Offset* parameter specifies the byte offset in the nonvolatile memory device where the data structure is to be stored. An example of how to initialize, store, and recall the example data structure is shown below.

```
NVOp(NV_OP_CLEAR, &NVEx, sizeof(NVEx), 0);
NVOp(NV_OP_SAVE , &NVEx, sizeof(NVEx), 0);
NVOp(NV_OP_OPEN , &NvEx, sizeof(NVEx), 0);
NVOp(NV_OP_FIX,   &NVEx, sizeof(NVEx), 0);
NVOp(NV_OP_SAVE , &NVEx, sizeof(NVEx), 0);
```

The clear, save, and open operations cause the nonvolatile device to be cleared and filled with the NVEx data structure; then the data structure is filled from nonvolatile memory. The fix and save operations are used to modify the nonvolatile device, which updates the internal data structures and then writes them back to the nonvolatile memory device.

If errors are encountered during the check, save, or compare operations, an error message is returned from the function *NVOp*. The error codes are listed below:

Table 9-8: *NVOP Error Codes*

Command:	Value:	Description:
NVE_NONE	0	No errors
NVE_OVERFLOW	1	Nonvolatile device write count exceeded
NVE_MAGIC	2	Bad magic number read from nonvolatile device
NVE_CKSUM	3	Bad checksum read from nonvolatile device
NVE_STORE	4	Write to nonvolatile device failed
NVE_CMD	5	Unknown operation requested

Command:	Value:	Description: (continued)
NVE_CMP	6	Data does not compare to nonvolatile device

See Also: NVFields.h

Seed

Synopsis:
```
void Seed(unsigned long Value)
```

Description: The **Seed** function sets the initial value for the random number generator command **rand**.

Serial Support

Synopsis:
```
unsigned char getchar (void)
void putchar(char c)
int KBHit(void)
int TxMT(void)
void ChBaud(int Baud)
```

Description: The serial support functions defined here provide the ability to read, write, and poll the monitor's console device, which provides the user interface. The serial port is configured at reset according to the nonvolatile memory configuration.

The function **getchar** reads characters from the console device. When called, this function does not return until a character has been received from the serial port. The character read is returned to the caller.

The function **putchar** writes the character *c* to the console device. If the serial port does not accept the character, the function eventually times out.

The function **KBHit** polls the console device for available characters. If the receiver indicates a character is available, this function returns TRUE; otherwise, it returns FALSE.

The function **TxMT** polls the console device to determine if the transmitter can accept more characters. If the transmitter indicates a character can be sent, this function returns TRUE; otherwise, it returns FALSE.

The function **ChBaud** modifies the console's baud rate. The argument *Baud* specifies the new baud rate to use for the port. This function accepts the same baud rates as the **baud_c** function (see Section).

See Also: **get_c**, **put_c**, **key_c**, **tx_empty**, **baud_c** on page 9-44.

Unexpected Interrupt Handler

Synopsis:
```
SetUnExpIntFunct(unsigned long Funct)
```

Description: If desired, a program can call the **SetUnExpIntFunct** function to attach its own interrupt handler to all unexpected interrupts. This function attaches the handler specified by *Funct*. The new interrupt handler must determine the source of the unexpected interrupt and remove it.

Strings

Synopsis:

```
int CmpStr(char *Str1, char *Str2)
int StrCmp(char *Str1, char *Str2)
void StrCpy(char *Dest, char *Source)
int StrLen(char *Str)
void StrCat(char *DestStr, char *SrcStr)
```

Description: These functions provide the basic string manipulation functions necessary to compare, copy, concatenate, and determine the length of strings.

The function *CmpStr* compares the two null terminated strings pointed to by *Str1* and *Str2*. If they are equal, it returns TRUE; otherwise, it returns FALSE. Note that this version does not act the same as the UNIX® **strcmp** function. **CmpStr** is not case-sensitive and only matches characters up to the length of *Str1*. This is useful for pattern matching and other functions.

The function **StrCmp** compares the two null terminated strings pointed to by *Str1* and *Str2*. If they are equal, it returns zero; otherwise, it returns the difference between the first two characters in the strings that fail to match (not case-sensitive). Note that this function is not the same as the UNIX **strcmp** function, which is case-sensitive.

The function **StrCpy** copies the null terminated string *Source* into the string specified by *Dest*. There are no checks to verify that the string is large enough or is null terminated. The only limit is the monitor-defined constant MAXLN (80), which is the largest allowed string length the monitor supports. The length of the string is returned to the calling function.

The function **StrLen** determines the length of the null terminated string *Str* and returns the length. If the length exceeds the monitor defined limit MAXLN, the function returns MAXLN.

The function **StrCat** concatenates the string *SrcStr* onto the end of the string *DestStr*.

Test Suite

Synopsis:

```
void TestSuite(unsigned long BaseAddr,
               unsigned long TopAddr,
               int TSPass)
void ByteAddrTest(unsigned char *BaseAddr,
```

```
                    unsigned char *TopAddr)
void WordAddrTest(unsigned short *BaseAddr,
                    unsigned short *TopAddr)
void LongAddrTest(unsigned long *BaseAddr,
                    unsigned long *TopAddr)
void RotTest(unsigned long *BaseAddr,
              unsigned long *TopAddr)
void PingPongAddrTest(unsigned long BaseAddr,
                      unsigned long TopAddr)
void Interact(int Mod,
               unsigned char *StartAddr,
               unsigned char *EndAddr)
```

Description: The function **TestSuite** and the memory tests which make up this function verify a memory interface. Each of these functions accepts two arguments, *BaseAddr* and *TopAddr*, which describe the memory region to be tested. The argument *TSPass* defines the number of passes to perform. Each test and the intended goals of the test are described briefly below.

The function **ByteAddrTest** performs a byte-oriented test of the specified memory region. Each location is tested by writing the lowest byte of the location address through the entire memory region and verifying each location.

The function **WordAddrTest** performs a word-oriented test of the specified memory region. Each location is tested by writing the lowest word of the location address through the entire memory region and verifying each location.

The function **LongAddrTest** performs a long-oriented test of the specified memory region. Each location is tested by writing the location address through the entire memory region and verifying each location.

The function **RotTest** performs a long word-oriented test of the specified memory region. Each memory location is tested by rotating a single bit through the long-word location.

The function **PingPongAddrTest** is used to test the reliability of memory accesses in an environment where the data addresses are varying widely. The intention is to cause the address buffers and multiplexors to change dramatically.

The function **Interact** is used to test byte interaction in the memory region specified by *StartAddr* and *EndAddr*. The main goal of this test is to check for mirrors in memory. This is accomplished by testing the interaction between bytes at different points in memory.

Timer

Synopsis:

```
void time_delay(unsigned long duration)
```

Description: The **time_delay** function causes a delay of *duration* microseconds. This function makes use of the time base counter on the CPU to generate the delay.

Printing

Description: This function serves as a System V UNIX®-compatible **printf()** without floating point. It implements all features of %d, %o, %u, %x, %X, %c, and %s. An additional control statement has been added to allow printing of binary values (%b).

The **xprintf** and **xsprintf** functions format an argument list according to a control string *Ctrl-Str*. The function **xprintf** prints the parsed control string to the console, while the function **xsprintf** writes the characters to the *Buffer*. The control string format is a string that contains plain characters to be processed as is, and special characters that are used to indicate the format of the next argument in the argument list. There must be at least as many arguments as special characters, or the function may be unreliable.

Special character sequences are started with the character %. The characters after the % can provide information about left or right adjustment, blank and zero padding, argument conversion type, precision, and more things too numerous to list.

If detailed information on the argument formats and argument modifiers is required, see your local C programmer's manual for details. Not all of the argument formats are supported. The supported formats are %d, %o, %u, %x, %X, %c, and %s.

Acronyms

ASCII	American Standard Code for Information Interchange
CPU	Central Processing Unit
CSA	Canadian Standards Association
EC	European Community
ECC	Error-correcting Code
EIA	Electronic Industries Alliance
EMC	Electromagnetic Compatibility
ESD	Electrostatic Discharge
ETSI	European Telecommunications Standards Institute
FCC	Federal Communications Commission
GPL	General Public License
I^2C	Inter-integrated Circuit
JTAG	Joint Test Action Group
LED	Light-emitting Diode
MAC	Medium/media Access Control/controller
PCI	Peripheral Component Interconnect
PLCC	Plastic-Leaded Chip Carrier
PLD	Programmable Logic Device
RMA	Return Merchandise Authorization
RTC	Real-time Clock
SDRAM	Synchronous Dynamic Random Access Memory
TBD	To Be Determined
UART	Universal Asynchronous Receiver/transmitter
UL	Underwriters Laboratories

Acronyms: (continued)

Index

A

abbreviations for monitor commands 9-12
air flow rate 2-9
arbiter 5-6
autoboot cancellation 9-38

B

baud rate 7-6
binary format records, downloading 9-31
block diagram, general system 1-2
board
  configuration, monitor 9-36
  product ID 2-12
  serial number 2-12
boot
  booting up 9-41
  commands 9-13
  device configuration 9-27
  jumper 4-2
booting applications
  from EPROM 9-14, 9-15
  from flash 9-15
  from FTP 9-16
  from ROM 9-15
  from serial port 9-15
bridge, PCI 5-1
bus speed, PCI 5-6

C

cache, CPU memory 3-3
caution statements
  install/remove the board 2-8
  serial cable strain relief 2-5
character arguments for monitor commands 9-13
checksum, S-records 9-32
circuit board dimensions 2-1
command reference 9-12
command-line editor 9-2
commands, monitor
  add 9-35
  bootbus 9-13
  booteprom 9-14, 9-15
  bootflash 9-15
  bootrom 9-15
  bootserial 9-15
  boottftp 9-16
  bootverify 9-17
  cachetest 9-30
  call 9-31
  checksummem 9-18
  clearmem 9-18
  cmpmem 9-18
  configboard 9-23, 9-36
  copymem 9-18
  displaymem 9-18
  div 9-35
  download 9-31
  enumerate_pci 9-36
  ethernetaddr 9-37
  ethertest 9-30
  fillmem 9-19
  findmem 9-19
  findnotmem 9-19
  findstr 9-19
  flashblkwr 9-22
  flashclrstat 9-22
  flasheraseblk 9-22
  flashsocketblkwr 9-22
  flashsocketerase 9-22
  getboardconfig 9-37
  help 9-37
  mul 9-35
  nvdisplay 9-15, 9-23, 9-29
  nvinit 9-26
  nvopen 9-26
  nvramtest 9-30
  nvset 9-26
  nvupdate 9-15, 9-27
  pci_show 9-36
  rand 9-35
  readmem 9-20
  rewritemonitor 9-22
  serialtest 9-30
  setmem 9-20
  sub 9-36
  swapmem 9-20
  testmem 9-20
  um 9-21
  writemem 9-21
  writestr 9-21
compliance 1-5
component map
  bottom 2-3
  top 2-2
connectors
  Ethernet 6-4
  overview 2-4
  PMC 5-9
  serial 7-6
contents, table of . . . . . ii-iii
CPC700
  bus support 8-1
  IIC ports 8-2
  interrupt controller 8-1
  memory interface 4-1
  PCI bridge 5-1
  processor interface 3-3
  timers 8-3
CPU
  cache memory 3-3
  CPC700 interface 3-3
  initialization 3-2
  interrupts 3-2
  reset 3-1
customer support. See technical support.

D

data width in user flash 4-3
defaults, monitor 9-48
diagnostics, power-up 9-1
download
  configuring the serial port 9-29
  from monitor 9-30
DRAM 4-4

E

editor commands, monitor 9-2
Emerson identifier, Ethernet 6-4
EPROM, booting from 9-14, 9-15
equipment for setup 2-8
error messages 9-38
ESC key 9-2
ESD prevention 2-1
Ethernet
  address 6-4, 9-37
  connector 6-4
  controller configuration 6-1
  interface 6-1
examples
  default boot device 9-27
  nvdisplay monitor command . . 9-25
  S-record file 9-34

F

features
components 1-1
general 1-1
PCI 5-1
figures, list of . . . . . iii-vii
flags
for memory monitor commands 9-17
for monitor commands 9-13
flash 4-2
memory commands 9-21
free memory 9-47
functions, monitor
atob 9-41
atod 9-41
atoh 9-40
atoo 9-41
atoX 9-41
baud_c 9-42
baud_d 9-42
BitToHex 9-41
BootUp 9-41
ByteAddrTest 9-52
ChBaud 9-51
clrHID0 9-39
clrMSR 9-40
CmpStr 9-52
config_MMU 9-44
ConfigCaches 9-44
ConfigEthernet 9-46
ConfigSerDevs 9-42
configSerDevs 9-44
connecthandler 9-43
disable_dcache 9-42
disable_icache 9-42
disconnecthandler 9-43
DispGroup 9-48
DisplayTemp 9-40
enable_dcache 9-42
enable_icache 9-42
FindBitSet 9-41
flush_dcache 9-42
flush_L2 9-42
FromFifo 9-45
get_c 9-44
get_d 9-44
get_dbatl_entry 9-40
get_dbatu_entry 9-40
getchar 9-51
getDEC 9-40
getHID0 9-39
getMSR 9-40
getPVR 9-40
getSRR0 9-40
getTBL 9-40
getTBU 9-40
HexToBin 9-41
InitFifo 9-45
Interact 9-53
IntrErr 9-46
invalidate_dcache 9-42
invalidate_icache 9-42
KBHit 9-51
key_c 9-44
L2_off 9-42
L2_on 9-42
LongAddrTest 9-53
maskints 9-46
MemBase 9-48
MemTop 9-48
mmu_data_disable 9-42
mmu_data_enable 9-42
mmu_inst_disable 9-42
mmu_inst_enable 9-42
NvHkOffset 9-48
NvMonAddr 9-48
NvMonOffset 9-48
NvMonSize 9-48
NVOp 9-48
PingPongAddrTest 9-53
Probe 9-43
put_c 9-44
put_d 9-44
putchar 9-51
RestartMon 9-46
RotTest 9-53
Seed 9-51
setDEC 9-40
setHID0 9-39
setMSR 9-40
SetNvDefaults 9-48
SetUnExpIntFunct 9-51
StrCat 9-52
StrCmp 9-52
StrCpy 9-52
StrLen 9-52
TestSuite 9-52
time_delay 9-53
ToFifo 9-45
tx_empty 9-44
tx_empty_d 9-44
TxMT 9-51
unmaskints 9-46
vectinit 9-43
WordAddrTest 9-53
fuse location 2-4

G

glossary of acronyms 10-1
grounding 2-1

I

initialization
error, nonvolatile memory 9-38
of board to defaults 9-45
of memory from the monitor . . 9-12
of nonvolatile memory, caution 9-26
installation of the board 2-8
installation on baseboard 2-9
interrupts
CPU 3-2
PCI 5-5

J

j key 9-2
JTAG/COP interface 3-4
jumper location 2-4
jumper, boot device select 4-2

K

k key 9-2
keys
ESC 9-2
j 9-2
k 9-2

L

LED locations 2-6
LEDs 2-5

M

mean time between failures (MTBF) 1-5
memory
boot ROM/flash 4-2
checksum 9-18
clearing 9-18
compare addresses 9-18
controller 4-1
copy 9-18
destructive test 9-21

memory *(continued)*
display 9-18
erase socketed flash 9-22
fill with specified value 9-19
flash block write 9-22
flash clear status 9-22
flash erase block 9-22
get configuration 9-37
initializing from the monitor. . . 9-12
management 9-47
map, NVRAM 4-5
modify 9-20
monitor commands 9-17
nondestructive test 9-20
read. 9-20
rewrite monitor image 9-22
search 9-19
search for string 9-19
socketed flash block write 9-22
swap 9-20
test failure. 9-1
user flash. 4-2
write 9-21
write ASCII string 9-21
memory map 1-3
monarch 5-5
monitor
character arguments 9-13
command syntax 9-12
commands. See commands, monitor.
flags 9-13
flash commands 9-21
functions. See functions, monitor.
group. See monitor group.
NVRAM commands 9-23
persistent memory 9-7, 9-12
power-up/reset sequence 9-4
typographic conventions 9-13
version number 2-12
monitor group
BootParams 9-25, 9-27
Cache 9-24
console 9-23
Download 9-24
HardwareConfig 9-25
Manufacturing 9-25
Misc 9-24, 9-29
Network 9-24
Service 9-25

N

non-persistent memory, monitor. . 9-5
nonvolatile memory
checking 9-26
commands 9-23
modifying 9-26
notation conventions 1-6
number bases for monitor arguments 9-12
numeric format 9-12

P

PCI bridge 5-1
PCI signals 5-7
persistent memory, monitor 9-7, 9-12
PMC module initialization. 9-36
power requirements 2-9
power-up
errors 9-38
monitor sequence 9-4
power-up diagnostics 9-1
cache test 9-30
Ethernet test. 9-30
NVRAM test 9-30
serial test 9-30
product code, Ethernet 6-4
product ID. 2-12
product repair. 2-13

R

records
data 9-32
data count 9-33
termination and start address . 9-33
user defined 9-32
references and manuals 1-6
registers
FCR 7-5
HCR0 2-7
IER 7-2
IIR 7-2
LCR 7-3
LSR 7-4
PIOR 5-5
reset control 3-1
regulatory certifications 1-5
reset
monitor sequence 9-4
PCI 5-7
returning boards 2-13

S

screen messages 9-37
SDRAM. See DRAM.
serial and version numbers. 2-12
serial I/O
baud rate 7-6
control 7-1
control from the monitor 9-51
registers 7-2
serial number 2-12
setup requirements 2-8
specifications
environmental 2-9
mechanical. 2-1
power. 2-9
S-records 9-31, 9-32
file example 9-34
static control 2-1
string format 9-12
symbol format 9-12, 9-13
synchronous. 4-4
syntax for monitor commands . . . 9-12

T

table of contents . . . . . ii-iii
tables, list of . . . . . iv-ix
technical references 1-6
technical support
terminology 1-6
troubleshooting, general 2-11
typographic conventions 9-13

U

UL certifications 1-5
user flash read/write array mode examples 4-3

V

vendor and device IDs, PCI 5-2
version
monitor 2-12
operating system 2-12
vi editing commands 9-2
vibration testing. 1-5

Notes

Emerson Network Power
The global leader in enabling
Business-Critical Continuity™

- AC Power Systems
- Connectivity
- DC Power Systems
- **Embedded Computing**
- Embedded Power
- Integrated Cabinet Solutions
- Outside Plant
- Power Switching & Controls
- Precision Cooling
- Services
- Site Monitoring
- Surge & Signal Protection

Emerson Network Power, Embedded Computing
8310 Excelsior Drive ▪ Madison, WI 53717-1935 USA
US Toll Free: 1-800-356-9602 ▪ Voice: +1-608-831-5500 ▪ FAX: 1-608-831-4249
Email: info@artesyncp.com

www.emersonembeddedcomputing.com

Business-Critical Continuity, Emerson Network Power and the Emerson Network Power logo are trademarks and service marks of Emerson Electric Co.
© 2007 Emerson Electric Co.

EMERSON. CONSIDER IT SOLVED.™