PPC440SP Evaluation Board Pass 2 Kit
User's Manual

APPLIED MICRO CIRCUITS CORPORATION

Applied Micro Circuits Corporation
6290 Sequence Dr., San Diego, CA 92121

Phone: (858) 450-9333 — (800) 755-2622 — Fax: (858) 450-9885

http://www.amcc.com

Contents

Figures

Tables

About This Book

This book contains information you need to install and use the AMCC® PowerPC 440SP Evaluation Board Pass 2 Kit, a hardware and software development tool for the PowerPC PPC440SP 32-bit embedded processor.

The PowerPC 440SP Evaluation Board Pass 2 Kit hardware includes the PowerPC 440SP Evaluation Board Pass 2, power supply, line cord, and board interface cables.

The PPC440SP evaluation board pass 2 kit software includes the AMCC PowerPC Initialization Boot Software (PIBS), which is resident in the flash memory on the board, PIBS source code, the Embedded PowerPC Operating System (EPOS), sample application programs, application development libraries and tools, a GNU cross development tool for PowerPC compiler/linker/assembler and associated binary utilities, and the files needed to support IBM® RISCWatch, a source-level debugger that runs on the host system.

The PPC440SP evaluation board pass 2 kit also includes technical specifications and board schematics. Connection of the evaluation board pass 2 to a host system is required for the exercises in this book.

Who Should Use This Book

This book is for hardware and software developers who want to evaluate the PowerPC PPC440SP microprocessor and use the debugging features of the PowerPC 440SP Evaluation Board Pass 2 Kit to support software development. Users should understand hardware and software development tools, concepts, and environments. Specifically, users should understand:

- The host's operating system
- The PowerPC Architecture™ and implementation-specific characteristics of the PowerPC microprocessor being used
- C and PowerPC Assembler language programming

How to Use This Book

This book contains the following sections organized in three parts:

To help readers find material in these chapters, the book contains:

Contacting the Support Center

For information about the PowerPC 440SP Evaluation Board Pass 2 Kit and the AMCC family of hardware and software products for embedded system developers, see the AMCC web site at:

http://www.amcc.com/

Please send any comments or questions regarding this product to the following Internet address:

support@amcc.com

Related Publications

Many of the following publications are included on the CD ROM that comes with the evaluation kit or can be found on the AMCC Web site at http://www.amcc.com. The others are available from your AMCC representative:

- Embedded Application Binary Interface (EABI) Publications

  PowerPC Embedded Application Binary Interface (EABI)

  System V Application Binary Interface, Third Edition, 0-13-0100439-5

  System V Application Binary Interface, PowerPC Processor Supplement
- Debugger Publications

  IBM RISCWatch Debugger User's Guide, 13H6964

  IBM RISCWatch Debugger Installation Guide, 13H6984
- PowerPC *440SP* Embedded Processor Publications

  PowerPC 440SP Embedded Processor Data Sheet

  PowerPC 440SP Embedded Processor User's Manual
- Evaluation Board Pass 2 Publications

  PowerPC 440SP Evaluation Board Pass 2 User's Manual
- EPOS Publications

  Embedded PowerPC Operating System User's Manual

Part I. Evaluation Board Kit Installation Setup

1. Evaluation Board Kit Overview

This section provides an introduction to hardware and software components contained in the PPC440SP evaluation board kit.

1.1 Hardware Components

The PPC440SP evaluation board kit contains the evaluation board, power supply, line cord, serial port, and Ethernet cables.

1.1.1 Evaluation Board Platform

Evaluation board features include:

- PowerPC™ PPC440SP processor
- 512 MB DDR SDRAM (at least)
- 3 PCI interfaces
- 1 built-in Ethernet port (MII/GMII, 10/100/1000 Mbps)
- 1 MB socketed FLASH memory (2 - 512-KB devices)
- 4 MB on-board FLASH
- 1 MB SRAM
- 3 serial ports
- I^{2}C EEPROMs
- 32 Kb FRAM

For detailed descriptions of the evaluation board specifications, features, and memory map, see the *PowerPC 440SP Evaluation Board Manual*.

1.1.2 Cables and Power Supply

The PPC440SP evaluation board kit includes a serial port interface cable for connecting the board's Serial Port 0 (J11 Bottom) to a terminal emulator running on the host.

The kit provides an Ethernet crossover cable to support direct Ethernet communication with the host system. The crossover cable supports 10/100/1000Mbps operation. One standard 10/100/1000Mbps Ethernet connector is provided on the evaluation board. The Ethernet crossover cable is for direct connection to a single host and should not be used with a hub, network switch or a building's Ethernet network.

Note: Using the crossover cable with a network switch or a building's network while using the gigabit-enabled emacs is possible but not recommended, because the PHYs autosense the receive and transmit cable wiring.

A power supply and line cord are also provided with the PPC440SP evaluation board kit.

1.2 Software Components

The PPC440SP evaluation board kit software consists of the Board Support Package (BSP), the IBM RISCWatch source level debugger, the PowerPC development tools, and a GNU cross development tool for PowerPC.

1.2.1 BSP Software

The BSP software includes the PowerPC Initialization Boot Software (PIBS), the Embedded PowerPC Operating System (EPOS), sample programs (including the Dhrystone benchmark program), and host tools. Source code is provided for the PIBS along with EPOS pre-compiled libraries and include files.

1.2.1.1 PowerPC Initialization Boot Software (PIBS)

The PIBS firmware is supplied in the socketed flash memory on the evaluation board. PIBS is accessed through a serial port connection between Serial Port 0 on the board (J11 Bottom) and a serial port on the host computer. PIBS initializes the 440SP processor and the board for both serial and Ethernet communications. Because PIBS supports communications with the host computer system, applications can be loaded from the host onto the board. See “PIBS Overview” on page 31 for additional information on PIBS.

1.2.1.2 Embedded PowerPC Operating System (EPOS)

EPOS is a real-time operating system developed for PowerPC processors. EPOS can be configured to meet the functional requirements and memory constraints of a wide variety of embedded applications. PIBS and the sample programs are EPOS applications. EPOS compiled libraries and include files are provided in the BSP and can be used for product development. If necessary, full EPOS source code can be provided upon request by contacting AMCC support through email at **support@amcc.com**. See the *Embedded PowerPC Operating System User's Manual* for additional information on EPOS.

1.2.1.3 Dhrystone Benchmark Program

The Dhrystone benchmark is a commonly available integer benchmark. It is included as a sample program to be built, loaded onto the board, and executed. The results of this benchmark may vary based on the compiler used, compiler options, and the system environment in which it is run.

1.2.1.4 Application Tools

Several host-based tools are provided to support ROM and application development for the evaluation board.

1.2.2 GNU Cross Development Tools/Utilities

The GNU cross development tools for PowerPC include the GNU compiler, assembler, linker, and various other utilities for application development. The GNU cross development tools must be used to recompile code for execution on the evaluation board because the EPOS libraries, which are required for a compilation, have been built using this toolset.

2. Host System Requirements

This section describes the hardware and software requirements of the host system to which the evaluation board will be connected. Supported host systems include Windows PCs.

2.1 PC Host System Requirements

Recommended hardware requirements of the host PC include:

- Pentium class CPU 500 MHz or faster with at least 64 MB RAM
- SVGA, XGA Display Monitor (1024 x 768)
- Approximately 200 MB of free disk space. This space is required for the PowerPC GNU cross development tools environment and the BSP software. The IBM RISCWatch installation requests approximately 13 MB. When planning disk space usage, consider disk space requirements for Windows and any other software packages.
- At least one available serial port for terminal emulation. Establishing an Ethernet host-to-board connection may require the installation of an Ethernet adapter card on the host (if not already installed) and additional connectivity hardware. That hardware may include any or all of the following: an Ethernet 10/100/1000 BaseT network transceiver, a twisted pair cable, and a hub. A point-to-point connection will require the Ethernet crossover cable supplied with the kit. The Ethernet crossover cable is for direct connection to a single host.

3. Installing the Software

This section describes the procedures for installing the evaluation board kit software on the host system.

3.1 What is Installed

The evaluation board software consists of the board support package (BSP), the GNU cross development tools for PowerPC, and the files required to support the IBM RISCWatch debugger.

3.2 PC Software Installation

Before beginning the installation, you must have:

- The CD labeled *AMCC PowerPC 440SP Evaluation Board Kit Software Package*, included in the kit
- A Windows PC

The installation procedure is as follows:

1. Insert the CD labeled *AMCC PowerPC 440SP Evaluation Kit Software Package* into the CD-ROM drive. A pop-up window displays, allowing you to install the evaluation board kit software packages.

   Note: If the window does not appear, do the following:

   1. Select **Start** from the Windows task bar.
   2. Select **Run**.
   3. Type D:\PPC440SPKit_install\kit440SP-x.xx.exe then press Enter to run the installation program (where D is your CD-ROM drive letter and x.xx is the level of the .exe).
2. Follow the installation program instructions that display in the window.

The default installation directory is C:\AMCC.

3.3 IBM RISCWatch Usage

IBM RISCWatch version 5.1 is required for use with the PowerPC 440SP processor and this board.

RISCWatch debugger installation and usage are not addressed by the *PPC440SP Evaluation Board Kit Software Package* automatic installation. RISCWatch setup.exe is available in the IBM RISCWatch directory of this CD and can be manually installed.

A RISCWatch processor configuration file (PCF) for the PPC440SP processor and Power PC 440SP Evaluation Board, and the related *How to compile 440SP PCF* readme file are also provided.

4. Host Configuration

The host configuration steps described in this section are required to enable communication between the board and the host computer.

4.1 Serial Port Setup - PC

A typical PC includes two serial ports to support communications through asynchronous data transfer. These ports are sometimes referred to as communication or COM ports. They are usually accessed from the back of the system unit. See your PC documentation to determine how many serial ports are available on your PC and where they are located.

The connection of the terminal emulator running on the host to the PIBS monitor running on the board, is made through the Serial Port 0 (J11 Bottom) on the board and a serial port on the PC. The host serial port must be configured for a baud rate of 9600, 8 data bits, 1 stop bit, no parity, and no flow control. Setting these parameters is discussed in “Using a Terminal Emulator” on page 26.

The serial port connection can be used to download applications from the host to the board but it can not be used to establish a debug connection with the RISCWatch debugger.

4.2 Ethernet Setup - PC

An Ethernet connection can be used for host-to-board communications. The Ethernet connection is made through an Ethernet adapter on the host and an Ethernet port on the board. Ethernet is used when downloading applications to the board or when using the RISCWatch debugger.

An Ethernet connection may require additional hardware. The evaluation board supports a standard Ethernet, twisted pair (10/100/1000 BaseT) connection. This connection requires a host PC with an appropriate Ethernet adapter. The host adapter is not included in the kit. See your PC and adapter documentation for requirements and installation instructions.

A 10/100/1000BaseT connection requires at least a crossover Ethernet twisted pair cable (included in the kit) for point-to-point communications. The Ethernet crossover cable is for direct connection to a single host. If more than two nodes are needed, then a hub/switch and straight-through twisted pair cables will be required.

Other hardware required depends on the type of Ethernet adapter you have on your PC and whether the board is being connected to an existing Ethernet network. Please see your system administrator and the documentation included with the adapter hardware for additional instructions.

Establishing an Ethernet interface requires a host IP address. If the host PC is connected to an existing Ethernet network, the host IP address should already be defined and there is no need to set it again. See your network administrator about how to obtain the host’s Ethernet IP address and how to add the board to the existing network.

To set the host IP address for the Ethernet connection:

1. Select the My Computer icon from the desktop.
2. Select Control Panel.
3. Select Network.
4. Add the appropriate Adapter network component for the Ethernet adapter being used (if not already added).
5. Add a protocol network component of Microsoft - TCP/IP (if not already added). Specify the IP address and netmask to be used.

For the update to take effect, TCP/IP may need to be restarted. This may require a reboot of the system and/or a restart of TCP/IP. Make a note of the host IP address assigned to the Ethernet adapter, because the PIBS software for host-to-board file transfers will need this value.

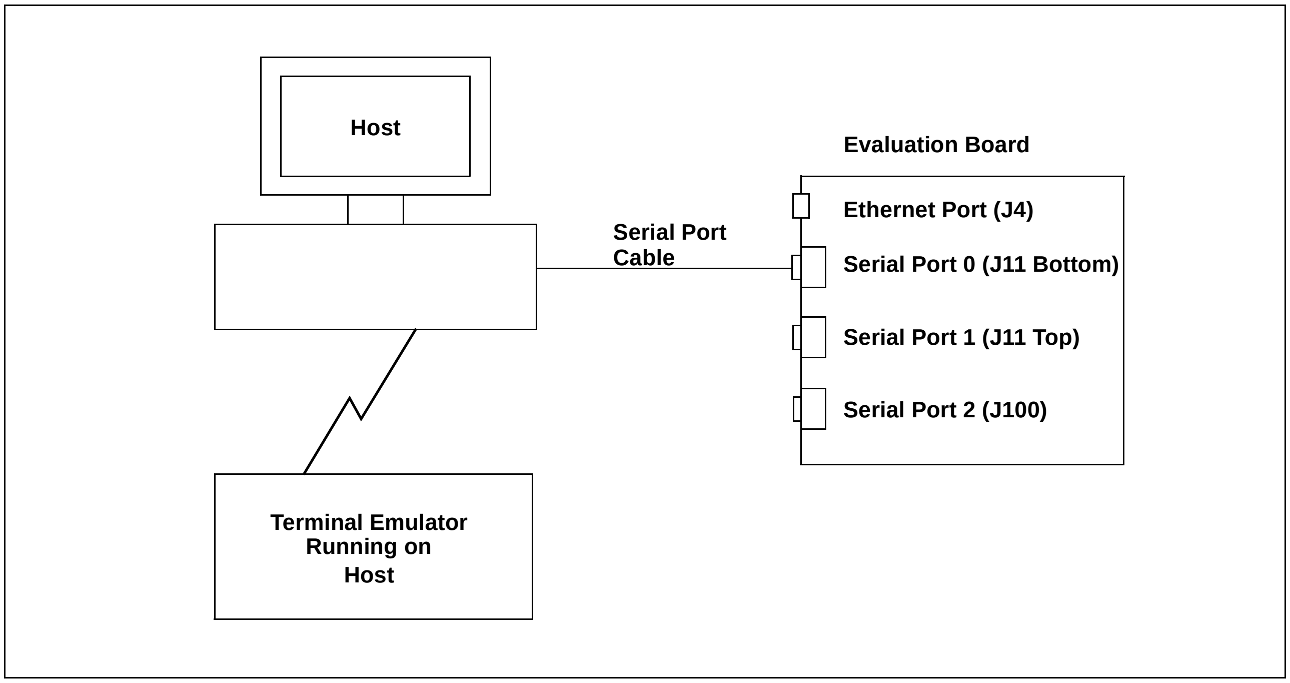
5. Board Connectors

For detailed descriptions of the connectors and jumpers on the evaluation board, see the *PowerPC 440SP Evaluation Board User's Manual*.

5.1 Connecting the Evaluation Board to the Host

To establish an operational environment, the evaluation board must be connected to a host system. PowerPC Initialization Boot Software (PIBS) access requires a connection between Serial Port 0 on the board (J11 Bottom) and a serial port on the host as shown in *Figure 5-1*. An Ethernet connection must be established when using the RISCWatch debugger in non-JTAG mode to debug an application running on the evaluation board.

Figure 5-1. Serial Port Connection



If a terminal emulator running on the host is going to be used for PIBS access, use the cable provided with the kit to connect Serial Port 0 (J11 Bottom) on the board to a serial port on the host. The host end might require a serial port adapter (not supplied) for connectivity.

The Ethernet connection can be made in one of two ways. If the connection is to be used exclusively between the host and the board and only 10Mb/s speed is required, the crossover cable provided with the kit can be used to directly connect the two nodes in a point-to-point Ethernet connection. Otherwise, a 10/100/1000BaseT hub/switch (not provided) must be used to connect the nodes.

Note: The Ethernet crossover cable supplied with the kit will **not** work if it is plugged into a hub switch. Ethernet straight-through cables (not included in the kit) are required if a hub or switch is used.

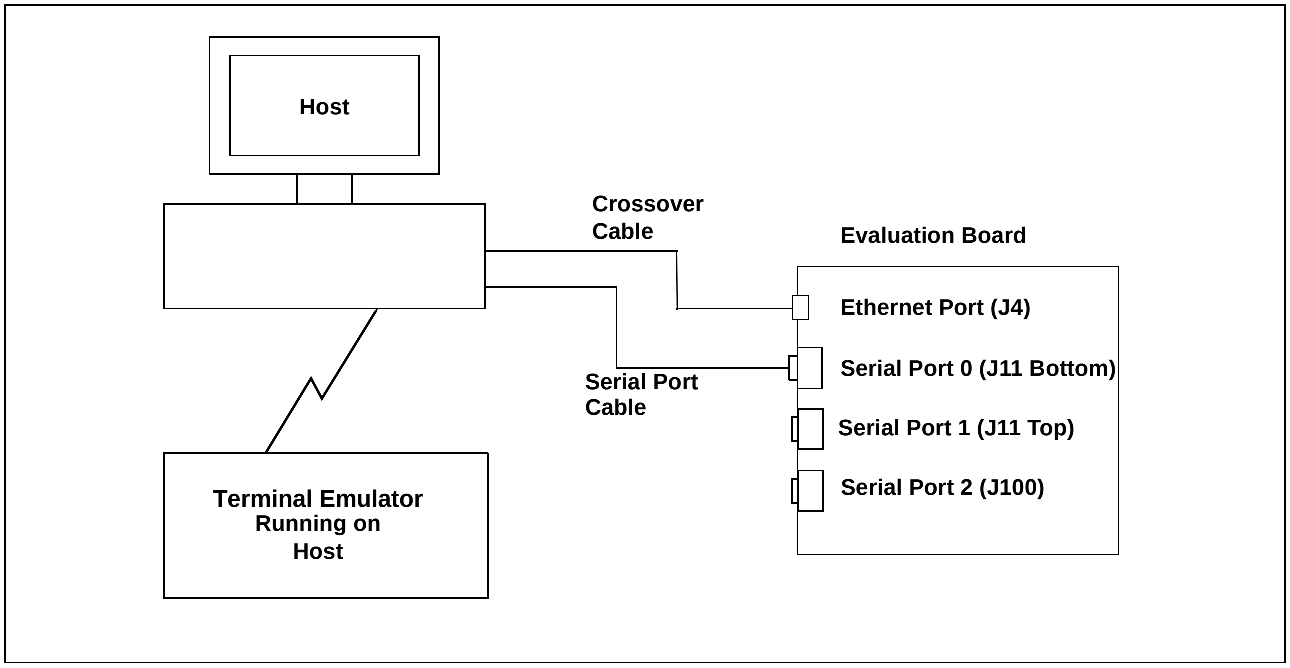
Figure 5-2 shows the connections and signal assignments required in a crossover cable.

Figure 5-2. Signal Assignments in a Crossover Cable

RJ-45 Connector		Cable		RJ-45
Twisted Pair	Signal Name	Pin	Pin	Signal Name
1	TD +	1	1	TD +
1	TD –	2	2	TD –
2	RD +	3	3	RD +
2	RD –	6	6	RD –
3,4	(Not used)	4, 5, 7, 8	4, 5, 7, 8	(Not used)

Figure 5-3 shows a point-to-point Ethernet connection using the provided crossover cable.

Figure 5-3. Point-to-Point Ethernet Connection



If the connection is to be made to an existing Ethernet network, users should see their network administrator to ensure correct connectivity.

5.2 Using a Terminal Emulator

The PowerPC Initialization Boot Software (PIBS) transmits and receives data through Serial Port 0 (J11 Bottom) on the board. PIBS can be accessed by connecting a VT100 (or compatible) terminal directly to Serial Port 0 on the board or by using a terminal emulator running on the host. When using a terminal emulator, access is gained by using a null-modem cable to connect Serial Port 0 on the board to an available serial (or COM) port on the host system. The terminal emulator session should be set for 9600 baud, 8 data bits, no parity, 1 stop bit, and no flow control if the **uart0baudrate** PIBS variable is not set.

5.2.1 PC Terminal Emulation

After the required host-to-board connections have been made and power has been supplied to the board, the Windows HyperTerminal program can be used as a terminal emulator to support communications with the PIBS monitor. To launch HyperTerminal, select Start **|** Programs **|** Accessories **|** Communications **|** HyperTerminal.

Follow the instructions in the Hyperterminal installation window provided by your system and select:

Connection type: Direct
Bits per second: 9600
Data bits: 8
Parity: None
Stop Bits: 1
Flow Control: None

After resetting the board, the PIBS shell prompt should display in the HyperTerminal window. If it does not, check the HyperTerminal settings and the connections between the host and the board.

5.3 Board Reset

When the connections have been established, press the board’s On/Off switch to apply power to the board. To reset the processor and the communications controllers, press the Reset switch. After PIBS (resident in flash) initializes the processor and board peripherals, and a correctly configured terminal emulator is attached to Serial Port 0 (J11 Bottom) on the board, the PIBS shell prompt is displayed. See *PowerPC Initialization Boot Software (PIBS)* on page 29 for details about PIBS operation.

Part II. PowerPC Initialization Boot Software (PIBS)

6. PIBS Overview

PowerPC Initialization Boot Software (PIBS) initializes the PowerPC microprocessor and the system-level board components and provides a way to load and execute application programs.

For the PPC440SP platform, PIBS consists of two bootable images: PIBS1 and PIBS2. The PIBS1 image enables system recovery if the PIBS2 image becomes corrupted or the unit is unable to boot because of invalid PIBS data. PIBS2 contains all other PIBS functionality.

All user interaction with PIBS occurs through the serial port. The serial port must be configured for 9600 baud, with 8 data bits, no parity, one stop bit, and no flow control. PIBS interaction takes place through UART0 using a traditional 9 pin serial null-modem cable through which the user interacts with the PIBS shell. The serial port speed can be set using the **set uart0baudrate=my_speed** command.

6.1 PPC440SP External Hardware Support

PIBS for the PPC440SP platform supports the PPC440SP on-chip Ethernet port.

Note: To use the Ethernet on the evaluation board an Ethernet hardware address must be set. A hardware address can be set using the set hwdaddr0=*your-hardware-address* command. See the command description on page 71 for details. After the address is set, the board must be reset so that the emacs are programmed with the correct hardware address.

6.2 Quick Start

Connect the PPC440SP evaluation board’s UART0 connector to the PC using a 9 pin null-modem cable. (The board’s UART0 connector is the 9-pin connector closest to the board.) On the host computer, start the terminal emulation software (on a Windows PC, Hyperterminal can be used as a terminal emulator). The required serial port connection settings are 9600 baud, 8 data bits, no parity, one stop bit, and no flow control. After the terminal emulation software package is running, power up the board. At this point, the PIBS shell prompt should appear on the screen. If it does not, verify the terminal emulation software configuration and make sure the board is powered on. After the PIBS shell displays, all PIBS shell commands can be issued. The PIBS commands are explained in *PIBS Commands* on page 33.

6.3 Programming PIBS2 Software in FLASH

This section describes different methods that can be used to program PIBS2 software in FLASH.

1. The PIBS **storefile** command can be used to reprogram PIBS2 into the FLASH with the **finalopibs2.bin** file. See *storefile* on page 61 for details.
2. PIBS1 and PIBS2 can be placed initially into FLASH using the **flash_pibs.cmd** RISCWatch command file. (RISCWatch version 5.1 is required.) To do this, first make sure the serial port is connected and that the host computer is running the terminal emulation software. For the Windows PC environment, Hyperterminal can be used. The terminal emulation settings required are the same ones used to operate the PIBS shell. After establishing the serial port connection, locate the files described in *Table 6-1* on page 32, add the path (on which the files were downloaded) to the RISCWatch file path search list (using RISCWatch **srchpath add** command), and execute the **flash_pibs.cmd** command file. After the command file finishes execution, if RISCWatch does not automatically enter “running” mode, type the **run** RISCWatch command and press <ENTER>.

Table 6-1. Files Needed to Install PIBS onto the PPC440SP Platform

File Name	Description
finaloddr	ELF file containing DDR SDRAM setup code
finalopibs1.bin	Binary file containing PIBS1
finalopibs2.bin	Binary file containing PIBS2
finaloram	ELF file containing SRAM setup code update
finaloutil	ELF file for PIBS1/PIBS2 FLASH utility
flash_pibs.cmd	RISCWatch command file

3. If RISCWatch is not available and PIBS2 is corrupted, PIBS1 can be used to reprogram PIBS2. To do this, during the first two seconds while PIBS is booting, press 1 to stop the boot sequence. When the boot sequence has stopped, the PIBS2 file **finalopibs2.bin** can be transferred to the board using the serial port and Kermit file transfer. For Windows PC users, HyperTerminal can be used to transfer the file. When the file transfer is complete, PIBS1 will burn the PIBS2 file in flash.

6.4 Recovering PIBS2

During the first two seconds while PIBS is booting, pressing 1 stops the boot sequence process and PIBS1 appears, allowing the user to reprogram the PIBS data and the PIBS2 software. If 1 is not pressed, the PIBS2 shell command prompt displays. Always use PIBS2 for normal use. Only use PIBS1 if PIBS2 is corrupted and needs to be reprogrammed.

6.5 Programming PIBS1 Software In FLASH

This section describes different methods that can be used to program PIBS1 software in FLASH.

1. Using PIBS2, the PIBS **storefile** command can be used to reprogram PIBS1 into the FLASH with the **finalopibs1.bin** file. See *storefile* on page 61 for details.
2. Using the **flash_pibs.cmd** RISCWatch command file. See *Programming PIBS2 Software in FLASH* on page 31.

6.6 PIBS Memory Map

PIBS1 occupies the top 256KB of memory (address from 0xFFFC0000 to 0xFFFFFFFF). The PIBS data occupies the next 64KB of FLASH memory (address from 0xFFFB0000 to 0xFFFBFFFF). PIBS2 occupies the next 704KB of memory (address from 0xFFF00000 to 0xFFFAFFFF). During runtime, PIBS uses RAM memory from 0x00001000 to 0x001FFFFF for PIBS stack space, heap and data sections. *Table 6-2* describes PIBS memory map.

Table 6-2. PIBS Memory Map

Physical Address (36 bits)	Logical Address (32 bits)	Contents
0x1FFFFF000 - 0x1FFFFFFFF	0xFFFFF000 - 0xFFFFFFFF	PIBS 1 CPU core installation
0x1FFFC0000 - 0x1FFFFEFFF	0xFFFC0000 - 0xFFFFEFFF	All other PIBS 1 code, including interrupt vectors
0x1FFFB0000 - 0x1FFFBFFFF	0xFFFB0000 - 0xFFFBFFFF	PIBS database
0x1FFFAF000 - 0x1FFFAFFFF	0xFFFAF000 - 0xFFFAFFFF	PIBS 2 CPU core initialization
0x1FFF00000 - 0x1FFFAEFFF	0xFFF00000 - 0xFFFAEFFF	All other PIBS 2 code, including interrupt vectors
0x1FFE00000 - 0x1FFEFFFFF	0xFFE00000 - 0xFFEFFFFF	External SRAM
0x1FFC00000 - 0x1FFDFFFFF	0xFFC00000 - 0xFFDFFFFF	Unused
0x1FF800000 - 0x1FFBFFFFF	0xFF800000 - 0xFFBFFFFF	User (Large) FLASH Space
0x000200000 - 0x07FFFFFFF	0x00200000 - 0x7FFFFFFF	Application Space (RAM)
0x000001000 - 0x0001FFFFF	0x00001000 - 0x001FFFFF	PIBS RAM
0x000000000 - 0x000000FFF	0x0000000 - 0x0000FFF	Application Space (RAM)

7. PIBS Commands

The following sections provide detailed descriptions of all the PIBS commands.

7.1 Command Syntax

This section explains the commands available to the user from the PIBS shell. The following explains the syntax used in the examples.

- `Courier` font indicates text typed in on the PIBS command line.
- *Italic* text indicates a general name substitution.
- **Bold** text indicates a command name, variable name, or file name.
- Text of the form (`option1|option2|option3`) indicates that the user must type in one, and only one, of the options.
- Text of the form (*general_text*)+ indicates the *general_text* token must be replicated at least once and possibly multiple times.
- `<ENTER>` indicates either the Enter or Return (↵) key on the computer keyboard is pressed.
- Text of the form [option] indicates the user may or may not include the option.
- The pound sign (#) indicates that a number should be substituted in its place.

7.2 List of PIBS Commands

alias
arp
boardinfo
bootfile
chipclk, chipclkreg
copyright
display
echo
enetmode
getbootstrap
help
ifconfig
l2config
linkmon_stop
netstat
pcixfreq
ping
reset
route
set
storefile
uartclk
unalias
version

7.3 Command Descriptions

The following pages provide detailed descriptions for each PIBS command in alphabetical order. The information provided includes a synopsis of the command, the syntax of the command, descriptions of all command parameters, and examples of command usage.

Synopsis

Assigns another name to a PIBS command

Syntax

alias *alias_name=system_command*

Parameters

alias_name	A name to be used as an alias.
system_command	Name of a system command to be aliased

Example

```
alias my_help=help <ENTER>
```

Result: Typing in **my_help** on the command line now brings up the help menu.

Notes

None

Synopsis

Configures system arp (address resolution protocol) table entries

Syntax

arp ([-n] *hostname*|[-n] -a|-d *hostname*|-s *hostname ether_addr* [temp] [pub]|-f [#])

Parameters

hostname	Displays the ARP entry for the hostname (can be an IP address or a name).
-n	Suppresses host name display and DNS lookup
-a	Displays all ARP entries
-d	Deletes an ARP entry
-s	Sets a static ARP entry
ether_addr	The twelve digit Ethernet hardware address supplied for the static ARP entry (must be specified in hex as XX:XX:XX:XX:XX:XX).
temp	Defines if the ARP entry is temporary or permanent. If temp is specified, a timeout is involved; if not specified, the entry is permanent.
pub	Defines if the ARP entry is public or private. A public entry can be used for a reverse arp lookup.
-f	Flushes all ARP table entries associated with a specific interface. If the interface number [#] is not provided, all interfaces are flushed.

Example

arp -a <ENTER>

Result: Displays all arp table entries.

arp 192.168.0.1 <ENTER>

Result: Displays the arp entry for 192.168.0.1 if it exists.

Notes

None

Synopsis

Displays system information and settings

Syntax

boardinfo

Parameters

None

Example

boardinfo <ENTER>

Result: Displays the current system settings. Depending on the PIBS version and the hardware platform, the results shown here may be different than what is actually displayed.

```
board config data version: 1.00
processor name          : 440SP
processor PVR value     : 0x53221850
total SDRAM DDR memory  : 536870912
ECC enabled
system clk period (ps)  : 30000
system clk frequency (Hz): 33333333
VCO frequency           : 1000000000 - PLL Bypassed in PPC440SP
CPU frequency           : 500000000
CPU frequency ind. method: 490043904
PLB frequency           : 166666666
OPB frequency           : 83333333
EBC frequency           : 83333333
MAL frequency           : 83333333
DDR frequency           : 166666666
PCI0 frequency          : 33333333
PCI1 frequency          : 33333333
PCI2 frequency          : 33333333
TMR frequency           : 25000000
uart 0 clk frequency    : 11059200
uart 1 clk frequency    : 11059200
uart 2 clk frequency    : 11059200
On-board PCI0 Arbiter   : External
On-board PCI1 Arbiter   : External
On-board PCI2 Arbiter   : External
Ethernet mode           : 10/100
SRAM address            : 0xffe00000
Small Flash address     : 0xfff00000
```

Notes

Table 7-1 explains the meaning of the fields.

Table 7-1. Board Configuration Information

Field Name	Board config field description
board config data version	The current version of the board configuration data structure.
processor name	The name of the PowerPC processor used in the specific platform.
processor PVR value	The PVR register contents of the PowerPC processor.
total SDRAM memory (byte)	The amount in bytes of SDRAM or DDR-SDRAM memory installed in the unit.
ECC	The ECC feature status: Enabled or Disabled
system clk period (ps)	The PowerPC chip input system clock period displayed in picoseconds.
system clk frequency (Hz)	The PowerPC chip input system clock frequency displayed in Hertz.
VCO frequency (Hz)	The Voltage Controlled Oscillator frequency of the PowerPC chip system PLL displayed in Hertz.
CPU frequency (Hz)	The calculated CPU core frequency, in Hertz, based on the input system clock frequency, PowerPC chip system PLL, and clock tree divider settings
CPU frequency ind. method (Hz)	The CPU core frequency as indicated by detecting the number of instructions executed during a specific time period. This can be used to verify the validity of the CPU core frequency.
PLB frequency (Hz)	The calculated Processor Local Bus frequency, in Hertz, based on the input system clock frequency, PowerPC chip system PLL, and clock tree divider settings.
OPB frequency (Hz)	The calculated On-chip Peripheral Bus frequency, in Hertz, based on the input system clock frequency, PowerPC chip system PLL, and clock tree divider settings.
EBC frequency (Hz)	The calculated External Bus Controller frequency, in Hertz, based on the input system clock frequency, PowerPC chip system PLL, and clock tree divider settings.
MAL frequency (Hz)	The calculated Media Access Layer frequency, in Hertz, based on the input system clock frequency, PowerPC chip system PLL, and clock tree divider settings.
DDR frequency	
PCI 0 frequency (Hz)	The frequency, in Hertz, of the core0 PCI-X bus.
PCI 1 frequency (Hz)	The frequency, in Hertz, of the core1 PCI-X bus.
PCI 2 frequency (Hz)	The frequency, in Hertz, of the core2 PCI-X bus.
TMR frequency (Hz)	The frequency, in Hertz, of the CPU core timebase.
uart0 clk frequency (Hz)	The frequency, in Hertz, of the UART0 core serial clock input.
uart1 clk frequency (Hz)	The frequency, in Hertz, of the UART1 core serial clock input.
uart2 clk frequency (Hz)	The frequency, in Hertz, of the UART2 core serial clock input.
On-board PCI0 Arbiter	Displays whether the PowerPC chip PCI arbiter (internal setting) or the board-supplied arbiter (external setting) is being used.
On-board PCI1 Arbiter	Displays whether the PowerPC chip PCI arbiter (internal setting) or the board-supplied arbiter (external setting) is being used.
On-board PCI2 Arbiter	Displays whether the PowerPC chip PCI arbiter (internal setting) or the board-supplied arbiter (external setting) is being used.
Ethernet mode	Displays the selected mode of the Ethernet link: 10/1000Mbps or 1000Mbps.
SRAM address	External SRAM address
Small Flash address	Small Flash address

Synopsis

Responsible for file loading, relocating and execution. File formats supported are the bootable image format, ELF file format, and binary file format. See *Bootable Image File Format Description* on page 85 for specific information about the bootable image format. ELF and bootable image formats are relocated by PIBS as indicated by their headers before execution begins. PIBS does not relocate a file stored in the binary file format. With a binary file, execution begins at the *addr* specified. If program relocation is necessary for a binary file, it is the program's responsibility to relocate itself.

After program loading and relocation but before PIBS branches to the user file, the system is put into a state such that system components cannot interfere with user program execution. This includes the following:

1. External interrupts are disabled. MSR.EE = 0
2. Critical interrupts are disabled. MSR.CE = 0
3. PIT and Watchdog timer interrupts are disabled.
4. All PLB masters, besides the CPU core, are disabled or stopped. This includes any PCI-X (if applicable), LCD (if applicable), External Bus Master, MAL and DMA operations.
5. External interrupts are disabled at the interrupt controller level.

PIBS also performs the following steps before branching to the user file.

1. If the EBC FLASH bank is currently marked as read-only, it is now made read/writeable.
2. The data cache is completely flushed and the instruction cache is invalidated.

Before execution of the user file starts, two parameters are passed in through the processor's general purpose registers. These parameters point to data structures residing in RAM, so the user program must not overwrite them unless they are not needed anymore.

1. GPR 3: A pointer to the user-supplied program information, such as boot parameters, located in the PIBS environment variable **userdata**. See *userdata* on page 78 for more information about the **userdata** environment variable.
2. GPR 4: A pointer to the board configuration structure, which contains system information such as CPU frequency, various bus frequencies, and the total amount of memory installed.

Syntax

bootfile (flash|flashb|eth|ethl|ethd|serial|seriall|ram) [*addr*]

Parameters

flash	Relocate and execute a bootable image or ELF file residing in FLASH. If the *addr* option is specified, PIBS tries to locate the bootable image or ELF file at that address. The **bootfile** command will fail if the bootable image or ELF file at the specified address is invalid. If the *addr* option is not used, PIBS uses the **imgaddr** environment value if it is valid. If **imgaddr** is not set to a valid address, PIBS searches through the FLASH and relocates/executes the first valid bootable image or ELF file found. If no valid bootable image or ELF file is found, the **bootfile** command fails.

flashb	Relocate and execute a bootable image or ELF file, or execute a binary file, residing in FLASH. If the *addr* option is specified, PIBS tries to locate a bootable image or ELF file at that address. If a bootable image or ELF file is not located at that address, PIBS treats the file as a binary file and starts program execution by branching to the specified address. If the *addr* option is not used, PIBS uses the **imgaddr** environment value instead if it is valid. If **imgaddr** is not set to a valid address, PIBS searches through the FLASH and branches to the first valid bootable image or ELF file found. If a bootable image or ELF file is located, then the *flashb* option behaves just like the *flash* option. The *flashb* option behaves differently only if the file is not a bootable image or an ELF file.
eth	Load, relocate, and execute a bootable image or ELF file across the network. If the *eth* option is specified, the *addr* option is not used. The *eth*, *ethl* and *ethd* options require a TFTP server/daemon running on the host computer that supplies the user program. See *Configuring TFTP* on page 83 for more information about TFTP server system requirements and installation. PIBS environment variables **ipdstaddr** and **bootfilename** supply the host computer's IP address and directory path, and filename information respectively.
ethl	Similar to the *eth* option but only performs the load step to get the bootable image or ELF file into memory. Program relocation and execution do not take place.
ethd	Similar to the *eth* option but adds debug output messages.
serial	Load, relocate, and execute a bootable image or ELF file using the serial port as the transport medium and Kermit as the file transfer protocol. If using the **serial** option, the *addr* option is not used.
seriall	Similar to the *serial* option but only performs the load step to get the bootable image or ELF file into memory. Program relocation and execution does not take place.
ram	If the **bootfile** command was previously executed with either the *ethl* or *seriall* option to get the bootable image or ELF file loaded into memory, calling **bootfile** with the *ram* option starts program relocation and execution. The *addr* option (if specified) provides the start address of the loaded file. If the *addr* option is not used the default RAM location is used instead.
addr	Used with the *flash*, *flashb*, or *ram* option and specifies where to look in the FLASH or RAM for the bootable image, ELF or binary file.

Example

bootfile flash 0xffa00000 <ENTER>

Result: Relocate/execute a bootable image or ELF file from FLASH address 0xFFA00000

bootfile flash <ENTER>

Result: If the **imgaddr** environment variable is set, the **bootfile** command uses it as the bootable image/ELF file starting address. If the **imgaddr** environment variable is not set, PIBS searches through the FLASH and relocate/execute the first bootable image or ELF file found.

bootfile flashb 0xffa00000 <ENTER>

Result: First, PIBS determines if the file at 0xFFA00000 is a bootable image or ELF file. If so, the file is relocated if necessary and executed. Otherwise, PIBS assumes it is a binary file and branches to address 0xFFA00000.

bootfile flashb <ENTER>

Result: Execute a binary file or relocate/execute a bootable image or ELF file from FLASH address specified by the PIBS **imgaddr** environment variable. If the **imgaddr** environment variable setting is valid, PIBS determines if the file at the **imgaddr** start address is a bootable image or ELF file. If so, the file is appropriately relocated and executed. Otherwise, PIBS assumes it is a binary file and branches to the address defined by **imgaddr**. If the **imgaddr** environment variable address setting is invalid, the **bootfile** command relocates/executes the first bootable image or ELF file found in FLASH storage. If no bootable image or ELF file is found, the command fails.

bootfile eth <ENTER>

Result: Load a bootable image or ELF file from the network and once loaded, perform relocation and branch to the program entry point. PIBS environment variables **ipdstaddr** and **bootfilename** supply the host computer's IP address and directory path/filename respectively. The host computer must be running a TFTP server/daemon and have access to the file at the specified directory path.

bootfile serial <ENTER>

Result: Load a bootable image or ELF file using the serial port kermit file transfer protocol and once loaded, perform relocation and start execution.

Notes

None

Synopsis

Sets the system clock timing or programs the chip clocking registers directly

Syntax

chipclk (reset|prom0|prom1)(bypass|pllouta|plloutb|cpu|ebc) <cpuf> <plbf> <opbf> <ebcf> <malf> <ddrf> <(pci0f|bypass)> <(pci1f|bypass)> <(pci2f|bypass)> [<sysp>]

Parameters

reset	Reset with the new PLL/clock tree settings.
prom0	Store the PLL/CLK tree strap values into EEPROM0
prom1	Store the PLL/CLK tree strap values into EEPROM1
bypass	Disable the PLL
pllouta	PLL output A is the feedback path
plloutb	PLL output B is the feedback path
cpu	CPU clock is the PLL feedback path
ebc	EBC clock is the PLL feedback path
cpuf	CPU frequency in Hertz
plbf	PLB frequency in Hertz
opbf	OPB frequency in Hertz
ebcf	EBC frequency in Hertz
malf	MAL frequency in Hertz
ddrf	DDR frequency in Hertz
pci0f	PCI0 frequency in Hertz or bypass PCI0 PLL feedback (bypass)
pci1f	PCI1 frequency in Hertz or bypass PCI1 PLL feedback (bypass)
pci2f	PCI2 frequency in Hertz or bypass PCI2 PLL feedback (bypass)
sysp	System clock period in picoseconds

Example

```
chipclk prom0 pllouta 500000000 166666666 83333333 83333333 83333333 166666666
33333333 33333333 33333333 30000
```

Result: The device is programmed to the following settings.

Table 7-2. Chipclk Example 1 Command Results

Name	Value
Boot Method	Store PLL/CLK tree strapping in EEPROM0
PLL Status	Enabled feedback path is PLL OUTPUTA

Table 7-2. Chipclk Example 1 Command Results (Continued)

CPU Frequency	500000000 Hz
PLB Frequency	166666666 Hz
OPB Frequency	83333333 Hz
EBC Frequency	83333333 Hz
MAL Frequency	83333333 Hz
DDR Frequency	166666666 Hz
PCI0 Frequency	33333333 Hz
PCI1 Frequency	33333333 Hz
PCI2 Frequency	33333333 Hz
System clock period in picoseconds	30 000

Note

Note1: If the method is \"reset\" the system clock period must not be specified.

If the method is \"prom0\" or \"prom1\", the system clock period must be specified.

Note2: In the following example:

chipclk reset cpu 500000000 166666666 83333333 83333333 83333333 166666666 33333333 0 bypass

PCI1 is not used and must be set to 0, and PCI2 is set to PCI PLL feedback bypass mode.

Synopsis

Allows direct programming of the serial device strap (SDR0_SDSTP0, SDR0_SDSTP1, SDR0_SDSTP2) and CPR0_PRIMAD[PRADV0] registers for chip bootstrapping

Syntax

chipclkreg (reset|prom0|prom1) <primad> <sdstp0> <sdstp1> <sdstp2>

Parameters

reset	If specified, PIBS loads the new *primad*, *sdstp0, sdstp1,* and *sdstp2* parameter settings into their corresponding CPR0 and SDR0 registers. A chip reset occurs so that the new CPR0 and SDR0 register settings take effect. If using the reset, prom0, or prom1 boot method, the *primad* parameter must also be specified and must be set as described under that parameter.
prom0	If specified, PIBS stores the new SDR0_SDSTP0, SDR0_SDSTP1, and SDR0_SDSTP2 register values (supplied by the *sdstp0*, *sdstp1*, and *sdstp2* parameters) into the EEPROM0 bootprom.
prom1	If specified, PIBS stores the new SDR0_SDSTP0, SDR0_SDSTP1, and SDR0_SDSTP2 register values (supplied by the *sdstp0*, *sdstp1*, and *sdstp2* parameters) into the EEPROM1 bootprom.
primad	If using the reset boot method, this parameter must be specified and must set the CPR0_PRIMAD[PRADV0] bit to a value from 1 to 8. If using either the prom0 or prom1 reset boot method, this parameter must also be specified and must set the CPR0_PRIMAD[PRADV0] bit to a value of 1.
sdstp0	The 32-bit Serial Device Strap Register 0 setting (in hexadecimal).
sdstp1	The 32-bit Serial Device Strap Register 1 setting (in hexadecimal).
sdstp2	The 32-bit Serial Device Strap Register 2 setting (in hexadecimal).

Example

```
chipclkreg prom0 1 857ce2e6 05800100 <ENTER>
```

Result: A value of 0x857ce2e6 is programmed into EEPROM0 for the SDR0_SDSTP0 serial device strap register setting. A value of 0x05800100 is programmed into EEPROM0 for the SDR0_SDSTP1 serial device strap register setting. For the new SDR0_SDSTP0 and SDR0_SDSTP1 bootstrap settings to take effect, the evaluation board must be reset with the bootstrap controller enabled and set to EEPROM0.

```
chipclkreg reset 1 857ce2e6 05800100 <ENTER>
```

Result: The 0x857ce2e6 value is decoded into separate bit fields, and each CPR0/SPR0 register with the corresponding bit definition is updated accordingly. See the preceding *reset* parameter definition for a list of the processor registers that get updated. Last, PIBS resets the processor such that the recently updated CPR0/SPR0 register settings take effect.

Notes

None

Synopsis

Displays the PIBS copyright string

Syntax

copyright

Parameters

None

Example

copyright

Result: Shows the PIBS version copyright on the display.

```
copyright
-----------------------------------------------
COPYRIGHT IBM COPORATION 2001, 2004
LICENSE MATERIAL - PROGRAM PROPERTY OF IBM
----------------------------------------------------
-----------------------------------------------
Copyright (C) 2005
Applied Micro Circuits Corporation
AMCC
All Rights Reserved
```

Notes

None

Synopsis

Displays memory contents as translated by the MMU. Memory contents are displayed in their hexadecimal value as well as their ASCII character value.

Syntax

display *mem_addr* [*num_bytes*]

Parameters

mem_addr	The internal 440 core 32 bit effective start address. This value can be either a 32-bit decimal or hexadecimal value. Hexadecimal values are proceeded by a 0x specifier.
num_bytes	The number of memory bytes to display, starting with the start address defined by the *mem_addr* parameter. This value is rounded up to the nearest value evenly divisible by 4. (For example, if 89 bytes is requested, 92 will be displayed.) If this parameter is not specified on the command line, 32 bytes is the default value used.

Example

display 300 99 <ENTER>

```
Result: Displays 100 bytes starting at address 0x12c (300 decimal).
0000012c:  0000012c 00000130 00000134 00000138 | ..., ...0 ...4 ...8 |
0000013c:  0000013c 00000140 00000144 00000148 | ...< ...@ ...D ...H |
0000014c:  0000014c 00000150 00000154 00000158 | ...L ...P ...T ...X |
0000015c:  0000015c 00000160 00000164 00000168 | ...\ ...` ...d ...h |
0000016c:  0000016c 00000170 00000174 00000178 | ...l ...p ...t ...x |
0000017c:  0000017c 00000180 00000184 00000188 | ...| .... .... .... |
0000018c:  0000018c                            | ....                |
```

display 0x300 101 <ENTER>

```
Result: Displays 104 bytes starting at address 0x300
00000300:  00000300 00000304 00000308 0000030c | .... .... .... .... |
00000310:  00000310 00000314 00000318 0000031c | .... .... .... .... |
00000320:  00000320 00000324 00000328 0000032c | ...  ...$ ...( ..., |
00000330:  00000330 00000334 00000338 0000033c | ...0 ...4 ...8 ...< |
00000340:  00000340 00000344 00000348 0000034c | ...@ ...D ...H ...L |
00000350:  00000350 00000354 00000358 0000035c | ...P ...T ...X ...\ |
00000360:  00000360 00000364                   | ...` ...d
```

Notes

None

Synopsis

Displays the value of a PIBS variable, or echoes text back to the display

Syntax

echo ($*variablename*|*text*)

Parameters

variablename	A valid name of a PIBS system variable. See *PIBS Environment Variables* on page 65 for valid PIBS system variable names and their descriptions.
text	Any generic text to get outputted back to the display.

Example

echo $emacdhcp0 <ENTER>

Result: The value of the **emacdhcp0** environment variable is outputted to the display. For example, if the **emacdhcp0** environment variable is set to true, then "true" appears on the display.

echo $bootfilename <ENTER>

Result: The value of the **bootfilename** environment variable appears on the display.

echo generic text <ENTER>

Result: "generic text" appears on the display.

Notes

None

Synopsis

Controls or displays the current Ethernet mode. If the command is entered without any parameters, the current status of the enet link is displayed.

Syntax

enetmode (autoneg|no_autoneg} (10|100|1000) (half|full)

Parameters

autoneg|no_autoneg	Ethernet negotiation mode.
10|100|1000	Ethernet speed in Mbits.
half|full	Ethernet duplex mode.

Example

enetmode <ENTER>

Result: enet mode: link up _ 100 _ full _ autonegotiation

enetmode no_autoneg 1000 full <ENTER>

Result: Gigabit full duplex mode is set.

Notes

If autoneg is selected, Ethernet speed and duplex mode are negotiated automatically.

Synopsis

Returns the system's current bootstrapped values or the settings stored in an onboard EEPROM. The settings returned are formatted into SDR0_SDTP0, SDR0_SDTP1, SDR0_SDTP2, and SDR0_SDTP3 values as defined by their respective register bit field definitions.

Syntax

getbootstrap (core|prom0|prom1|sdstp)

Parameters

core	Returns the current system PLL, clock tree, PCI, SDR0_CUST0, SDR0_CUST1, and other miscellaneous settings. The settings returned are formatted into SDR0_SDTP0, SDR0_SDTP1, SDR0_SDTP2, and SDR0_SDTP3 values as defined by their respective register definitions.
prom0	Returns the bootstrap settings in EEPROM0.
prom1	Returns the bootstrap settings in EEPROM1.
sdstp	Returns the actual SDR0_SDTP0, SDR0_SDTP1, SDR0_SDTP2, and SDR0_SDTP3 register settings. Note that the *sdstp* and *core* parameters may or may not produce the same results.

Example

getbootstrap prom0 <ENTER>

Result: The SDR0_SDTP0, SDR0_SDTP1, SDR0_SDTP2, and SDR0_SDTP3 values are read from EEPROM0 and displayed on the PIBS shell.

getbootstrap sdstp <ENTER>

Result: PIBS returns the bootstrapped settings as stored in registers SDR0_SDTP0, SDR0_SDTP1, SDR0_SDTP2, and SDR0_SDTP3.

Notes

None

Synopsis

Displays the help screen

Syntax

help [*command_name*]

Parameters

command_name	If provided, displays help information for *command_name*.

Example

help <ENTER>

Result: Displays and gives a brief description of the available PIBS commands. Depending on the PIBS version level and the hardware platform, the help results shown here may be different than what is actually displayed.

```
Very simple shell for PIBS can be used to:
1. Execute the following commands:
  alias          : assigns an alias to a command
  arp            : modifies system arp table entries
  boardinfo      : displays the current system settings
  bootfile       : loads file and jumps to the entry point
  chipclk        : sets chip clock bootstrapping
  chipclkreg     : programs the serial device strap settings
  copyright      : displays the current copyright information
  display        : displays memory contents
  echo           : displays the value of a PIBS variable
  enetmode       : resets/stores Ethernet mode information
  getbootstrap   : returns bootstrapping information
  help           : displays command-specific help information
  ifconfig       : configures/displays network interface info
  l2config       : configures the onchip L2 cache/SRAM operation
  linkmon_stop   : disables link monitor for all network devices
  netstat        :  displays network statistics
  pcixfreq       : sets PCI-0:3 bus freq advertised to the adapters
  ping           : sends an echo request to a network host
  reset          : resets the board
  route          : manipulates routing tables
  set            : assigns a value to a PIBS variable
  storefile      : stores file in flash
  uartclk        : controls the UART input clock source
  unalias        : removes an alias to a command
  version        : displays the PIBS version string

2. Set and display PIBS variables. PIBS variable values
  are preserved across reboots. PIBS variables are:
  aliaslist
  autoboot
  autobootdelay
  bootfilename
```

emacdhcp0
hwdaddr0
ifconfigcmd0
imgaddr
ipdstaddr0
l2configcmd
routecmd
uartclkcmd
userdata

PIBS variables are assigned using the set command, and are displayed using the echo command.

3. Display command help. For example: help bootfile.

help alias <ENTER>

Result: Gives a more detailed description for the **alias** command.

Notes

None

Synopsis

Configures the network interface

Syntax

ifconfig [*addr* [*dest_ad*]] [up] [down] [*netmask*] [*metric*] [arp|-arp]

Parameters

addr	The host address
dest_ad	The destination address
up	Bring the interface up
down	Bring the interface down
netmask	Specifies the network mask
metric	Number of hops to destination
arp	Indicates use of arp
-arp	Indicates that arp is not used

Example

ifconfig up <ENTER>

Result: Brings the interface up

ifconfig down <ENTER>

Result: Brings the interface down

ifconfig 192.168.0.1 <ENTER>

Result: Sets the interface's TCP/IP address to 192.168.0.1

Notes

When using DHCP to provide the IP information, do not use the **ifconfig** command to change system settings.

Synopsis

Configures the onchip SRAM as an L2 cache or as an SRAM. If enabled as an L2 cache, further control is provided to allow setup as a D-cache, as an I-cache or both enabled simultaneously.

Syntax

l2config (sram|icache|dcache|cache)

Parameters

sram	Enable the on-chip SRAM memory as a 256KB SRAM.
icache	Enable the on-chip SRAM memory as a 256KB I-cache.
dcache	Enable the on-chip SRAM memory as a 256KB D-cache.
cache	Enable the on-chip SRAM memory as a 256KB I-cache and D-cache.

Example

```
l2config icache <ENTER>
```

Result: Onchip memory is configured as L2 I-cache only storage.

Notes

None

Synopsis

Disables the link monitor for all network devices. The link monitor brings an interface up if link is detected. If link status is lost, the link monitor brings the corresponding interface down.

Syntax

linkmon_stop

Parameters

None

Example

linkmon_stop <ENTER>

Result: The link monitor is stopped for all network devices. The only way to restart the link monitor is to reset the evaluation board.

Notes

None

Synopsis

Displays network statistics

Syntax

netstat -i -t -m -s -p [-d|-b #|-c #|-i #|-M #|-m #|-n #|-r #|-s #|-t #|-u #|-z #]

Parameters

-i	Display interface information
-t	Display TCP control block information
-m	Display TCP/IP buffer usage
-s	Display routing statistics
-p	Set/display various TCP/IP values
-d	Display current TCP/IP options
-b	Set sb_max
-c	Set connect timeout
-i	Set initial KEEPALIVE timeout
-M	Set MSL2 timeout (in 500ms ticks)
-m	Set MSL2 timeout for local sockets
-n	Set KEEPALIVE retry timeout
-r	Set tcp_recvspace
-s	Set tcp_sendspace
-t	Set KEEPALIVE retry timeout count
-u	Set udp_recvspace
-z	Set udp_sendspace
	All timer values are in units of 500 ms. KEEPALIVE timeout formula is init timeout + (retry timeout* retry timeout count)

Example

netstat -s <ENTER>

Result: Displays all routing statistics.

netstat -i <ENTER>

Result: Displays all routing statistics.

Notes

None

Synopsis

Sets the PCI-X frequency selection installed in PCI-X 133 MHz-capable adapter cards. The command does not actually control the PCI-X frequency; it sets the PCI-X frequency of the PCI-X 133 MHz-capable adapter cards. The command allows the user to decrease the PCI-X bus frequency if multiple PCI-X 133 MHz-capable adapter cards are installed. This command is not needed if one or more of the PCI-X adapter cards is not 133 MHz capable; it is also not needed if the system is running in PCI conventional mode.

Syntax

pcixfreq (pci0|pci1|pci2) (reset|prom0|prom1) (0|1|2)

Parameters

(pci0|pci1|pci2)	Identifies the PCI core to configure
reset	PIBS stores the PCI-X settings into register SDR0_XCR[PXFS], and a chip reset takes place so the new selection takes effect.
prom0	PIBS stores the PCI-X settings into the EEPROM0 bootprom.
prom1	PIBS stores the PCI-X settings into the EEPROM1 bootprom.
0	100 -133 MHz frequency selection.
1	66 -100 MHz frequency selection.
2	50 -133 MHz frequency selection.

Example

```
pcixfreq pci0 prom0 0 <ENTER>
```

Result: For pci0, the 100 - 133 MHz PCI-X frequency selection is written out to EEPROM 0.

Notes

None

Synopsis

Sends an icmp echo request to a network host

Syntax

ping [-r] [-v] *host* [*size*] [*packets*]

Parameters

-r	Bypass the normal routing tables
-v	Verbose output
host	IP address destination
size	The size in bytes of the data portion of the ping packet.
packets	The number of echo request packets to send.

Example

ping 192.168.0.1 256 4 <ENTER>

Result: Transmits four icmp ping packets to IP address 192.168.0.1. Each ping packet includes a 256 byte data section.

Notes

If the *size* parameter is not specified, the data portion of the ping packet defaults to 64 bytes. If the *packets* parameter is not specified, the **ping** command defaults to sending out three icmp echo requests.

Synopsis

Resets the evaluation board causing a reboot of the firmware

Syntax

reset (core|chip|sys)

Parameters

core	Reset the 440 processor core.
chip	Reset the 440SP chip.
sys	Reset the 440SP chip and board peripherals.

Example

reset sys <ENTER>

Result: Resets the system, including the chip and onboard peripherals.

Notes

None

Synopsis

Manipulates network routing tables

Syntax

route [-dnqv] *command* [[-<*qualifiers*>] *args*]

Parameters

-d	Debug only.
-n	Do not query for host name.
-q	Quiet.
-v	Verbose.
command	Get, change, add, delete, monitor, flush destination and gateway address.
-args	Command line args for the command.

Example

route add 192.168.0.40 <ENTER>

Result: IP address 192.168.0.40 is added as a route to the router table.

route delete 192.168.0.40 <ENTER>

Result: Delete router entry 192.168.0.40.

Notes

None

Synopsis

Assigns a value to a system environment variable or displays the current system variables and their values

Syntax #1

set [*pibs_variable*=*value*]

Parameters

pibs_variable	Name of the variable in PIBS to use. Type in **set** <ENTER> on the command line for a complete list.
value	Value to which to set the variable.

Example

set emacdhcp0=true <ENTER>

Result: The next time the board is booted, PIBS starts up the DHCP client service for emac0.

set <ENTER>

Result: The current setting of all the PIBS variables is outputted to the display.

Notes

Depending on the PIBS version level and the hardware platform, the environment variable name in the above example may be different than what is actually used.

Synopsis

Stores a file into system FLASH. A file may either be a bootable image, ELF or raw binary file.

Syntax

storefile (bin|pibs|p1)(eth|serial)[*addr*]

Parameters

bin	Takes a binary, bootable image or ELF file and stores it into FLASH. If using the *bin* parameter, the *addr* parameter must be specified.
pibs	Used to reprogram PIBS2 into the FLASH. The *addr parameter* is ignored with this parameter. If using the *pibs* parameter, the source PIBS2 file must be a binary file.
p1	Used to reprogram PIBS1 into the FLASH. The *addr* parameter is ignored with this parameter. If using the *p1* parameter, the source PIBS1 file must be a binary file.
eth	Load the file from another host over the network. The *eth* parameter requires a TFTP server/daemon running on the host computer that supplies the user program. See *Configuring TFTP* on page 83 for more information on TFTP server system requirements and installation. PIBS environment variables **ipdstaddr** and **bootfilename** supply the host computer's IP address and directory path/filename.
serial	Load the file over the serial port using kermit file transfer protocol.
addr	Address to place the loaded file into FLASH. This parameter must be specified if using the *bin* parameter.

Example

storefile bin eth 0xFFA00000 <ENTER>

Result: Load in a file over the network and store it into FLASH at address 0xffa00000.

storefile bin serial 0xFFA00000 <ENTER>

Result: Load in a file over the serial port and store it into FLASH at address 0xffa00000.

storefile pibs eth <ENTER>

Result: Load in PIBS2 over the network and store it into FLASH. A reboot soon follows.

storefile pibs serial <ENTER>

Result: Load in PIBS2 over the serial port and store it into FLASH. A reboot soon follows.

Notes

None

Synopsis

Controls the source of the UART input clock. Sets UART clocking to internal or external chip clocking.

Syntax

uartclk (uart0|uart1|uart2) (int|ext)

Parameters

uart0	The clocking change is for UART0.
uart1	The clocking change is for UART1.
uart2	The clocking change is for UART2.
int	Switch the specified uart to use the chip provided UART clock.
ext	Switch the specified uart to use the board provided UART clock.

Example

```
uartclk uart0 int <ENTER>
```

Result: UART0 is set up to use the internal serial clock.

```
uartclk uart1 ext <ENTER>
```

Result: UART1 is set up to use the externally supplied board serial clock.

Notes

None

Synopsis

Removes an alias to a command

Syntax

unalias *alias_name*

Parameters

alias_name	Name of the alias to remove from the system.

Example

```
unalias my_help <ENTER>
```

Result: If **my_help** was an alias to another command it no longer works.

Notes

None

Synopsis

Displays the PIBS version string

Syntax

version

Parameters

None

Example

version <ENTER>

Result: The PIBS version string is outputted to the display.

Notes

None

8. PIBS Environment Variables

The following sections provide detailed information about the PIBS environment variables.

8.1 Environment variable Overview

The PIBS environment variables fall into two categories. *Table 8-1* lists variables that control PIBS system initialization before the shell comes up. *Table 8-2* lists variables that indirectly control the function of various PIBS shell commands.

Table 8-1. PIBS System Initialization Environment Variables

Variable name	Variable description
autoboot	Controls automatic program loading on board bootup
autobootdelay	Controls the auto boot delay
emacdhcp0	Enables/disables DHCP support on EMAC0
hwdaddr0	The MAC hardware address for EMAC0
ifconfigcmd0	Controls Ethernet device initialization during PIBS boot for Emac0
l2configcmd	Controls L2/SRAM configuration
PCI Scanning	Controls the automatic PCIX bus scanning
pinconfig	Selects the shared I/O
routecmd	Controls routing on board bootup
uart Speed	Controls the speed of UART
uartclkcmd	Controls initial UART_CLOCK selection on board bootup

Table 8-2. PIBS Runtime Environment Variables

Variable name	Variable description
aliaslist	Allows users to supply alternate names for the PIBS commands.
bootfilename	Contains the path and filename of the boot file residing on the host computer. Used in conjunction with TFTP on the host computer.
imgaddr	Contains the FLASH address of the boot file.
ipdstaddr0	Contains the host computer's IP address containing the boot file. Used in conjunction with TFTP on the host computer.
userdata	Gives a place in FLASH to store user supplied program information.

8.2 Environment Variable Descriptions

The following pages provide detailed descriptions for each PIBS environment variable in alphabetical order. The information provided includes a synopsis of the variable, the syntax of the variable, descriptions of all variable parameters, and examples of variable usage.

Synopsis

Enables users to supply alternate names for PIBS commands

Syntax

set aliaslist= [(*alias_name* =*cmd_name*);+]

Parameters

alias_name	Name of the alias to use.
cmd_name	Name of the PIBS command to which to set the alias.

Usage

runtime

Example

set aliaslist=h=help;v=version; <ENTER>

Result: Typing h <ENTER> on the command line initiates the **help** command. Typing v <ENTER> on the command line initiates the **version** command.

Notes

Using the **alias** and **unalias** PIBS commands indirectly modifies the **aliaslist** variable.

Synopsis

Contains the parameters for the **bootfile** command. After system initialization, PIBS can be instructed to automatically issue a **bootfile** command. Automatic boot is enabled by placing correct bootfile parameters in the **autoboot** variable. Automatic boot is disabled by clearing the **autoboot** variable. The **autoboot** variable can be cleared by omitting the parameters that would normally follow the equal sign.

Syntax

set autoboot=*bootfile_parms*

Parameters

bootfile_parms	**bootfile** command parameters to be issued to the **bootfile** command. For more information about the **bootfile** command, see *bootfile* on page 39.

Usage

boottime

Example

set autoboot=flashb 0xffa00000 <ENTER>

Result: After system boot, PIBS executes a bootfile flashb 0xffa00000 command. Note that the bootfile command name should not be specified in the **autoboot** variable.

Notes

The **autoboot** variable is used in conjunction with the **autobootdelay** variable.

Synopsis

If the autoboot function is enabled (as described on page 67), **autobootdelay** sets the amount of delay between the time that the system is initialized and the time that the autoboot operation begins. This enables the user to abort the autoboot process.

Syntax

set autobootdelay=*time*

Parameters

time	Delay time ranging from 1 second to 999 seconds. Any time value outside of this range causes the **autobootdelay** to default to 10 seconds.

Usage

boottime

Example

set autobootdelay=10 <ENTER>

Result: After the next system reboot (if the autoboot function is enabled), the **bootfile** command with the parameters specified in the **autoboot** variable will be executed after a 10 second delay unless the user presses a key on the console.

Notes

None

Synopsis

Holds the path and filename of the boot file residing on the host computer

Syntax

set bootfilename=*file_location*

Parameters

file_location	Path and filename of the boot file.

Usage

runtime

Example

set bootfilename=c:\images\bootfile.img <ENTER>

Notes

The **bootfile** and **storefile** command use the PIBS **bootfilename** environment variable in conjunction with the **ipdstaddr0** variable to locate the boot file on the host computer using TFTP. See *Configuring TFTP* on page 83 for more information about configuring TFTP on the host computer.

Synopsis

Controls DHCP client operation. If the value of this variable is set to *true*, after a reset, PIBS will enable the associated DHCP client. If the value of the this variable is set to *false*, DHCP client support for that device will be disabled after rebooting. If the DHCP client for a particular device is disabled, the user is responsible for configuring the Ethernet networking through the **ifconfig** and **route** commands.

Syntax

set emacdhcp0=(true|false)

Parameters

true	Enable DHCP support for the chosen interface.
false	Disable DHCP support for the chosen interface.

Usage

boottime

Example

set emacdhcp0=true <ENTER>

Result: DHCP support for emac0 is enabled on the next reboot.

set emacdhcp0=false <ENTER>

Result: DHCP support for emac0 is disabled on the next reboot.

Notes

If using DHCP to provide the IP information, do not use the **ifconfig** command to change system settings for the same Ethernet device. When using DHCP, PIBS must be rebooted when connecting the platform to a different network.

Synopsis

Programs the emac's Ethernet hardware (MAC) address

Syntax

set hwaddr0=[*mac_addr*]

Parameters

mac_addr	The Ethernet's assigned 6-digit hexadecimal hardware address. The number must be specified in hexadecimal and no numeric prefix, to denote that hexadecimal notation is used. See the following example to see how to specify the hexadecimal number.
hwdaddr0	Interface to set the hardware address for hwdaddr0.

Usage

boottime

Example

set hwdaddr0=0123456789ab <ENTER>

Result: On the next system reboot, emac0's Ethernet MAC address will be programmed to 0123456789ab.

Notes

The Ethernet hardware (MAC) address must be set in order to use the EMAC in PIBS.

Synopsis

Contains TCP/IP network information

Syntax

set ifconfigcmd0=[config_parm]

Parameters

config_parm	Command parameters to be issued to the **ifconfig** command. for more information, see *ifconfig* on page 52.

Usage

boottime

Example

set ifconfigcmd0=up 192.168.0.1 <ENTER>

Result: After the next system reboot, PIBS brings up the interface with an IP address of 192.168.0.1.

set ifconfigcmd0=up 192.168.0.2 <ENTER>

Result: After the next system reboot, PIBS brings up the interface with an IP address of 192.168.0.2.

Notes

If DHCP client support is enabled for a particular network interface, the **ifconfigcmd0** variable is not used to configure the respective device.

Synopsis

Holds the address of the bootable image file stored in FLASH

Syntax

set imgaddr=[*flash_addr*]

Parameters

flash_addr	Start address in FLASH of the bootable image, ELF or raw binary program.

Usage

runtime

Example

set imgaddr=0xFFA00000 <ENTER>

Result: Used in conjunction with the **bootfile** command.

Notes

The PIBS **bootfile** command uses the **imgaddr** variable as the FLASH boot file address if one is not provided as a command line option for the **bootfile** command. See *bootfile* on page 39 for more information on the **bootfile** command.

Synopsis

Holds the host computer's IP address. This variable is used with the following commands:

1. The **bootfile** command (*bootfile* on page 39) with the eth, ethl or ethd option to specify the IP address of the host computer running the TFTP server/daemon.
2. The **storefile** command (*storefile* on page 61) with the eth option to specify the IP address of the host computer running the TFTP server/daemon.

Syntax

set ipdstaddr0=[*host_addr*]

Parameters

host_addr	Host computer IP address.

Usage

runtime

Example

```
set ipdstaddr0=192.168.0.2 <ENTER>
```

Result: The next time the **bootfile** command is executed with the eth, ethl or ethd option, the host computer at IP address 192.168.0.2 will be queried for the boot file.

Notes

The PIBS **bootfile** and **storefile** commands use the **ipdstaddr0** variable as the TFTP source address. The PIBS **bootfile** and **storefile** commands use the **bootfilename** variable setting as the TFTP source directory path/filename locator. For more information about TFTP host computer setup, see *Configuring TFTP* on page 83.

Synopsis

Contains the bootup parameters for the **l2config** command. After system initialization, PIBS can be instructed to automatically issue a **l2config** command with the supplied parameters in the **l2configcmd** environment variable. This can be used to automatically configure the on-chip SRAM as either an SRAM or cache storage upon system bootup.

Syntax

set l2configcmd=[*l2config_parms*]

Parameters

l2config_parms	Command parameters to be issued to the **l2config** command. For more information about the **l2config** command, see *l2config* on page 53.

Usage

boottime

Example

set l2configcmd=dcache <ENTER>

Result: After the next system boot, PIBS executes a l2config dcache command. Notice that the **l2config** command name should not be specified in the **l2configcmd** variable.

Notes

If no value is specified for this variable, the board comes up with L2 cache enabled for both I and D caching by default.

Synopsis

Contains system routing information

Syntax

set routecmd=[*route_parm*]

Parameters

route_parm	Command parameters to be issued to the **route** command. For more information about the **route** command, see *route* on page 59.

Usage

boottime

Example

set routecmd=add default 192.0.0.1 10 <ENTER>

Result: After the next system reboot, PIBS adds 192.0.0.1 to the route table as a default route for all networks.

Notes

If DHCP client support is enabled, the **routecmd** variable is not used.

Synopsis

Contains the bootup parameters for the **uartclk** command. After system initialization, PIBS can be directed to automatically issue a **uartclk** command by means of the parameters supplied in the **uartclkcmd** environment variable. This can be used to set the UART clocking to internal or external chip clocking automatically at bootup.

Syntax

set uartclkcmd=[*uartclk_parms*]

Parameters

uartclk_parms	Command parameters to be issued to the **uartclk** command. For more information about the **uartclk** command, see *uartclk* on page 62.

Usage

boottime

Example

set uartclkcmd=uart0 int <ENTER>

Result: After the next system boot, PIBS executes a **uartclk** uart0 int command. Note that the **uartclk** command name should not be specified in the **uartclkcm**d variable.

Notes

None

Synopsis

Stores user-defined information. For example, operating system boot parameters can be stored in the **userdata** variable and accessed by the operating system during the boot process.

Syntax

set userdata=[*data*]

Parameters

data	Up to 256 data bytes can be assigned to the **userdata** variable. PIBS stores the data as a string.

Usage

runtime

Example

set userdata=my user program <ENTER>

Result: A pointer to the content of this variable is passed to the user program through general purpose register R3 just before PIBS branches to the start of the user program.

Notes

This variable is intended to be accessed by user-supplied programs. PIBS does not use the contents of this variable.

9. PIBS Software Development

This section describes the PIBS software development process.

9.1 Directory Structure

The EPOS 1.7 kernel is required in order to build a bootable image. For this kernel level, pre-compiled library files and include files are delivered on the AMCC PPC440SP Evaluation Kit Software Package CD. The following directory structure is installed on your Windows PC during evaluation kit automatic installation:

Figure 9-1. Directory Structure

```
\installdir
    |installed cygwin readme
    |---- \bin
    |---- \...
    |
    |---- \opt
    |  |---- \powerpc-440-linux-gnu (Compilation Environment)
    |  |         (contains installed cross compiler readme file)
    |  |   |---- \gcc-3.2.3-glibc-2.2.5
    |  |
    |  |---- \ppc440sp_kit
    |  |   |---- \bsp_440sp (PowerPC Initialization Boot Software (PIBS)
    |  |   |
    |  |   |---- \kernel (Embedded PowerPC Operating System (EPOS)
    |  |   |
    |  |   |---- \doc (Documentation Directory)
    |  |   |
    |  |   |---- \rwsupport (IBM RISCWatch Support)
    |  |   |
    |  |   |---- \bin (Host Tools)
    |  |   |
```

PIBS source files will be installed on your Windows PC during the automatic installation of the evaluation kit. EPOS pre-compiled libraries and include files corresponding to level_1.7 will also be installed on your Windows PC during this process. EPOS source files corresponding to level_1.7 will *not* be installed on your Windows PC during this process. If you want to use them to recompile libraries, contact AMCC support by email at **support@amcc.com**.

After selecting Start**=>**Programs=>AMCC and launching CYGWIN from the icon in the resulting window, you have direct access to kernel and bsp_440sp build directories.

9.2 Build Tools

The PIBS software can be built using the *powerpc-440-linux-gnu* environment provided. After selecting Start**=>**Programs=>AMCC and launching CYGWIN from the icon in the resulting window, you have direct access to kernel and bsp_440sp build directories.

9.3 Makefile Setup

When compiling PIBS software, default options are stored in the **make.opt** file located in the *\opt\ppc440sp_kit\bsp_440sp\make* directory. No specific setup needs to be performed in order to recompile PIBS software in the *powerpc-440-linux-gnu* environment provided.

9.4 Build Process

Important: A full rebuild of the PIBS/EPOS libraries as described in the following procedure is not mandatory. Pre-built EPOS libraries are provided in the *\opt\ppc440sp_kit\kernel\lib* directory. Pre-built PIBS libraries are provided in the *\opt\ppc440sp_kit\bsp_440sp\lib* directory.

The complete build process is composed of three main steps. The following procedure enables the user to erase and then rebuild all libraries.

1. Build the EPOS kernel libraries.

   The following procedure can be done only if EPOS source files were provided to you. If you do not have these source files, *do not* perform Step a.

   a. **cd** opt\ppc440sp_kit\kernel\lib
   b. **make** *clobber* (for a first-time build or to remove all binaries)
   c. **make** (builds all the kernel libraries)
2.
   a. **cd** \opt\ppc440sp_kit\bsp_440sp\lib
   b. **make** *clobber* (for a first-time build or to remove all binaries)
   c. **make** (builds all the PIBS binaries)
3. Build the PIBS binaries.
   a. **cd** \opt\ppc440sp_kit\bsp_440sp\sample
   b. **make** *clobber* (for a first-time build or to remove all binaries)
   c. **make** *ram* (in sample directory). This is used to build a bootable image designed to be executed from DDR-SDRAM memory instead of the Flash for PIBS testing purposes. See *Table 9-1* for a list of the files created.

Table 9-1. Files Produced from a make ram

Files produced	File descriptions
sample.o	Intermediate object file
finaloram.map	Map file for the PIBS finaloram bootable image and ELF files
finaloram.img	Bootable image file containing PIBS meant to run from RAM. This file contains debug information because the files were compiled with the gdwardf-2 compiler option by default.
finaloram	ELF file containing PIBS meant to run from RAM. This file contains debug information because the files were compiled with the gdwardf-2 compiler option by default.
debug thread.0	Intermediate object file
dhry_1.0	Intermediate object file
dhry_2.0	Intermediate object file

d. **make** *pibs1* (in **sample** directory) is used to build the FLASH executable PIBS1 bootable image and binary files. See *Table 9-2* for a list of the files produced.

Table 9-2. Files Produced from a make pibs1

Files produced	File descriptions
sample_pibs1.o	intermediate object file
finalopibs1.map	Map file for the PIBS1 finalopibs1 binary and ELF files
finalopibs1.bin	binary file containing PIBS1 used to burn into FLASH. This file is used by the RISCWatch flash_pibs.cmd file to program PIBS1 into system FLASH. This file can also be programmed into the FLASH using the PIBS `storefile` command with the `p1` parameter.
finalopibs1	ELF file containing PIBS1 used to provide RISCWatch debug information if necessary.

e. **make** *pibs2* (in **sample** directory) is used to build the FLASH executable PIBS2 bootable image and binary files. See *Table 9-3* for a list of the files produced.

Table 9-3. Files Produced from a make pibs2

Files produced	File descriptions
sample_pibs2.o	intermediate object file
finalopibs2.map	Map file for the PIBS finalopibs2 binary and ELF files
finalopibs2.bin	binary file containing PIBS2 used to burn into FLASH. This file is used by the RISCWatch flash_pibs.cmd file to program PIBS2 into system FLASH. This file can also be programmed into the FLASH using the PIBS `storefile` command with the `pibs` parameter.
finalopibs2	ELF file containing PIBS2 used to provide RISCWatch debug information if necessary.
debug_thread.0	Intermediate object file

f. **make** *util* (in **sample** directory) is used to build executable files used in conjunction with the **flash_pibs.cmd** RISCWatch command file to program the PIBS1 and PIBS2 binaries into FLASH storage.

Table 9-4. Files Produced from a make util Command

Files produced	File descriptions
flash_pibs.o	intermediate object file
finaloutil.map	Map file for the finaloutil bootable image and ELF files
finaloutil.img	Image file of the finaloutil utility. This file is not used.
finaloutil	ELF file used by the RISCWatch flash_pibs.cmd file to program the finalopibs1.bin and finalopibs2.bin files into system FLASH.

g. **make** *ddr* (in **sample** directory) is used to build executable files used in conjunction with the **flash_pibs.cmd** RISCWatch command file to program the PIBS1 and PIBS2 binaries into FLASH storage

Table 9-5. Files Produced from a make ddr command

Files produced	File descriptions
finaloddr.img	Image file of the finaloddr utility. This file is not used.
finaloddr	Elf file used by the RISCWatch flash_pibs.cmd file to initialize the DDR memory interface.
finaloddr.map	Map file for the final ddr bootable image and elf files
ddr_util.0	Intermediate object file
Note: This code runs from SRAM and initializes the memory interface so RISCWatch can load in the DDR resident finaloddr elf file.	

Note: Step b through Step g can be bypassed by using the **make** all command, which will create all previously described at one time.

10. Configuring TFTP

In order to use the facilities of PIBS for downloading applications, through Ethernet, the host PC must be configured to support the TFTP daemon. The following section describes the steps required to configure the TFTP daemon on the PC.

The configuration process consists of two steps:

1. The **tftpaccess.ctl** files on the host must be customized to match system requirements.
2. the TFTP daemon (or server) must be made available.

If you decide to use the TFTP daemon provided with this package, you might need to modify a system environment variable to specify the location of the **tftpaccess.ctl** file. If the **tftpaccess.ctl** file is located in a directory other than the directory that contains the TFTP programs, the ETC environment variable will have to be updated. The ETC environment variable specifies an alternate directory where the **tftpaccess.ctl** file is located.

A sample TFTP daemon control access file (**tftpaccess.ctl**) is included with the package. The **tftpaccess.ctl** file can be modified or copied to another directory and modified appropriately. Entries in **tftpaccess.ctl** describe directories that can be accessed by the TFTP daemon.

When creating or modifying the **tftpaccess.ctl** file, observe the following rules:

- Blank lines and lines beginning with "#" are ignored.
- Each entry must be entered on a single line.
- Each entry must start with the **allow** keyword, followed by a colon (:), which would be followed by a directory name. A directory name specifies a directory that the TFTP daemon is allowed to access.

For example, the following entry would allow the tftpd server to access the **\default\bsp_440sp\samples** directory:

allow:\default\bsp_440sp\samples

Each entry should be entered on a single line.

It is possible to run **tftpd.exe** automatically every time Windows is started. To make the program run automatically every time Windows is started, perform the following steps:

1. Select Start from the Windows task bar.
2. Select Settings.
3. Select Taskbar.
4. Select Start Menu Programs (or Advanced).
5. Select Add...
6. In the command line field enter tftpd.exe. Include the full directory path of the tftpd.exe file if necessary.
7. Select Next.
8. In the Select Program Folder window, select the Programs/Startup folder.
9. Select Next.
10. Select Finished.

The TFTP daemon will start automatically the next time Windows is restarted.

11. Bootable Image File Format Description

The bootable image file format is designed to enable specifying multiple text, symbol, and data sections. The file format consists of multiple sections prefixed by section headers (of type rel_block_t), with a boot header (of type, boot_block_t) prefixing the entire file. The boot header specifies the initial target of the image and the image entry point. PIBS software only processes the information in the boot header when loading bootable image files.

11.1 Boot Header

The bootable image file is preceded by the boot header. The PIBS loader uses information in the boot header to verify the validity of the bootable image file and to determine the image file destination. The boot header is stripped off by PIBS loader and is not saved in memory. The boot header has the following format:

```
typedef struct boot_block {
   unsigned long magic;
   unsigned long dest;
   unsigned long num_512blocks;
   unsigned long debug_flag;
   unsigned long entry_point;
   unsigned long reserved[3];|
} boot_block_t;
```

magic	Identifies this image as a legitimate bootable image and must have the value 0x0052504F.
dest	Target address for the image (after the boot header is stripped off).
num_512blocks	Bootable images are padded to a multiple of 512 byte blocks. This field specifies the number of 512 byte blocks.
debug_flag	Currently unused.
entry_point	Specifies the address of the bootable image entry point.

11.2 Section Headers

PIBS places the entire image at the address specified in the boot header. The entry point specified in the boot header is assumed to be a branch, followed by the first section header of type **info_block_t**. This is to allow the bootstrap code to gain immediate addressability to the first section header. The format of the first section header is shown below:

```
typedef struct info_block {
   long magic_num;
   long text_start;
   long text_size;
   long data_start;
   long data_size;
   long elf_hdr_size;
   long sym_start;
   long num_syms;
   long toc_ptr;
   struct rel_block *next;
} info_block_t;
```

magic_num	is used for verification purposes and must be set to 0x004D5054.

text_start	is the address of the first text section.
text_size	is the size in bytes of the first text section.
data_start	is the address of the first data section.
data_size	is the size in bytes from the of the first data section.
elf_hdr_size	is currently unused.
sym_start	is the address of the symbol table.
num_syms	is the number of symbols.
next	points to the next section header.

There are three additional section header types that can follow the first section header. Generally, they can occur in the image in any order, but they are usually arranged in ascending address order. A section header has the following format:

```
typedef struct rel_block {
  unsigned long type;
  unsigned long dest_addr;
  unsigned long size;
  union {
    struct data_info {
      unsigned long size_to_fill;
      unsigned long char_to_fill;
    } data_info_str;
    struct text_info {
      unsigned long toc_pointer;
      unsigned long entry_pt;
    } text_info_str;
    unsigned long number_symbols;
  } section_info;
  struct rel_block *next;
  struct rel_block *bptr;
} rel_block_t;
```

The **type** field is one of the following manifest constants:

```
#define TEXT_SECT 0x00000001
#define DATA_SECT 0x00000002
#define SYMB_SECT 0x00000004
```

The **dest_addr** specifies the target for the section, while **size** is the extent of the section, not counting the header. The bootstrap program uses this information to move the section to the destination specified at link time. **next** and **bptr** are the section header forward and backward pointers, respectively.

For a text section, the union **section_info** contains the structure **text_info**, specifying the entry point of the text section.

For a data section, the union **section_info** contain the structure **data_info**, specifying **size_to_fill** and **char_to_fill**. These parameters are used to optionally fill a region past the **size** with a character **char_to_fill** for **size_to_fill** bytes.

For symbols, the union **section_info** contains the number of symbols in the section. The data in this section consists of the symbol table from the original object file.

Part III. Sample Applications

12. Sample Applications Overview

This section describes the steps necessary to build, load, and execute the sample application programs included in the evaluation board kit. At this point it is assumed that the serial port and Ethernet board-to-host connections have been made and properly configured. Users can ensure Ethernet connectivity by using the **ping** command from the PIBS shell prompt.

The sample application programs are compiled, assembled, and linked using the GNU cross development tool for PowerPC included with the evaluation board kit software. EPOS libraries are used during the link step to create an executable file in ELF format. The ELF executable can then be downloaded from the host to the board using either the PIBS **storefile** command or the RISCWatch debugger.

Note: If you receive an error message regarding the cygwin1.dll file when using GNU cross development tool for PowerPC, there may be another application on the host that uses a different version of this file. Update your PATH system environment variable (Control Panel->System->Advanced->Environment Variables) so that the GNU cross development tool for PowerPC directory is first in your path.

13. How to Load Application Programs

The PIBS **bootfile** command supports file transfer using a serial port or Ethernet network connection. To load an ELF file using the serial port, issue the **bootfile serial** command from the PIBS prompt, then transfer the ELF file through the Kermit protocol, using the terminal emulator running on the host. For Windows HyperTerminal users, select Transfer, Send File, set the Filename, Kermit Protocol.

Before loading an ELF file over Ethernet, ensure that the board's Ethernet interface has been set up correctly through the PIBS **ifconfig** command. Board-to-host connectivity can be verified with the PIBS **ping** command or by pinging the board's IP address from the host. After verifying Ethernet connectivity, ensure that the PIBS **ipdstaddr** and **bootfilename** environment variables are set correctly, supplying the host computer's IP address and the directory path/filename of the file to load. To load an ELF file from the host, issue the **bootfile eth** command from the PIBS prompt. PIBS will send a TFTP request for the bootfile to the host using the Ethernet network connection. The host computer must be running a TFTP server/daemon with access to the **bootfilename** specified. See *Configuring TFTP* on page 83 for additional information.

After a successful download, PIBS will perform relocation and branch to the program's entry point to start execution. See *bootfile* on page 39 for additional information.

14. EPOS Shell and Dhrystone Sample

To create the EPOS shell sample program, issue the **make** *ram* command from the **sample** directory. This creates the **finaloram** ELF file suitable for execution on the board. This sample also includes the Dhrystone benchmark which can be run from the EPOS shell. After loading the **finaloram** file using either the PIBS **bootfile serial** or **bootfile eth** command, output similar to that in *Figure 14-1* should be observed.

Figure 14-1. Sample Output from EPOS

```
-------------------------------------
text start=0x00100000
text end  =0x00136e7c
data start=0x00136e7c
data end  =0x0013d050
bss start =0x0013d050
bss end   =0x0013e8c0
heap start=0x0013e8c0
heap end  =0x001c1740
speed     =400000000
-------------------------------------------
 XXXXX   XX XXX     XXXX   XXXXXXX
XX    X   XX  XX   XX  XX  XX
XXXXXXX   XX  XX   XX  XX  XXXXXXX
XX        XXXXX    XX  XX       XX
 XXXXX    XX        XXXX   XXXXXXX
         XXXX
-------------------------------------------
Very simple shell for EPOS
type "help()" for help
type "exit()" to quit
EPOS $
```

The Dhrystone benchmark is also included with this sample and can be run by entering **dhry()** at the EPOS shell prompt. Output similar to that below should be observed.

Dhrystone Benchmark, Version 2.1 (Language: C)

Program compiled without ‘register’ attribute

Please give the number of runs through the benchmark:

At this point, enter the number of desired iterations. The test is designed not to give results if the selected iterations completes in less two seconds, so pick a large number (Š 10000000). After the test completes, a check screen will be displayed, followed by the benchmark results. The results may vary based on the system environment, the compiler, and the compiler options used.

15. Debugging Application Programs

PIBS includes a Debug Monitor that allows applications to be loaded and debugged using the RISCWatch debugger in non-JTAG "sim" target mode. The RISCWatch debugger running on the host communicates with the Debug Monitor using the Ethernet connection between the host and the board. The debug connection must be made over Ethernet because debug using the serial port is not supported. The Debug Monitor function is provided by the EPOS debug library (debuglib.a), with which PIBS is built. The Debug Monitor can be included in any EPOS application to provide RISCWatch debug capability.

The debug sample found in the **debug_thread.c** file shows how an application can be debugged using the PIBS Debug Monitor and the RISCWatch debugger in "sim" target mode. The code in the **debug_thread.c** file is linked in the **finalopibs2** and **finaloram** images. The code in the **debug_thread.c** file can also be used to create a standalone image file, **finalodebug**.

It is important to note that when the **debug_thread.c** code is used to create a standalone image file, the application code in the **debug_thread.c** file must not use interrupts and must not conflict with PIBS memory space usage. Doing so may cause the PIBS Debug Monitor-to-RISCWatch debugger connection to fail. If the application program that is being debugged requires the use of interrupts, the EPOS shell sample code (**finaloram** image file) can be loaded on to the board and the RISCWatch debugger can be used to attach to the Debug Monitor in the EPOS shell sample image. Once attached, the Instruction Address Register (IAR) is set to the start of the application code in the **debug_thread()** function. Only application code can be debugged using the Debug Monitor. The Debug Monitor cannot debug EPOS operating system code or first level interrupt handlers.

Before starting the RISCWatch debugger on the host system, the RISCWatch environment file, rwppc.env, must be updated as follows:

```
TARGET_TYPE = sim

TARGET_NAME = x.x.x.x
```

where x.x.x.x is the IP address of the evaluation board.

To build the debug sample, issue the "make debug" command from the **sample** directory. This creates the **finalodebug** ELF file suitable for execution on the board. After verifying host-to-board Ethernet connectivity by pinging the board's IP address from the host computer, start RISCWatch on the host system. After a successful connection to the PIBS Debug Monitor is made, the main RISCWatch command window should display. If it does not, verify all settings and ensure Ethernet connectivity between the host and the board.

From the RISCWatch command line, use the RISCWatch **srchpath add *pathname*** command to add the **sample** directory to the RISCWatch search path. Issue the RISCWatch **load file finalodebug** command to load the file on to the board using the Debug Monitor connection. To allow source level debug of the debug sample, issue the **load host finalodebug** command from the RISCWatch command line. RISCWatch can now be used as a source level debugger via its connection to the Debug Monitor.

When using RISCWatch with the Debug Monitor code, users should be aware of the following:
- Cache displays are not supported
- Hardware breakpoints are not supported
- Reset command is not supported
- General purpose register R1 should not be manually modified
- Do not use memory that may affect Debug Monitor operation
- Future versions of RISCWatch may use a TARGET_TYPE other than "sim" to connect to the Debug Monitor.

Revision Log

Revision Date	Level	Contents of Modification
05/12/2005	1.03	Document open for 440SP pass 2 updates. Updated Framemaker variable definitions. Implemented all pass 2 updates. GJG
09/23/2004	1.02	Converted to AMCC template and made all concomitant changes.
04/15/2004	1.01	Corrected lists and text regarding features and capacities. Corrected PIBS memory map, descriptions of commands and environment variables. Corrected PIBS Software Development and Sample Applications Overview sections. Edited all text for readability and consistency.
03/10/2004		Implemented first comments from engineers. Some comments require further input from engineers. Work will proceed as comments become available.
02/20/2004		This is the working document only to be used as the basis. The information in this manual pertains to PPC440GX kit user’s manual. The idea is to reuse relevant information and review, update, modify and add additional PPC440SP specific information.

Index