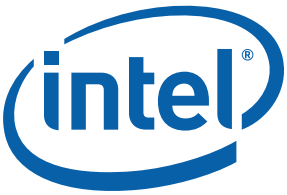


Host Bus Adapter

Customer Reference Board Manual for 8134x I/O Processors

September 2006

Order Number: 315056-001US

Contents

Figures

Tables

Revision History

Date	Revision	Description
September 2006	001	Initial public release

1.0 Introduction

1.1 Document Purpose and Scope

This document describes the Host Bus Adapter Customer Reference Board for 8134x I/O Processors (hereafter referred to as CRB). The CRB is a development and evaluation platform intended to enable rapid development of hardware and software applications.

Note: The Intel® 81348 and Intel® 81341 and Intel® 81342 products are I/O Processors. The Intel® 413808 and Intel® 413812 products are I/O Controllers. From this point on in the manual for simplicity, we refer to these products as “processors”.

This document tells you how to install the board in a server system and establish connections between the CRB and a separate system. Instructions are provided for attaching software debug hardware to the JTAG port on the CRB and loading software into the flash memory or CPLD. You can also use this manual as a reference when you are creating boards for your own custom applications.

This document assumes that you are familiar with installing and using computer boards and drivers, and that you are familiar with industry or Intel standards such as PCI-X, PCI Express, Serial Attached SCSI (SAS), JTAG, etc.

Common uses of the CRB development kit include:

- Software Development - The board comes pre-loaded with a modified version of Red Hat, Inc.’s RedBoot* embedded bootstrap and debug environment (this is often referred to as a “bootloader”). RedBoot provides a bootstrap environment, including network downloading and debugging. It also provides a simple flash file system for boot images.
- Software Debug - The processor supports source level software debugging through the use of JTAG hardware and either Intel’s XDB or Accelerated Technology, Inc.’s Edge* debugger applications. Refer to Appendix E or Appendix F for tutorials on using these tools.
- Reference Design - This document can also be used as a reference by designers creating boards for custom applications. A high level Bill of Materials is included in this document and schematics are available in a separate document on Intel’s Web site.

Note: The design for the CRB does not fully comply with the specifications included in the *Intel® 81348 I/O Processor* or *Intel® 81341 and 81342 I/O Processors Design Guide*. When using the CRB as a reference design, specifications in the Design Guide take precedence over anything included in the schematics.

1.2 Product Descriptions

The products available from Intel in the 8134x family include:

- Intel® 81348 I/O Processor-based host bus adapter with 8-port SAS/SATA controller.
- Intel® 81341 I/O Processor-based host bus adapter with one processing core.
- Intel® 81342 I/O Processor-based host bus adapter with two processing cores.
- Intel® 413808 I/O Controller-based host bus adapter with 8-port SAS/SATA controller
- Intel® 413812 I/O Controller-based host bus adapter with 8-port SAS/SATA controller.

Note: Certain information in this manual applies only to specific versions of the product set. The instances in the manual where only specific versions apply are labeled appropriately.

1.3 Using this Manual

"Customer Reference Board Features" on page 12 provides an overview of the board and processor features.

"Getting Started" on page 15 describes how to unpack and set up your board for use as a development platform.

"Using the CRB as a Software Development Platform" on page 20 provides reference information for software programmers.

"Hardware/Architecture Reference" on page 27 provides reference material for designers intending to use their package as an example on which to base a custom or derivative design.

1.3.1 Related Documents

Table 1. Related Documentation List (Sheet 1 of 2)

Document	Intel Order Number (or SIG URL)
Intel® 81348 I/O Processor Developer's Manual	315036-001
Intel® 81341 and Intel® 81342 I/O Processor Developer's Manual	315037-001
Intel® 81348 I/O Processor Datasheet	315038-001
Intel® 81341 and Intel® 81342 I/O Processor Datasheet	315039-001
Intel® 81348 I/O Processor Design Guide	315053-001
Intel® 81341 and Intel® 81342 I/O Processor Design Guide	315054-001
Intel® 81348 I/O Processor Specification Update	315041-001
Intel® 81341 and Intel® 81342 I/O ProcessorSpecification Update	315042-001
Host Bus Adapter Schematics for Intel(R) 8134x I/O Processors	315367-001

Table 1. Related Documentation List (Sheet 2 of 2)

Document	Intel Order Number (or SIG URL)
Intel® Flash Recovery Utility (FRU) Reference Manual	274071
IEEE Standard Test Access Port and Boundary-Scan Architecture (IEEE JTAG-1149.1-1990)	http://www.ieee.org
PCI Local Bus Specification, Revision 2.3 - PCI Special Interest Group PCI Express Specification, Revision 1.0a - PCI Special Interest Group PCI Express Base Specification 1.0a - PCI Special Interest Group PCI Express Card Electromechanical Specification 1.0a - PCI Special Interest Group PCI Local Bus Specification, Revision 2.3 - PCI Special Interest Group PCI-X Specification, Revision 1.0b - PCI Special Interest Group PCI Bus Power Management Interface Specification, Revision 1.1 - PCI Special Interest Group PCI Bus Hot-Plug Specification, Revision 1.1 - PCI Special Interest Group	http://www.pcisig.com/specifications

1.3.2 Terms Used in this Document

Table 2. Terms and Definitions (Sheet 1 of 2)

Acronym / Term	Definition
ARM	Refers to both the microprocessor architecture and the company that licenses it.
ATU	Address Translation Unit
BSP	Board Support Package
BTB	Branch Target Buffer
CAS	Column Address Strobe
CMD	Command
CPLD	Complex Programmable Logic Device
CPSR	Current Program Status Register
CPU	Central Processing Unit
CRB	Customer Reference Board
DBCON	Data Breakpoint Control
DDR	Double Data Rate
DIMM	Dual Inline Memory Module
DIP	Dual In-Line Package
DRAM	Dynamic Random Access Memory
ECC	Error Correction Code
EDE	Embedded Development Environment
EEPROM	Electrically Erasable Programmable Read-Only Memory
ELF	Executable and Linkable Format
Gb	Gigabit
GNU	G N U stands for “GNU's Not Unix.” Therefore, the G in “GNU” also stands for “GNU,” making the acronym recursive.
GPIO	General Purpose I/O
HD	Hard Drive
HSD	High-Speed Data

Table 2. Terms and Definitions (Sheet 2 of 2)

Acronym / Term	Definition
I/O	In/Out
ICE	In-Circuit Emulator – A piece of hardware used to mimic all the functions of a microprocessor.
IOC	I/O Controller
IOP	I/O processor
JTAG	Joint Test Action Group – A hardware port supplied on Intel XScale® microarchitecture CRBs used for in-depth testing and debugging.
LAN	Local Area Network
LED	Light Emitting Display
MAC	Media Access Control
MB	MegaBytes
MHz	MegaHertz
MMU	Memory Management Unit
NVRAM	Nonvolatile Random Access Memory
OS	Operating System
PBI	Peripheral Bus Interface
PBIU	Peripheral Bus Interface Unit
PCIe	PCI Express bus
PCI-X	PCI-X bus
PHY	Physical Layer
PSU	Power Supply Unit
R/W	Read/Write
RAM	Random Access Memory
ROM	Read Only Memory
SAS	Serial Attached Small Computer Systems Interface (SCSI)
SATA	Serial Advanced Technology Attachment
SDRAM	Synchronous Dynamic Random Access Memory
SGPIO	Storage General Purpose I/O
TTB	Translation Table Base
UART	Universal Asynchronous Receiver-Transmitter
XDB	XScale Debugger

1.4 Getting Support

Table 3. Electronic Information

Support Type	Location / Contact
The Intel World-Wide Web (WWW) Location:	http://www.intel.com
Customer Support (US and Canada):	1-916-377-7000

2.0 Customer Reference Board Features

2.1 81348 Processor Features

The 81348 processor takes the next step in integration of storage features. The 81348 combines a high performance I/O Processor and an eight-port 3Gbps SAS/SATA I/O controller into a single component. Both the IOP and IOC functions in 81348 utilize the newest core based on Intel XScale® technology. Combining these two capabilities into a single component helps reduce board space and system power while improving performance in comparison to discrete components. In addition to combining two functions into a single component, 81348 makes advances towards end-to-end data protection. 81348 utilizes byte-parity checking on it's internal buses to protect data while it travels through the 81348. It supports ECC for external memory to detect errors. Also, 81348 includes a hardware RAID 5/6 acceleration engine to improve RAID throughput and protect the user's data in case of a drive error.

81348 has options for both PCIe and PCI-X, offering Root Complexes and Central Resources as well as target mode. It is fully compliant with the *PCI-X Addendum to the PCI Local Bus Specification,* Revision 1.0 and *PCI Express Specification,* Revision 1.0a.

The 81348 processor includes eight General Purpose I/O (GPIO) pins, and eight ACTIVITY/STATUS pin pairs which are used for SAS links for activity and status indicators. Each SAS link uses one ACTIVITY/STATUS pin pair.

Table 4. 81348 Processor Features

- Address Translation Unit
- Messaging Unit
- Flash Interface Unit
- Eight SAS Link Engines with integrated PHYs
- I^2C Bus Interface Units
- Multi-Port SRAM Integrated Memory
- Application DMA Controllers
- Transport DMA Controllers
- UART Units
- Address and Data Bus Parity Protection
- TPMI (Messaging Interface)
- Inter-Processor Communication
- Timers
- Watchdog Timers

2.2 Customer Reference Board Features

Figure 1 is a block diagram of the CRB major functional areas.

Figure 1. CRB Functional Block Diagram

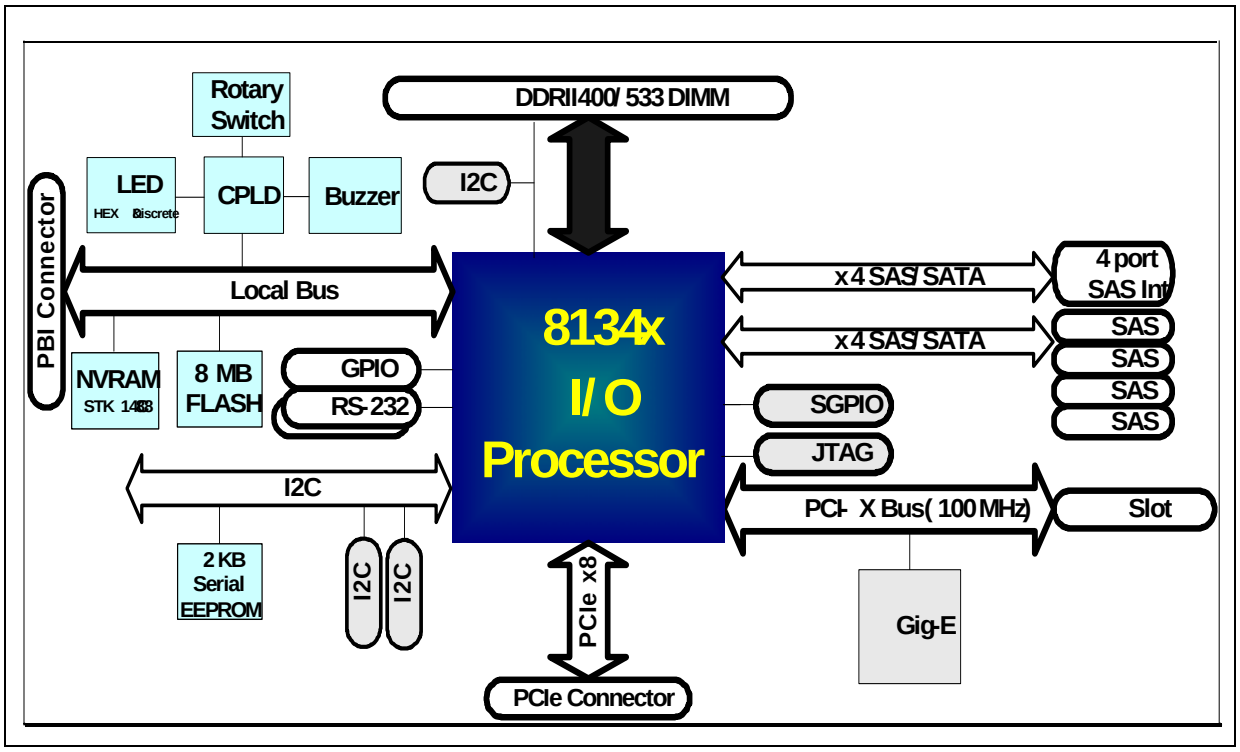


Figure 2 is a diagram of the CRB connectors, jumpers and key components.

Figure 2. CRB Layout

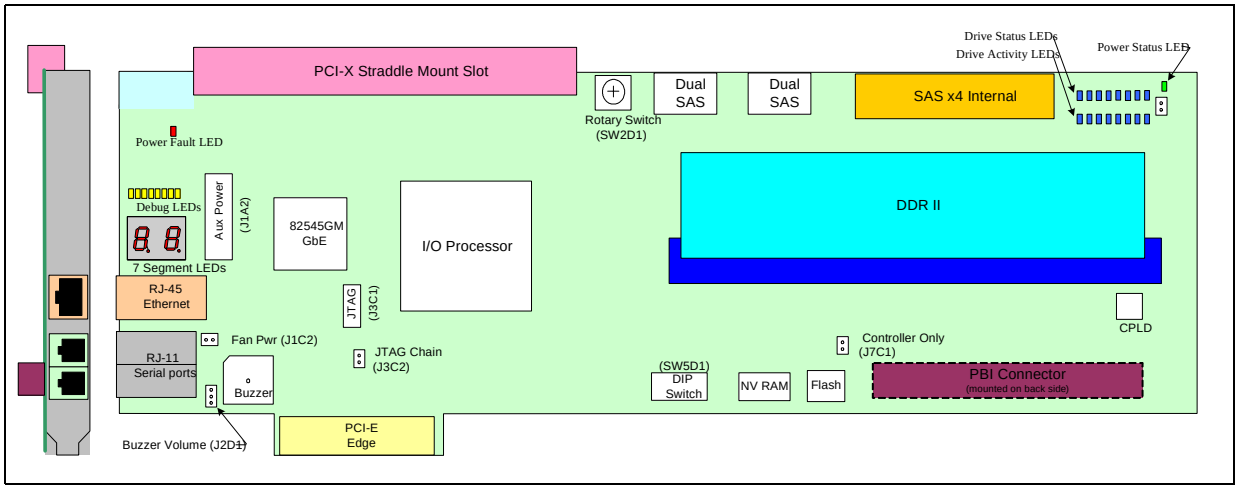


Table 5 summarizes the major CRB features.

Table 5. CRB Features

Feature	Description
Alarm	Audible buzzer
Discrete LEDs	Eight discrete LED indicators for debug (amber)
DRAM	DDR II DIMM connector: Ships with 256 MB 533 MHz DIMM or 512 MB DIMM Supports up to 2 GB DDRII 400/533 MHz
Drive Activity LEDs	Sixteen discrete LED indicators for SAS/SATA drive activity and status (blue)
Ethernet Port	One Gigabit Ethernet Port - RJ45
Flash ROM	8 Mbytes Flash ROM
Form Factor	Full length PCIe add-in card (312 mm x 145 mm)
Hex Display	Two 7-segment Hex LED displays
JTAG Port	10-pin mini JTAG Header Adapter card interfaces to 20-pin JTAG cable
NVRAM	32 Kbytes NVRAM
PBI Connector	4x25 receptacle connector for peripheral bus auxiliary circuit prototyping
PCI Express	x8 PCIe edge finger connection to host system
PCI-X Bus	64-bit PCI-X slot - 100 MHz
Power LED	Power on LED indicator (green)
Processor	Integrated 800 MHz Intel XScale® core-based Processor
Rotary Switch	16-position rotary switch for software debug
SAS Ports	One internal SAS/SATA wide port connector: Four SAS / SATA narrow port connectors
Serial EEPROM	2 KByte serial EEPROM
Serial Port	Two Serial Console Ports (RJ-11 connector)
Slot Power Fault LED	Red LED illuminates when PCI-X card is plugged into the slot but slot power is not turned on

3.0 Getting Started

This section tells you how to unpack the board and install it into an available PCI Express slot in a server or backplane for use as a development platform. Once you have completed the setup, refer to "Using the CRB as a Software Development Platform" on page 20 for reference material you can use when configuring the board for a test or development environment.

Note: Software updates and additional offerings from vendors can change frequently. To keep up-to-date, please visit http://www.intel-ioprocessortools.com/CRB-registration/ for the latest updates.

3.1 Visual Inspection

Warning: Static charges can severely damage the board. Make sure you are properly grounded before removing the board from the anti-static bag.

The first time you unpack the CRB, check to make sure the board was not damaged during shipment. If anything is damaged or missing, contact your Intel representative.

3.2 Hardware Requirements

To make full use of the board as a SW development or debug environment you will need the following components (these are NOT included in your kit):

- Server system or backplane with an available x8 or longer PCI Express slot.
- Additional computer system (such as a laptop) with at least one available parallel port and one available serial port (two USB ports). You'll use this system to communicate with the CRB through a serial connection using a terminal program such as Microsoft Windows* HyperTerminal. The setup instructions in this document assume that you will use a development/debug system in addition to the server system or backplane.
- JTAG Emulator debug device. You can obtain a debugger from the vendors listed in Table 6.

3.3 Set-up and Configuration

The CRB must be installed in an available PCI Express slot (x8 slot) in a server system or backplane. Since server and backplane configurations may vary, your precise setup steps may vary. To set up your board follow the general guidelines below; as needed, refer to the additional information provided.

1. Install the board in a PCI-Express slot in your server system. Refer to Section 3.3.1 for more information about the types of servers or backplanes you can use.
   — You can install an optional PCI-X or PCI card in the slot on top of the board. Note that additional power connections are required to use this slot. Refer to Section 3.3.1.1 for full details.

— You may want to connect another system (such as a laptop computer) to the JTAG port on the CRB. Debug software on the development/debug system can then be used to debug software applications running on the CRB. Refer to Section 3.3.1.2 for details on using the JTAG port to establish a debug connection.

— Before the server system can recognize the board, appropriate drivers must be installed. Other firmware may also be required to make full use of the board's features. Refer to Section 3.3.2 for details on driver/firmware availability and setup information for the SAS and SATA ports.

2. To program and configure the board, connect a serial cable between the board and another system (such as a laptop computer); then you can use connection software running on the laptop to communicate with the board. Refer to Section 3.4 for full details on establishing this connection.
3. The software on the board used to operate the board and exercise board features is a bootstrap program called RedBoot. Refer to Section 4.0 for more information about the RedBoot operating system.

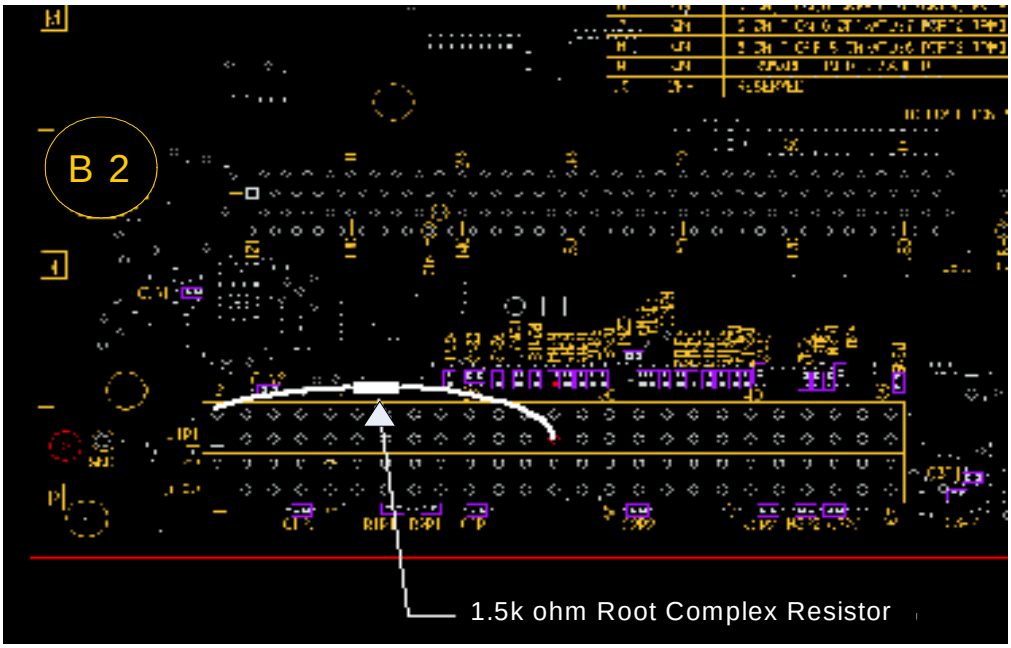
3.3.1 Installing CRB into a Host System

The board is a full-length host bus adapter card that requires a PCI Express slot free from obstructions. The CRB has a x8 edge connector. Follow the host system manufacturer instructions for installing a PCI Express adapter card.

Note: The CRB form factor may not fit in all servers. The PBI connector on the back side of the board (J1P1 and J1P2) exceeds the maximum PCIe secondary side component height specification.

To operate the CRB in a passive backplane, the I/O processor must be put into PCIe root complex mode. Root complex mode can be activated by installing a 1.5K ohm resistor in the PBI connector socket (J1P1) between pins 2 and 25. Connector J1P1 is located on the back side of the board. Figure 3 shows where to install the root complex resistor for passive backplane operation.

Figure 3. PCIe Root Complex Resistor Placement For Passive Backplane Operation



3.3.1.1 PCI-X Slot

Power for the CRB comes from the host system through the PCI-Express card edge fingers. When the board is powered-on the green power status indicator illuminates.

An additional PCI-X slot is located on the top of the board. This slot does *not* receive power from the CRB PCI Express card edge fingers; power for this slot must be provided through the auxiliary power connector (J1A2). The auxiliary power connector mates with a standard ATX power supply 4-pin peripheral power connector. A red power fault LED illuminates when a PCI or PCI-X card is installed in the PCI-X slot and the CRB is powered up without power being applied to the auxiliary power connector.

When the power fault LED is illuminated, turn off power to the board and connect power to the auxiliary power connector, or remove the card from the CRB PCI-X slot.

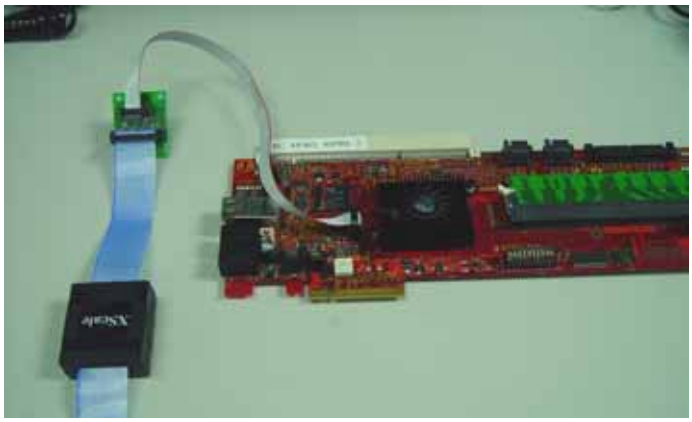
3.3.1.2 Connecting an Optional JTAG Debug Device

A JTAG debug device can be connected to the CRB through the 10-pin mini JTAG header (J3C1). A converter board and cable is included in your kit to interface the 10-pin CRB JTAG connector to a conventional 20-pin JTAG connector. You can obtain a JTAG debugger from the vendors listed in Table 6.

Be careful to align the polarizing notch on the 10-pin header shroud with the polarizing feature on the JTAG cable receptacle connector. If the 10-pin mini JTAG cable is plugged in backward a red LED on the JTAG converter board will illuminate to indicate a cable polarity problem.

Figure 1 shows the JTAG Adapter board and ribbon cable required to communicate with a JTAG Emulator.

Figure 1. JTAG Adapter Board



3.3.2 Installing Drivers and Firmware

When you power up the host server or backplane connected to the CRB do not let the Microsoft Windows*-based Operating System (OS) load any Plug and Play or other drivers for the CRB. Typically, the board will appear to Windows as an “Unknown Memory Device”. Loading Windows drivers may corrupt the bootstrap program already loaded on the board.

Warning: Select Cancel when prompted to load drivers for the CRB by the server operating system.

To use some CRB features, such as the SAS ports, you will need to load appropriate drivers or Transport Firmware on the CRB. These drivers are available from Intel. Contact your Field Sales Representative for more information.

Once you have obtained the Transport Firmware, you can use the SATA or SAS ports as described in the next section.

3.4 Communicating with the CRB

When the CRB boots, a Redboot bootloader is loaded into memory and executed. A development/debug system can communicate with the Redboot software through the serial ports. A pair of RJ-11 cables and an RJ-11 to DB9 converter is supplied with the board.

To communicate between a development/debug system and the CRB Redboot software, plug an RJ-11 cable into either RJ-11 connector.

Note: Use the lower port (UART1) for Redboot communication.

Plug an RJ-11 to DB9 converter board onto the other end of the RJ-11 cable and into a development/debug computer DB9 serial port connector.

Start a HyperTerminal session on the development/debug computer with the following parameters:

Communications Port: Com port connected to the DB9 to RJ-11 converter.
Baud Rate: 115200
Data Bits: 8
Parity Bits: None

Stop Bits: 1
Flow Control: None

When the CRB is powered-up it displays boot-up messages and after approximately 30 seconds displays the Redboot prompt in the HyperTerminal window.

3.4.1 Using the Ethernet Port

The CRB has an on-board gigabit Ethernet device and connector. Programmers can use this device for software download via TFTP and for disk management over Ethernet applications.

3.4.2 Using the Serial ATA or Serial Attached SCSI Ports

The CRB has eight SAS/SATA disk drive ports connected to the 81348 storage interface. Four ports are implemented using dual-stacked narrow-port connectors and four ports are implemented using an internal x4 wide-port connector.

Once the transport firmware has been installed, connect one or more supplied narrow-port and/or wide-port SAS or SATA cables to the CRB SAS/SATA connectors. Connect the other end(s) to one or more SAS/SATA drives. Connect the 4-pin power cable(s) attached to the SAS/SATA cables to standard 4-pin power supply peripheral power connectors.

Note: The narrow-port SAS cables come with two port connections per cable. These are designed to be able to redundantly connect one drive to two different disk controller cards for systems that include fail-over protection. If the host system does not have multiple disk controller cards that support redundant disk connections, one of the narrow-port cables on each dual narrow-port cable should be left unconnected.

4.0 Using the CRB as a Software Development Platform

4.1 Third Party Software Support

This build of the CRB is using pre-production silicon, and is the first build of boards that has been delivered to OS vendors for software porting. Therefore, there is no commercially available software support for these boards at the time of publishing this manual. Gold releases of software solutions will only be publicly available coincident with the production release of the 81348 silicon, and the Customer Reference Board.

4.1.1 Proposed Solutions

Table 6 indicates Software Tools Vendors and Operating Systems that are getting CRBs.

Table 6. Software Tools Vendors and Operating Systems

Vendor	OS
Bactrian*	JTAG
American Arium*	JTAG
ARM*	JTAG
Embedded Performance Inc.*	JTAG
GreenHills*	Integrity RTOS
Macraigor Systems*	Raven and MpDemon JTAG
Mentor Graphics*	Nucleus OS and EDGE debugger.
TimeSys*	Linux OS and Tools
Windriver Systems*	VxWorks and Vision-ICE JTAG

The boards are delivered with the RedBoot* Bootloader installed in flash memory. Source files for this solution are available on request, and updated binaries are released as bugs are found and fixed.If you would like a status on the porting of a particular solution, or RedBoot information please send an email to: **crb@intel-ioprocessortools.com**.

4.1.2 Third Party Contact Information

Since these boards are delivered under a Corporate Non-Disclosure Agreement, it is highly likely that your sales contact at the companies listed above may not have heard of the CRB, or the 81348 silicon. Please send an email to: **crb@intel-ioprocessortools.com** requesting contact information, for example, if you would like an early drop of a particular solution.

4.1.3 Other Solutions

When your favorite OS or tools company is missing from the list above, please send an email to **crb@intel-ioprocessortools.com** requesting that they be added.

4.2 Software Developer Reference Information

This section provides reference information for software developers.

4.2.1 RedBoot* Software

Once you see the RedBoot prompt in the HyperTerminal window, you know the board is functioning properly. The RedBoot bootstrap software supports diagnostic tools for testing the memory and basic board features. To access the RedBoot diagnostic features, refer to Section 4.2.1.1. This will display a list of supported commands and syntax. For full details, refer to Appendix A, "Advanced Redboot Commands" for information on Redboot commands including the following:

- "Removing the "+FLASH configuration checksum error or invalid key" message"
- "Rebuilding RedBoot"
- "Using RedBoot to Update the RedBoot Image"
- "Using RedBoot to Update the Transport Firmware Image"
- "Initialize the Flash File System "
- "Display the Current File System in Flash"

Also refer to the RedBoot User's Guide at this link:

http://sources.redhat.com/ecos/docs-latest/redboot/redboot-guide.html

4.2.1.1 Redboot Diagnostics

The CRB comes with RedBoot loaded in the Flash. RedBoot is the RedHat debug monitor which initializes the board and has some debug and diagnostic functions.

Note: You should read the RedBoot manual and be familiar with Serial Port applications before performing diagnostics.

Note: The CRB board is shipped preflashed with jumpers and settings set in factory settings. It also is shipped with the necessary cabling hardware (cable and adapter/converter) to perform this procedure.

To perform diagnostics:

1. Refer to "Communicating with the CRB" on page 18 for instructions on how to set up the CRB and get to the Redboot prompt.
2. At the Redboot prompt, enter the diagnostic command with **DIAG** or **DIA** and the RedBoot Diagnostic function is invoked. The list of Redboot diagnostics displays. Figure 4 shows the diagnostic startup and utilities list.
3. Perform the tests that are desired.

Figure 4. Redboot Diagnostics Command and Hardware Test Listing

```
RedBoot> diag

Entering Hardware Diagnostics - Disabling Data Cache!

IQ8134 CRB Hardware Tests

1 - Memory Tests

2 - Random-write Memory Tests

3 - Repeating Memory Tests

4 - Repeat-On-Fail Memory Test

5 - Enhanced Memory Tests

6 - Rotary Switch S1 Test

7 - 7 Segment LED Tests

8 - i82545 Ethernet Configuration

9 - Timer Test

10 - PCI Bus test

11 - CPU Cache Loop (No return)

12 - Batch Memory Tests

13 - Multi-Path Memory Test

14 - NVRAM Test

0 - quit

Enter the menu item number (0 to quit):
```

The Redboot hardware tests are listed in Table 7.

Table 7. Redboot Hardware Tests (Sheet 1 of 2)

Number	Test Name	Description
1	Memory Tests	Performs Walking One, Zero, 32-bit Address/Address Bar, and 8-bit Address/Address Bar test to selected memory range Note: Redboot uses memory between 00000000H - 0001C000H. Memory tests should be restricted to addresses above 0001C000H to avoid conflicting with Redboot.
2	Random-Write Memory Tests	Writes and then verifies a random 32-bit pattern to selected memory range.
3	Repeating Memory Tests	Performs the Memory Test diag (#1) ad infinitum to selected memory range.

Table 7. Redboot Hardware Tests (Sheet 2 of 2)

Number	Test Name	Description
4	Repeat-On-Fail Memory Test	Performs Address/Address Bar memory test to the selected memory range in a continuous loop until a failure is encountered and then will continually read/ write to the failing location ad infinitum.
5	Enhanced Memory Tests	Executes a memory diagnostic to the selected memory range with several different memory patterns.
6	Rotary Switch S1 Test	Shows the current location of the rotary switch on the Right 7-segment LED.
7	7 Segment LED Tests	Displays 0-F on the left and then right LED.
8	i82545 Ethernet Configuration	Programs the i82545 EEPROM with the desired MAC address.
9	Timer Test	Outputs a period "." every second.
10	PCI Bus test	Scans the PCI-X bus for any devices present and prints out device/vendor ID and BAR information.
11	CPU Cache Loop (No return)	Places processor in a tight cache loop with no return.
12	Batch Memory Tests	Executes the Memory Tests Diagnostic (#1) to all available SDRAM addresses.
13	Multi-Path Memory Test	Executes a parallel memory test using all 3 ADMA units and a core memory copy.
14	NVRAM Test	Executes a standard memory test, but also checks if data has already been written in a previous test run to verify if the memory retained its values since the last execution.
0	Quit	Exits the application.

4.2.2 Reflashing the Memory

The boards are delivered with the RedBoot* Bootloader installed in Flash. You can load your own software into memory using a variety of tools. Appendix E provides a tutorial for using Intel's XDB to load custom software onto the CRB. Appendix F provides a tutorial for using Accelerated Technology, Inc.'s Nucleus Edge software to load custom software onto the CRB.

If these tools are unavailable, you can also use Intel's Flash Recovery Utility that is provided on the CD included in your kit.

4.2.2.1 Using the FRU Utility to Load Software into Memory

FRUCMD.EXE is a host-based program that lets you load your own software into the flash memory on the CRB and run diagnostic tests. FRU version 6.06 is included on the CD in your kit. The manual contains instructions for using the FRU utility.

4.2.3 81348 Peripheral Bus Memory Map

The Table 8 is the physical memory map of the devices on the 81348 peripheral bus register. Refer to Section 5.7 through Section 5.10 for register descriptions.

Table 8. Peripheral Bus Memory Map

Address Range (in Hex)	Size	Data Bus Width	Description
F000 0000 - F07F FFFF	8 MB	8-bit or 16-bit	Flash memory (re-mapped)
F200 0000 -F200 FFFF	64 KB	8-bit	Product Code register
F201 0000 -F201 FFFF	64 KB	8-bit	Board Stepping register
F202 0000 -F202 FFFF	64 KB	8-bit	CPLD Firmware Revision register
F203 0000 -F203 FFFF	64 KB	8-bit	Discrete LEDs register
F204 0000 -F204 FFFF	64 KB	8-bit	Hex Display Left register
F205 0000 -F205 FFFF	64 KB	8-bit	Hex Display Right register
F206 0000 -F206 FFFF	64 KB	8-bit	Buzzer Control register
F207 0000 -F207 FFFF	64 KB	8-bit	32KB NVRAM
F20D 0000 -F20D FFFF	64 KB	8-bit	Rotary Switch register

4.2.4 RedBoot Memory Map

Table 9 lists the memory map established by Redboot when the CRB boots.

Table 9. RedBoot* Memory Map

Virtual Address	Physical Address (Internal Bus)	Size (MB)	Description
0x00000000	0x0.0000.0000	2048	SDRAM - 64-bit ECC
0x80000000	0x1.8000.0000	128	ATU-X Outbound Memory Translation Window
0x88000000	0x2.8800.0000	128	ATUe Outbound Memory Translation Window
0x90000000	0x0.9000.0000	1	ATU-X Outbound IO Window
0x90100000	0x0.9010.0000	1	ATUe Outbound IO Window
0x90200000	----------	254	Unused
0xA0000000	0x0.0000.0000	512	SDRAM - 64-bit ECC - Uncached Alias
0xC0000000	0x0.C000.0000	1	Cache Flush, DCache Lock
0xC0100000	----------	255	Unused
0xD0000000	----------	512	Unused (32-bit ECC not enabled)
0xF0000000	0x0.F000.0000	32	Flash (PCE0#)
0xF2000000	0x0.F200.0000	1	PCE1#
0xF2100000	0x0.F200.0000	1	PCE1# - cached/buffered
0xF2100000	----------	14	Unused
0xF3000000	0x0.F300.0000	1	unused
0xF3100000	----------	204	Unused

4.2.5 RedBoot Files Included in the CRB Kit

The CD included in your kit contains a copy of the RedHat eCos for the CRB.

- The RedBoot initialization code source files from the following location:
  From the installed directory:
  ..\RedHat\eCos\packages\hal\arm\xscale\IQ81348SC\current\include

- The RedBoot binary image files (downloadable onto Flash) from the following location:
  From the installed directory:
  ..\RedHat\eCos\loaders\

If you need to rebuild the RedBoot image that comes already loaded onto the board, refer to "Redboot Diagnostics" on page 21.

To access GNUPro tools for compiling and building RedBoot, you may also go to the following location on the Intel site:

- http://developer.intel.com/design/intelxscale/dev_tools/031121/index.htm

Also refer to the RedBoot user's guide:

- http://sources.redhat.com/ecos/docs-latest/redboot/redboot-guide.html

4.2.6 RedBoot* Initialization Sequence

In order to set the ECC bits correctly, the DDR memory system must be written to with a known value. This process requires writing to the entire DDR memory intended for use. The following explains the sequence for memory initialization by RedBoot on a CRB with an ECC DIMM.

Initialization Sequence overview:

1. Disable interrupts. (Technically they are disabled at reset, but for soft reset this is included.
2. Init PBIU (Peripheral Bus Interface Unit) chip selects.
3. Enable I cache.
4. Move Flash to 0xF0000000.
5. Set TTB and Enable MMU.
6. Lock data cache as temporary RAM to create stack space for c-function calls.
7. Read SDRAM SPD device for memory parameters.
8. Set Memory Parameters.
9. Enable SDRAM.
10. Scrub SDRAM by writing zeros to all locations in order to initialize the ECC fields in the memory.
11. Delete the temporary data cache RAM.
12. Enable ECC.
13. Copy page table entries to SDRAM.
14. Update TTB register.
15. Initialize UART.
16. Initialize PCI-X bus.
17. Load GigE Ethernet driver.

4.2.7 Software Reset

Software reset through the JTAG port resets the 81348 processor only and breaks the PCI Express communication link to the server system. This type of reset clears the registers that have been loaded by the server system for communication with the board.

Resetting the CRB through JTAG should only be done when communication to the host system through the PCI express interface is not required. To reset the CRB without losing the host interface register settings, use the "hot debug" feature in the XDB JTAG debugger. Refer to Appendix E and Appendix F for instructions on how to use the JTAG hot debug feature.

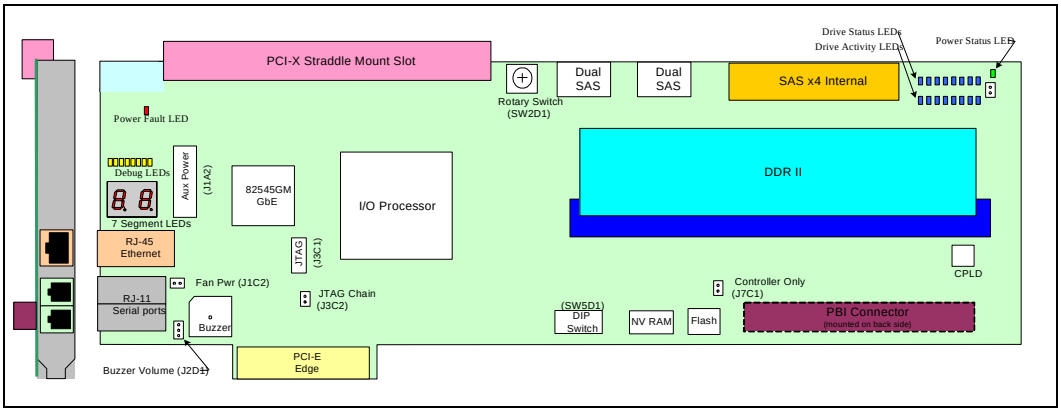
5.0 Hardware / Architecture Reference

This document can be used as a reference by designers creating boards for custom applications. A high level Bill of Materials is included in "Bill of Materials" on page 46 and schematics are available in a separate document on Intel's Web site.

Note: The design for the CRB does not fully comply with the specifications included in the Design Guide for 81348. When using the CRB as a reference design, specifications in the Design Guide take precedence over anything included in the schematics.

Figure 5 shows a diagram of the CRB connectors, jumpers and key components.

Figure 5. CRB Layout



5.1 Memory Subsystem

The memory subsystem is composed of DDR II SDRAM, flash and non-volatile RAM circuits.

5.1.1 DDR II SDRAM

The CRB memory subsystem has the following features:

- One DDRII SDRAM DIMM socket supporting the following configurations:
- 400 MHz or 533 MHz DIMM
- Registered or unbuffered DIMM
- With or without ECC
- CAS latency of 3, 4 or 5
- Single-sided or double-sided DIMM
- I2C bus 2 connected to SDRAM for serial presence detection

The CRB features a single 240-pin right-angled DDRII DIMM connector for 64-bit or 72-bit DDRII DIMM modules. The 3-bit base SPD address is set for binary '000', so the complete SPD address is binary '1010000'. The 81348 component supports registered or unbuffered DIMMs with or without ECC, memory speeds of 400 MHz or 533 MHz, in sizes ranging from 256 MB up to 2 GB and CAS latency of 3, 4, or 5. Memory device widths of x8 or x16 are supported.

5.1.2 Flash ROM

The Flash component used on the CRB is an Intel StrataFlash® 28F640J3C120, which has an 8 Mbyte density. Depending on the switch setting which dictates the width of the PBI bus, the flash can support 8-bit or 16-bit transactions.

5.1.3 NVRAM

The CRB has an STK14C88-3 32Kbyte NVRAM device. This memory is used by some RAID stack software vendors to store configuration information. It is not used by the Intel Redboot software.

5.2 Connectors

5.2.1 Serial-UART Communication

The I/O processor has two integrated UARTs. Each asynchronous serial port supports all the functions of a 16550 UART. The UART signals are connected to a dual RS-232 buffer and then to a pair of RJ-11 serial port connectors mounted on the CRB I/O bracket.

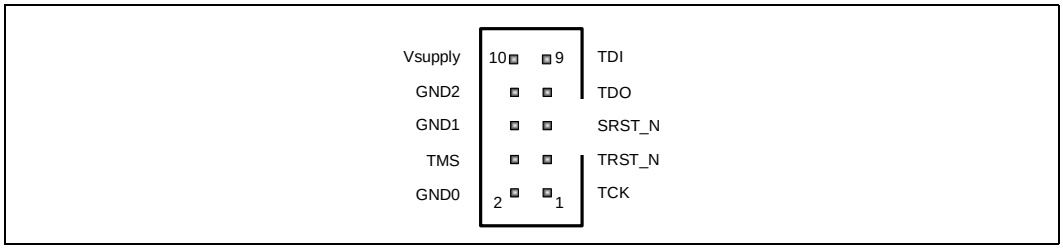
5.2.2 PBI Connector

The PBI connector interface consists of a 100-pin interface to an optional daughter card module. The interface is created using two 50-pin connectors. Signals on the PBI connector interface include the PBI bus (data, address, and control), battery backup support signals, and other miscellaneous signals. The connector accepts PBI personality modules. It can also be used for prototyping auxiliary circuits.

5.2.3 JTAG Port

This board includes a mini JTAG header for debug. The JTAG header is a 10-pin 2mm shrouded header. Figure 6 shows the pinout for the JTAG header.

Figure 6. JTAG Port Pin-out



A 2-pin jumper controls which components are included in the JTAG scan chain. When the jumper is not installed only the I/O processor is in the JTAG chain. When the jumper is installed all the major components are included in the JTAG scan chain. The scan chain order is:

1. I/O processor

2. Gigabit Ethernet
3. CPLD

Refer to Section 5.5, “Switch and Jumper Settings” on page 30 for jumper details.

5.2.4 Network Communication

Using a standard network connection, the user can communicate with the CRB via the Ethernet port. RedBoot also allows the user to remotely boot the platform using a BOOTP server through the network connection.

The CRB has a gigabit Ethernet port that is connected to the 81348 PCI-X interface. It utilizes the Intel® 82545GM controller (82545GM) with integrated Media Access Control (MAC) and Physical Layer (PHY). The controller is capable of transmitting and receiving data rates of 1000 Mbps, 100 Mbps, or 10 Mbps. The CRB implements the PCI-X bus as a 64-bit bus with a maximum frequency of 100 MHz.

5.2.4.1 10/100/1000 Base-T RJ45 Ethernet Connector

The CRB has an RJ45 1 Gbit Ethernet jack with built-in magnetics and LEDs. One green LED and one amber/green LED is built into the RJ-45 LAN connector. Table 10 describes the LED states when the board is powered up and the Ethernet subsystem is operating.

Table 10. Ethernet Connector LED States

LED	LED Color	LED State	Condition
Speed (Right)		Off	10 Mbit/s data rate is selected.
	Green	On	100 Mbit/s data rate is selected.
	Amber	On	1000 Mbit/s data rate is selected.
Link / Active (Left)		Off	LAN link is not established.
	Green	On (steady state)	LAN link is established.
	Green	Blinking	The system is transmitting or receiving data.

5.3 Storage Clock

The storage input clock to the 81348 component is generated from an ICS 843021 LVPECL clock generator. The clock generator output is level-shifted and AC coupled to match the 81348 component storage clock input voltage level requirements. The 81348 component has internal termination so the AC coupled clock does not need any external biasing between the AC coupling capacitors and the 81348 storage clock input.

5.4 Resetting the Board

There is no reset switch provided on the board.

Software reset through the JTAG port resets the 81348 processor only and breaks the PCI express communication link to the server system. This type of reset clears the registers that have been loaded by the server system for communication with the board.

5.5 Switch and Jumper Settings

The CRB has a 10-pin DIP switch, rotary switch and 4 jumpers for board configuration. Please refer to the *Intel® 81348 I/O Processor Developer's Manual* for a more detailed description of these configuration options.

5.5.1 DIP Switch SW5D1

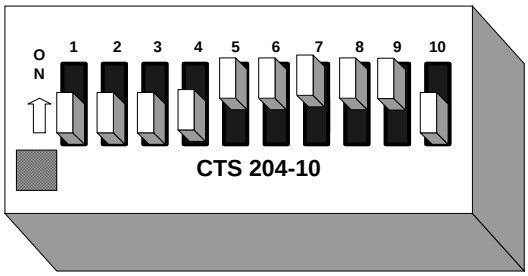
Most configuration options are set by a 10-position DIP switch (SW5D1). Table 11 describes the function of each switch. The default switch setting is shown in BOLD type.

Table 11. SW5D1 Default Switch Position Descriptions

Switch	Setting			Description	Default
1	On **Off**			8-bit Flash interface **16-bit Flash interface**	x
2	On **Off**			PCI Configuration Cycles Enabled **PCI Configuration Cycles Disabled**	x
3	On **Off**			Hold CPU X0 in Reset **Allow CPU X0 to Run Normally**	x
4	On **Off**			Hold CPU X1 in Reset **Allow CPU X1 to Run Normally**	x
5	**On** Off			**533 MHz DDRII SDRAM Interface Speed** 400 MHz DDRII SDRAM Interface Speed	x
6, 7 and 8	6	7	8	Note: 6, 7 and 8 used in conjunction to determine storage port setting.	
	On	**On**	**On**	**8 Ports on ATU, Zero Port on TPMI**	x
	On	On	Off	7 Ports on ATU, One Port on TPMI	
	On	Off	On	6 Ports on ATU, Two Ports on TPMI	
	On	Off	Off	Reserved	
	Off	On	On	4 Ports on ATU, Four Ports on TPMI	
	Off	On	Off	Reserved	
	Off	Off	On	Reserved	
	Off	Off	Off	0 Ports on ATU, Eight Ports on TPMI	
9	**On** Off			**Firmware Timer Disabled** Firmware Timer Enabled	x
10	On **Off**			Reserved - always set this switch to OFF (Default)	x

Figure 7 shows the default SW5D1 DIP switch settings.

Figure 7. DIP Switch SW5D1 10-Position Default Switch Settings



5.5.2 Rotary Switch SW2D1

A 16-position rotary switch is included for debug to enable software to boot into different modes. The Redboot software installed on this board does not currently use the rotary switch. The default rotary switch setting is position 0.

5.5.3 Configuration Jumpers

The CRB has four configuration jumpers. Table 12 describes the configuration jumper options.

Table 12. Configuration Jumper Descriptions

Jumper	Pins Shorted or Open	Description	Default
J2D1	All Open **1-2 Shorted** 2-3 Shorted	Buzzer Disabled **Buzzer Loud** Buzzer Soft	x
J3C2	**1-2 Open** 1-2 Shorted	**I/O Processor only in JTAG chain** JTAG chain includes I/O Processor, Ethernet and CPLD	x
J7C1	**1-2 Open** 1-2 Shorted	**Enable I/O Processor mode** Disable I/O Processor and run board as disk controller only	x
J9A1	1-2 Open **1-2 Shorted**	Route SGPIO signals to the x4 SAS Connector SGPIO pins **Route SGPIO signals to the Hard Drive LEDs**	x

5.6 GPIOs

All GPIO signals are routed to the PBI connector. GPIO<4..0> are also routed to the CPLD for design flexibility.

5.7 CPLD

The CRB contains the CPLD for monitoring and controlling various board features. The CPLD is preprogrammed at the factory with support for debug LEDs, buzzer, rotary switch and backup battery.

5.7.1 CPLD Register Descriptions

Table 13 through Table 20 shows the CPLD register definitions. The read/write column indicates whether the bits are read only (R) or can be read and written (R/W). Writes have no effect on bits that are read only.

Table 8 on page 24 shows the addresses for the PBI devices including CPLD registers.

Table 13 defines the Battery Status Register bits.

Table 13. BAT_STAT Register Definition (Sheet 1 of 2)

Bit	Read/ Write	Name	Definition
0	R	Battery Present	0 = DDR backup battery is present. 1 = No DDR backup battery.
1	R	Battery Charged	0 = DDR backup battery is not fully charged. 1 = DDR backup battery is fully charged.

Table 13. BAT_STAT Register Definition (Sheet 2 of 2)

Bit	Read / Write	Name	Definition
2	R	Battery Discharged	0 = DDR backup battery is not fully discharged. 1 = DDR backup battery is fully discharged.
3	R/W	Battery Enable	0 = Disable DDR backup battery. 1 = Enable DDR backup battery.
4-7		Reserved	Undefined

Table 14 defines the Product Code Register bits.

Table 14. PROD_CODE Register Definition

Bit	Read / Write	Name	Definitions
3-0	R	Product SKU	Identifies variants within a family of products. This value is hard coded in the CPLD firmware. Product SKU definition: 0x1 = PCIE CRB
7-4	R	Product Code	Identifies the product family. This value is hard coded in the CPLD firmware. 0x6 = 8134x CRBs.

Table 15 defines the Board Stepping Register bits.

Table 15. BOARD_STEPPING Register Definition

Bit	Read / Write	Name	Definition
3-0	R	Board Rework Level	Identifies rework completed on the board. This value is hard coded in the CPLD firmware.
7-4	R	Board Stepping	Identifies the board stepping. This value is hard coded in the CPLD firmware.

Table 16 defines the CPLD Firmware Revision Register bits.

Table 16. CPLD_FW Register Definition

Bit	Read / Write	Name	Definition
7-0	R	CPLD Code Revision	Identifies the CPLD code revision level. This value is hard coded into the CPLD.

5.8 Audio Buzzer

The CRB has a self-oscillating audio buzzer, which can be turned on and off by writing to the Buzzer Control Register located in the CPLD. Jumper J2D1 adjusts the volume from off, to low, to high. Please refer to Section 5.5, “Switch and Jumper Settings” on page 30 for jumper details.

Table 17 defines the Buzzer Control Register bits.

Note: The audio buzzer register is not implemented in the 8134x CRB CPLD.

Table 17. BUZZER_CTRL Register

Bit	Read/ Write	Name	Definition
0	R/W	Buzzer On	0 = Buzzer sound off. 1 = Buzzer sound on.
7-1		Reserved	Undefined.

5.9 Rotary Switch

Table 18 defines the Rotary Switch Register bits.

Table 18. ROT_SW Register

Bit	Read/ Write	Name	Definition
3-0	R	Rotary switch	Rotary switch setting.
7-4	R	Reserved	Undefined

The CRB provides a rotary switch to enable user selection of different software configuration options. Redboot does not currently support any rotary switch options on this CRB. The rotary switch settings may be used with other software applications. The rotary switch setting can be read from the Rotary Switch Register in the CPLD. Please refer to Section 4.2.3, “81348 Peripheral Bus Memory Map” on page 23 for the Rotary Switch Register address range.

5.10 LEDs

5.10.1 Seven-Segment Displays

Two 7-segment displays are provided for software debug. The CPLD provides registers that are used to turn on each segment of the display. The usage of the displays is dependent upon the firmware implementation. Redboot provides boot codes on these displays.

Table 19 defines the seven-segment LED register bits. Figure 8 shows how the LED segments are positioned in the seven-segment display. When reset is asserted the CPLD latches 0x7F and illuminates all segments except segment G. When reset is de-asserted the CPLD latches 0x7E and turns off the decimal point.

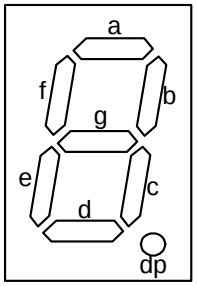
Table 19. SEVEN_SEG_LED Register

Bit	Read/ Write	Name	Definition
0	R/W	Decimal Point	0 = Decimal point off. 1 = Decimal point illuminated.
1	R/W	Segment A	0 = Segment off. 1 = Segment illuminated.
2	R/W	Segment B	0 = Segment off. 1 = Segment illuminated.
3	R/W	Segment C	0 = Segment off. 1 = Segment illuminated.
4	R/W	Segment D	0 = Segment off. 1 = Segment illuminated.

Table 19. SEVEN_SEG_LED Register

Bit	Read / Write	Name	Definition
5	R/W	Segment E	0 = Segment off. 1 = Segment illuminated.
6	R/W	Segment F	0 = Segment off. 1 = Segment illuminated.
7	R/W	Segment G	0 = Segment off. 1 = Segment illuminated.

Figure 8. Seven-Segment Display



5.10.2 Descrete LED Array

Eight amber descrete LEDs are provided for debug. Each LED is lit when the corresponding bit in the CPLD Descrete LED register is set.

Table 20 defines the Discrete LED Register bits.

Table 20. DISCRETE_LED Register

Bit	Read / Write	Name	Definition
7-0	R/W	LED	0 = LED off. 1 = LED on.

5.10.3 Storage Interface LEDs

The storage interface has 16 blue LEDs to display disk activity and status for each of the eight storage ports. These LEDs are controlled by the I/O processor. The CRB multiplexes four of the storage interface LED signals with Storage General Purpose I/O signals (SGPIO). The SGPIO signals are routed to the internal x4 SAS connector and comply with the SFF-8485 specification. A jumper selects whether these four storage interface signals are routed to the storage interface LEDs or to the internal x4 SAS connector SGPIO pins.

5.10.4 Power LED

A green LED on the board indicates when power is applied to the board. The LED is lit when the main +3.3 V power rail is powered on.

5.10.5 Power Fault LED

When the PCI-X slot is used it must be powered through the auxiliary power connector. When a PCI or PCI-X card is plugged into the PCI-X slot and the CRB is powered up without power being applied to the auxiliary power connector a red power fault LED will illuminate to indicate the slot needs power.

Appendix A Advanced Redboot Commands

This appendix provides information on the following advanced Redboot commands:

Three versions of RedBoot are used with 81348:

1. **ROM**: Programmed at 0x0 in Flash. It executes out of Flash, but copies its data section, including MMU tables into RAM from Flash for faster access. This version does not work with the Transport Firmware.
2. **ROMRAM**: Programmed at 0x20_0000 in Flash. It starts execution in Flash, but will copy itself into SDRAM and continue execution out of SDRAM. It is designed to work with the Transport Firmware Image.
3. **RAM**: Does not do any board initialization. Its main purpose is to provide the ability to update the ROM image of RedBoot.

A.1 Removing the "+FLASH configuration checksum error or invalid key" message

This message is due to the fact that the configuration area of Flash that RedBoot uses for configuration parameters has not been initialized yet. Therefore, it has not had a valid checksum programmed into it yet. To initialize the configuration area, use the "fconfig" command. Use the "-i" option to force the config area to be written, even if you don't change any parameters. Answer the questions appropriately for your environment.

Note: Note: User entry text is shown in **bold courier font**.

1. RedBoot> **fconfig i**
2. Initialize non-volatile configuration - continue (y/n)? **y**
3. Run script at boot: **false**
4. Use BOOTP for network configuration: **false**
5. Gateway IP address:
6. Local IP address: **10.0.0.111** [user selected IP]
7. Local IP address mask: **255.255.255.0** [user selected netmask]
8. Default server IP address: **10.0.0.5** [user selected server IP]
9. Console baud rate: **115200**
10. DNS server IP address:
11. GDB connection port: **9000**
12. Force console for special debug messages: **false**
13. Network debug at boot time: **false**
14. Update RedBoot non-volatile configuration - continue (y/n)? **y**

A.2 Rebuilding RedBoot

To rebuild Redboot, perform the following steps:

1. % **mkdir /tmp/redboot_build**
2. % **cd /tmp/redboot_build**

Note: The environment variable ECOS_REPOSITORY must point to the eCos/packages source tree.

3. % **ecosconfig new IQ81348SC redboot**

```
U CYGHWR_REDBOOT_ARM_LINUX_EXEC_ADDRESS_DEFAULT, new inferred value 0xA0008
U CYGNUM_HAL_ARM_XSCALE_CORE_GENERATION, new inferred value 3
U CYGNUM_HAL_ARM_XSCALE_IOP_GENERATION, new inferred value 4
U CYG_PCI_MAX_DEV, new inferred value 32
U CYGSEM_HAL_ARM_XSCALE_IOP_PROVIDE_I2C_API, new inferred value 1
U CYGNUM_HAL_ARM_XSCALE_IOP_FLASH_PHYS_BASE, new inferred value 0xF0000000
U CYGNUM_HAL_ARM_XSCALE_IOP_MMRS_PCI_TVR, new inferred value 0xFFD00000
U CYGNUM_HAL_ARM_XSCALE_IOP_BAR0_FUNCTION, new inferred value RAM
U CYGNUM_HAL_ARM_XSCALE_IOP_BAR1_FUNCTION, new inferred value NOTHING
U CYGNUM_HAL_ARM_XSCALE_IOP_BAR2_FUNCTION, new inferred value NOTHING
U CYGSEM_HAL_USE_ROM_MONITOR, new inferred value 0
```

4. % **ecosconfig import**

```
${ECOS_REPOSITORY}/hal/arm/xscale/IQ81348SC/current/misc/redboot_ROM.ecm
```

Note: Substitute ROM with the version you want, either: ROM, ROMRAM, or RAM

5. % **ecosconfig tree**
6. % **make**
   The resulting RedBoot files will be in the *install/bin* folder.

A.3 Using RedBoot to Update the RedBoot Image

This procedure is used on Redboot versions **ROM** and **ROMRAM**. For the **ROM** version, perform all steps that follow. For the **ROMRAM** version, only perform Steps 4 and 5.

To determine which version of Redboot you are running, type “ver” at the RedBoot prompt and look at the header:

```
RedBoot(tm) bootstrap and debug environment [ROM]
```

1. Load the RAM-based RedBoot over tftp or xmodem.

- tftp:
  RedBoot> **load -m tftp redboot_ram.elf**
- xmodem
  RedBoot> **load -m xmodem**

Note: At this point, you need to start an xmodem transfer in your Terminal program to send the redboot_ram.elf file

2. Execute the RAM-based RedBoot
   RedBoot> **go**
3. Wait for the RAM-based RedBoot to start .
4. Load the new Boot Loader over xmodem or tftp but specify a location in RAM as the base address (-b) and load as a “raw” (-r) format.

Note: This example shows a Redboot file named "redboot_july21.bin". Substitute the name of the Redboot file being downloaded in place of this file name.

- tftp:
  RedBoot> **load -m tftp -r -b 0x500000 redboot_july21.bin**
- xmodem
  RedBoot> **load -m xmodem -r -b 0x500000**

Note: At this point, you need to start an xmodem transfer in your Terminal program to send the redboot binary

5. Program the new Boot Loader into flash.
   RedBoot> **fis init**

```
About to initialize [format] FLASH image system - continue (y/n)? y
```

RedBoot> **fis unlock RedBoot**

RedBoot> **fis create RedBoot**

Answer **yes** to over-write the current RedBoot image

```
RedBoot> fis lock RedBoot
```

Example 2. Example of updating RedBoot ROM image via TFTP

```
Ethernet eth0: MAC address 00:0e:0c:80:64:a6
IP: 10.0.0.210/255.255.255.0, Gateway: 0.0.0.0
Default server: 10.0.0.1, DNS server IP: 0.0.0.0

RedBoot(tm) bootstrap and debug environment [ROM]
Intel Intermediate Development release
        version 2.1-experimental
        built 13:35:58, Jul 21 2005

Platform: IQ81348SC (XScale) Core1, DDR2-533
IF_PCIX: 1 COnly: 1 PCIe EP PCI-X CR: PCIX-100
Copyright (C) 2000, 2001, 2002, Red Hat, Inc.

RAM: 0x00000000-0x10000000, 0x0001b280-0x0ffd1000 available
FLASH: 0xf0000000 - 0xf0800000, 64 blocks of 0x00020000 bytes each.
RedBoot> load -m tftp redboot_RAM.elf
Entry point: 0x00050040, address range: 0x00050000-0x0007d1ac
RedBoot> go
+Sending BOOTP Requests...can take up to 30secs to timeout...
Ethernet eth0: MAC address 00:0e:0c:80:64:a6
IP: 10.0.0.210/255.255.255.0, Gateway: 0.0.0.0
Default server: 10.0.0.1, DNS server IP: 0.0.0.0

RedBoot(tm) bootstrap and debug environment [RAM]
Intel Intermediate Development release
        version 2.1-experimental
        built 08:11:49, Jul 11 2005

Platform: IQ81348SC (XScale) Core1, DDR2-533
IF_PCIX: 1 COnly: 1 PCIe EP PCI-X CR: PCIX-100
Copyright (C) 2000, 2001, 2002, Red Hat, Inc.

RAM: 0x00000000-0x10000000, 0x0008ef00-0x0ffd1000 available
FLASH: 0xf0000000 - 0xf0800000, 64 blocks of 0x00020000 bytes each.
RedBoot> fis init
About to initialize [format] FLASH image system - continue (y/n)? y
*** Initialize FLASH Image System
   Warning: device contents not erased, some blocks may not be usable
... Unlock from 0xf07e0000-0xf0800000: .
... Erase from 0xf07e0000-0xf0800000: .
... Program from 0x0ffdf000-0x0ffff000 at 0xf07e0000: .
... Lock from 0xf07e0000-0xf0800000: .
RedBoot> fis list
Name              FLASH addr  Mem addr    Length      Entry point
RedBoot           0xF0000000  0xF0000000  0x00040000  0x00000000
RedBoot config    0xF07C0000  0xF07C0000  0x00001000  0x00000000
```

```
FIS directory    0xF07E0000  0xF07E0000  0x00020000  0x00000000
RedBoot> load -r -b 0x500000 redboot_july21_rom.bin
Using default protocol (TFTP)
Raw file loaded 0x00500000-0x005393eb, assumed entry at 0x00500000
RedBoot> fis unlock RedBoot
... Unlock from 0xf0000000-0xf0040000: ..
RedBoot> fis create RedBoot
An image named 'RedBoot' exists - continue (y/n)? y
... Erase from 0xf0000000-0xf0040000: ..
... Program from 0x00500000-0x005393ec at 0xf0000000: ..
... Unlock from 0xf07e0000-0xf0800000: .
... Erase from 0xf07e0000-0xf0800000: .
... Program from 0x0ffdf000-0x0ffff000 at 0xf07e0000: .
... Lock from 0xf07e0000-0xf0800000: .
RedBoot> fis lock RedBoot
... Lock from 0xf0000000-0xf0040000: ..
RedBoot>
```

Note: At this point, you would reset.

Example 3. Example of Updating from ROM Image to ROMRAM and SAS Transport Firmware

```
+Sending BOOTP Requests...can take up to 30secs to timeout...
Ethernet eth0: MAC address 00:0e:0c:80:64:a6
IP: 10.0.0.210/255.255.255.0, Gateway: 0.0.0.0
Default server: 10.0.0.1, DNS server IP: 0.0.0.0

RedBoot(tm) bootstrap and debug environment [ROM]
Intel Intermediate Development release
        version 2.1-experimental
        built 13:35:58, Jul 21 2005

Platform: IQ81348SC (XScale) Core1, DDR2-533
IF_PCIX: 1 COnly: 1 PCIe EP PCI-X CR: PCIX-100
Copyright (C) 2000, 2001, 2002, Red Hat, Inc.

RAM: 0x00000000-0x10000000, 0x0001b280-0x0ffd1000 available
FLASH: 0xf0000000 - 0xf0800000, 64 blocks of 0x00020000 bytes each.
RedBoot> load redboot_RAM.elf
Using default protocol (TFTP)
Entry point: 0x00050040, address range: 0x00050000-0x0007d1ac
RedBoot> go
+Sending BOOTP Requests...can take up to 30secs to timeout...
Ethernet eth0: MAC address 00:0e:0c:80:64:a6
IP: 10.0.0.210/255.255.255.0, Gateway: 0.0.0.0
Default server: 10.0.0.1, DNS server IP: 0.0.0.0

RedBoot(tm) bootstrap and debug environment [RAM]
Intel Intermediate Development release
        version 2.1-experimental
        built 08:11:49, Jul 11 2005
```

```
Platform: IQ81348SC (XScale) Core1, DDR2-533
IF_PCIX: 1 COnly: 1 PCIe EP PCI-X CR: PCIX-100
Copyright (C) 2000, 2001, 2002, Red Hat, Inc.

RAM: 0x00000000-0x10000000, 0x0008ef00-0x0ffd1000 available
FLASH: 0xf0000000 - 0xf0800000, 64 blocks of 0x00020000 bytes each.
RedBoot> load -r -b 0x500000 Pre-Alpha_Transport_FW_2MB.bin
Using default protocol (TFTP)
Raw file loaded 0x00500000-0x006fffff, assumed entry at 0x00500000
RedBoot> fis list
Name              FLASH addr  Mem addr    Length      Entry point
RedBoot           0xF0000000  0xF0000000  0x00040000  0x00000000
RedBoot config    0xF07C0000  0xF07C0000  0x00001000  0x00000000
FIS directory     0xF07E0000  0xF07E0000  0x00020000  0x00000000
RedBoot> fis unlock -f 0xf0000000 -l 0x200000
... Unlock from 0xf0000000-0xf0200000: ................
RedBoot> fis write -b 0x500000 -f 0xf0000000 -l 0x200000
* CAUTION * about to program FLASH
          at 0xf0000000..0xf01fffff from 0x00500000 - continue (y/n)? y
... Erase from 0xf0000000-0xf0200000: ................
... Program from 0x00500000-0x00700000 at 0xf0000000: ................
RedBoot> load -r -b 0x500000 redboot_romram.bin
Using default protocol (TFTP)
Raw file loaded 0x00500000-0x0053bfff, assumed entry at 0x00500000
RedBoot> fis unlock -f 0xf0200000 -l 0x40000
... Unlock from 0xf0200000-0xf0240000: ..
RedBoot> fis create -b 0x500000 -l 0x40000 -f 0xf0200000
RedBoot_ROMRAM
... Erase from 0xf0200000-0xf0240000: ..
... Program from 0x00500000-0x00540000 at 0xf0200000: ..
... Unlock from 0xf07e0000-0xf0800000: .
... Erase from 0xf07e0000-0xf0800000: .
... Program from 0x0ffdf000-0x0ffff000 at 0xf07e0000: .
... Lock from 0xf07e0000-0xf0800000: .
RedBoot> Go
+Sending BOOTP Requests...can take up to 30secs to timeout...
Ethernet eth0: MAC address 00:0e:0c:80:64:a6
IP: 10.0.0.210/255.255.255.0, Gateway: 0.0.0.0
Default server: 10.0.0.1, DNS server IP: 0.0.0.0

RedBoot(tm) bootstrap and debug environment [ROMRAM]
Intel Intermediate Development release
        version 2.1-experimental
        built 08:53:43, Aug 26 2005

Platform: IQ81348SC (XScale) Core1, DDR2-533
IF_PCIX: 1 COnly: 1 PCIe EP PCI-X CR: PCIX-100
Copyright (C) 2000, 2001, 2002, Red Hat, Inc.
```

```
RAM: 0x00000000-0x10000000, 0x0004de68-0x0ffd1000 available
FLASH: 0xf0000000 - 0xf0800000, 64 blocks of 0x00020000 bytes each.
RedBoot> fis list
Name              FLASH addr  Mem addr    Length     Entry point
RedBoot           0xF0000000  0xF0000000  0x00040000  0x00000000
RedBoot config    0xF07C0000  0xF07C0000  0x00001000  0x00000000
FIS directory     0xF07E0000  0xF07E0000  0x00020000  0x00000000
RedBoot_ROMRAM    0xF0200000  0xF0200000  0x00040000  0x00500000
RedBoot> fis init
About to initialize [format] FLASH image system - continue (y/n)? y
*** Initialize FLASH Image System
    Warning: device contents not erased, some blocks may not be usable
... Unlock from 0xf07e0000-0xf0800000: .
... Erase from 0xf07e0000-0xf0800000: .
... Program from 0x0ffdf000-0x0ffff000 at 0xf07e0000: .
... Lock from 0xf07e0000-0xf0800000: .
RedBoot> fis list
Name              FLASH addr  Mem addr    Length     Entry point
(reserved)        0xF0000000  0xF0000000  0x00200000  0x00000000
RedBoot           0xF0200000  0xF0200000  0x00040000  0x00000000
RedBoot config    0xF07C0000  0xF07C0000  0x00001000  0x00000000
FIS directory     0xF07E0000  0xF07E0000  0x00020000  0x00000000
RedBoot>
```

Note: At this point, you would reset.

A.4 Using RedBoot to Update the Transport Firmware Image

Note: This procedure only applies to the **ROMRAM** Version.

Note: This example shows a transport firmware file named "Pre-Alpha_Transport_FW_2MB.bin". Substitute the name of the transport firmware file being downloaded in place of this file name.

Perform these steps:

1. Download the Transport Firmware Image into SDRAM:.
   RedBoot> **load -m tftp -r -b 0x500000 Pre-Alpha_Transport_FW_2MB.bin**
2. Initialize the Flash File system.
   RedBoot> **fis init**
3. Unlock the Flash Blocks where the Transport FW will be stored.
   RedBoot> **fis unlock -f 0xf0000000 -l 0x200000**
4. Write the Transport FW to Flash.
   RedBoot> **fis write -b 0x500000 -f 0xf0000000 -l 0x200000**
5. Lock the Transport FW image.
   RedBoot> **fis lock -f 0xf0000000 -l 0x200000**

Example 4. Example of Updating the Transport Firmware Image via TFTP

```
Ethernet eth0: MAC address 00:0e:0c:80:64:a6
IP: 10.0.0.210/255.255.255.0, Gateway: 0.0.0.0
Default server: 10.0.0.1, DNS server IP: 0.0.0.0
```

```
RedBoot(tm) bootstrap and debug environment [ROMRAM]
Intel Intermediate Development release
        version 2.1-experimental
        built 11:20:12, Jul 21 2005

Platform: IQ81348SC (XScale) Core1, DDR2-533
IF_PCIX: 1 COnly: 1 PCIe EP PCI-X CR: PCIX-100
Copyright (C) 2000, 2001, 2002, Red Hat, Inc.

RAM: 0x00000000-0x10000000, 0x00049e68-0x0ffd1000 available
FLASH: 0xf0000000 - 0xf0800000, 64 blocks of 0x00020000 bytes each.
RedBoot> fis init
About to initialize [format] FLASH image system - continue (y/n)? y
*** Initialize FLASH Image System
   Warning: device contents not erased, some blocks may not be usable
... Unlock from 0xf07e0000-0xf0800000: .
... Erase from 0xf07e0000-0xf0800000: .
... Program from 0x0ffdf000-0x0ffff000 at 0xf07e0000: .
... Lock from 0xf07e0000-0xf0800000: .
RedBoot> fis list
Name              FLASH addr  Mem addr    Length     Entry point
(reserved)        0xF0000000  0xF0000000  0x00200000  0x00000000
RedBoot           0xF0200000  0xF0200000  0x00040000  0x00000000
RedBoot config    0xF07C0000  0xF07C0000  0x00001000  0x00000000
FIS directory     0xF07E0000  0xF07E0000  0x00020000  0x00000000
RedBoot> load -m tftp -r -b 0x500000 Pre-Alpha_Transport_FW_2MB.bin
Raw file loaded 0x00500000-0x006fffff, assumed entry at 0x00500000
RedBoot> fis unlock -f 0xf0000000 -l 0x200000
... Unlock from 0xf0000000-0xf0200000: ................
RedBoot> fis write -b 0x500000 -f 0xf0000000 -l 0x200000
* CAUTION * about to program FLASH
        at 0xf0000000..0xf01fffff from 0x00500000 - continue (y/n)? y
... Erase from 0xf0000000-0xf0200000: ................
... Program from 0x00500000-0x00700000 at 0xf0000000: ................
RedBoot> fis lock -f 0xf0000000 -l 0x200000
... Lock from 0xf0000000-0xf0200000: ................
RedBoot>
```

Note: At this point, you would reset.

A.5 Initialize the Flash File System

RedBoot> **fis init -f**

About to initialize [format] FLASH image system - continue (y/n)? y

*** Initialize FLASH Image System

... Erase from 0xf0040000-0xf07c0000: .....

.....

... Erase from 0xf07e0000-0xf07e0000:

... Erase from 0xf0800000-0xf0800000:

... Unlock from 0xf07e0000-0xf0800000: .

... Erase from 0xf07e0000-0xf0800000: .

... Program from 0x0ffdf000-0x0ffff000 at 0xf07e0000: .

... Lock from 0xf07e0000-0xf0800000: .

A.6 Display the Current File System in Flash

RedBoot> **fis list**

Name	FLASH addr	Mem addr	Length	Entry point
RedBoot	0xF0000000	0xF0000000	0x00040000	0x00000000
RedBoot config	0xF07C0000	0xF07C0000	0x00001000	0x00000000
FIS directory	0xF07E0000	0xF07E0000	0x00020000	0x00000000

Appendix B Troubleshooting and FAQ

B.1 Common Problems and Possible Solutions

You may need to start HyperTerminal on the attached development/debug system before booting the CRB.

B.2 Frequently Asked Questions

Q1: Why doesn't the audio buzzer work?

A1: The Audio buzzer is disabled in CPLD code due to CPLD resource constraints.

Appendix C Bill of Materials

C.1 Key Components

Table 21 provides a vendor list as a service to our customers for reference only. The inclusion of this list should not be considered a recommendation or product endorsement by Intel® Corporation.

Table 21. Component Reference

Component	Part Number	Additional Information
MAX8544	MAX8544	Manufacturer: Maxim Semiconductor* URL: http://www.maxim-ic.com/quick_view2.cfm/qv_pk/4309
Switching FET	Si4390DY	Manufacturer: Vishay Siliconix* URL: http://www.vishay.com
Switching FET	Si4842DY	Manufacturer: Vishay Siliconix URL: http://www.vishay.com
Storage Clock Generator	ICS843021	Manufacturer: Integrated Circuit Systems, Inc.* URL: http://www.icst.com/hiperclocks.aspx
DDR DIMM Connector	AU0401H-R3[1]	Manufacturer: Foxconn* URL: http://www.foxconn.com
82545GM Controller	RC82545GM	Manufacturer: Intel® Corporation URL: http://developer.intel.com/design/network/products/lan/controllers/82545gm.htm
EEPROM	AT93C46	Manufacturer: Atmel/Catalyst* URL: http://www.atmel.com
Crystal	MA-406 25.00M 20 pF	Manufacturer: EPSON* URL: http://www.eea.epson.com
CPLD	LC4064Z_M56C	Manufacturer: Lattice* URL: http://www.latticesemi.com/
Rotary Switch	94HAB16W	Manufacturer: Grayhill* URL: http://www.grayhill.com/index.htm
Intel StrataFlash®	28F640J3C120	Manufacturer: Intel® Corporation URL: http://intel.com/design/flcomp/datashts/290667.htm
PCI-X Connector	89402-9203	Manufacturer: Molex* URL: http://www.MOLEX.com/

1. This exact part number does not show up on the Foxconn web site. The closest match is AU04017-R3. The only difference between these parts is that the AU04017-R3 has 15 u" gold plating and the AU0401H-R3 has 30 u" gold plating and is lead free.

C.2 Complete Bill of Materials

Contact your Intel representative for a copy of the Bill of Materials.

Appendix D Schematics

Contact your Intel representative for a copy of the schematic files or refer to Table 1, “Related Documentation List” on page 9 for manual information.

Appendix E Intel® C++ Software Development Tool Suite 2.0 Tutorial

E.1 Introduction

This appendix is a tutorial for building and debugging code on a CRB using the Intel® C++ Software Development Tool Suite 2.0. Historically, most of our customers have used the GNU compiler, so this tutorial will build an application using the Integrated Development Environment and using a makefile with a GNU compiler. Both builds can then be debugged using the Intel® XDB Debugger. Also, a normal connection which resets the target and a Hot-Debug connection which does not reset the target will be discussed.

E.1.1 Purpose

The purpose of this appendix is to help the user setup and become familiar with the CRB and other related hardware and software. This appendix steps the user through an example program using the Intel® C++ Software Development Tool Suite 2.0 and the Macraigor* Raven* JTAG. It includes hardware setup, software setup, building a project, debugging the project, and a tour of some of the debug features.

E.1.2 Required Hardware and Software

This tutorial requires a PC, the Intel® C++ Software Development Tool Suite 2.0, a Macraigor* Raven*, a CRB and accompanying cables, and a PCI Express host system or a PCI Express back plane. (If a back plane is used, the CRB must be placed in root complex mode. Refer to Section 3.3.1 for more information).

E.1.3 Related Web Sites

- Macraigor Systems: http://www.macraigor.com/
- Intel Developer's page for I/O processors: http://developer.intel.com/design/iio/
- Intel Developer's page for I/O processors documentation: http://developer.intel.com/design/iio/docs/iop348.htm

E.2 Setup

E.2.1 Hardware Setup

Perform the following steps to set up the hardware:

1. Set the dip switches:
   - Turn Switches 1, 2, 4, 9, and 10 off
   - Turn Switch 3 on
   - Turn Switches 5 through 8 on.

Note: When Switch 3 is on, core 0 is held in reset. Core 0 will run the storage firmware that can be loaded on the CRB, so it will not be discussed in this tutorial. To keep core 0 from competing for the bus and other devices in the address space, holding it in reset is a good option.

2. If there is a jumper on J3C2, remove it.
3. With power turned off, plug the CRB into a PCI Express slot on the back plane or host system.
4. Connect the parallel cable to the Raven* and to the parallel port on the PC.
5. Connect the 20-pin connector at the end of the Raven's ribbon cable to the 20-pin header on the 20-pin to 10-pin adaptor card which came with the CRB. The connector is keyed to prevent plugging the connector in backward.
6. Connect the small ribbon cable to the 10-pin header. The connector is keyed to prevent plugging the connector in backward.
7. Connect the other end of the small ribbon cable to the small, keyed header on the CRB near the processor cooling fan. The connector is keyed to prevent plugging the connector in backward.
8. Connect one end of the serial cable to the CRB using either RJ11 connector.
9. Connect the other end to the RJ11 to DB9F adaptor and connect the adaptor to the serial port on the PC. (If the PC has USB and does not have a serial port, then use a USB to serial adaptor).
10. Apply power to the PC, the Macraigor* Raven*, and the back plane or host system.

E.2.2 Software Setup

Perform the following steps to set up the software:

1. Close all applications and execute the installation file for Intel® C++ Software Development Tool Suite 2.0. When prompted to restart the computer during the installation, restart it at that time.
2. Using BIOS setup or Control Panel in Windows, set the parallel port to EPP mode. (0378-037F resource allocation is recommended, but other ranges will also work.)

Note: Tera Term*, HyperTerminal*, or a similar console program will be needed for the serial connection.

3. Download the code files from http://developer.intel.com/design/iio/swsup/IQ81348SC_for SDT_2_0.htm and place them in a new project directory. The name of the directory does not matter. After unzipping, the directory should contain the following files:

- Blink.c
- Blink.h
- LED.c
- LED.h
- Start.xdb
- Config_JTAG.xsf
- Makefile
- Temp_path.bat
- Readme.txt

Note: The Help Menu for Intel® C++ Software Development Tool Suite 2.0 is quite extensive. While using this tutorial, use Help in parallel.

4. Download the RedBoot.bin file for the CRB from the following location: http://developer.intel.com/design/iio/swsup/RedBoot.htm. Place RedBoot in a convenient directory.

Note: Be sure to get the ROM version of RedBoot for the CRB.

E.3 New Project Setup

E.3.1 Creating a New Project

Perform the following steps:

1. Launch the Intel® Integrated Development Environment.
2. Select *Project, Create New Workspace and Project,* or use the icon on the Toolbar.
3. The “New Project” window displays.
4. Select *C application for Intel XDB JTAG Debugger* under Project Template and name the project *IQ81348SC* in the name field. Click *OK*.
5. Click on the plus signs in the project tree to expand the tree. Notice that the project is populated with some files to run a hello world (hello.c).
6. Close the Readme file. (It is a file in the project tree so you can read it later.)
7. Select *Tools, Build Manager* or click on the hammer icon.
8. Under “Tool list”, click on each item and notice the changes in the other fields.
9. Under “Tool list”, click on *JTAG Debugger*.
10. The “Options” field has the options that are typically found in a makefile.
11. Delete *-startup*. This option suppresses the startup window of the debugger which you want to examine. Leave the rest of the options unchanged.
12. Under “Tool list”, click on *Linker*.
13. Change the load base to **`-base 0x00020000`** and leave the rest of the options unchanged.
14. Click *OK*.
15. Click *Project, Save Project* to save these changes.
16. Click on the *Build* icon the first time you build.

Note: Use the “Rebuild All” icon for any subsequent builds.

17. Examine the results in the Build Window. The last line should display “Build Completed Successfully”.
18. Launch XDB from the IDE, by clicking on the icon that looks like a “bug under a magnifying glass”.
19. The “Edit Settings of config_jtag.xsf” dialog box pops up.
20. Click on the *CPU-JTAG* tab.
21. **`Make Target Connection lpt1:`**
22. Board: **`IOP348_mz1_tlu`**
23. Check *try Hot-Debug connection*. This will connect to the target without a reset.

Note: A normal connection to the processor will start the session with a reset. This allows you to debug the code from the beginning, but the BIOS configuration of the bus will be lost in the card being debugged. A Hot-Debug connection connects without resetting the card under debug and continues from the point of entry in the code. The code can only be debugged from that point forward; however, the bus enumeration is intact. This tutorial uses a Hot-Debug connection.

24. Click *OK* and XDB will connect to the target.

25. To save time, close XDB and the IDE and unzip the downloaded project files into the CRB workspace directory that you just created. Place files Blink.c, Blink.h, LED.c, LED.h into the same directory as hello.c.
26. Find *Start.xdb* and *Config_JTAG.xsf* and overwrite them.
27. Open the IDE again, right click on the source directory under the project in the workspace window and select *Insert file to project*. Add blink.c and led.c to the project.
28. Right-click on `hello.c` and deleted it from the project.
29. Save the changes to the project.
30. Highlight *IQ81348.bd* and then click on the *Rebuild All* icon (three blue down-arrows).
31. Examine the results in the Build Window. The last line should read “Build Completed Successfully”.

The compiler produces an IQ81348SC.elf file and then converts it into two files that the XDB debugger uses: IQ81348SC.bd and IQ81348SC.hx. One file is the binary executable file and the other is the symbols file. Refer to the *Object Converters Reference Manual* in the help documents for more information.Actual above those is an .ELF file - same questions for that too.

E.4 RedBoot*

E.4.1 Overview

RedBoot* is a debug monitor which initializes the board, has some debug and diagnostic functions, and is capable of serial communication with a console program such as Tera Term* or HyperTerminal*. Since this tutorial is building and debugging a simple piece of code in RAM, RedBoot will be used for target initialization.

E.4.2 Checking and Configuring RedBoot

Perform the following steps to check and configure Redboot:

1. Check RedBoot by powering on the Host system with the CRB plugged into it.
2. Open the console program. (Tera Term* will be used for the purposes of this tutorial.)
3. Select *serial port* and *COM1*. (The COM port can vary, of course, so you may need to check with the Device Manager to determine the appropriate serial port.)
4. Check to see if the following settings display:

- Baud rate: 115200.
- Data: 8 bit.
- Parity: none.
- Stop: 1 bit.
- Flow control: none.

5. Press Enter several times and the RedBoot prompt should redisplay. If this is the case, then RedBoot is healthy and there is no need to reflash.

Note: To reach the RedBoot prompt faster after reset, type **fconfig** and press Enter after the RedBoot prompt.

6. Set **`BOOTP to false`** and save.
   The results should be as follows:

```
RedBoot> fconfig
Run script at boot: false
Use BOOTP for network configuration: false
Gateway IP address:
Local IP address:
Local IP address mask:
Default server IP address:
Console baud rate: 115200
DNS server IP address:
GDB connection port: 9000
Force console for special debug messages: false
Network debug at boot time: false
Update RedBoot non-volatile configuration - continue (y/n)? y
... Unlock from 0xf07c0000-0xf07c1000: .... Erase from 0xf07c0000-0xf07c1000: .
... Program from 0x0ffd2000-0x0ffd3000 at 0xf07c0000: .
... Lock from 0xf07c0000-0xf07c1000: .
RedBoot>
```

E.5 Flashing with JTAG

E.5.1 Overview

Intel® C++ Software Development Tool Suite 2.0 and the Raven* are capable of reading from, writing to, and erasing the contents of the Flash on the evaluation board. The flash programmer only supports the binary file format, so for this tutorial, we will reflash using a RedBoot.bin file.

E.5.2 Configuring and Using the Flash Programmer

1. Turn dip switch 3 on to hold core 0 in reset.
2. Power on the host system with the CRB plugged into the PCI Express slot.
3. Launch the Intel XDB Debugger.
4. In the start up box, click browse and find:
5. C:\Program Files\Intel\SDT2.0\xdb\configurations\ocd\IPO348_mz1_tlu_RV.xsf
6. In the start up box, click *start*.
   Wait for everything to connect properly. The Command window shows the connection details and should end with an XDB prompt. If there are errors in the connection, the error messages will be reflected in this window.

7. Select *Flash, Open Configuration,* and select *IOP348_LED.fcf*.
8. Select *Flash, Burn Flash*.
9. To the right of the Data File field, click the "…" button and find the RedBoot.bin file that was previously downloaded.
10. Make sure all three boxes are checked and click *burn*.
11. After the burn completes, click *close*.
12. Exit XDB and turn off power to the host system.
13. Power up the host system again.

E.6 Building an Executable File

14. Power on the host system. The LEDs on the CRB should display **0.8. > SL > A1**. The board ships with RedBoot loaded into flash, so this is the expected pattern. If the LEDs do not reach A1, then the CRB will need to be reflashed with RedBoot.
15. Launch the Intel IDE.
16. Open project IQ81348SC.
17. Highlight *IQ81348.bd* and then click on the *Rebuild All* icon (three blue down-arrows).
18. Examine the results in the Build Window. The last line should read "Build Completed Successfully".
19. Close Tera Term* or any other console program if it is open. The serial output in this case will go to an output window in the debugger.
20. Launch XDB from the IDE by clicking on the icon that looks like a bug under a magnifying glass.
21. The "Edit Settings of config_jtag.xsf" dialog box pops up.
22. Click on the *CPU-JTAG* tab.
23. Make Target Connection lpt1:
24. Board: **`IOP348_mz1_tlu`**
25. Check *try Hot-Debug connection*. This will connect to the target without a reset. The LEDs should remain at "A1". (If the LEDs change to 0.8., then a reset occurred.)
26. Click *OK* and XDB will connect to the target.
27. Click the *Run* button and then click the *Stop* button.
28. Click the *ProjLoad* button.
29. Click the `setStart` button.
30. Click the *Run* button. The LEDs will scroll and the Output window will display print statements.
31. Click the *Stop* button.

E.7 Touring Some of the Debugger Features

1. Move the cursor arrow over a few buttons. Within a few seconds, a text description of the button appears.

2. Find the Module List Window button and click on it several times. Notice that the window toggles open and closed. Most buttons work this way. The project files are displayed in this window.
3. Double-click on any file in the Module window and it is displayed. Double-click on *blink.c.*
4. Move the cursor arrow to the blue dot on line 17. Right-click and select *Toggle Breakpoint*. Now press *go* a few times. Then, right-click there again and toggle the breakpoint to clear it.
5. More elaborate breakpoints can be set by selecting *Debug, Set Breakpoint*: code, data, conditional, hardware, etc.
6. Click on the *Local Variables* Window button.
7. Click anywhere in line 16 of blink.c and then click the *Run Until* button. The code will stop at a breakpoint on line 16.
8. Click the *Step Into* button a few time (single-steps in C).
9. Click the *Instruction Step* button a few times (single-steps in assembly).
10. Click the *Step Until Caller* button.
11. Close the *Local Variables* window.
12. Click the *Create Memory* window button.
13. Type **`0x4000`** for the address, make size **`long`**, and click **`OK`**.

Caution: This is the memory management unit table, so do not change it unless you are going to reset.

14. Close this window and open a new one at **`0x60000`**, make size **`long`**. Double-click on any data location and change it to **`0xA5A5A5A5`**.
15. Close this window.
16. Click on the *Register* button, do a few single steps, notice the changes to the registers, and close the window.
17. Click on the U button. This window displays all of the memory-mapped registers for the CRB. Click on the plus signs to expand the tree. Double-click on any register to drill down to a more verbose register window.
18. Open the M and C windows and explore them.

Note: The “X” and “D” buttons do not pertain to the CRB so do not use them.

19. Click on the T button. Type **`0x20000`** into the virtual address field and press Enter. Then, type **`0xA0040000`** into the virtual address field and press Enter. Close the window.
20. Click on the *Trace* window button (looks like a camera). This information is a result of the contents of the trace buffer. The window displays the instructions leading up to the breakpoint. Close the window.
21. Click on the *I$* button. This window displays the L1 instruction cache. Close the window. Explore the *D$* and *L2$* windows.
22. Open the *Call Stack* window and do some stepping. Close the window.
23. Open the *Evaluations* window. Double-click to highlight a local variable and drag it into the window. Do this for some other variables. Then do some stepping and watch the changes. Close the window.
24. The help and documentation files can be accessed centrally from Windows. From the Start menu, select *All Programs, Intel Software Development Tools, Intel C++ Software Development Tools Suite 2.0, Documentation, Documentation Index*.

This index has references for the compiler, assembler, linker, the IDE, the debugger, the flash programmer, etc.

E.8 The Configuration Files behind the GUI

1. Launch the IDE and open project IQ81348SC.
2. Double-click to open *config_JTAG.xsf* and *start.xdb*.
3. Click on the *XDB* button to launch it. The "Edit settings of config_jtag.xsf" box pops up. Any changes in this box are captured in the file which is part of the project, so make all changes here rather than trying to manually edit the file.

- Start.xdb is the initial batch file which means that it is run immediately on connection. A batch file can contain any command line commands in it. A batch file can even run another batch file, so these initial files can be set up to take care of routine tasks. In this case, we have defined two buttons that load our program, change the program counter to the beginning of the program, and then run the program up to the beginning of main().
- The swigui.dll plug-in and the SWI heap and stack definitions are used to send the printf statements to the output window.
- If you do a search for all XDB batch files (*.xdb), you will find examples of batch files that can be used as templates and tutorials. The XDB command line command set is very powerful and is very useful in batch files.
- To run a batch file from the GUI, select *file, batch*, find the file, click *OK*, and it runs.

E.9 Compiling with a makefile

Alternatively, you may prefer using GNU make and a GNU GCC compiler.

1. Place the project files (.c, .h, and makefile) in a project directory.
2. Install an Intel XScale-elf-gcc compiler by performing the following steps:
   a. Add a path environment variable to point to **`xscale-elf-gcc.exe`**.
   b. Open a Command Window in the project directory.
   c. Type **`make clean`** and press Enter if this is not the first build and then type **`make`** and press Enter. Otherwise, if this is the first build, type **`make`** and press Enter.
   d. Type **`dwarf2bd hx IQ81348.elf`** and press Enter. This will produce a .bd and a .hx file for the XDB debugger.
   e. Launch XDB and connect with *IOP348_mz1_tlu_RV.xsf*.
   f. Check *try Hot-Debug connection* to connect without a reset.
   g. Run a script that defines the ProjLoad and setStart buttons used in the start.xdb script to load the code into RAM, change the program counter to the start of the code, and run to main and stop.
   h. The user will be prompted to point the debugger to the source code.

E.10 Conclusion

At this point, the user should have a good picture of the evaluation board and the development tools. The tools have many features that have not been covered in this tutorial. The tutorial's purpose was to get the board and tools up and running and to

give the user a feel for the evaluation board and the development tools. Now that a simple project is running, the help menu can provide much information on the features and capabilities of the tools as application development proceeds.

Appendix F Accelerated Technology's Nucleus Edge* Tutorial

F.1 Introduction

This appendix is a tutorial for building and debugging code on a CRB using Accelerated Technology's Nucleus Edge*. Historically, most of our customers have used the GNU compiler, so this tutorial will build an application using the Integrated Development Environment and using a makefile with a GNU compiler. Both builds can then be debugged using the Nucleus Edge* debugger. Also, a normal connection which resets the target and a Hot-Debug connection which does not reset the target will be discussed.

F.2 Purpose

The purpose of this appendix is to help the user setup and become familiar with the CRB and other related hardware and software. This appendix steps the user through an example program using Nucleus Edge* and the Macraigor* Raven* JTAG. It includes hardware setup, software setup, building a project, debugging the project, and a tour of some of the debug features.

F.2.1 Necessary Hardware and Software

This tutorial requires a PC, Nucleus Edge*, a Macraigor* Raven*, a CRB and accompanying cables, and a PCI Express host system or a PCI Express back plane. (If a back plane is used, the CRB must be placed in root complex mode. Refer to Section 3.3.1 for more information).

F.2.2 Related Web Sites

- Macraigor Systems: http://www.macraigor.com/
- Intel Developer's page for I/O processors: http://developer.intel.com/design/iio/
- Intel Developer's page for I/O processors documentation: http://developer.intel.com/design/iio/docs/iop348.htm

F.3 Setup

F.3.1 Hardware Setup

Perform the following steps to set up the hardware:

1. Set the dip switches:
2. Turn Switches 1, 2, 4, 9, and 10 off
3. Turn Switch 3 on
4. Turn Switches 5 through 8 on.

Note: When Switch 3 is on, core 0 is held in reset. Core 0 will run the storage firmware that can be loaded on the CRB, so it will not be discussed in this tutorial. To keep core 0 from competing for the bus and other devices in the address space, holding it in reset is a good option.

5. If there is a jumper on J3C2, remove it.
6. With power turned off, plug the CRB into a PCI Express slot on the back plane or host system.
7. Connect the parallel cable to the Raven* and to the parallel port on the PC.
8. Connect the 20-pin connector at the end of the Raven's ribbon cable to the 20-pin header on the 20-pin to 10-pin adaptor card which came with the CRB. The connector is keyed to prevent plugging the connector in backward.
9. Connect the small ribbon cable to the 10-pin header. The connector is keyed to prevent plugging the connector in backward.
10. Connect the other end of the small ribbon cable to the small, keyed header on the CRB near the processor cooling fan. The connector is keyed to prevent plugging the connector in backward.
11. Connect one end of the serial cable to the CRB using either RJ11 connector.
12. Connect the other end to the RJ11 to DB9F adaptor and connect the adaptor to the serial port on the PC. (If the PC has USB and does not have a serial port, then use a USB to serial adaptor).
13. Apply power to the PC, the Macraigor* Raven*, and the back plane or host system.

F.3.2 Software Setup

Perform the following steps to set up the software:

1. Close all applications and execute the installation file for Nucleus Edge*. Be sure to select "Intel XScale® Tools 3.1" for installation. When prompted to restart the computer during the installation, restart it at that time.

2. Using BIOS setup or Control Panel in Windows, set the parallel port to EPP mode. (0378-037F resource allocation is recommended, but other ranges will also work.)

Note: Tera Term*, HyperTerminal*, or a similar console program will be needed for the serial connection.

3. Download the code files from: http://developer.intel.com/design/iio/swsup/IQ81348SC_for EDGE.htm and place them in a new project directory. The name of the directory does not matter. After unzipping, the directory should contain the following files:

- Blink.c
- Blink.h
- LED.c
- LED.h
- Makefile
- Temp_path.bat
- Readme.txt

Note: The help menu for Nucleus Edge* is quite extensive. During this tutorial, use the Nucleus Edge* Help and Nucleus Debugger Help in parallel. The search function is also quite good. For example, search for "glossary" or "Eclipse notes" to get Eclipse specific terminology.

4. Download the RedBoot.elf file for the CRB from: http://developer.intel.com/design/iio/swsup/RedBoot.htm. Place RedBoot in a convenient directory.

Note: Be sure to get the ROM version of RedBoot for the CRB.

F.4 New Project Setup

F.4.1 Creating a New Project

Peform the following steps:

1. Launch *Nucleus Edge*.
2. On the Tool Bar, click on *Nucleus Edge Projects.*
3. Select *File* > *New* > *Project* > *Basic Project*.
4. Name the project IQ81348SC. Leave the check in "Use default".
5. Under Create from Template, select *Intel XScale® Tools 3.1* for the tools set and **`application`** for project type.
6. Click on *finish*.
7. In the Navigator view, right click on IQ81348SC and select *Import* > *Sources Files* > *Next*> *Browse,* and find the directory where the source files are located. Add the two **`.c files`** and click *finish*.

F.4.2 Configuration

Perform the following steps:

1. With project IQ81348SC highlighted in the Navigator view, select *Window* > *Preferences.*
2. Click on the plus sign beside *Nucleus Edge, Builder,* and *Debug* to expand the tree.
3. Click on *Nucleus Edge* and the licensing view is displayed.
4. Click on *Builder* and then click on the plus sign by *Intel XScale® Tools 3.1* to expand the tree.
5. Click on *Intel XScale® C Compiler* and look under Tool Configuration. Notice that the compile only and output options are taken care of, so they will not need to be specified later under options.
6. Click on *Intel XScale® Assembler* and change xscale-elf-as{exe} to **`xscale-elf-gcc{exe}`**.
7. Click on *Intel XScale® Linker* and change xscale-elf-ld{exe} to **`xscale-elf-gcc{exe}`**.
8. Click on *Intel XScale® Tools 3.1* and add **`rem`** in front of **`set GCC_EXEC_PREFIX….`**
9. Click *OK*.
10. Select *`Project > Properties > Build Settings`*.
11. Highlight **`IQ81348SC`** under Folders.
12. Under "Tools", select *Intel XScale® Assembler.*
13. Click on the plus sign by *Target Specific*. For "Processor", choose **`xscale`**.
14. "–mcpu=xscale" will appear in the command line options field.
15. Add **`specs=redboot.specs`** to the command line options field so that it now reads "–specs=redboot.specs –mcpu=xscale".
16. Under "Tools", select *Intel XScale® Linker* and add **`specs=redboot.specs`** to the command line field.
17. Under "Tools", select *Intel XScale® C*.

18. Click on the plus sign by *Debug* to expand the tree and select *Default Level* for Generate Debug Info-Dwarf-2.
19. Click on the plus sign by *Directory* and for *Override Default Specs* add **`redboot.specs`** to the value field.
20. Click on the plus sign by *Output* and select **`yes`** for "Print Compilation Commands (verbose)".
21. Click on the plus sign by *Target Specific* and choose **`xscale`** for "Specify Target Processor".
22. Click the plus sign by *Warnings* and choose *Enable All Warnings* for "All Warnings".
23. Click *Apply* and *OK*.

F.5 RedBoot

F.5.1 Overview

Since this tutorial is building and debugging a simple piece of code in RAM, RedBoot will be used for target initialization. RedBoot is a debug monitor which initializes the board, has some debug and diagnostic functions, and is capable of serial communication with a console program such as Tera Term* or HyperTerminal*.

F.5.2 Checking and Configuring RedBoot

Perform the following steps to check and configure Redboot:

1. Check RedBoot by powering on the Host system with the CRB plugged into it.
2. Open the console program. (Tera Term* will be used for the purposes of this tutorial.)
3. Select ***serial port*** and ***COM1***. (The COM port can vary, of course, so you may need to check with the Device Manager to determine the appropriate serial port.)
4. Check to see if the following settings display:

- Baud rate: 115200.
- Data: 8 bit.
- Parity: none.
- Stop: 1 bit.
- Flow control: none.

5. Press Enter several times and the RedBoot prompt should redisplay. If this is the case, then RedBoot is healthy and there is no need to reflash.

Note: To reach the RedBoot prompt faster after reset, type **fconfig** and press Enter after the RedBoot prompt.

6. Set **`BOOTP to false`** and save.
   The results should be as follows:

RedBoot> **`fconfig`**

Run script at boot: false

Use BOOTP for network configuration: false

Gateway IP address:

Local IP address:

Local IP address mask:

Default server IP address:

Console baud rate: 115200

DNS server IP address:

GDB connection port: 9000

Force console for special debug messages: false

Network debug at boot time: false

Update RedBoot non-volatile configuration - continue (y/n)? **`y`**

... Unlock from 0xf07c0000-0xf07c1000: .... Erase from 0xf07c0000-0xf07c1000: .

... Program from 0x0ffd2000-0x0ffd3000 at 0xf07c0000: .

... Lock from 0xf07c0000-0xf07c1000: .

RedBoot>

F.6 Flash Programming

F.6.1 Overview

Nucleus Edge* and the Raven* are capable of reading from, writing to, and erasing the contents of the Flash on the evaluation board. The flash programmer only supports certain file formats: Intel Hex, Motorola srec and standard elf (executable and linking format). For this tutorial, we will reflash using a RedBoot.elf file.

F.6.2 Configuring and Using the Flash Programmer

1. Turn dip switch 3 on to hold core 0 in reset.
2. Power on the host system.
3. Click on the down arrow next to the green bug icon and select Debug, or from the main menu, select *Run > Debug*.
4. In the lower left, click new.
5. Select *XScale > Intel > Intel XScale Core 3(SINGLECORE 2-TAP)*.

6. Click *next*.
7. Select *Raven*.
8. Click *finish*.
9. Under the "Connection" tab, set the following:

- OCD connection speed = **1**.
- OCD debug speed =**1**.
- Connection mode – core 1 = **halt**.
- JTAG bypass instruction register length –core 2 = **7**.
- JTAG bypass instruction – core 2 = **0xff.**
- JTAG bypass length – core 2 = **1**.
- XScale – Hot Debug – core 1 = **false**.

10. Under the "Core" tab, select *core = XSCALE_CORE3[ARM/Intel XScale]_1* and check *Do not download image to target.*
11. Under the "Core" tab, select *core = Unknown Device[ARM/Unknown Device]_2* and check *Do not download image to target*.
12. Click *Apply* and *Close*.
13. Click on the *"green" bug icon*. The Debug view should show "OCD Thread (Suspended)". Since Hot Debug was not selected, a target reset should have occurred. The LEDs should have returned to 0.8. on connection, and the program counter in the disassembly window should be at 0x00000000.
14. Select *Run > Flash*, and the OCDemon Flash Memory Programmer comes up.
15. Select *File > Open* and find *..\edge\bin\transports\Ocd\Intel_Xscale_81348.OCD*.
16. Select *Configuration > Communications*.
17. Comm Method = **Raven Parallel**.
18. Comm Port = **LPT1**.
19. Debug Port Clock Rate = **1:8.0MHz**.
20. Click *OK*.
21. Click *Program*.
22. Check *Erase Target Flash Sectors before Programming*.
23. Browse for the RedBoot.elf file.
24. Click *Program*.
25. Use the instructions in Section F.5.2 to verify the flash has been successfully rewritten.

F.7 Building an Executable File

1. In the Navigator view, right-click on project IQ81348SC and select *Build Project*. The Build Console view will then fill with verbose output for the build.
2. In the Navigator view, click on the plus signs to expand the tree.

- Objects should contain blink.o and led.o.
- Output should contain IQ81348.elf.

3. After this initial build, clean the project before each build. Right-click on the project IQ81348SC and select *Clean Project*. Then, select *Build Project*.

F.8 Launching and Configuring the Debugger

1. Power on the host system and the LEDs on the CRB should display **0.8. > SL > A1**. The board ships with RedBoot loading into flash, so this is the expected pattern. If the LEDs do not reach A1, then the CRB needs to be reflashed with RedBoot.
2. Use Tera Term* to get a RedBoot prompt. Press Enter several times to establish an active prompt.

Note: A quirk with RedBoot is that if the connection is not established by pressing Enter at the prompt, the print statements in our code will not print to the screen, so be sure to get an active prompt by pressing Enter.

3. Click on the down arrow next to the green bug icon and select Debug, or from the main menu, select *Run* > *Debug*.
4. In the lower left, click *new*.
5. Select *XScale* > *Intel* > *Intel XScale Core 3(SINGLECORE 2-TAP)*.
6. Click *next*.
7. Select *Raven*.
8. Click *finish*.
9. Under the "Connection" tab, set the following:

- OCD connection speed = `1`.
- OCD debug speed = `1`.
- Connection mode – core 1 = **`halt`**.
- JTAG bypass instruction register length –core 2 = `7`.
- JTAG bypass instruction – core 2 = **`0xff`**.
- JTAG bypass length – core 2 = `1`.
- XScale – Hot Debug – core 1 = **`true`**.

A normal connection to the processor will start the session with a reset. This allows the user to debug the code from the beginning, but the BIOS configuration of the bus will be lost in the card being debugged. A Hot-Debug connection connects without resetting the card under debug and continues from the point of entry in the code. The code can only be debugged from that point forward; however, the bus enumeration is intact. This tutorial will use a Hot-Debug connection for this reason.

10. Under the "Core" tab, select *core = XSCALE_CORE3[ARM/Intel XScale]_1* and check *Do not download image to target*.
11. Under the "Core" tab, select *core = Unknown Device[ARM/Unknown Device]_2* and check *Do not download image to target*.
12. Click *Apply* and *Close*.
13. Click on the *"green" bug icon*. The Debug view should show "OCD Thread (Suspended)". Since Hot Debug was selected, the LEDs should have remained at A1 which means that the target did not reset on connection. If a target reset occurred, then the LEDs would return to 0.8. on connection.
14. Select *Run* > *Resume* to run.
15. Select *Run* > *Suspend* to halt. (*Run* > *Terminate* disconnects the JTAG connection. Using Terminate, the user can connect and disconnect indefinitely without closing Nucleus Edge*.)

16. In the Debug view, hold the mouse arrow over the icon that looks like a processor with a blue arrow pointing down. Text will appear that says "Load a new image".
17. Click on the icon.
18. In the "Load Target Image" view, select *Load Project Output* and select IQ81348SC in select project field. Check *set program counter*.
19. You can perform one of the following:

- Click *OK* to load all sections.
- Click *Cancel* for source not found redboot-crt0.s.
- Click *Resume* to run the code and the LEDs will scroll.
- Click *Suspend* to halt and the source code will come up.
- Double-click on any source file in the Navigator view to display it and set breakpoints.

F.9 Touring Some of the Debugger Features

1. Launch Nucleus Edge* and make sure that project IQ81348SC is open.
2. Hold the cursor arrow over the button that looks like a "green bug". The debugger connection from Section F.8 should have been retained and the text should read "Intel XScale® processor 3(SINGLECORE 2-TAP)". If so, click on the *green bug button*. If not, go through the steps in Section F.8, "Launching and Configuring the Debugger".
3. The Debug view will display the results of the JTAG connection. Since we are initializing with RedBoot and are loading our program later, the connection tree should read "Ocd Thread (Suspended)", "<no Source>" and the address of the program counter on connection. A reset connection would result in (0x00000000). A Hot-Debug connection would result in an address somewhere at (0xF000XXXX). The exact address depends on where the running RedBoot code was when the debugger broke in. In this case, we should have had a Hot-Debug connection.
4. Press *resume* and let RedBoot run.
5. Open your serial console and look for the RedBoot prompt. Press Enter a few times to see that the prompt is responding.
6. Click *suspend* on the debugger.
7. Click the *Load a new image* button on the Debug view. It looks like a processor with a blue down-arrow.
8. Load project output, IQ81348SC. Check the *Set program counter*.
9. Perform one of the following:

- Click *OK* to load all sections.
- Click *Cancel* for source not found redboot-crt0.s.
- Double-click on *blink.c* in the Navigator view.
- Right-click to the left of "int main(void)" and select *Add Breakpoint*.

10. Press *resume* and the code will run up to and stop at "main".
11. Press *resume* again and watch the LEDs scroll and check the console for "printf" output.
12. Click *suspend*.

Note: Notice that there are 5 groups of views. In each group, you can move around by clicking on the tabs, and by clicking on the full screen button in the top-right corner of the view, the view will fill the whole screen. Clicking in the top-right corner will restore the view to its group.

13. Click on the *Breakpoints* tab. The breakpoint at main is displayed.
14. Click on the *Register* tab and maximize the view to full screen. Click on the plus signs to expand and view the registers for the various modes and the coprocessor registers. Then, restore the view to its group.
15. Add a **`breakpoint`** to line 15 in blink.c and click *resume*.
16. Click on the *Variables* tab, click on the plus signs to expand the tree, click *step into* several times, and watch the changes.
17. Click on the *Watch* tab. In led.c, double-click to highlight and copy *display_left* and *display_right* and paste them into the *names field of the watch view*.
18. Click *resume* to stop at line 15 in blink.c.
19. Click *step into, step return, step into, step return, step into, step return*. Observe the changes in the watch window. Adjust the field sizes as appropriate to read the values.
20. Click the *Trace On* button.
21. Press *resume* and then *suspend*.
22. Click the *Show Trace Results* button. Turn Trace *off*.
23. In the group of views at the bottom, click on the *Memory view* tab.
24. In the address field, type **`0xF0000000`** and press Enter. This address is the beginning of flash after RedBoot has completed initialization. Notice that there are icons for this view on the right.
25. Click on the *Fill Memory* button. At this point, RAM is at 0x0, so fill *0x50000-0x50100* with **`a5a5a5a5`**.
26. Click on the *Breakpoints* tab (upper right) and right-click the break point on line-15 of blink.c. Change the properties so that the breakpoint doesn't occur until the 7th occurrence. Then, press resume and count the LED patterns to see if the pattern repeats itself 7 times before stopping.

F.10 Compiling with a makefile

Alternatively, the user may prefer using GNU make and a GNU GCC compiler.

1. Place the project files (.c, .h, and makefile) in a project directory.
2. Install an Intel XScale-elf-gcc compiler by performing the following steps (or the compiler we just used in Nucleus Edge* will work.):
   a. Add a path environment variable to point to xscale-elf-gcc.exe.
   b. Open a Command Window in the project directory.
   c. Type **`make clean`** and press Enter if this is not the first build and then **`make`** and Enter. Otherwise, if this is the first build, type **`make`** and press Enter.
   d. Type **`dwarf2bd hx IQ81348.elf`** and press Enter. This will produce a .bd and a .hx file for the XDB debugger.
   e. Launch XDB and connect with *IOP348_mz1_tlu_RV.xsf*.
   f. Check *try Hot-Debug connection* to connect without a reset.

g. Run a script that defines the ProjLoad and setStart buttons used in the start.xdb script to load the code into RAM, change the program counter to the start of the code, and run to main and stop.

h. The user will be prompted to point the debugger to the source code.

F.11 Conclusion

At this point, the user should have a good picture of the evaluation board and the development tools. The tools have many features that have not been covered in this tutorial. The tutorial's purpose was to get the board and tools up and running and to give the user a feel for the evaluation board and the development tools. Now that a simple project is running, the help menu can provide much information on the features and capabilities of the tools as application development proceeds.

§ §