IQ80960RP Evaluation Platform User's Guide

June 1996

Order Number: 272913-001

Copies of documents which have an ordering number and are referenced in this document, or other Intel literature, may be obtained from:

Intel Corporation
P.O. Box 7641
Mt. Prospect IL 60056-764
or call 1-800-548-4725

FIGURES

TABLES

1

INTRODUCTION

CHAPTER 1
INTRODUCTION

1

This user's guide describes the IQ Software Developer's Kit (IQ-SDK) for Intel's i960® RP processor. The i960 RP processor combines an i960 Jx processor core with two PCI bus interfaces, as well as memory control, DMA channels, an interrupt controller interface, and an I^2C Serial Bus. The IQ-SDK platform is a full-length PCI adapter board that can be installed in any PCI host system that complies with the *PCI Local Bus Specification* Revision 2.1. PCI devices can be connected to the secondary bus on the IQ-SDK platform to build powerful intelligent I/O subsystems.

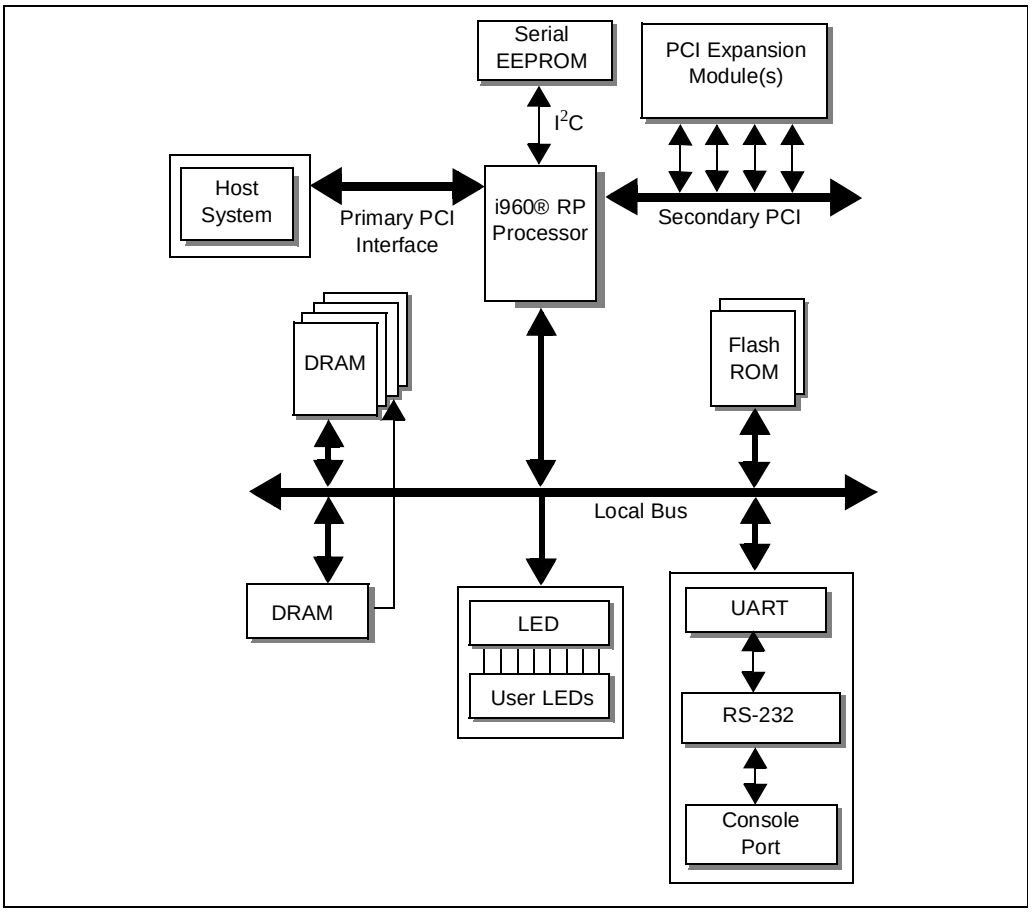

Figure 1-1. IQ-SDK Platform Functional Block Diagram

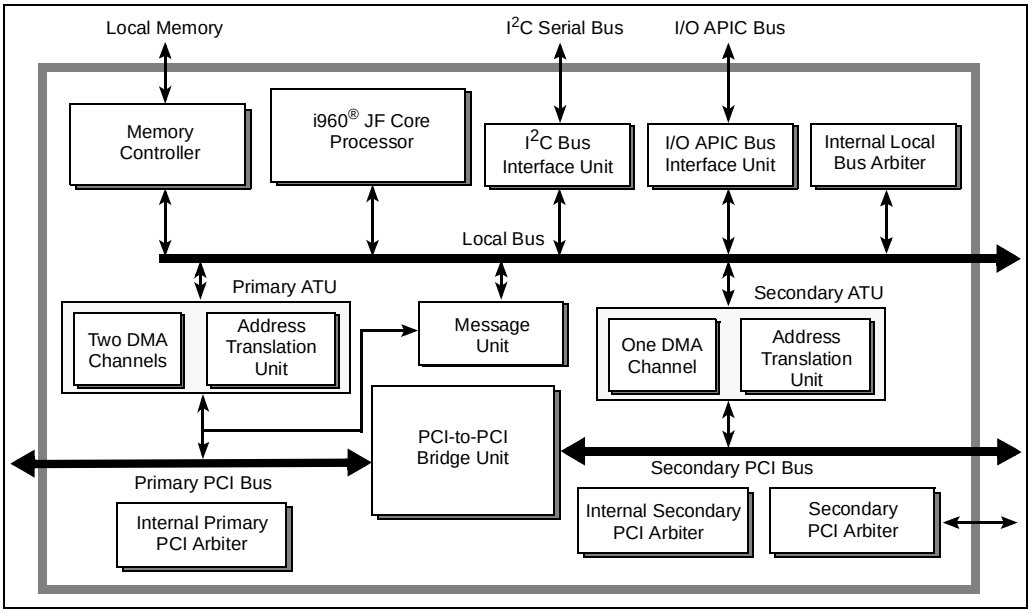

Figure 1-2. i960® RP Processor Block Diagram

1.1 i960 RP PROCESSOR ADVANTAGES AND FEATURES

The i960 RP processor serves as the main component of a high performance, PCI-based intelligent I/O subsystem. The IQ-SDK platform allows the developer to connect PCI devices to the i960 RP processor using removable modules, the specifications for which are included in this manual. The features of the IQ-SDK platform are enumerated below and shown in Figures 1-1 and 1-2.

- i960 RP processor
- PCI long-card form factor
- Primary PCI bus interface
- Secondary PCI bus connected to the primary PCI interface with a PCI-to-PCI bridge
- DMA channels on both PCI buses
- APIC Bus interface
- I^2C Serial Bus
- SIMM sockets supporting 2 to 64 Mbytes DRAM
- Serial console port based on 16C550 UART
- Eight user-programmable LEDs
- Flash ROM sockets
- 512 Kbyte serial EEPROM (24C04) connected to I^2C Bus port
- APIC Bus header
- Emulator header
- JTAG header

1.2 ABOUT THIS MANUAL

A brief description of the contents of this manual follows.

Chapter 1 INTRODUCTION	Introduces the IQ-SDK and its features. Also defines notational conventions and related documentation.
Chapter 2 GETTING STARTED	Provides step-by-step instructions for installing the IQ-SDK platform in a host system and downloading and executing an application program. This chapter also describes Intel's software development tools, the MON960 Debug Monitor, software installation, and hardware configuration.
Chapter 3 HARDWARE REFERENCE	Describes the locations of connectors, switches and LEDs on the IQ-SDK platform. Header pinouts and register descriptions are also provided in this chapter.
Chapter 4 i960® RP PROCESSOR OVERVIEW	Presents an overview of the capabilities of the i960 RP processor and includes the CPU memory map.
Chapter 5 MON960 ON THE IQ-SDK PLATFORM	Describes a number of features added to MON960 to support application development on the i960 RP processor.
Chapter 6 IQ MODULE INTERFACE	Describes the physical and electrical characteristics of the IQ Module interface, which allows PCI devices on removable modules to be connected to the secondary PCI bus on the IQ-SDK platform.
Appendix A, PARTS LIST	Lists each IQ-SDK component and quantity, the component's reference name as it appears on the PC board, a description of size or rating, and the manufacturer's part number. To order replacement parts contact the manufacturer listed in Table A-1.

1.3 NOTATIONAL CONVENTIONS

The following notation conventions are consistent with other 80960RP documentation and general industry standards.

# and $\overline{\text{overbar}}$	In code examples the pound symbol (*#*) is appended to a signal name to indicate that the signal is active. Normally inverted clock signals are indicated with an overbar above the signal name (e.g., $\overline{\text{RAS}}$).
Bold sans serif	Indicates user entry and/or commands.
Bold serif	In text, PLD signal names are in bold lowercase letters (e.g., **h_off**, **h_on**). In code examples, `typewriter font` is used.
Italics	Indicates a reference to related documents; also used to show emphasis.
`Typewriter font`	Indicates code examples and file directories and names.
Asterisks (*)	On non-Intel company and product names, a trailing asterisk indicates the item is a trademark or registered trademark. Such brands and names are the property of their respective owners.
UPPERCASE	In text, signal names are shown in uppercase. When several signals share a common name, each signal is represented by the signal name followed by a number; the group is represented by the signal name followed by a variable *(n)*. In code examples, signal names are shown in the case required by the software development tool in use.
Designations for hexadecimal and binary numbers	In text—instead of using subscripted "base" designators (e.g., FF_{16}) or leading "0x" (e.g., 0xFF)—hexadecimal numbers are represented by a string of hex digits followed by the letter *H*. A zero prefix is added to numbers that begin with *A* through *F*. (e.g., *FF* is shown as *0FFH*.) In examples of actual code, "0x" is used. Decimal and binary numbers are represented by their customary notations. (e.g., 255 is a decimal number and 1111 1111 is a binary number. In some cases, the letter *B* is added to binary numbers for clarity.)

1.4 TECHNICAL SUPPORT, SCHEMATICS, AND PLD EQUATIONS

For technical assistance with the IQ-SDK, contact the Intel Technical Support Hotline. For information about technical support in other geographical areas, contact Intel's North America Technical Support Hotline.

You can also use your PC with a modem to download IQ-SDK schematics from Intel's Bulletin Board Service (BBS).

Up-to-date product and technical information is available electronically from these sources.

Intel's World-Wide Web (WWW) Location:	http://www.intel.com
IQ-SDK Product Information	http://www.intel.com
FaxBACK Service:	
US and Canada	800-628-2283
Europe	+44(0)793-496646
worldwide	916-356-3105
Application Bulletin Board Service:	
up to 14.4-Kbaud line, worldwide	916-356-3600
dedicated 2400-baud line, worldwide	916-356-7209
Europe	+44(0)793-432955

1.4.1 Intel Customer Support Contacts

Customer Support (US and Canada)		800-628-8686
Country	**Literature**	**Technical Support**
Australia National Sydney	Contact local distributor	008-257-307 61-2-975-3300 61-3-810-2141
Belgium, Netherlands, Luxembourg	010-4071-111	010-4071-111
Canada	800-468-8118	Contact local distributor
Finland	358-0-544-644	358-0-544-644
France	33-1-30-57-70-00	33-1-30-57-72-22
Germany	49-89-90992-257	Hardware: 49-89-903-8529 Software: 49-89-903-2025
Israel	972-3-498080	972-3-548-3232
Italy	39-02-89200950	39-02-89200950
Japan	Contact local distributor	0120-1-80387
Sweden	46-8-7340100	46-8-7340100
United States	800-548-4725	800-628-6249

1.4.2 Additional Information

To order manuals from Intel, contact your local sales representative or Intel Literature Sales (1-800-879-4683).

Product	Document Name	Company/ Order #
All	*Intel Solutions960®* catalog	Intel # 270791
80960RP	*i960® RP Microprocessor User's Manual*	Intel # 272736-001
	80960RP Intelligent I/O Microprocessor Data Sheet	Intel #272737
	MON960 Debug Monitor User's Guide	Intel #484290
	I/O APIC Emulation Software for the i960® RP Processor	Intel #272905-001
	Connector Specification for Using In-Circuit Emulators and Logic Analyzers with the i960® RP Processor	Intel #272812-001
	PCI Local Bus Specification Revision 2.1	PCI Special Interest Group 1-800-433-5177
	24C08 Serial EEPROM Data Sheet	Xicor, Inc.

Contact Cyclone Microsystems for additional information about their products:

Cyclone Microsystems **25 Science Park** **New Haven CT 06511**	Phone: 203-786-5536
	FAX: 203-786-5025
	e-mail: info@cyclone.com WWW: http://www.cyclone.com

2

GETTING STARTED

CHAPTER 2
GETTING STARTED

2

This chapter contains instructions for installing the IQ-SDK platform in a host system, and explains how to download and execute an application program using the MON960 Debug Monitor.

2.1 PRE-INSTALLATION CONSIDERATIONS

This section provides a general overview of the components required to develop and execute a program on the IQ-SDK platform. The *MON960 Debug Monitor User's Guide* (order number 484290) fully describes several of these components, including MON960 commands, the Host Debugger Interface Library (HDIL), and the MONDB.EXE utility.

2.1.1 Software Development Tools

A number of software development tools are available for the i960 processor family[1]. The installation instructions presented in this chapter were verified using GNU/960 and CTOOLS960—Intel's i960® processor software development tools. Advanced C-language compilers for the i960 processor family are available for DOS-based systems and a variety of UNIX workstation hosts. These products provide execution profiling and instruction scheduling optimizations and include an assembler, a linker, and utilities designed for embedded processor software development. If you are using other software development tools, read through this example to gain a general understanding of how to use your tools with this board.

2.1.2 MON960 Debug Monitor

The IQ-SDK platform is equipped with Intel's MON960, an on-board software monitor that allows you to execute and debug programs written for i960 processors. The monitor provides program download, breakpoint, single step, memory display, and other useful functions for running and debugging a program.

The IQ-SDK platform works with source-level debuggers, such as DB960 and GDB960. The source-level debugger must support the Host Debugger Interface Library (HDIL) defined by MON960.

1. Refer to Intel's *Solutions960*® catalog for a complete list of i960 processor software development and debug tools.

2.1.3 Host Communications

MON960 allows you to communicate and download programs developed for the IQ-SDK platform across a host system's serial port or PCI interface. The IQ-SDK platform supports two methods of serial download: terminal emulation and Host Debugger Interface Library (HDIL).

2.1.3.1 Terminal Emulation Method

Terminal emulation software on your host system can communicate to MON960 on the IQ-SDK platform via an RS-232 serial port. The IQ-SDK platform supports port speeds from 300 to 115,200 bps. Serial downloads to MON960 require that the terminal emulation software support the XMODEM protocol.

Configure the serial port on the host system for 300-115,200 baud, 8 bits, one stop bit, no parity.

2.1.3.2 Host Debugger Interface Library (HDIL) Method

The MONDB utility provided with MON960 allows application code to be downloaded, executed, and debugged on the IQ-SDK platform. This utility differs from standard terminal emulation programs in that it allows you to download executable images through a serial port or via the PCI bus (see Section 2.1.3.3). When used for serial download, MONDB can operate at any of the permissible port speeds (300-115,200 bps).

MON960 can detect an HDIL connection requested only after the IQ-SDK platform has been powered up or reset. Once the user initiates a terminal connection, HDIL requests are ignored.

2.1.3.3 PCI Download Support

Application code can be downloaded to the IQ-SDK platform via the host system's PCI bus. MONDB must be used for PCI downloads, unless a debugger is available that supports PCI downloads with the HDIL interface. The command line syntax for MONDB is documented in the *MON960 User's Guide*.

2.1.3.4 Source Level Debugger

You may use a source-level debugger, such as Intel's DB960 or GDB960, to establish serial communications with the IQ-SDK platform. The MON960 Host Debugger Interface Library (HDIL) provides the interface between MON960 and the debugger, so any debugger used with the IQ-SDK platform must support HDIL.

HDIL connection requests cannot be detected by MON960 if the user has already initiated a connection using a terminal emulator. In this case, the IQ-SDK platform must be reset before HDIL requests can be processed.

2.2 SOFTWARE INSTALLATION

2.2.1 Installing Software Development Tools

If you haven't done so already, install your development software as described in its manuals. All references in this manual to CTOOLS960 or GNU/960 assume that the default directories were selected during installation. If this is not the case, substitute the appropriate path for the default path wherever file locations are referenced in this manual.

The example program provided on the MON960 diskette enables you to use your development tools to compile a sample application program. If you are using software tools other than CTOOLS960 or GNU/960, these instructions still are generally applicable; however, you will need to consult your tools documentation for equivalent commands.

2.3 HARDWARE INSTALLATION

Follow these instructions to get your new IQ-SDK platform running. Be sure all items on the checklist were provided with your IQ-SDK.

WARNING:
STATIC CHARGES CAN SEVERELY DAMAGE THE IQ-SDK PLATFORM. BE SURE YOU ARE PROPERLY GROUNDED BEFORE REMOVING THE IQ-SDK PLATFORM FROM THE ANTI-STATIC BAG.

2.3.1 Installing IQ Modules

Installing IQ Modules on the IQ-SDK platform is a simple procedure. The module is turned so that its component side faces the component side of the IQ-SDK platform, and the connectors are mated and gently pushed together. If the module includes connectors that are accessible from the outside of the host system, the slot plate on the IQ-SDK platform must be replaced with a cutout slot plate. In this case, slide the connectors on the module through the cutout in the slot plate before mating the connectors. Nylon standoffs and screws are used to secure the module to the IQ-SDK platform; these and cutout slot plates are available from Intel on request.

2.3.2 Installing the IQ-SDK Platform in the Host System

If you are installing the IQ-SDK platform for the first time, visually inspect the board for any damage that may have occurred during shipment. If there are visible defects, return the board for replacement. Follow the host system manufacturer's instructions for installing a PCI adapter. The IQ-SDK platform is a full-length PCI adapter and requires a PCI slot that is free from obstructions. See the *PCI Local Bus Specification* Revision 2.1 for the dimensions of a full-length PCI adapter.

2.3.3 Verify IQ-SDK Platform is Functional

These instructions assume that you have already installed the IQ-SDK platform in the host system as described in Section 2.3.2.

- To connect the serial port for communicating with and downloading to the IQ-SDK platform, connect the RS-232 cable (provided with the IQ-SDK) from a free serial port on the host system to the phone jack-style connector on the IQ-SDK platform.
- Upon power-up, the red Fail LED should turn off, indicating that the processor has passed its self-test, and the green Run LED should light, indicating that the processor is performing bus cycles.
- Press return on a terminal connected to the IQ-SDK platform to bring up the MON960 prompt. MON960 automatically adjusts its baud rate to match that of the terminal at startup. At baud rates other than 9600, it may be necessary to press return several times.

2.4 CREATING AND DOWNLOADING EXECUTABLE FILES

To download code to the IQ-SDK platform, your compiler must produce a COFF-format object file. The Companion Diskette included with the IQ-SDK contains a sample makefile and a sample linker directive file which can be used to build applications for the IQ-SDK platform with the CTOOLS960 or GNU/960 toolset. These files need to be adapted for other compilers.

During a download, MON960 checks the link address stored in the COFF file, and stores the file at that location on the IQ-SDK platform. If the executable file is linked to an invalid address on the IQ-SDK platform, MON960 will abort the download.

2.4.1 MONDB-to-IQ-SDK Platform Communication Support

MONDB is a command line program that serves as a host interface to download and debug features of MON960, the ROM-based monitor that runs on the IQ-SDK platform. This example demonstrates the use of MONDB to download an application to the IQ-SDK platform. A batch file, DWNLD.BAT, is provided on the Companion Diskette with the command line options to start a PCI download to the IQ-SDK platform. To start a download, enter DWNLD followed by the name of the COFF-format file to download. The command line parameters in DWNLD.BAT can be changed with a text editor. MONDB command line options are documented in the *MON960 User's Manual* under the name EXE960.

2.4.2 PCI Download

PCI download to the IQ-SDK platform requires the use of MONDB. See the *MON960 User's Manual* for a description of MONDB's command line syntax. If your application produces output to the console port on the IQ-SDK platform, you need to connect a terminal.

2.4.3 Terminal Emulation-to-IQ-SDK Platform Communication Support

To use a terminal emulator to communicate with the IQ-SDK platform:

- Invoke the terminal emulation program.
- To establish communication between the terminal emulation program and MON960, reset the IQ-SDK platform using the host's reset switch and press <ENTER>. The MON960 banner appears, followed by the command prompt.
- At the command prompt, enter 'do' to download:
  ```
  => do
  ```
- Start your terminal emulation program's XMODEM transfer mode and send the COFF-format object file. The following message appears when transfer is complete:
  ```
  -- Download complete --
     Start address is: XXXXXXXX
  ```
- To execute your program, enter 'go':
  ```
  => go
  ```

An error during download usually indicates that the application is improperly linked. Checking a linker-generated map file against the IQ-SDK platform address map (Figure 4-1) usually reveals this type of error. Code and data segments should be located in the range assigned to Flash ROM or RAM.

More information on the MON960 commands mentioned in this section can be found in the *MON960 Debug Monitor User's Guide*.

3

HARDWARE REFERENCE

CHAPTER 3
HARDWARE REFERENCE

The location and function of physical connectors, switches, and LEDs is described in this section. Refer to Figure 3-1, the physical diagram of the IQ-SDK platform, for the locations of components discussed in this section.

3.1 CONNECTORS, SWITCHES, AND LEDS

Figure 3-1 shows the physical locations of the major components on the IQ-SDK platform. The functions of these components are listed in Table 3-1. For a complete list of components on the IQ-SDK platform, refer to Appendix A, Parts List.

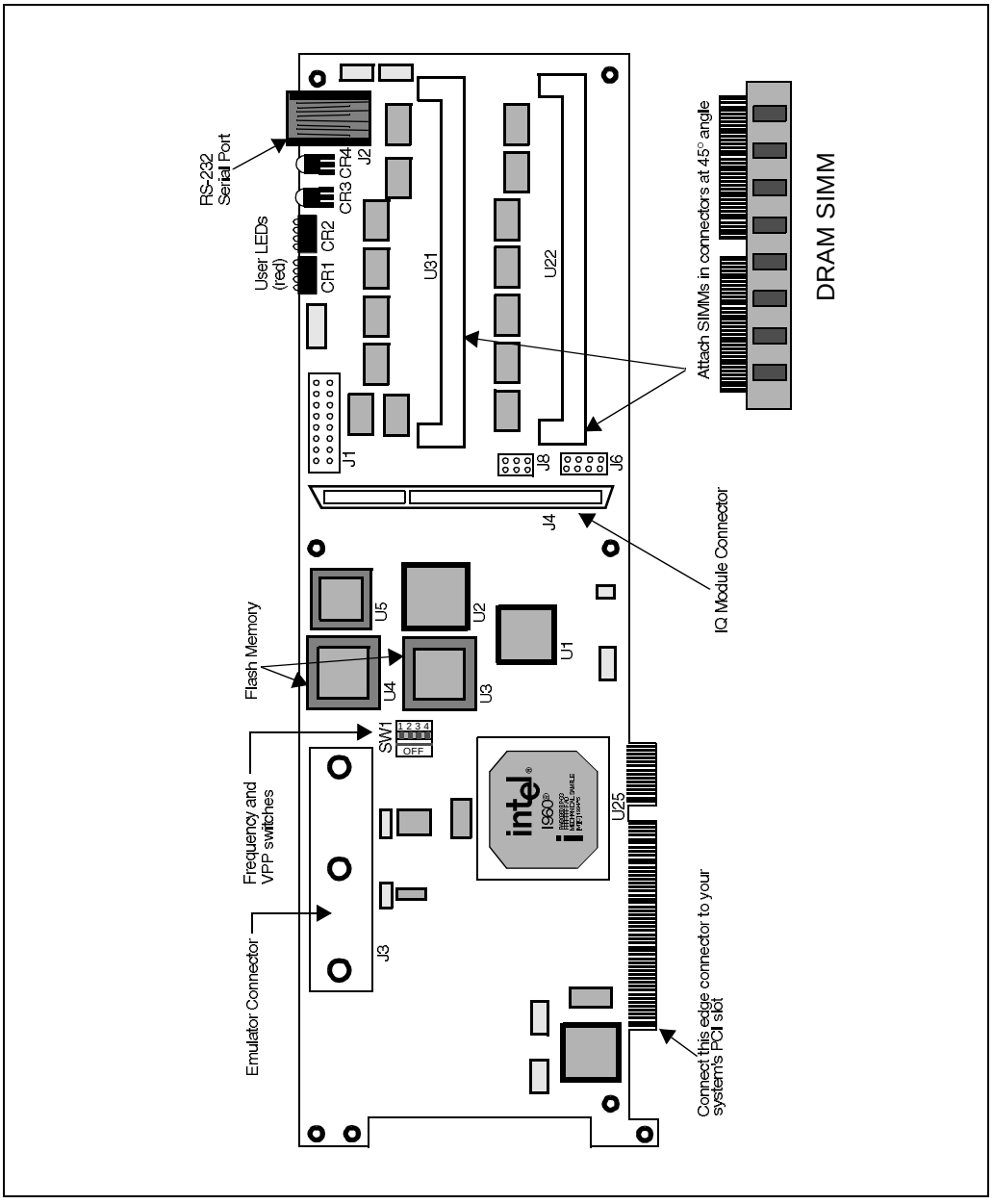

Figure 3-1. IQ-SDK Platform Physical Diagram

Table 3-1. IQ-SDK Platform Connectors and LEDs

Reference	Description
J1	JTAG connector
J2	Serial port connector
J3	Emulator connector
J4	IQ Module connector (secondary PCI bus)
J6	I^2C connector
J8	APIC connector
CR1, CR2	Eight user LEDs (small red)
CR3	Fail LED (red)
CR4	Run LED (green)
SW1-1	Enables programming voltage, V_{PP}, to Flash ROMs
SW1-2	ROMSWAP: Determines which Flash ROM is the boot ROM. ROMSWAP = ON: boot from U3 ROMSWAP = OFF: boot from U4
SW1-3	ROM DISABLE: Disables booting from either Flash ROM device. Allows processor to boot from devices connected to the emulator header.
SW1-4	Not used.

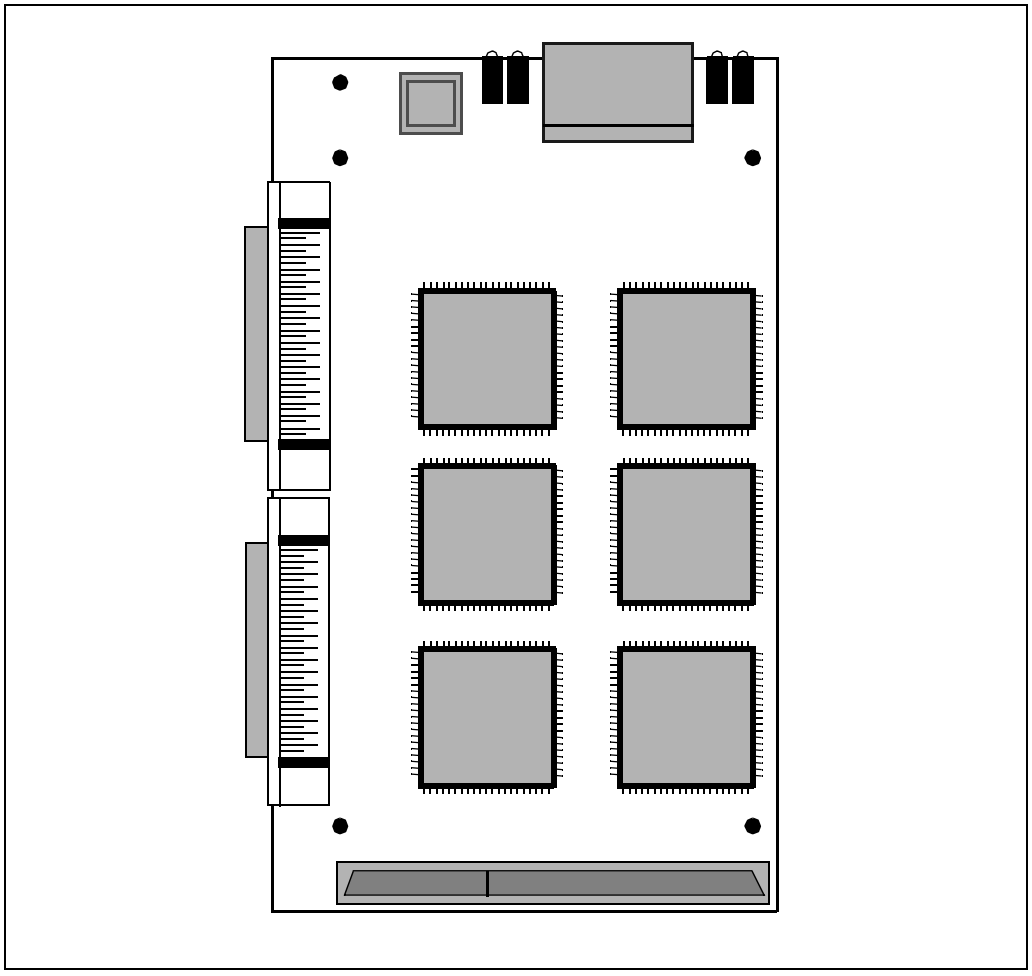

Figure 3-2. IQ Module Physical Diagram

3.2 POWER REQUIREMENTS

The IQ-SDK platform draws power from the PCI bus. The power requirements of the IQ-SDK platform are shown in Table 3-2. The numbers do not include the power required by an IQ Module mounted on the IQ-SDK platform.

Table 3-2. IQ-SDK Platform Power Requirements

Voltage	Typical Current	Maximum Current
+3.3 V	not used	not used
+5 V	3.0 A	3.9 A
+12 V	1 mA	1 mA
-12 V	1 mA	1 mA

NOTE: Does not include the power required by an IQ Module mounted on the IQ-SDK platform.

3.3 INTERLEAVED DRAM

The IQ-SDK platform can be populated with 2 to 64 Mbytes of interleaved SIMM DRAM. The DRAM is accessible from either of the PCI buses on the IQ-SDK platform, and can be used to implement shared communications areas for IQ Modules installed on the IQ-SDK platform.

3.3.1 DRAM Performance

The IQ-SDK platform uses interleaved fast page mode DRAM to achieve a zero wait state burst at 33 MHz. The memory runs with two wait states in the first cycle of a burst and no wait states during the burst. No additional recovery cycles are required beyond the intrinsic i960 Jx core recovery cycle. Table 3-3 shows the performance numbers for the IQ-SDK platform.

Table 3-3. DRAM Performance

Cycle Type	Table Clocks	Wait States	Performance Bandwidth
Read Single	4-1	2	27 Mbytes/sec
Read Burst	4-1-1-1-1	2-0-0-0	66 Mbytes/sec
Write Single	4-1	2	27 Mbytes/sec
Write Burst	4-1-1-1-1	2-0-0-0	66 Mbytes/sec

NOTE: Bandwidth is sustained bandwidth—not peak.

3.3.2 Upgrading DRAM

On-board DRAM is located in two SIMM sockets as shown in Figure 3-1. The IQ-SDK platform may be equipped with 2 to 64 Mbytes of DRAM. The various memory combinations are shown in Table 3-4. Only 72 pin Fast Page Mode DRAM modules rated at 60 or 70 ns should be used on the IQ-SDK platform. Either x32 or x36 devices may be used, since DRAM parity is not enabled on the IQ-SDK platform. Both SIMM sockets must be populated for the IQ-SDK platform to function.

Table 3-4. DRAM Configurations

Total Memory	SIMM Module Type
2 Mbytes	256K x 32 or 256K x 36 (1 Mbyte)
4 Mbytes	512K x 32 or 512K x 36 (2 Mbytes)
8 Mbytes	1M x 32 or 1M x 36 (4 Mbytes)
16 Mbytes	2M x 32 or 2M x 36 (8 Mbytes)
32 Mbytes	4M x 32 or 4M x 36 (16 Mbytes)
64 Mbytes	8M x 32 or 8M x 36 (32 Mbytes)

3.4 ROM AND FLASH ROM

Two standard 32-pin PLCC ROM sockets are included on the IQ-SDK platform. The primary ROM socket, U4, is populated with a 27C020 PROM or 28F020 Flash ROM containing MON960. The secondary socket, U3, contains a 28F020 Flash ROM and may be used to store user applications. MON960 includes features to erase and program Flash ROM and download code directly into Flash ROM.

3.4.1 ROMSWAP and ROM-DISABLE Switches

The ROMSWAP switch allows the user to determine the socket from which the processor boots. The ROM-DISABLE switch disables both Flash ROMs, allowing the processor to boot from a device located on the emulator connector. Table 3-5 describes the ROMSWAP and ROM-DISABLE switch positions.

Table 3-5. ROMSWAP and ROM-DISABLE Switch Positions

ROMSWAP	ROM-DISABLE	U3 Address	U4 Address	Boot Device
OFF	OFF	FEF8 0000H	FEFC 0000H	U4
ON	OFF	FEFC 0000H	FEF8 0000H	U3
X	ON	not available	not available	Emulator Connector

3.4.2 V_{PP} Switch

The V_{PP} switch (SW1-1) enables/disables V_{PP} to the boot Flash ROMs, U3 and U4. It is recommended that this switch remains set to OFF (the default setting from the factory). When V_{PP} is enabled (switch is in ON position), the processor may not be able to boot from ROM if the power sequencing of +5 V and +12 V is not correct.

3.5 CONSOLE SERIAL PORT

The console serial port on the IQ-SDK platform, based on a 16C550 UART, is capable of operation from 300 to 115,200 bps. The port is connected to a phone jack-style plug on the IQ-SDK platform. The DB25 to RJ-45 cable included with the IQ-SDK can be used to connect the console port to any standard RS-232 port on the host system.

The UART on the IQ-SDK platform is clocked with a 1.843 MHz clock, and may be programmed to use this clock with its internal baud rate counters. The UART register addresses are shown in Table 3-6; refer to the 16C550 device data book for a detailed description of the registers and device operation. Note that some UART addresses refer to different registers depending on whether a read or a write is being performed.

Table 3-6. UART Register Addresses

Address	Read Register	Write Register
E000 0000H	Receive Holding Register	Transmit Holding Register
E000 0004H	Unused	Interrupt Enable Register
E000 0008H	Interrupt Status Register	FIFO Control Register
E000 000CH	Unused	Line Control Register
E000 0010H	Unused	Modem Control Register
E000 0014H	Line Status Register	Unused
E000 0018H	Modem Status Register	Unused
E000 001CH	Scratchpad Register	Scratchpad Register

3.6 SECONDARY PCI BUS EXPANSION CONNECTOR

The i960 RP processor provides a secondary PCI bus to which expansion modules may be connected (see Section 4.5 for details). Modules are connected to the secondary PCI bus using a 120 pin AMP connector (see Section 6.4). Pin assignments for the secondary PCI connector differ slightly from normal PCI pin assignments, in that four clock signals, S-CLK3:0; four request signals, S-REQ3:0#; and four grant signals, S-GNT3:0# have been added. These signals replace the following signals from the standard PCI edge connector, which are unnecessary on the IQ-SDK platform's expansion connector: PRSNT1#, PRSNT2#, CLK, GNT#, REQ#, IDSEL, and six reserved pins. The pinout of the expansion connector can be found in Table 6-1.

Information on the expansion modules can be found in Chapter 6.

3.7 SERIAL EEPROM (I^2C)

A 512 Kbyte serial EEPROM is connected to the I^2C bus on the IQ-SDK platform at address 0. Intel does not define the contents of this device, so it is available for use by the developer. The EEPROM is read and written using the I^2C bus; consult the *i960® RP Microprocessor User's Manual* and the *24C08 Serial EEPROM Data Sheet* for more information.

3.8 HEADERS

A number of headers are provided on the IQ-SDK platform to allow the adapter to be connected to various external devices. This section details these headers.

3.8.1 I²C Bus Header

A four post I²C header is included on the IQ-SDK platform. The pinout for this header is shown in Table 3-7.

A 512 Kbyte serial EEPROM is connected to the I²C bus on the IQ-SDK platform; see Section 3.7 for details.

Table 3-7. I²C Header Pinout

PIN	SIGNAL	PIN	SIGNAL
1	+5 V fused	2	SCL
3	+5 V fused	4	GND
5	+5 V fused	6	SDA
7	+5 V fused	8	GND

3.8.2 Emulator Header

The IQ-SDK platform is equipped with an emulator header. For a diagram of signal traces from the i960 RP processor to the emulator header, refer to *Connector Specification for Using In-Circuit Emulators and Logic Analyzers with the i960® RP Processor*. The pinout for the emulator header is shown in Table 3-8.

Table 3-8. Emulator Header Pinout (Sheet 1 of 2)

Connector Pin #	DUT Pin #	Signal Name	Connector Pin #	DUT Pin #	Signal Name	Connector Pin #	DUT Pin #	Signal Name
A001	AE005	Width/ HLTD1/ Retry	B019	F003	RAS3#	C036	C019	LRDYRCV#
A002	AF005	Width/ HLTD1/ Sync	B020	GND		C037	C022	W/R#
A003	AE007	P_RST#	B021	B013	AD16	C038	C023	BLAST#/ EBM#
A004	AE003	STEST	B022	E003	RAS0#	C039	C024	TMS
A005	GND		B023	C002	DP0	C040	GND	
A006	AB002	ICECLK	B024	C004	MA8	D001	M002	DALE1
A007	AB001	ICEVLD#	B025	GND		D002	L002	LEAF0#
A008	AA002	ICEMSG#	B026	A005	MA6	D003	L001	CE1#
A009	AA001	ICEBRK#	B027	C007	MA0	D004	K002	DWE1#
A010	GND		B028	C009	AD26	D005	GND	
A011	Y001	ICEBUS5	B029	A010	AD24	D006	K001	DWE0#
A012	W002	ICEBUS3	B030	GND		D007	J002	MWE2#
A013	W001	ICEBUS2	B031	C012	AD17	D008	J001	MWE1#
A014	V002	ICEBUS0	B032	C014	AD11	D009	H002	CAS7#
A015	GND		B033	C016	AD5	D010	GND	
A016	R002	XINT7#	B034	C017	AD2	D011	H001	CAS6#
A017	R001	XINT6#	B035	GND		D012	G002	CAS4#
A018	P002	XINT4#	B036	C020	ALE	D013	F002	CAS1#
A019	B003	MA11	B037	A022	BE0#	D014	F001	CAS0#
A020	GND		B038	B024	TCK	D015	GND	
A021	B008	AD31	B039	D026	TDI	D016	E002	RAS2#
A022	F025	S_CLK	B040	GND		D017	E001	RAS1#
A023	B015	AD10	C001	N002	S_INTB#/ XINT1#	D018	D003	DP1
A024	B016	AD7	C002	N001	S_INTA#/ XINT0#	D019	D001	DP2
A025	GND		C003	AD006	LRST#	D020	GND	
A026	B017	AD4	C004	AD005	FAIL#	D021	A013	AD15
A027	A017	AD3	C005	GND		D022	B004	MA9
A028	B018	AD1	C006	AD004	LOCK#/ ONCE#	D023	B005	MA7
A029	A018	AD0	C007	AC002	ICESEL#	D024	B006	MA4
A030	GND		C008	AB003	ICELOCK#	D025	GND	
A031	B019	RDYRCV#	C009	Y003	ICEBUS4	D026	B007	MA2
A032	B021	ADS#	C010	GND		D027	A007	MA1
A033	A021	BE3#	C011	V003	HOLDA	D028	A008	AD30
A034	B022	BE1#	C012	T003	NMI#	D029	B009	AD28

Table 3-8. Emulator Header Pinout (Sheet 2 of 2)

Connector Pin #	DUT Pin #	Signal Name	Connector Pin #	DUT Pin #	Signal Name	Connector Pin #	DUT Pin #	Signal Name
A035	GND		C013	P003	S_INTC#/ XINT2#	D030	GND	
A036	B023	DT/R#	C014	N003	WAIT#	D031	A009	AD27
A037	A023	Den#/ BIMODE#	C015	GND		D032	B010	AD25
A038	C025	TRST#	C016	M001	DALE0	D033	B011	AD22
A039	D025	TDO	C017	K003	MWE3#	D034	A011	AD21
A040	GND		C018	H003	CAS5#	D035	GND	
B001	AC003	MSGFRM#	C019	G001	CAS3#	D036	B012	AD19
B002	Y002	ICEBUS6	C020	GND		D037	A012	AD18
B003	AF004	D/C# RSTMODE#	C021	B014	AD13	D038	A015	AD09
B004	NC		C022	A014	AD12	D039	C021	BE2#
B005	GND		C023	D002	DP3	D040	GND	
B006		VCC (target)	C024	C003	MA10			
B007	AC001	ICEADS#	C025	GND				
B008	AA003	ICEBUS7	C026	C005	MA5			
B009	W003	ICEBUS1	C027	C006	MA3			
B010	GND		C028	C008	AD29			
B011	V001	HOLD	C029	C010	AD23			
B012	R003	XINT5#	C030	GND				
B013	P001	S_INTD#/ XINT3#	C031	C011	AD20			
B014	M003	LEAF1#	C032	C013	AD14			
B015	GND		C033	C015	AD8			
B016	L003	CE0#	C034	A016	AD6			
B017	J003	MWE0#	C035	GND				
B018	G003	CAS2#						

3.8.3 JTAG Header

The JTAG header allows debugging hardware to be quickly and easily connected to some of the i960 RP processor's logic signals.

The JTAG header is a 16 pin header. A 3M connector (part number 2516-6002UG) is required to connect to this header. The pinout for the JTAG header is shown in Table 3-9. The header and connector are keyed using a tab on the connector and a slot on the header to ensure proper installation.

Each signal in the JTAG header is paired with its own ground connection to avoid the noise problems associated with long ribbon cables. Signal descriptions are found in the *i960® RP Microprocessor User's Manual*.

Table 3-9. JTAG Header Pinout

PIN	SIGNAL	INPUT/OUTPUT TO 80960RP	PIN	SIGNAL
1	TRST#	IN	2	GND
3	TDI	IN	4	GND
5	TDO	OUT	6	GND
7	TMS	IN	8	GND
9	TCK	IN	10	GND
11	RSTIN#	IN	12	GND
13	RSTOUT#	OUT	14	GND
15	PWRVLD	OUT	16	GND

3.8.4 APIC Header

An APIC (Advanced Programmable Interrupt Controller) header is included on the IQ-SDK platform at J8. The APIC interface allows the IQ-SDK platform to act as an intelligent interrupt controller for the host system. The APIC header is a 3 by 2 header. The pinout for the APIC header is shown in Table 3-10, and the signal definitions for the APIC header can be found in the *i960® RP Microprocessor User's Manual*. For APIC header software information, refer to *I/O APIC Emulation Software for the i960® RP Processor.*

Table 3-10. IQ-SDK Platform APIC Header Pinout

PIN	SIGNAL	PIN	SIGNAL	PIN	SIGNAL
1	PIC CLK	3	PICDATA0	5	PIC D1
2	GND	4	GND	6	GND

3.9 USER LEDS

The IQ-SDK platform has a bank of eight user-programmable LEDs, located on the upper edge of the adapter board. These LEDs are controlled by a write-only register and used as a debugging aid during development. Software can control the state of the user LEDs by writing to the LED Register, located at E004 0000H. Each of the eight bits of this register correspond to one of the user LEDs. Clearing a bit in the LED Register by writing a “0” to it turns the corresponding LED on, while setting a bit by writing a “1” to it turns the corresponding LED off. Resetting the IQ-SDK platform results in clearing the register and turning all the LEDs on. The LED Register bitmap is shown in Figure 3-3.

The user LEDs are numbered in descending order from left to right, with LED7 being on the left when looking at the component side of the adapter.

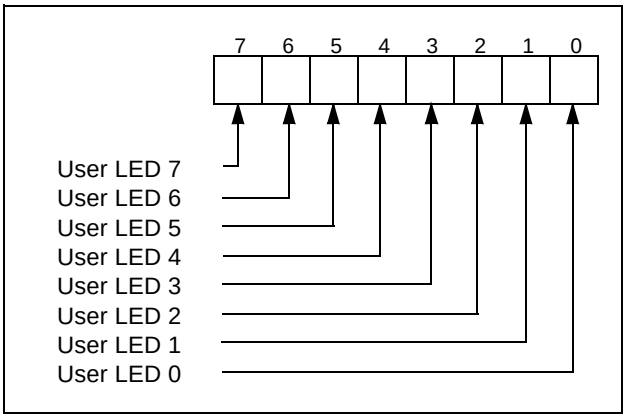

Figure 3-3. LED Register Bitmap

3.10 DRAM STATUS REGISTER

The DRAM status register is included on the IQ-SDK platform so that initialization code can configure the adapter for the size and speed of installed DRAM. The register is a read-only register at address E004 0000H.

.

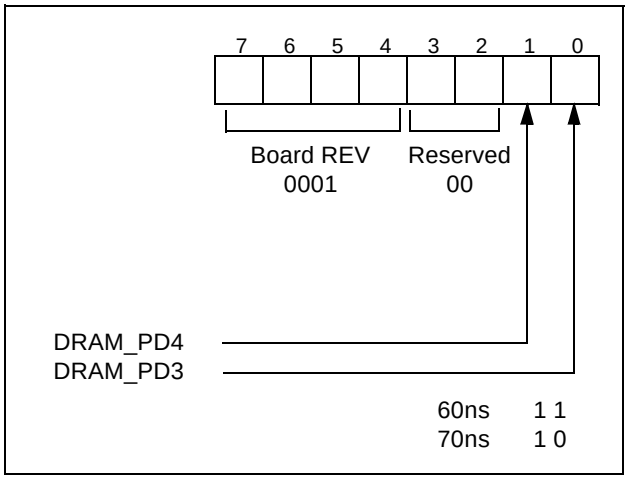

Figure 3-4. DRAM Status Register

4

i960® RP PROCESSOR OVERVIEW

CHAPTER 4
i960® RP PROCESSOR OVERVIEW

This chapter describes the features and operation of the processor on the IQ-SDK platform. For more detail, refer to the *i960® RP Microprocessor User's Manual*.

4.1 CPU MEMORY MAP

The memory map for the IQ-SDK platform is shown in Figure 4-1. All addresses below 8202 0000H on the IQ-SDK platform are reserved for various functions of the i960 RP processor, as shown on the memory map. Documentation for these areas, as well as the processor memory mapped registers at FF00 0000H and the IBR, can be found in the *i960® RP Microprocessor User's Manual*.

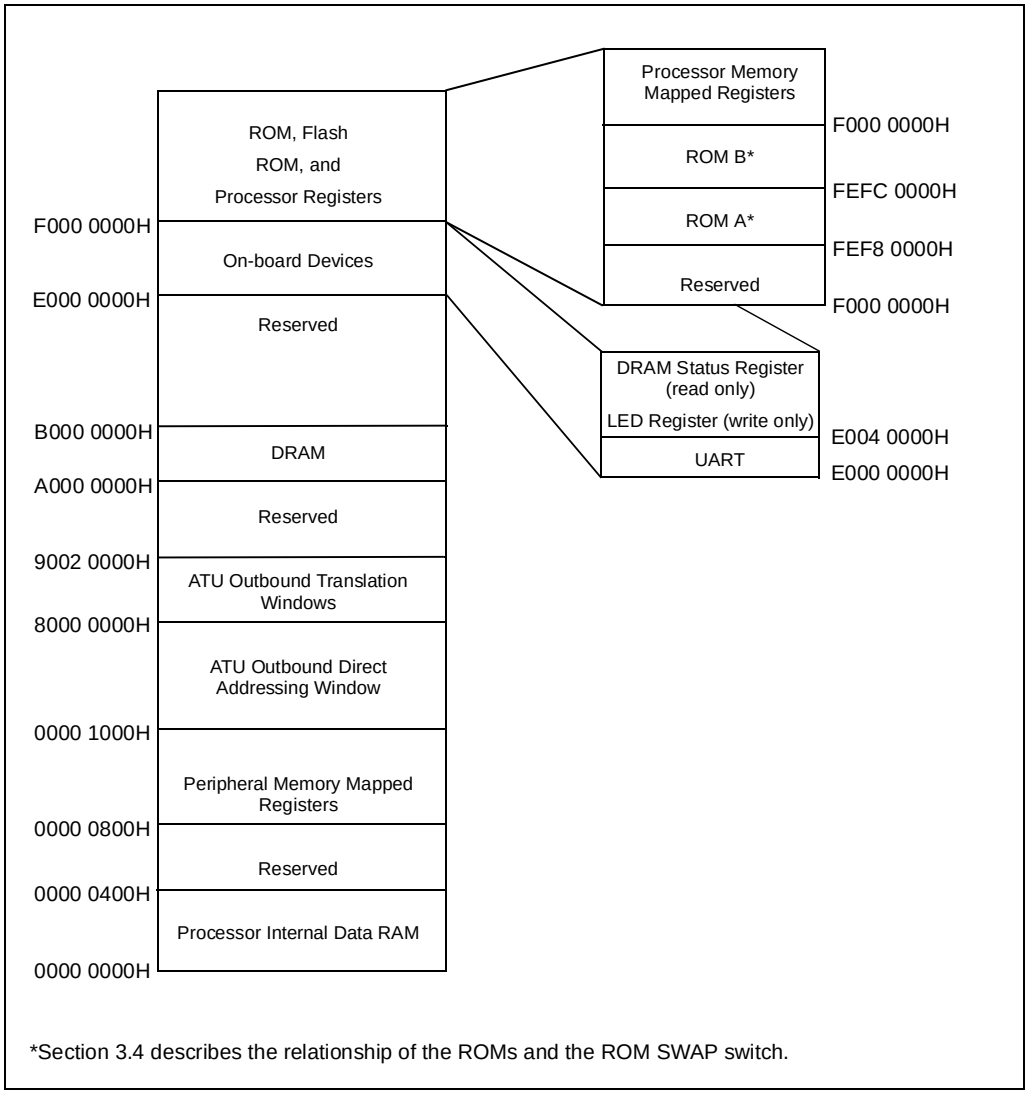

Figure 4-1. IQ-SDK Platform Memory Map

4.2 LOCAL INTERRUPTS

The i960 RP processor is built around an i960 Jx core, which has nine external interrupt lines designated XINT0# through XINT7# and NMI#. In the i960 RP processor, these interrupt lines are not directly connected to external interrupts, but pass through a layer of internal interrupt routing logic. Figure 4-2 shows the interrupt connections on the i960 RP processor.

XINT0# through XINT3# on the i960 Jx core can be used to receive PCI interrupts from the secondary PCI bus, or these interrupts can be passed through to the primary PCI interface, depending on the setting of the XINT Select bit of the PCI Interrupt Routing Select Register in the i960 RP processor. On the IQ-SDK platform, XINT0# through XINT3# are configured to receive interrupts from the secondary PCI bus.

XINT4# and XINT5# on the i960 RP processor may be connected to interrupt sources external to the processor. On the IQ-SDK platform, XINT4# is unused and XINT5# is connected to the 16C550 UART.

XINT6#, XINT7#, and NMI# receive interrupts from internal sources as well as their respective external interrupt lines. Since all of these interrupts accept signals from multiple sources, a status register is provided for each of them to allow service routines to identify the source of the interrupt. Each of the possible interrupt sources is assigned a bit position in the status register. The interrupt sources for these lines are shown in Figure 4-2. On the IQ-SDK platform, these interrupts are not connected to any external interrupt sources and receive interrupts only from the internal devices on the i960 RP processor. Note that error indications are received on NMI#, whereas XINT6# and XINT7# receive normal device interrupts.

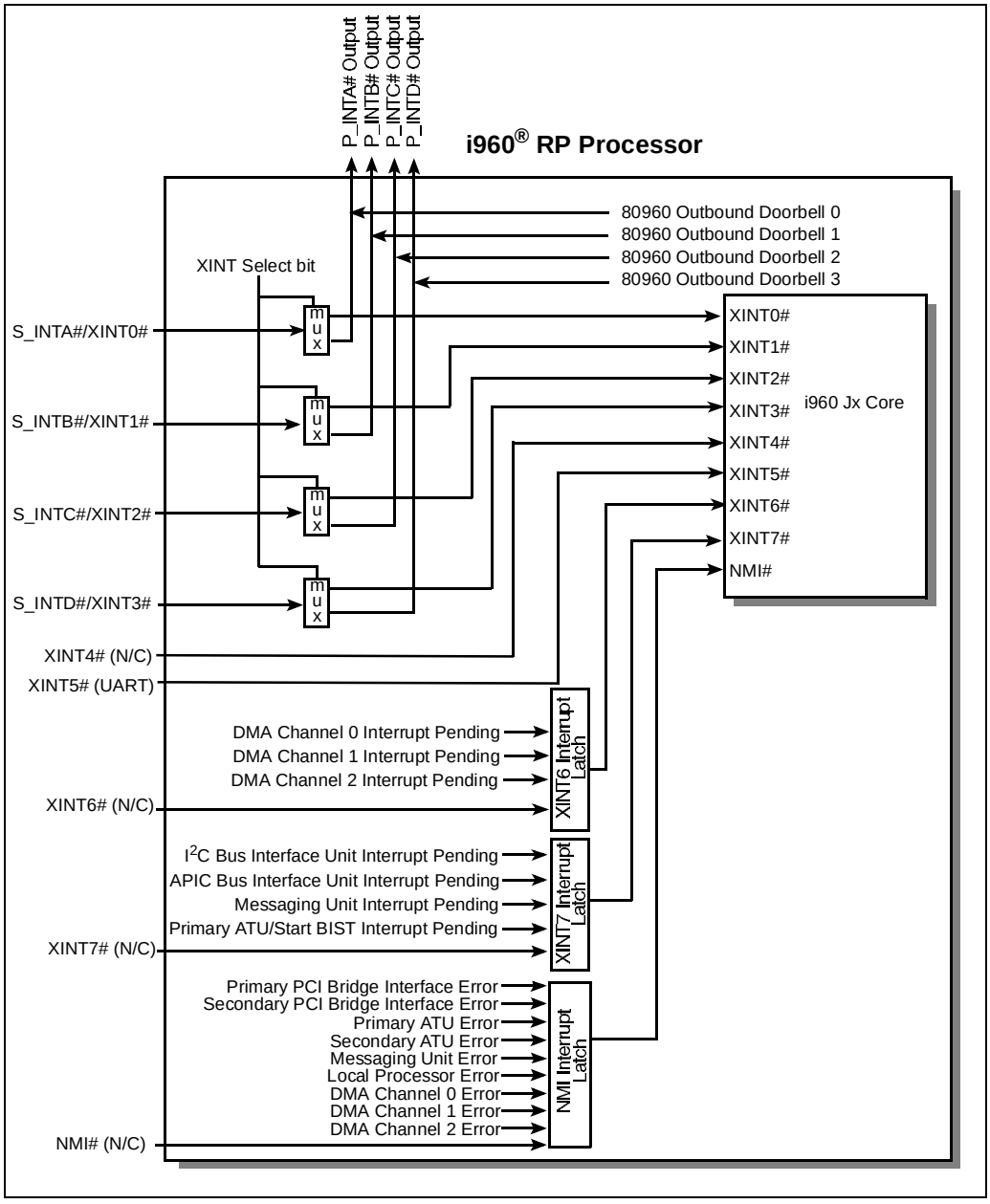

Figure 4-2. i960® RP Interrupt Controller Connections

4.3 CPU COUNTER/TIMERS

The i960 RP processor is equipped with two on-chip counter/timers which are clocked with the i960 RP processor clock signal. The i960 RP processor receives its clock from the primary PCI interface clock, generated by the motherboard. Most motherboards generate a 33 MHz clock signal, although the PCI specification only requires a clock frequency between 0 and 33 MHz. The timers can be programmed for single-shot or continuous mode, and can generate interrupts to the processor when the countdown expires.

4.4 PRIMARY PCI INTERFACE

The primary PCI interface on the IQ-SDK platform provides the i960 RP processor with a connection to the PCI bus on the host system. Only the PCI-to-PCI bridge unit on the i960 RP processor is directly connected to the primary PCI interface. Devices installed on IQ Modules are connected to the PCI bus via the bridge unit on the i960 RP processor. The PCI-to-PCI bridge accepts Type 1 configuration cycles destined for devices on the secondary bus, and will forward them as Type 0 or Type 1 configuration cycles, or as special cycles.

4.5 SECONDARY PCI INTERFACE

The secondary PCI interface provided by the i960 RP processor is used to connect PCI-based IQ Modules to the host system's PCI bus. IQ Modules are attached to the IQ-SDK platform with a connector (see Section 3.6) and may contain up to four separate PCI devices. The i960 RP processor provides PCI-to-PCI bridge functionality to map installed PCI devices onto the host PCI bus, and supports transaction forwarding in both directions across the bridge. PCI devices on IQ Modules can therefore act as masters or slaves on the host system's PCI bus. Additional PCI-to-PCI bridge devices are supported by the i960 RP processor on its secondary PCI interface and can be designed into IQ Modules. In addition, the i960 RP processor supports "private" PCI devices on its secondary bus. Private devices are hidden from initialization code on the host system, and are configured and accessed directly by the i960 RP processor. These devices are not part of the normal PCI address space, but they can act as PCI bus masters and transfer data to and from other PCI devices in the system.

Unless designated as private devices, PCI devices installed on the secondary PCI interface of the IQ-SDK platform are mapped into the system-wide PCI address space by configuration software running on the host system. No logical distinction is made at the system level between devices on the primary PCI bus and devices on secondary buses; all transaction forwarding is handled transparently by the PCI-to-PCI bridge. Configuration cycles and read and write accesses from the host are forwarded through the PCI-to-PCI bridge unit of the i960 RP processor. Master read and write cycles from devices on the secondary PCI bus are also forwarded to the host bus by the PCI-to-PCI bridge unit.

4.6 DMA CHANNELS

The i960 RP processor features three independent DMA channels, two of which operate on the primary PCI interface, whereas the remaining one operates on the secondary PCI interface. All three of the DMA channels connect to the i960 RP processor's local bus and can be used to transfer data from PCI devices to memory on the IQ-SDK platform. Support for demand mode, chaining, and scatter/gather is built into all three channels. The DMA can address the entire 2^{64} bytes of address space on the PCI bus and 2^{32} bytes of address space on the i960 local bus.

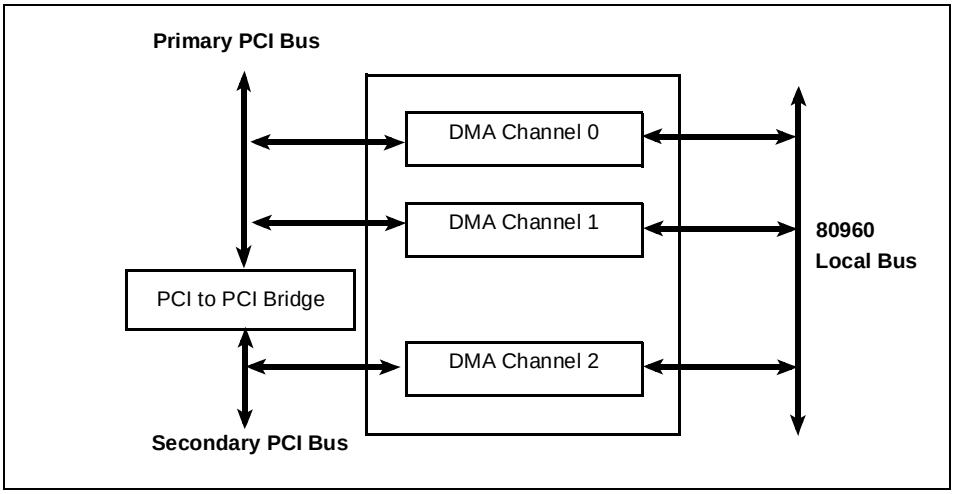

Figure 4-3. i960® RP Processor DMA Controller

5

MON960 ON THE IQ-SDK PLATFORM

CHAPTER 5
MON960 ON THE IQ-SDK PLATFORM

A number of additions have been made to MON960 to fully support the IQ-SDK. This section describes these additions. For complete documentation on the operation of MON960, see the *MON960 Debug Monitor User's Guide*.

5.1 SECONDARY PCI BUS EXPANSION CONNECTOR

The IQ-SDK platform contains a secondary PCI bus expansion connector to give users access to the secondary PCI bus of the 80960RP using either a standard or custom I/O module. Extensions to MON960 perform secondary PCI bus initialization including the establishment of a secondary PCI bus address map. Routines compatible with the *PCI Local Bus Specification* Revision 2.1 allow the software on the IQ-SDK platform to search for devices on the secondary PCI bus and read and write the configuration space of those devices.

5.2 FIRMWARE COMPONENTS

The IQ-SDK firmware package consists of four main components: initialization firmware, the MON960 kernel, MON960 extensions, and diagnostics/example code. These four components together are referred to as MON960.

5.2.1 MON960 Initialization

The main function of MON960 initialization is to put the IQ-SDK platform into a known, functional state that allows the host processor to perform PCI initialization and allows both the MON960 kernel and the MON960 extensions to load and execute correctly. MON960 initialization is the first activity performed after a RESET condition. MON960 initialization encompasses all major portions of the 80960RP and IQ-SDK platform including i960 Jx core initialization, Memory Controller initialization, DRAM initialization, Primary PCI Address Translation Unit (ATU) initialization, and PCI-to-PCI Bridge Unit initialization.

The IQ-SDK platform is designed to use the Configuration Mode of the 80960RP. Configuration Mode allows the i960 Jx core to initialize and control the initialization process before the PCI host configures the 80960RP processor. By utilizing Configuration Mode, the user is given the ability to initialize the PCI configuration registers to values other than the default power-up values. Configuration Mode gives the user maximum flexibility to customize the way in which the 80960RP processor and IQ-SDK platform appear to the PCI host configuration software.

5.2.2 i960 Jx Core Initialization

The i960 Jx core begins the initialization process by reading its Initial Memory Image (IMI) from a fixed address in the boot ROM (FEFF FF30H in the i960 address space). The IMI includes the Initialization Boot Record (IBR), the Process Control Block (PRCB), and several system data structures. The IBR provides initial configuration information for the core and integrated peripherals, pointers to the system data structures and the first instruction to be executed after processor initialization, and checksum words that the processor uses in its selftest routine. In addition to the IBR and PRCB, the required data structures are: the System Procedure Table, the Control Table, the Interrupt Table, the Fault Table, and a User Stack (application dependent), Supervisor Stack, and Interrupt Stack.

5.2.3 Memory Controller Initialization

Since the 80960RP Memory Controller is integral to the design and operation of the IQ-SDK platform, the operational parameters for Bank 0 and Bank 1 are established immediately after processor core initialization. Memory Bank 0 is associated with the ROMs on the IQ-SDK platform. Memory Bank 1 is associated with the UART, the LED Control Register, and the DRAM Status Register. Parameters such as Bank Base Address, Read Wait States, and Write Wait States must be established to ensure the proper operation of the IQ-SDK platform. The Memory Controller is initialized so as to be consistent with the IQ-SDK platform memory map shown in Figure 4-1.

5.2.4 DRAM Initialization

DRAM Initialization includes not only establishing the operational parameters for the 80960RP's DRAM controller, but also sizing and clearing the installed DRAM configuration. To configure the system properly, an algorithm is used in which the DRAM controller is configured for the largest supportable bank size, a memory test is run to determine the actual bank size, and the DRAM controller is reconfigured for the proper bank size. The DRAM controller is also initialized to enable refresh cycles. Once the DRAM controller is configured, the DRAM is cleared in preparation for the C language runtime environment. The actual DRAM size is stored for later use (e.g. to establish the size of the IQ-SDK platform PCI Slave image). The DRAM Controller is initialized to be consistent with the IQ-SDK platform memory map shown in Figure 4-1.

5.2.5 Primary PCI Interface Initialization

The IQ-SDK platform is a multi-function PCI device. On the primary PCI bus, two functions (from a PCI Configuration Space standpoint) are supported.

- Function 0 is the PCI-to-PCI Bridge of the 80960RP, which optionally provides access capability between the primary PCI bus and the secondary PCI bus.
- Function 1 is the Primary ATU which provides access capability between the primary PCI bus and the local i960 bus.

Since the IQ-SDK platform is operating in Configuration Mode, the PCI host will receive PCI Retries when it attempts to access the Configuration Space of the 80960RP's primary PCI interface until the Configuration Cycle Disable bit in the Extended Bridge Control Register (EBCR) is cleared. For this reason, and to prevent PCI host problems, Primary PCI Initialization occurs at the earliest possible opportunity after Memory and DRAM controller initialization. The completion of primary PCI interface initialization is signalled by having the i960 Jx core clear the Configuration Cycle Disable bit in the EBCR.

5.2.6 Primary ATU Initialization

Primary ATU (Bridge) initialization includes initialization by the i960 Jx core and initialization by the PCI host processor. Local initialization occurs first and consists mainly of establishing the operational parameters for access to the local IQ-SDK platform bus. The Primary Inbound ATU Limit Register (PIALR) is initialized to establish the block size of memory required by the Primary ATU. The PIALR value is based on the installed DRAM configuration. The Primary Inbound ATU Translate Value Register (PIATVR) is initialized to establish the translation value for PCI-to-Local accesses. The PIATVR value is set to reference the base of local DRAM. The Primary Outbound Memory Window Value Register (POMWVR) is initialized to establish the translation value for Local-to-PCI accesses. The POMWVR value remains at its default value of “0” to allow the IQ-SDK platform to access the start of the PCI Memory address map, which is typically occupied by PCI host memory. Likewise, the Primary Outbound I/O Window Value Register (POIOWVR) remains at its default value of “0” to allow the IQ-SDK platform to access the start of the PCI I/O address map. PCI Doorbell-related parameters are also established to allow for communication between the IQ-SDK platform and a PCI bus master using the doorbell mechanism.

By default, Primary Outbound Configuration Cycle parameters are not established and Dual Address Cycle (DAC) support is not enabled. The ATU Configuration Register (ATUCR) is initialized to establish the operational parameters for the Doorbell Unit and ATU interrupts (both primary and secondary), and to enable the primary and secondary ATUs. The PCI host is responsible for allocating PCI address space (Memory, Memory Mapped I/O, and I/O), and assigning the PCI Base addresses for the IQ-SDK platform.

5.2.7 PCI-to-PCI Bridge Initialization

PCI-to-PCI Bridge initialization includes initialization by the i960 Jx core and initialization by the PCI host processor. Local initialization occurs first and consists mainly of establishing the operational parameters for the secondary PCI interface of the PCI-to-PCI bridge. On the IQ-SDK platform, the secondary PCI bus is configured to consist of private devices (not visible to PCI host configuration cycles). To support a private secondary PCI bus, the Secondary IDSEL Select Register (SISR) is initialized to prevent the secondary PCI address bits [20:16] from being asserted during conversion of PCI Type 1 configuration cycles on the primary PCI bus to PCI Type 0 configuration cycles on the secondary PCI bus. Secondary PCI bus masters are prevented from initiating transactions that will be forwarded to the primary PCI interface. The PCI host is responsible for assigning and initializing the PCI bus numbers, allocating PCI address space (Memory, Memory Mapped I/O, and I/O), and assigning the IRQ numbers to valid interrupt routing values.

5.2.8 Secondary ATU Initialization

Secondary ATU (Bridge) initialization consists mainly of establishing the operational parameters for access between the local IQ-SDK platform bus and the secondary PCI devices. The Secondary Inbound ATU Base Address Register (SIABAR) is initialized to establish the PCI base address of IQ-SDK platform local memory from the secondary PCI bus. By convention, the secondary PCI base address for access to IQ-SDK platform local memory is "0". The Secondary Inbound ATU Limit Register (SIALR) is initialized to establish the block size of memory required by the secondary ATU. The SIALR value is based on the installed DRAM configuration. The Secondary Inbound ATU Translate Value Register (SIATVR) is initialized to establish the translation value for Secondary PCI-to-Local accesses. The SIATVR value is set to reference the base of local DRAM. The Secondary Outbound Memory Window Value Register (SOMWVR) is initialized to establish the translation value for Local-to-Secondary PCI accesses. The SOMWVR value is left at its default value of "0" to allow the IQ-SDK platform to access the start of the PCI Memory address map. Likewise, the Secondary Outbound I/O Window Value Register (SOIOWVR) is left at its default value of "0" to allow the IQ-SDK platform to access the start of the PCI I/O address map.

On the secondary PCI bus, the IQ-SDK platform assumes the duties of PCI host and, as such, is required to configure the devices of the secondary PCI bus. Secondary Outbound Configuration Cycle parameters are established during secondary PCI bus configuration. Secondary PCI bus configuration is accomplished via MON960 Extension routines. Secondary PCI bus DAC support is not initialized.

5.3 MON960 KERNEL

The MON960 Kernel (monitor) provides the IQ-SDK user with a software platform on which application software can be developed and run. The monitor provides several features that the IQ-SDK user can use to speed application development. Among the available features are:

- Communication with a terminal or terminal emulation package on a host computer through a serial cable with automatic baud rate detection
- Communication with a software debugger such as DB960 or gdb960 (available from Intel) using the software interface called Host Debugger Interface (HDI)
- Communication with the host computer via the primary PCI bus
- Downloads of COFF object files via the primary PCI bus or via the serial console port at baud rates up to 115,200 baud
- Downloads of COFF object files via the primary PCI bus
- On-board erasure and programming of Intel 28F020 Flash ROMs
- Memory display and modification capability
- Breakpoint and single-step capability to support debugging of user code
- Disassembly of i960 instructions

5.4 MON960 EXTENSIONS

The monitor has been extended to include the secondary PCI bus initialization and also the BIOS routines which are contained in Revision 2.1 of the PCI BIOS Specification.

5.4.1 Secondary PCI Initialization

MON960 extensions are responsible for initializing the devices on the secondary PCI bus of the IQ-SDK platform. Secondary PCI initialization involves allocating address spaces (Memory, Memory Mapped I/O, and I/O), assigning PCI base addresses, assigning IRQ values, and enabling PCI mastership. Devices containing PCI-to-PCI bridges and hierarchical buses are not supported.

5.4.2 PCI BIOS Routines

The PCI BIOS routines are accessible at the MON960 layer and can also be accessed by application level software by using the i960 system call mechanism and the System Procedure Table. The supported BIOS functions are described in the subsections that follow.

pci_bios_present()
find_pci_device()
find_pci_class_code()
generate_special_cycle()
read_config_byte()
read_config_word()
read_config_dword()
write_config_byte()
write_config_word()
write_config_dword()
get_irq_routing_options()
set_pci_irq()

Although the calling interface is different from that used on a DOS-based host, these functions preserve, as closely as possible, the parameters and return values described in *PCI Local Bus Specification* Revision 2.1. Functions that return multiple values do so by filling in the fields of a structure passed by the calling routine.

5.4.2.1 pci_bios_present

This function allows the caller to determine whether the PCI BIOS interface function set is present, and the current interface version level. It also provides information about the hardware mechanism used for accessing configuration space and whether or not the hardware supports generation of PCI Special Cycles.

Calling convention:

```
int pci_bios_present (
       PCI_BIOS_INFO *info
       );
```

Return values:

This function always returns SUCCESSFUL.

5.4.2.2 find_pci_device

This function returns the location of PCI devices that have a specific Device ID and Vendor ID. Given a Vendor ID, a Device ID, and an Index, the function returns the Bus Number, Device Number, and Function Number of the Nth Device/Function whose Vendor ID and Device ID match the input parameters.

Calling software can find all devices having the same Vendor ID and Device ID by making successive calls to this function starting with the index set to “0”, and incrementing the index until the function returns DEVICE_NOT_FOUND. A return value of BAD_VENDOR_ID indicates that the Vendor ID value passed had a value of all “1”s.

Calling convention:

```
int find_pci_device (
        int     device_id,
        int     vendor_id,
        int     index
        );
```

Return values:

This function returns SUCCESSFUL if the indicated device is located, DEVICE_NOT_FOUND if the indicated device cannot be located, or BAD_VENDOR_ID if the vendor_id value is illegal.

5.4.2.3 find_pci_class_code

This function returns the location of PCI devices that have a specific Class Code. Given a Class Code and an Index, the function returns the Bus Number, Device Number, and Function Number of the Nth Device/Function whose Class Code matches the input parameters.

Calling software can find all devices having the same Class Code by making successive calls to this function starting with the index set to “0”, and incrementing the index until the function returns DEVICE_NOT_FOUND.

Calling convention:

```
int find_pci_class_code (
     int  class_code,
     int  index
     );
```

Return values:

This function returns SUCCESSFUL if the indicated device is located or DEVICE_NOT_FOUND if the indicated device cannot be located.

5.4.2.4 generate_special_cycle

This function allows for generation of PCI Special Cycles. The generated special cycle is broadcast on a specific PCI Bus in the system.

PCI Special Cycles are not supported on the IQ-SDK platform secondary PCI bus.

Calling convention:

```
int generate_special_cycle (
     int bus_number,
     int special_cycle_data
     );
```

Return values:

Since PCI Special Cycles are not supported by the IQ-SDK platform, this function always returns FUNC_NOT_SUPPORTED.

5.4.2.5 read_config_byte

This function allows the caller to read individual bytes from the configuration space of a specific device.

Calling convention:

```
int read_config_byte (
     int  bus_number,
     int  device_number,
     int  function_number,
     int  register_number,/* 0,1,2,...,255 */
     UINT 8*data
     );
```

Return values:

This function returns SUCCESSFUL if the indicated byte was read correctly or ERROR if there is a problem with the parameters.

5.4.2.6 read_config_word

This function allows the caller to read individual shorts (16 bits) from the configuration space of a specific device. The Register Number parameter must be a multiple of two (i.e., bit 0 must be set to “0”).

Calling convention:

```
int read_config_word (
     int  bus_number,
     int  device_number,
     int  function_number,
     int  register_number,/* 0,2,4,...,254 */
     UINT 16*data
     );
```

Return values:

This function returns SUCCESSFUL if the indicated word was read correctly or ERROR if there is a problem with the parameters.

5.4.2.7 read_config_dword

This function allows the caller to read individual longs (32 bits) from the configuration space of a specific device. The Register Number parameter must be a multiple of four (i.e., bits 0 and 1 must be set to “0”).

Calling convention:

```
int read_config_dword (
     int  bus_number,
     int  device_number,
     int  function_number,
     int  register_number,/* 0,4,8,...,252 */
     UINT 32*data
     );
```

Return values:

This function returns SUCCESSFUL if the indicated long was read correctly or ERROR if there is a problem with the parameters.

5.4.2.8 write_config_byte

This function allows the caller to write individual bytes to the configuration space of a specific device.

Calling convention:

```
int write_config_byte (
     int  bus_number,
     int  device_number,
     int  function_number,
     int  register_number,/* 0,1,2,...,255 */
     UINT 8*data
     );
```

Return values:

This function returns SUCCESSFUL if the indicated byte was written correctly or ERROR if there is a problem with the parameters.

5.4.2.9 write_config_word

This function allows the caller to write individual shorts (16 bits) to the configuration space of a specific device. The Register Number parameter must be a multiple of two (i.e., bit 0 must be set to "0").

Calling convention:

```
int write_config_word (
     int  bus_number,
     int  device_number,
     int  function_number,
     int  register_number,/* 0,2,4,...,254 */
     UINT 16*data
     );
```

Return values:

This function returns SUCCESSFUL if the indicated word was written correctly or ERROR if there is a problem with the parameters.

5.4.2.10 write_config_dword

This function allows the caller to write individual longs (32 bits) to the configuration space of a specific device. The Register Number parameter must be a multiple of four (i.e., bits 0 and 1 must be set to “0”).

Calling convention:

```
int write_config_dword (
     int  bus_number,
     int  device_number,
     int  function_number,
     int  register_number,/* 0,4,8,...,252 */
     UINT 32*data
     );
```

Return values:

This function returns SUCCESSFUL if the indicated long was written correctly or ERROR if there is a problem with the parameters.

5.4.2.11 get_irq_routing_options

This routine returns the PCI interrupt routing options available on the IQ-SDK platform. Two values are provided for each PCI interrupt pin for each device. One of these values is a bitmap that shows the interrupt to which 80960RP XINT (XINT3:0) is connected. The second value is a link value that provides a way of specifying which PCI interrupt pins are wire-OR’ed together on the motherboard. Interrupt pins with the same link value are wired together.

The PCI Interrupt routing fabric on the IQ-SDK platform is not reconfigurable (fixed mapping relationships).

Calling convention:

```
int get_irq_routing_options (
     PCI_IRQ_ROUTING_TABLE *table
     );
```

Return values:

This function always returns SUCCESSFUL.

5.4.2.12 set_pci_irq

The PCI Interrupt routing fabric on the IQ-SDK platform is not reconfigurable (fixed mapping relationships); therefore, this function is not supported.

Calling convention:

```
int set_pci_irq (
     int  int_pin,
     int  irq_num,
     int  bus_dev
     );
```

Return values:

This function always returns FUNC_NOT_SUPPORTED.

5.4.3 Additional MON960 Commands

The following commands have been added to the UI interface of MON960 to support the IQ-SDK platform.

5.4.3.1 print_pci Utility

A print_pci command to MON960 is accessed through the MON960 command prompt. This command displays the contents of the PCI configuration space on a selected adapter on the secondary PCI interface. For more information on the meaning of the fields in PCI configuration space, refer to the *PCI Local Bus Specification* Revision 2.1. The syntax of this command is:

pp <bus number> <device number> <function number>

5.5 DIAGNOSTICS / EXAMPLE CODE

IQ-SDK platform diagnostic routines serve a twofold purpose: to verify proper hardware operation and to provide example code for users who need similar functions in their applications. Diagnostic routines fall into two categories: board level diagnostics and PCI expansion module diagnostics.

5.5.1 Board Level Diagnostics

Board level diagnostics exercise all basic areas of the IQ-SDK platform. Diagnostic routines include DRAM tests, UART tests, LED tests, internal timer tests, I^2C bus tests, and primary PCI bus tests. Primary PCI bus tests exercise the primary ATU, the PCI Doorbell unit, and the PCI DMA controller. Interrupts from both local and PCI sources are generated and handled. The PCI bus tests require an external test suite running on a PC to verify complete functionality of the IQ-SDK platform.

5.5.2 PCI Expansion Module Diagnostics

PCI expansion module (IQ Module) diagnostics exercise the secondary PCI bus and the expansion modules, and also demonstrate the use of the PCI BIOS routines present in MON960 and illustrate basic functions in the expansion modules (e.g., sending and receiving Ethernet packets and reading and writing SCSI disk blocks). All Cyclone-built PCI expansion modules are supported by MON960 diagnostic code.

6

IQ MODULE INTERFACE

CHAPTER 6
IQ MODULE INTERFACE

6.1 INTRODUCTION

The IQ Module Interface allows PCI devices to be connected to the secondary PCI interface of the i960 RP processor. Up to four devices may be included in an IQ Module (the i960 RP processor allows up to six devices on its secondary bus), including PCI-to-PCI bridges. The standard signals defined for 32-bit PCI edge connectors are used for IQ Modules, with the exceptions noted in Section 6.3. The timing for devices on IQ Modules is the same as the timing for any other PCI device; see the *PCI Local Bus Specification* Revision 2.1 for details.

6

A number of IQ Modules are available from Intel. This section is intended for users interested in developing their own modules.

6.2 PHYSICAL ATTRIBUTES

The physical dimensions of IQ Modules are shown in the following two figures.

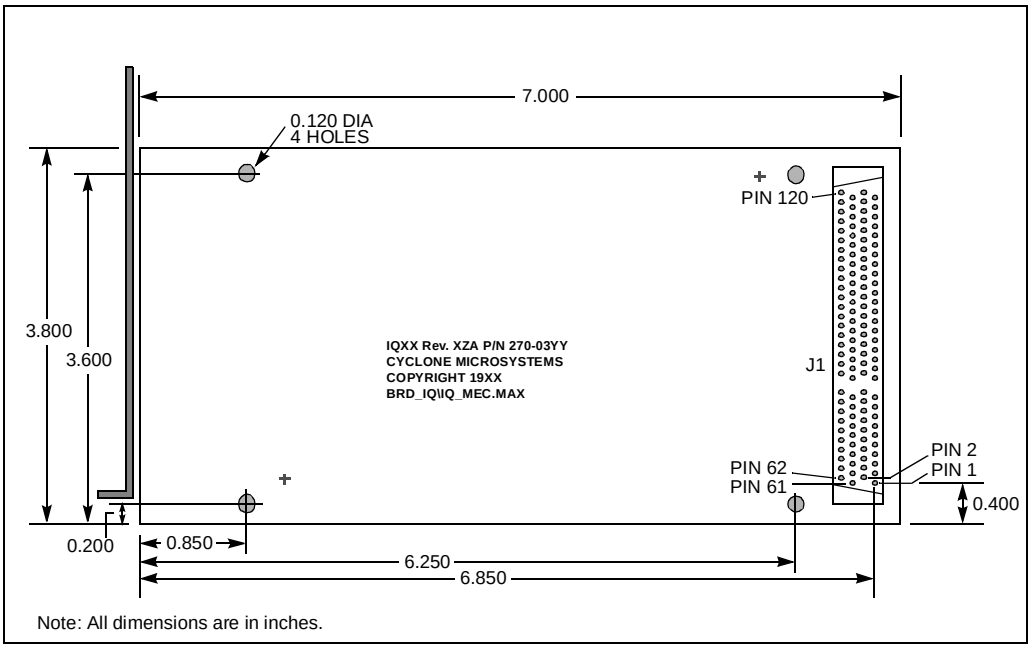

Figure 6-1. IQ Module Physical Diagram

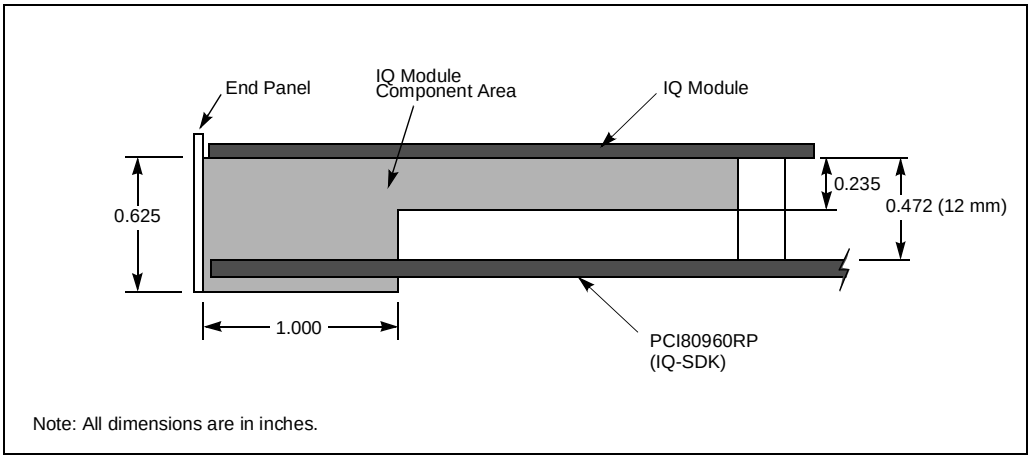

Figure 6-2. IQ Module Component Clearance Drawing

6.3 IQ MODULE SIGNAL DEFINITIONS

IQ Modules use the signals defined in the *PCI Local Bus Specification* Revision 2.1 for 32-bit PCI devices, with a few minor changes.

- Added four clock signals, S_CLK3:0, to the IQ Module connector
- Added four request signals, S_REQ3:0#, to the IQ Module connector.
- Added four grant signals, S_GNT3:0# to the IQ Module connector.
- Removed the following signals from the standard PCI edge connector: PRSNT1#, PRSNT2#, CLK, GNT#, REQ#, IDSEL, and six reserved pins.

The replaced pins are either of no use in this implementation, or their function is duplicated by one of the added pins.

The added signals correspond directly to signals documented in the *PCI Local Bus Specification* Revision 2.1; there is one signal for each possible PCI device on an IQ Module. S_CLK3:0 follow the description for CLK, S_REQ3:0# follow the description for REQ#, and S_GNT3:0# follow the description for REQ# in the PCI Specification. When the appropriate signals are connected to PCI devices on an IQ Module, each device has the full complement of PCI signals defined in the specification.

IDSEL signals are not provided. The designer of an IQ Module should connect the proper S_AD signal to a device's IDSEL pin. S_AD31:11 may be used depending on whether the device is to be a public or private PCI device.

6.4 IQ MODULE CONNECTOR

IQ Modules use an AMP Champ .050" FH board-to-board connector with 120 pins. Plug AMP P/N 176380-4 is located on the IQ-SDK platform and attaches to receptacle AMP P/N 176372-5. This connector combination allows for a 12mm (0.472") board-to-board spacing. See Figure 6-1 and Figure 6-2 for dimensions and component clearance details.

Table 6-1. IQ Module Connector Pinout (Sheet 1 of 2)

Pin	Signal	Pin	Signal
1	S_TRST#	61	-12 V
2	+12 V	62	S_TCK
3	S_TMS	63	GND
4	S_TDI	64	N/C
5	+5 V	65	+5 V
6	S_INTA#	66	+5 V
7	S_INTC#	67	S_INTB#
8	+5 V	68	S_INTD#
9	CLK_C	69	S_REQ3#
10	+5 V	70	S_REQ1#
11	CLK_D	71	S_GNT3#
12	GND	72	GND
13	GND	73	GND
14	S_GNT1#	74	CLKA
15	S_RST#	75	GND
16	+5 V	76	CLKB
17	S_GNT0#	77	GND
18	GND	78	S_REQ0#
19	S_REQ2#	79	+5 V
20	S_AD30	80	S_AD31
21	3.3 V[1]	81	S_AD29
22	S_AD28	82	GND
23	S_AD26	83	S_AD27
24	GND	84	S_AD25
25	S_AD24	85	3.3 V[1]
26	S_GNT2#	86	S_C/BE3#
27	3.3 V[1]	87	S_AD23
28	S_AD22	88	GND

Table 6-1. IQ Module Connector Pinout (Sheet 2 of 2)

Pin	Signal	Pin	Signal
29	S_AD20	89	S_AD21
30	GND	90	S_AD19
31	S_AD18	91	3.3 V[1]
32	S_AD16	92	S_AD17
33	3.3 V[1]	93	S_C/BE2#
34	S_FRAME#	94	GND
35	GND	95	S_IRDY#
36	S_TRDY#	96	3.3 V[1]
37	GND	97	S_DEVSEL#
38	S_STOP#	98	GND
39	3.3 V[1]	99	S_LOCK#
40	N/C	100	S_PERR#
41	N/C	101	3.3 V[1]
42	GND	102	S_SERR#
43	S_PAR	103	3.3 V[1]
44	S_AD15	104	S_C/BE1#
45	3.3 V[1]	105	S_AD14
46	S_AD13	106	GND
47	S_AD11	107	S_AD12
48	GND	108	S_AD10
49	S_AD9	109	GND
50	S_C/BE0#	110	S_AD8
51	3.3 V[1]	111	S_AD7
52	S_AD6	112	3.3 V[1]
53	S_AD4	113	S_AD5
54	GND	114	S_AD3
55	S_AD2	115	GND
56	S_AD0	116	S_AD1
57	+5 V	117	+5 V
58	N/C	118	N/C
59	+5 V	119	+5 V
60	+5 V	120	+5 V

NOTE:
1. 3.3 V pins are not supplied with 3.3 V power and are reserved for future use. Do not decouple 3.3 V pins to ground with 0.01 µF capacitors to provide AC return paths.

6.5 RIGHT-ANGLE MODULE EXTENDER CARD

For users developing their own IQ Modules, a right-angle module extender card is available from Intel (Intel P/N IQEXTENDER*)*. This card allows IQ Modules to be connected to the IQ-SDK platform at a right angle, and provides 0.025" posts for all PCI signals, simplifying the connection of a logic analyzer. The extender card is shown in Figure 6-3.

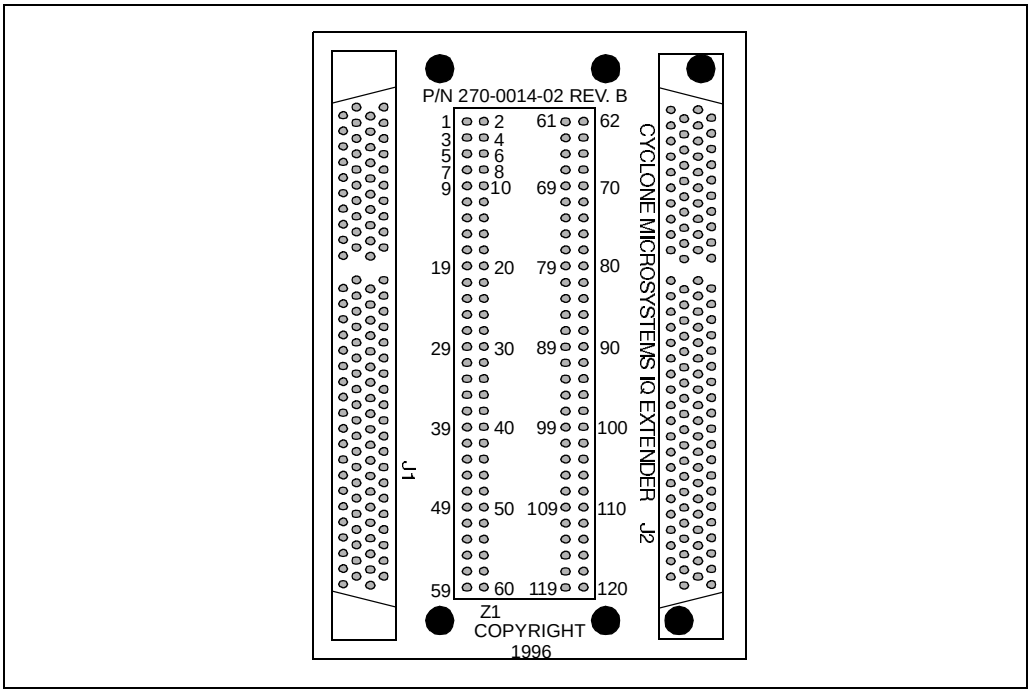

Figure 6-3. Module Extender Card Mechanical Drawing

A

PARTS LIST

APPENDIX A
PARTS LIST

This appendix identifies IQ-SDK platform components and quantities, component reference name as it appears on the PC board, description of size or rating, and the manufacturer's part number. To order replacement parts, contact the manufacturer listed in Table A-1. A manufacturer is not specified for those items that can be obtained from a variety of manufacturers.

Table A-1. IQ-SDK Platform Bill of Materials (Sheet 1 of 3)

Item	Qty	Reference	Description	Mfg. Part#	Manufacturer
1	2	U18, U20	IC	74ALS04	Motorola
2	1	U7	IC	74ABT273	Texas Instruments
3	5	U17, U10, U27, U29, U12	IC	74ABT573	Texas Instruments
4	8	U9, U11, U14, U15, U23, U24, U26, U28	IC	74ABT245	National Semiconductor
5	1	U6	IC	74ABT541	Texas Instruments
6	2	U13, U30	IC	QS3245SO	Quality Semiconductor
7	1	U16	IC	1488A	National Semiconductor
8	1	U8	IC	1489A	National Semiconductor
9	1	U5	PAL	PALCE16V8H-10JC	AMD
10	1	U25	80960RP-33 Processor	GC80960RP33	Intel
11	1	U2	UART	16C550 PLCC	Texas Instruments
12	2	U3, U4	Flash	28F020-150	Intel
13	2	U31, U22	Memory Module 256Kx32-70		
14	1	U21	EEPROM	24C08	Xicor
15	28	C1-C14, C18, C27-C29, C32-C36, C38, C40-C43	Capacitor 0.1UF		

Table A-1. IQ-SDK Platform Bill of Materials (Sheet 2 of 3)

Item	Qty	Reference	Description	Mfg. Part#	Manufacturer
16	12	C19-C25, C16, C31, C37, C39, C44	Capacitor 0.01 µF		
17	2	C26, C30	Capacitor 33 µF		
18	1	C17	Capacitor 4.7 µF		
19	1	C15	Capacitor 22 µF		
20	10	R33-R37, R62-R63, R66-R68	Resistor Pack 10 KΩ		
21	2	R46, R50	Resistor 1/8W, 5%, 1 KΩ		
22	18	R30-R32, R39-R42, R47, R51, R53, R57, R58, R61, R19-R20, R23, R64, R49	Resistor 1/8W, 5%, 10 KΩ		
23	1	R29	Resistor 1/8W, 1%, I5 Ω		
24	16	R14-R15, R17-R18, R21-R22, R24-R25, R48, R52, R54-R56, R59-R60, R65	Resistor 1/8W, 5%, 2.7 KΩ		
25	10	R3-R10, R12-R13	Resistor 1/8W, 5%, 470 Ω		
26	4	R1-R2, R11, R16	Resistor 1/8W, 5%, 4.7 KΩ		
27	7	R26-R28, R38, R43-R45	Resistor Pack 22 Ω		
28	1	J4	120-Pin SM Connector Plug		
29	2	U22, U31	72-Pin SIMM Connector	822134-3	AMP
30	1	J2	TJ6 6/6 LP Thru Hole Connector	GM-N-66	Kycon
31	1	J1	Hdr 16 Pin w/Shell Connector	103308	AMP
32	2	Z1, Z2	Jumper 2x1		
33	1	J8	Jumper 2x3		
34	1	J6	Jumper 2x4		
35	1	SW1	SM Switch DIP 4	DHS-4S	Apem Mors

Table A-1. IQ-SDK Platform Bill of Materials (Sheet 3 of 3)

Item	Qty	Reference	Description	Mfg. Part#	Manufacturer
36	1	U1	OSC 1.8432 MHz 1/2 THole	KHOHC1CSE	Kyocera
37	1	U19	Clock Chip	CY7B991-7J	Cypress
38	1	CR4	LED Green		
39	1	CR3	LED Red		
40	2	CR1, CR2	LED Red Small Group		
41	1	F1	Fuse Lamp		
42	1	U5	LP Socket		
43	2	U3, U4	LP Socket		

INDEX

P

R

S

T

U

V