Integrator™/AP

ASIC Development Motherboard

User Guide

ARM DUI 0098B

Integrator/AP
User Guide

Release Information

Description	Issue	Change
8 September 1999	A	New document
26 April 2001	B	New note added to FCC statement regarding conformance conditions. New subsections in *System bus* on page 3-3. These describe the system buses and signal routing. Address errors in Chapter 4 corrected. The section *Logic module region* on page 4-4 now to refers to distributed address decoding used by Integrator and requirement for modules to decode their own space. New subsections added to *EBI configuration registers* on page 4-19 to better describe EBI functionality. Appendix now contains ASB and AHB signals.

Proprietary Notice

Conformance Notices

This section contains conformance notices.

Federal Communications Commission Notice

This device is test equipment and consequently is exempt from part 15 of the FCC Rules under section 15.103 (c).

CE Declaration of Conformity

This equipment has been tested according to ISE/IEC Guide 22 and EN 45014. It conforms to the following product EMC specifications:

The product herewith complies with the requirements of EMC Directive 89/336/EEC as amended.

Confidentiality Status

This document is Open Access. This document has no restriction on distribution.

Product Status

The information in this documents is Final (information on a developed product).

Web Address

`http://www.arm.com`

Contents

Integrator/AP User Guide

Preface

About this document ... xiv
Further reading ... xvi
Feedback ... xviii

Chapter 1 Introduction

1.1 About the Integrator/AP ... 1-2
1.2 Integrator/AP system features ... 1-4
1.3 Connectors ... 1-9
1.4 LEDs ... 1-10
1.5 Test points ... 1-12

Chapter 2 Setting up the Integrator/AP

2.1 About setting up the Integrator/AP ... 2-2
2.2 Installing core modules and logic modules ... 2-3
2.3 Setting the DIP switches ... 2-6
2.4 Connecting power ... 2-7
2.5 Installing the Integrator/AP in a CompactPCI card rack ... 2-9
2.6 Using the boot monitor ... 2-10

Chapter 3 Hardware Description

3.1 System controller FPGA ... 3-2

3.2 System bus ... 3-3
3.3 External bus interface ... 3-11
3.4 Reset controller ... 3-12
3.5 Clock generator ... 3-15
3.6 Interrupt controller ... 3-19
3.7 Peripherals ... 3-22

Chapter 4 Programmer's Reference

4.1 About the Integrator memory map ... 4-2
4.2 System memory map regions ... 4-4
4.3 Accesses to boot ROM and flash ... 4-9
4.4 System control registers ... 4-10
4.5 EBI configuration registers ... 4-19
4.6 Counter/timer registers ... 4-23
4.7 Alphanumeric display, LED, and DIP switch registers ... 4-25
4.8 Interrupt controller registers ... 4-29
4.9 Peripheral registers ... 4-36

Chapter 5 PCI Subsystem

5.1 About the PCI subsystem ... 5-2
5.2 System to local bus bridge operation ... 5-6
5.3 V360EPC PCI to Host Bridge operation ... 5-9
5.4 PCI to PCI bridge operation ... 5-15
5.5 Initializing the PCI subsystem ... 5-16
5.6 PCI subsystem interrupts ... 5-17

Appendix A Connector Pinouts

A.1 Inter-module connectors HDRA and EXPA ... A-2
A.2 Core module connector HDRB ... A-5
A.3 Logic module connector EXPB ... A-8
A.4 Expansion module connector EXPM ... A-11
A.5 Serial interface connectors ... A-14
A.6 Keyboard and mouse connectors ... A-15

Appendix B Specifications

B.1 Mechanical details ... B-2
B.2 Electrical specification ... B-3
B.3 Timing specification ... B-4

Appendix C Interfacing to the System Bus

C.1 About the system bus ... C-2
C.2 Interfacing with the ASB system bus ... C-3
C.3 Interfacing to the AHB system bus ... C-5

List of Tables
Integrator/AP User Guide

ii

Table 1-1	Connector summary	1-9
Table 1-2	LED functional summary	1-11
Table 1-3	Test point functions	1-12
Table 2-1	DIP switch settings	2-6
Table 3-1	CFGSEL[1:0] encoding	3-5
Table 3-2	Rotated signal assignment	3-6
Table 3-3	Arbitration signal assignment	3-8
Table 3-4	EBI chip-select assignment	3-11
Table 3-5	Reset signal descriptions	3-13
Table 3-6	SYSCLK divider values	3-16
Table 3-7	P_CLK divider values	3-17
Table 3-8	UARTCLK divider values	3-18
Table 4-1	Top level Integrator/AP memory map	4-4
Table 4-2	Recommended logic module address decoding	4-4
Table 4-3	Aliased core module SDRAM mapping	4-5
Table 4-4	ROM, RAM, and peripherals region	4-6
Table 4-5	External bus interface chip selects	4-6
Table 4-6	Peripheral registers	4-7
Table 4-7	System controller status and control registers	4-10
Table 4-8	SC_ID register	4-11
Table 4-9	SC_OSC register	4-12
Table 4-10	System control register	4-13

Table 4-11	SC_DEC register	4-14
Table 4-12	SC_ARB register	4-15
Table 4-13	SC_PCI register	4-15
Table 4-14	SC_LOCK register	4-16
Table 4-15	SC_LBFADDR register	4-16
Table 4-16	SC_LBFCODE register	4-17
Table 4-17	System controller flag registers	4-18
Table 4-18	EBI configuration register locations	4-19
Table 4-20	EBI_CSRx default values	4-20
Table 4-19	EBI_CSRx register	4-20
Table 4-21	EBI_LOCK register	4-22
Table 4-22	Counter/timer registers	4-23
Table 4-23	TIMERx_CTL register	4-24
Table 4-24	LED control and switch registers	4-25
Table 4-25	LED_ALPHA bit assignment	4-26
Table 4-26	LED_ALPHA bit-to-segment mapping	4-26
Table 4-27	LED_LIGHTS register	4-28
Table 4-28	LED_SWITCH register	4-28
Table 4-29	IRQ register bit assignments	4-31
Table 4-30	IRQ register addresses	4-32
Table 4-31	FIQ registers addresses	4-33
Table 4-32	Software interrupt register	4-34
Table 4-33	Peripheral register locations	4-36
Table 4-34	GPIO register summary	4-36
Table 4-35	RTC register summary	4-38
Table 4-36	RTC_DR register	4-38
Table 4-37	RTC_MR register	4-39
Table 4-38	RTC_STAT/RTC_EOI registers	4-39
Table 4-39	RTC_LR register	4-39
Table 4-40	RTC_CR register	4-40
Table 4-41	UART register summary	4-41
Table 4-42	UARTx_DR register	4-42
Table 4-43	UARTx_RSR	4-43
Table 4-44	UARTx_LCRH register	4-44
Table 4-45	UARTx_LCRM register	4-45
Table 4-46	UARTx_LCML register	4-46
Table 4-47	Typical baud rates and divisors	4-46
Table 4-48	UARTx_CR register	4-47
Table 4-49	UARTx_FR register	4-49
Table 4-50	UARTx_IIR register	4-50
Table 4-51	KMI register summary	4-51
Table 4-52	KMI_CR register	4-52
Table 4-53	KMI_STAT register	4-53
Table 4-54	KMI_DATA transmit/receive data register read/write bits	4-54
Table 4-55	KMI_CLKDIV register read/write bits	4-54
Table 4-56	KMI_IR register	4-55
Table 5-1	PCI outbound access windows	5-10

Table 5-2	PCI inbound access windows	5-13
Table 5-3	PCI interrupts	5-17
Table 5-4	System bus fault codes	5-19
Table 5-5	PCI IDSEL and interrupt assignments for the PCI expansion bus	5-20
Table 5-6	PCI IDSEL and interrupt assignments for the CompactPCI bus	5-21
Table 5-7	PCI bus interrupt signal to interrupt controller assignment	5-21
Table 5-8	Interrupt to connector pin assignment	5-22
Table A-1	Bus bit assignment	A-3
Table A-2	HDRB signal description	A-6
Table A-3	EXPB signal description	A-9
Table A-4	EXPM signal description	A-11
Table A-5	Serial interface signal descriptions	A-14
Table A-6	Mouse and keyboard port signal descriptions	A-15
Table B-1	Electrical characteristics	B-3
Table B-2	Clock and reset parameters	B-4
Table B-3	AHB slave input parameters	B-5
Table B-4	AHB slave output parameters	B-5
Table B-5	Bus master input timing parameters	B-6
Table B-6	Bus master output timing parameters	B-6
Table B-7	AHB arbiter input parameters	B-6
Table B-8	AHB arbiter output parameters	B-7
Table B-9	Clock and reset parameters	B-7
Table B-10	ASB slave input parameters	B-8
Table B-11	ASB slave output parameters	B-8
Table B-12	Bus master input parameters	B-8
Table B-13	Bus master output parameters	B-9
Table B-14	ASB decoder input parameters	B-9
Table B-15	ASB decoder output parameters	B-9
Table B-16	ASB arbiter input parameters	B-10
Table B-17	ASB arbiter output parameters	B-10
Table B-18	ASB arbiter combinatorial parameters	B-10
Table C-1	CFGSEL[1:0] encoding	C-2

List of Figures
Integrator/AP User Guide

Figure 1-1	Integrator/AP (not to scale)	1-3
Figure 1-2	ARM Integrator/AP block diagram	1-5
Figure 1-3	LED locations	1-10
Figure 1-4	Test points	1-12
Figure 2-1	Assembled Integrator development system	2-3
Figure 2-2	Assembled Integrator/AP development system	2-4
Figure 2-3	DIP switches	2-6
Figure 2-4	Power supply connector	2-7
Figure 3-1	System controller FPGA functional block diagram	3-2
Figure 3-2	System bus architecture	3-4
Figure 3-3	Signal rotation scheme	3-7
Figure 3-4	Arbitration signal assignment to core and logic modules	3-8
Figure 3-5	Integrator/AP reset control	3-12
Figure 3-6	Clock generator block diagram	3-15
Figure 3-7	Interrupt controller architecture	3-19
Figure 3-8	nIRQ[3:0] signal routing	3-21
Figure 3-9	Counter/timer block diagram	3-22
Figure 3-10	RTC block diagram	3-24
Figure 3-11	Serial interface	3-26
Figure 3-12	KMI block diagram	3-28
Figure 4-1	Integrator memory map for core modules	4-2
Figure 4-2	Integrator memory map for logic modules	4-3
Figure 4-3	Core module alias memory mapping	4-5

Figure 4-4 Alphanumeric display segment designation 4-26
Figure 4-5 Interrupt control 4-29
Figure 4-6 GPIO direction control(1 bit) 4-37
Figure 5-1 PCI subsystem 5-3
Figure 5-2 System bus to PCI space mapping 5-4
Figure 5-3 System to local bus bridge transactions 5-6
Figure 5-4 PCI to host bridge 5-9
Figure A-1 Connector pin numbering A-2
Figure A-2 HDRB pin numbering A-5
Figure A-3 EXPB socket pin numbering A-8
Figure A-4 EXPM pin numbering A-11
Figure A-5 Serial interface connector pinout A-14
Figure A-6 KMI connector pinouts A-15
Figure B-1 Board outline B-2

Preface

This preface introduces the ARM Integrator/AP *ASIC development platform* and its reference documentation. It contains the following sections:

- *About this document* on page xiv
- *Further reading* on page xvi
- *Feedback* on page xviii.

About this document

This document provides a guide to how to set up and use the ARM Integrator/AP.

Intended audience

This document has been written for experienced hardware and software developers as an aid to using the ARM Integrator/AP as the basis of a the development platform for ARM-based products.

Organization

This document is organized into the following chapters:

Chapter 1 ***Introduction***

Read this chapter for an introduction to the ARM Integrator/AP. This chapter identifies the main components and connectors.

Chapter 2 ***Setting up the Integrator/AP***

Read this chapter for a description of how to set up and start using the Integrator/AP. This chapter describes how to attach modules to the Integrator/AP and how to apply power.

Chapter 3 ***Hardware Description***

Read this chapter for a description of the hardware architecture of the Integrator/AP. This includes clock control, resets control, interrupts control and peripherals.

Chapter 4 ***Programmer's Reference***

Read this chapter for a description of the system memory map and the system control, interrupt control, and peripheral registers.

Chapter 5 ***PCI Subsystem***

Read this chapter for a description of the PCI subsystem. This chapter provides details about the on-board PCI expansion and backplane interfaces

Appendix A ***Connector Pinouts***

Refer to this appendix for a description of the signals that appear on the card connectors.

Appendix B *Specifications*

Refer to this appendix for electrical, timing, and mechanical specifications.

Appendix C *Interfacing to the System Bus*

Refer to this appendix for information about the AMBA AHB and ASB buses as implemented on the Integrator/AP.

Typographical conventions

The following typographical conventions are used in this book:

`typewriter`	Denotes text that can be entered at the keyboard, such as commands, file and program names, and source code.
`typewriter`	Denotes a permitted abbreviation for a command or option. The underlined text can be entered instead of the full command or option name.
`typewriter italic`	Denotes arguments to commands and functions where the argument is to be replaced by a specific value.
italic	Highlights important notes, introduces special terminology, denotes internal cross-references, and citations.
bold	Highlights interface elements, such as menu names and buttons. Also used for terms in descriptive lists, where appropriate.
`typewriter bold`	Denotes language keywords when used outside example code and ARM processor signal names.

Further reading

This section lists related publications by ARM Limited and other companies that may provide additional information.

ARM publications

The following publications provide information about related ARM Integrator modules:

- *ARM Integrator/CM920T-ETM User Guide* (ARM DUI 0149)
- *ARM Integrator/CM940T, CM920T, CM740T, and CM720T User Guide* (ARM DUI 0157)
- *ARM Integrator/CM946E-S and CM966E-S User Guide* (ARM DUI 0138)
- *ARM Integrator/CM7TDMI User Guide* (ARM DUI 0126)
- *ARM Integrator/AM User Guide* (ARM DDI 0133)
- *ARM Integrator/LM-XCV600E and LM-EP20K600E User Guide* (ARM DUI 0146)
- *ARM Integrator/LM-XCV400+ User Guide* (ARM DUI 0130)

The following publications provide reference information about ARM architecture:

- *AMBA Specification* (ARM IHI 0011)
- *ARM Architectural Reference Manual* (ARM DDI 0100)
- *ARM PrimeCell UART (PL010) Technical Reference Manual* (ARM DDI 0139*)*
- ARM PrimeCell *RTC (PL030) Technical Reference Manual* (ARM DDI 0140)
- ARM PrimeCell *KMI (PL050) Technical Reference Manual* (ARM DDI 0143).

The following publication provides information about the ARM Firmware Suite:

- *ARM Firmware Suite Reference Guide* (ARM DUI 0102)

The following publications provide information about ARM SDT 2.5:

- *ARM Software Development Toolkit User Guide* (ARM DUI 0040)
- *ARM Software Development Toolkit Reference Guide* (ARM DUI 0041).

The following publications provide information about the ARM Developer Suite:

- *Getting Started* (ARM DUI 0064)
- *ADS Tools Guide* (ARM DUI 0067)
- *ADS Debuggers Guide* (ARM DUI 0066)
- *ADS Debug Target Guide* (ARM DUI 0058)
- *ADS Developer Guide* (ARM DUI 0056)
- *ADS CodeWarrior IDE Guid*e (ARM DUI 0065).

The following publication provide information about the Multi-ICE:

- *Multi-ICE User Guide* (ARM DUI 0048).

Other publications

The following publication provides information about the clock controller chip used on the Integrator modules:

- *MicroClock OSCaR User Configurable Clock Data Sheet* (MDS525), MicroClock Division of ICS, San Jose, CA.

The following publications provide information and guidelines for developing products for Microsoft Windows CE:

- *Standard Development Board for Microsoft® Windows® CE*, 1998, Microsoft Corporation
- *HARP Enclosure Requirements for Microsoft® Windows® CE*, 1998, Microsoft Corporation

Further information on these topics is available from the Microsoft website.

For further information about the PCI local bus and CompactPCI, consult the following websites:

- PCI Special Interest Group: `http://www.pcisig.com`
- PCI Industrial Computer Manufacturers Group: `http://www.picmg.com`

Feedback

ARM Limited welcomes feedback both on the ARM Integrator/AP, and on the documentation.

Feedback on this document

If you have any comments about this document, please send email to `errata@arm.com` giving:

- the document title
- the document number
- the page number(s) to which your comments refer
- an explanation of your comments.

General suggestions for additions and improvements are also welcome.

Feedback on the ARM Integrator/AP

If you have any comments or suggestions about this product, please contact your supplier giving:

- the product name
- an explanation of your comments.

Chapter 1
Introduction

This chapter introduces the Integrator/AP. It contains the following sections:

- *About the Integrator/AP* on page 1-2
- *Integrator/AP system features* on page 1-4
- *LEDs* on page 1-10
- *Test points* on page 1-12.

1.1 About the Integrator/AP

The Integrator/AP is an ATX form-factor motherboard that supports the development of applications and hardware for ARM processor-based products. It supports up to four processors on plug-in modules and provides clocks, bus arbitration, and interrupt handling for them. The Integrator/AP also provides operating system support with flash memory, boot ROM, and input and output resources.

The Integrator/AP can be expanded in the following ways:

- by fitting up to five plug-in core modules or logic modules
- by fitting up to three PCI expansion cards
- by installing the Integrator/AP in a CompactPCI card rack.

The Integrator/AP can be used in one of three ways:

- as a desktop development system
- in an ATX PC case
- in a CompactPCI card rack.

Figure 1-1 on page 1-3 shows the layout of Integrator/AP.

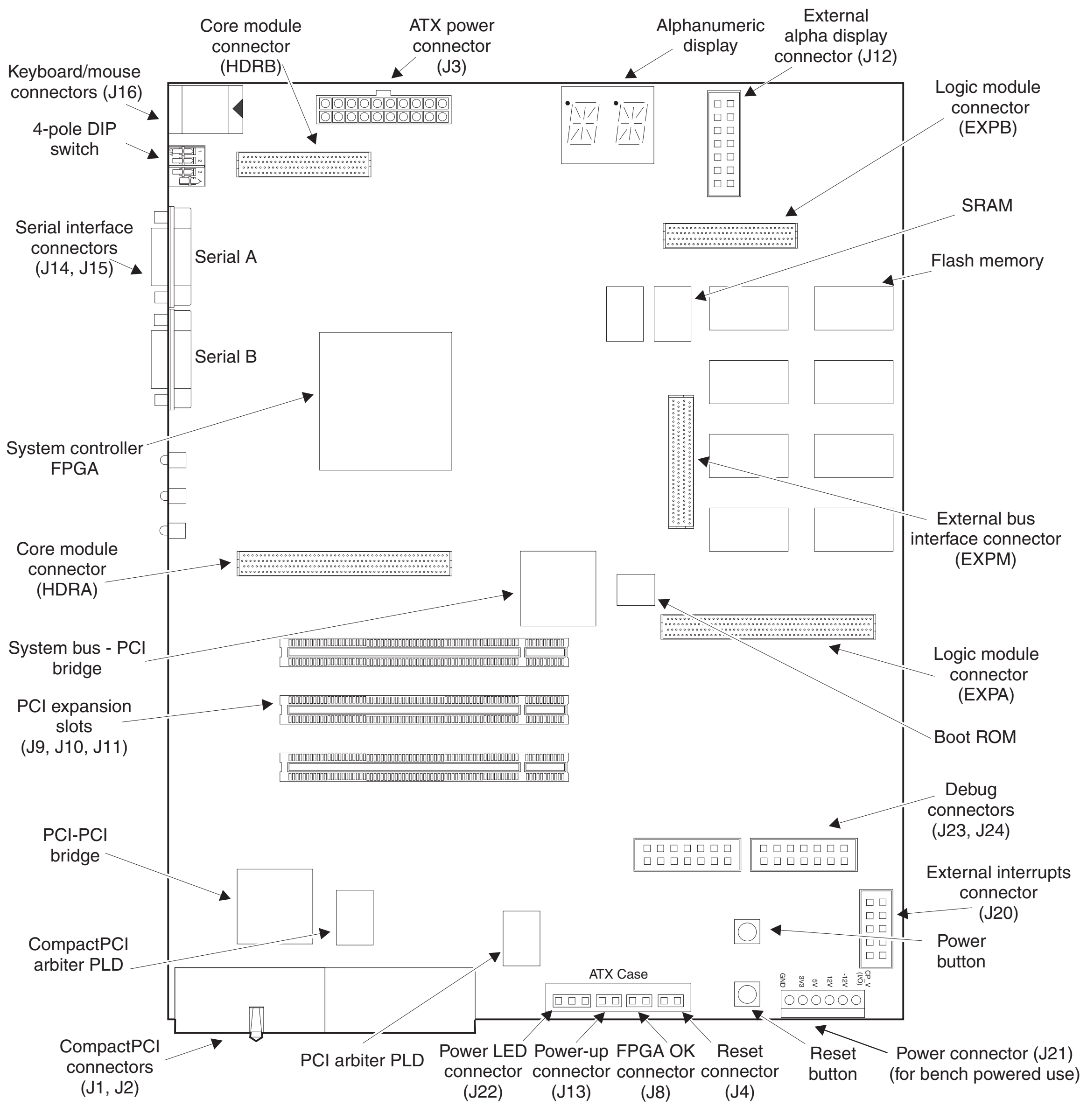

Core module connector (HDRB)
ATX power connector (J3)
Alphanumeric display
External alpha display connector (J12)
Keyboard/mouse connectors (J16)
Logic module connector (EXPB)
4-pole DIP switch
SRAM
Serial interface connectors (J14, J15)
Serial A
Flash memory
Serial B
System controller FPGA
External bus interface connector (EXPM)
Core module connector (HDRA)
System bus - PCI bridge
Logic module connector (EXPA)
PCI expansion slots (J9, J10, J11)
Boot ROM
Debug connectors (J23, J24)
PCI-PCI bridge
External interrupts connector (J20)
CompactPCI arbiter PLD
Power button
ATX Case
GND
3V3
5V
12V
-12V
CP_V (I/O)
CompactPCI connectors (J1, J2)
PCI arbiter PLD
Power LED connector (J22)
Power-up connector (J13)
FPGA OK connector (J8)
Reset connector (J4)
Reset button
Power connector (J21) (for bench powered use)

Figure 1-1 Integrator/AP (not to scale)

1.2 Integrator/AP system features

The major features on the Integrator/AP are as follows:

- system controller *Field Programmable Gate Array* (FPGA) that implements:
  - system bus interface to core and logic modules
  - system bus arbiter
  - interrupt controller
  - peripheral input and output controllers
  - three counter/timers
  - reset controller
  - system status and control registers
- clock generator
- 32MB flash memory
- 256KB boot ROM
- 512KB SSRAM
- two serial ports (RS232 DTE)
- system expansion, supporting core and logic modules (up to 5 in total)
- PCI bus interface, supporting expansion on-board or in a CompactPCI card rack
- *External Bus Interface* (EBI), supporting memory expansion.

1.2.1 System architecture

Figure 1-2 illustrates the architecture of the Integrator/AP.

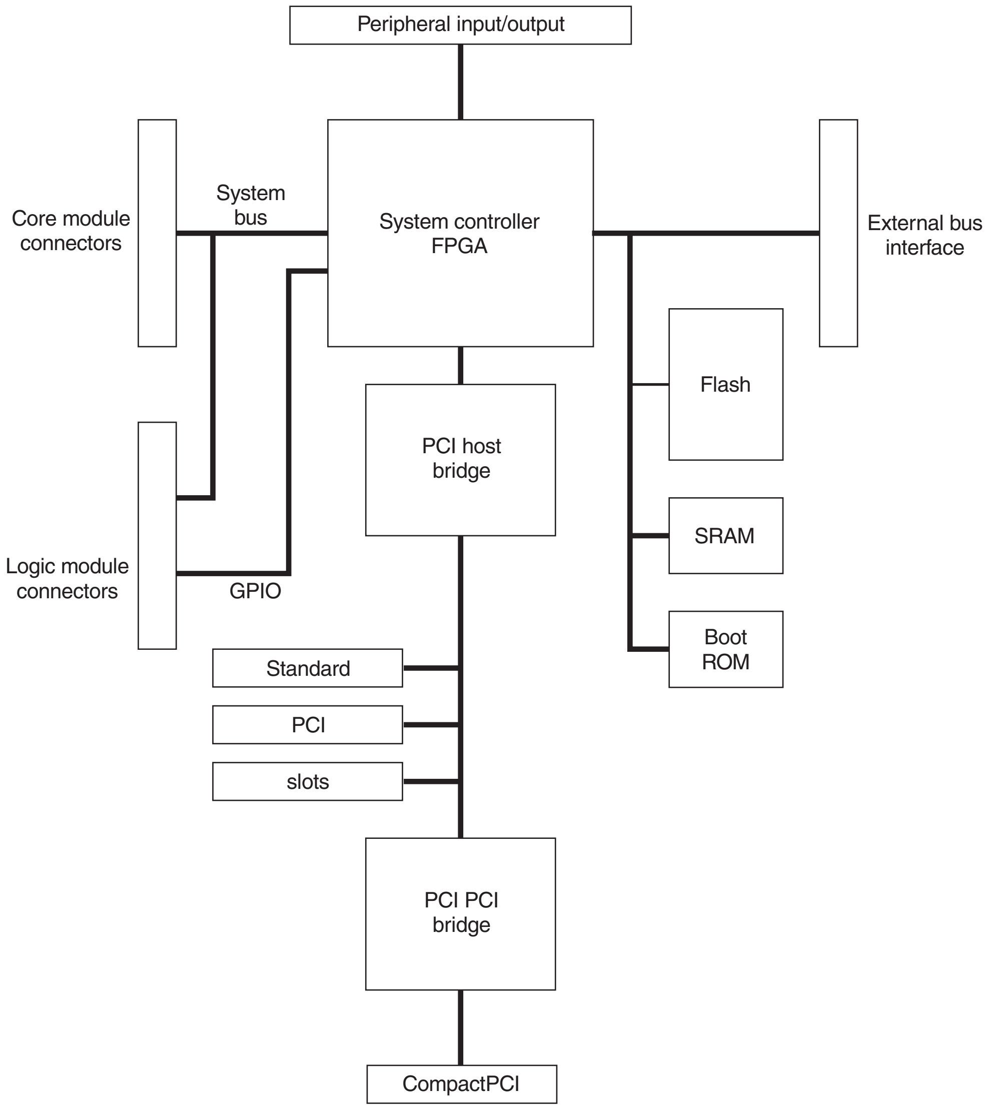


Figure 1-2 ARM Integrator/AP block diagram

1.2.2 System controller FPGA

The FPGA provides system controller functions for the Integrator/AP-based development system. These functions are outlined in this section and described in detail in Chapter 3 *Hardware Description*.

System bus interface

The system bus interface is incorporated into the system controller FPGA. It provides arbitration for Integrator/AP and any attached modules. The system bus interface also supports transfers to and from the:

- *Advanced Peripheral Bus* (APB) peripherals implemented within the FPGA
- PCI bus
- EBI.

System bus arbiter

The system bus arbiter provides bus arbitration for a total of six bus masters. These can include:

- up to five masters on core modules or logic modules
- PCI bus bridge.

The PCI bus bridge is the highest priority, with the remaining masters being allocated mastership on a round-robin arbitration scheme.

Interrupt controller

The interrupt controller handles IRQs and FIQs for up to four ARM processors. These originate from the peripheral controllers, from the PCI bus, and from devices on any attached logic modules. The interrupt controller allows interrupt requests from any of these sources to be assigned to any of the processors. Interrupts are enabled, acknowledged, and cleared using registers in the system controller FPGA.

Peripheral input/output controllers

The FPGA incorporates several peripheral devices. These include:

- two ARM PrimeCell UARTs (PL010)
- ARM PrimeCell *Keyboard and Mouse Interface* (KMI) (PL050)
- ARM PrimeCell *Real Time Clock* (RTC) (PL030)
- three 16-bit counter/timers
- GPIO controller
- alphanumeric display and LED controller, and switch reader.

The peripheral hardware is described in Chapter 3 *Hardware Description*, and programming information is provided in Chapter 4 *Programmer's Reference*.

Reset controller

The reset controller initializes the Integrator/AP when the system is reset. It allows the Integrator-based development system to be reset from several sources, including:

- push button reset switch
- PCI backplane
- under software control using register accesses
- from core or logic modules
- from Multi-ICE.

System status and control registers

The system controller status and control register space allows software configuration and control of the operation of the Integrator/AP. Controls include:

- clock speeds
- software reset
- flash memory write protection.

1.2.3 Clock generator

The Integrator/AP provides clock generators that supply clocks for:

- the system bus
- the UARTs
- the PCI sub-system
- the real-time clock, counter/timers, and KMI.

See *Clock generator* on page 3-15.

1.2.4 Memory

The memory on the Integrator/AP comprises:

- 256KB of boot ROM
- 32MB of 32-bit wide flash
- 512KB of 32-bit wide SSRAM

Reads from the flash memory, boot ROM, SSRAM, and external bus interface are controlled by the *Static Memory Interface* (SMI). The FPGA provides write-protection for the flash memory.

1.2.5 PCI bus interface

The Integrator/AP provides a V3 PCI host-bridge controller and a five-slot PCI arbiter. The on-board PCI interface provides three expansion connectors.

A PCI-PCI bridge, an additional eight-slot arbiter, and CompactPCI J1 and J2 connectors allow the card to be installed in a CompactPCI card cage.

1.2.6 System expansion

System expansion allows:

- up to four core modules to be stacked on the connectors HDRA and HDRB
- up to four logic modules to be stacked on the connectors EXPA and EXPB
- PCI expansion cards to be added to J9, J10, and J11.

Note

The combined total of logic and core modules that can be attached is five.

1.2.7 External bus interface

The Integrator/AP provides an external bus interface connector EXPM to allow additional memory to be added to the development system. Registers within the system controller are used to configure the interface to suit the application.

1.3 Connectors

This section provides a summary of the connectors on the Integrator/AP. These are illustrated in Figure 1-1 on page 1-3 and listed in Table 1-1.

Table 1-1 Connector summary

Legend	Function
J1 and J2	CompactPCI backplane connectors.
J3	PC ATX type power supply input.
J4	Reset. Can be connected to a panel mounted push button.
J5 and J6	Logic module connectors EXPA and EXPB.
J7	Expansion module connector EXPM.
J8	FPGA OK. Can be connected to a panel mounted LED to function as a System OK indicator.
J9, J10, J11	PCI bus expansion.
J12	Alphanumeric display extension. Can be connected to a panel mounted alphanumeric display. (Not fitted as standard.)
J13	Power button. Can be connected to a panel mounted push button.
J14 and J15	Serial channels A and B.
J16	Mouse (top) Keyboard (lower).
J18 and J19	Core module connectors HDRA and HDRB.
J20	Can be used to connect external interrupt sources. (Not fitted as standard.)
J21	Power supply input. Can be used to connect power from a bench power supply.
J22	Power LED. Can be connected to a panel mounted LED to function as a Power ON indicator.
J23	Test points for the PCI arbiter PLD. (Not fitted as standard.)
J24	Test points for the CompactPCI PLD. (Not fitted as standard.)

Connector pinouts and signal descriptions are provided in Appendix A *Connector Pinouts*.

1.4 LEDs

The Integrator/AP provides nine LEDs for status indication. These are illustrated in Figure 1-3.

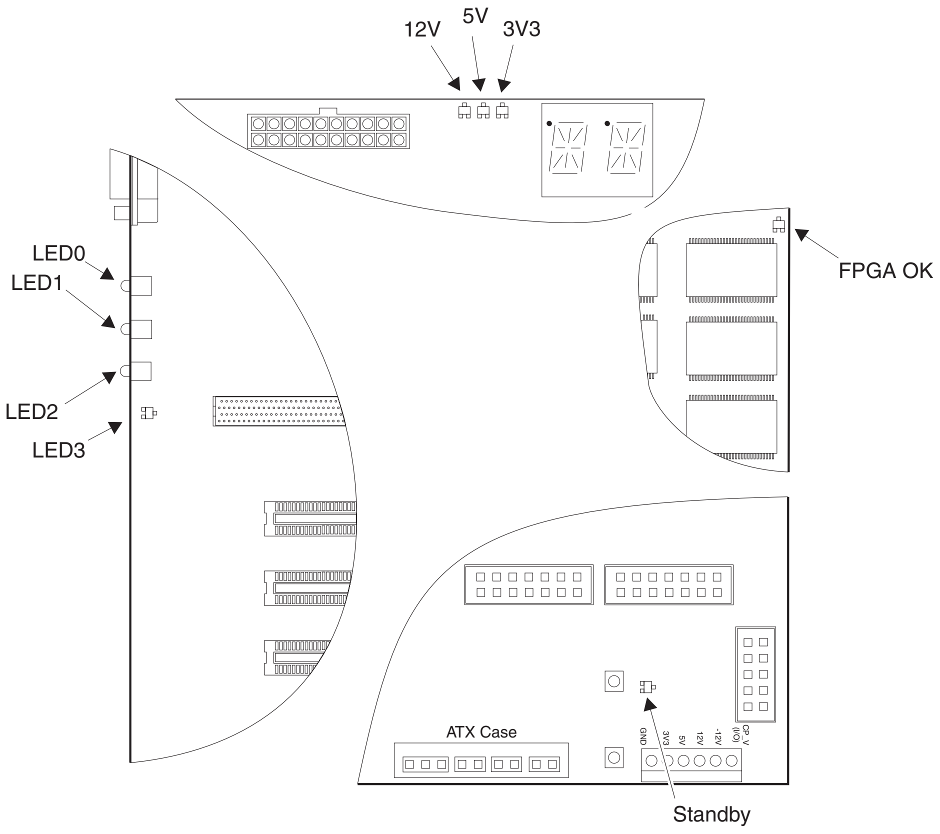


Figure 1-3 LED locations

1.4.1 LED functional summary

The function of the LEDs is summarized in Table 1-2.

Table 1-2 LED functional summary

LED	Color	Function
LED0	Green	This is a general purpose LED controlled by writing to bit 0 in the LED_LIGHTS register.
LED1	Yellow	This is a general purpose LED controlled by writing to bit 1 in the LED_LIGHTS register.
LED2	Red	This is a general purpose LED controlled by writing to bit 2 in the LED_LIGHTS register.
LED3	Green	This is a general purpose LED controlled by writing to bit 3 in the LED_LIGHTS register.
3V3	Green	Indicates that a 3.3V supply is available
5V	Green	Indicates that a 5V supply is available
12V	Green	Indicates that a 12V supply is available
STANDBY	Red	This LED illuminates when power is applied to the ATX power connector to warn that the power supply unit is in standby mode. To power on the Integrator/AP, press the POWER button.
FPGA OK	Green	This LED illuminates when the system controller FPGA has successfully loaded its configuration data following power on.

1.4.2 Power button

Press this surface-mounted button to power up the Integrator/AP when power is applied to the ATX power connector J3.

1.4.3 Reset button

Press the reset button to reset the Integrator/AP and any attached core and logic modules.

1.5 Test points

The Integrator/AP provides five test points as an aid to debug. These are illustrated in Figure 1-4.

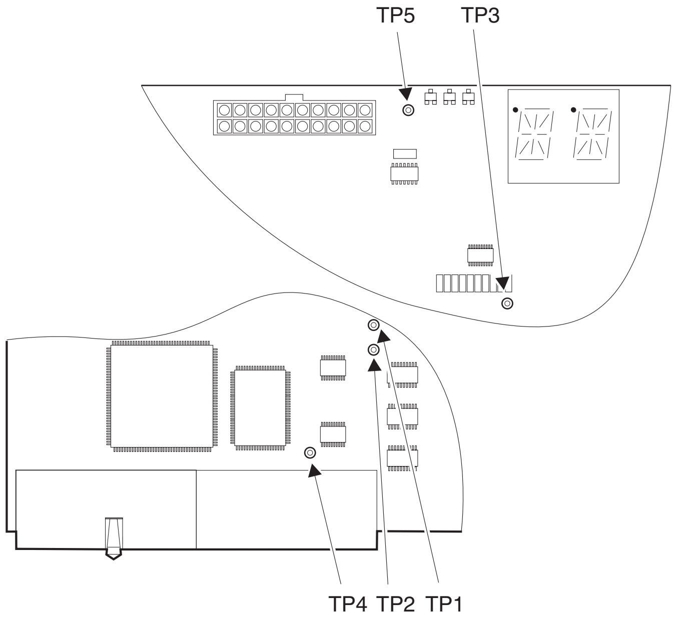


Figure 1-4 Test points

The function of the test points is summarized in Table 1-3.

Table 1-3 Test point functions

Test point	Signal	Function
TP1	**CLK24MHZ**	Crystal oscillator output
TP2	**UARTCLK**	UART clock input to the system controller
TP3	**SYSCLK**	System bus clock crystal oscillator output
TP4	**CP_CLK**	PCI subsystem clock crystal oscillator output
TP5	**nSYSRST**	Reset signal

Chapter 2
Setting up the Integrator/AP

This chapter describes how to set up and use the Integrator/AP. It contains the following sections:

- *About setting up the Integrator/AP* on page 2-2
- *Installing core modules and logic modules* on page 2-3
- *Setting the DIP switches* on page 2-6
- *Connecting power* on page 2-7
- *Installing the Integrator/AP in a CompactPCI card rack* on page 2-9
- *Using the boot monitor* on page 2-10.

2.1 About setting up the Integrator/AP

To set up the Integrator/AP carry out the following main steps:

1. Install a core module. Up to four core modules can be installed. See *Installing core modules and logic modules* on page 2-3.
2. Optionally, install a logic module. Up to four modules can be installed. See *Installing core modules and logic modules* on page 2-3
3. Set the DIP switches. See *Setting the DIP switches* on page 2-6.
4. Power the Integrator/AP by either:
   - connecting a bench power supply (for use as a bench-top development system), or
   - installing the Integrator/AP in a CompactPCI card rack, or
   - installing the Integrator/AP into a ATX PC case.

     Note

     The CompactPCI connectors make the Integrator/AP larger than the standard ATX footprint and prevents it being fitted into some ATX cases. See *Mechanical details* on page B-2.

   See *Connecting power* on page 2-7 and *Installing the Integrator/AP in a CompactPCI card rack* on page 2-9.
5. Optionally, connect a terminal or terminal emulator using a serial cable. See *Using the boot monitor* on page 2-10

2.2 Installing core modules and logic modules

An assembled Integrator/AP system is illustrated in Figure 2-1 showing the locations of the attached core modules and logic modules.

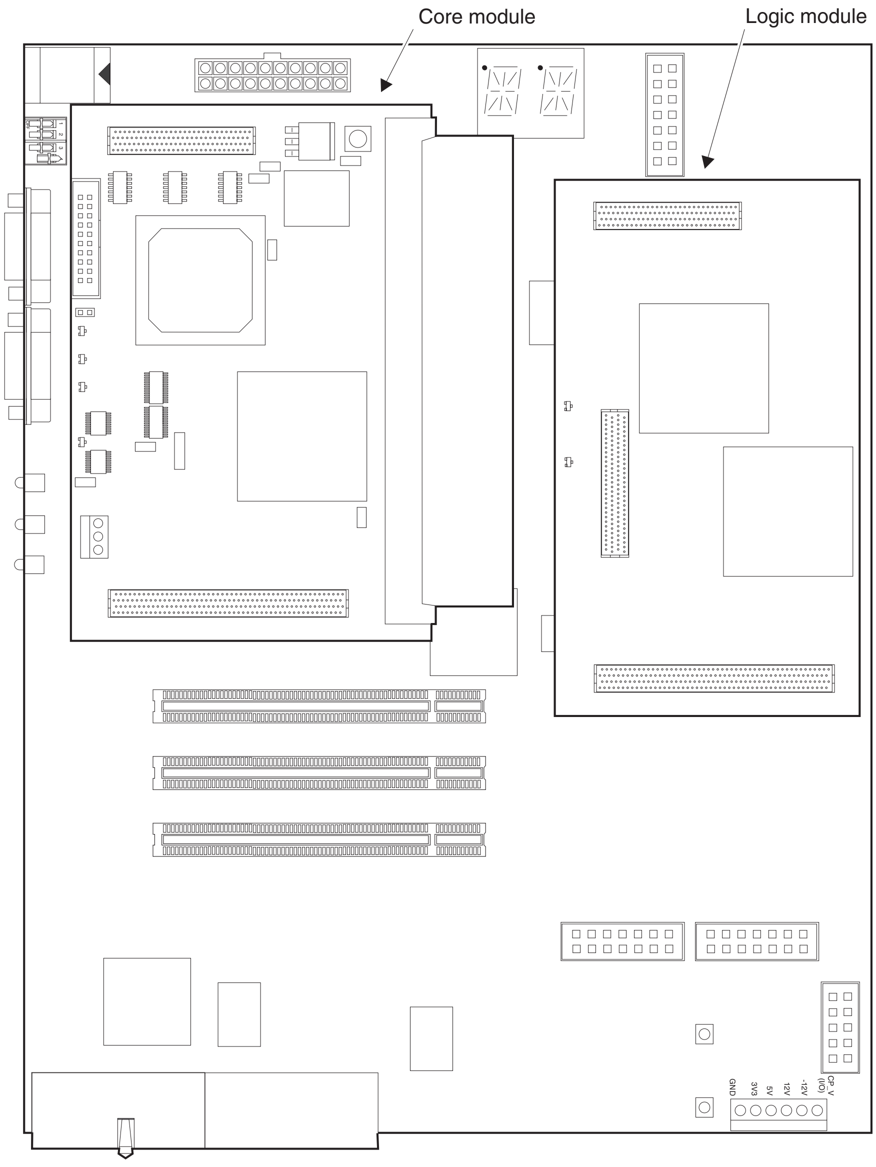


Figure 2-1 Assembled Integrator development system

The core modules, logic modules, and expansion cards mount onto the Integrator/AP as follows:

- core modules on the connectors HDRA and HDRB
- logic modules on the connectors EXPA and EXPB
- memory expansion module on the connector EXPM.

Figure 2-2 shows four core modules and one logic module attached to an Integrator/AP.

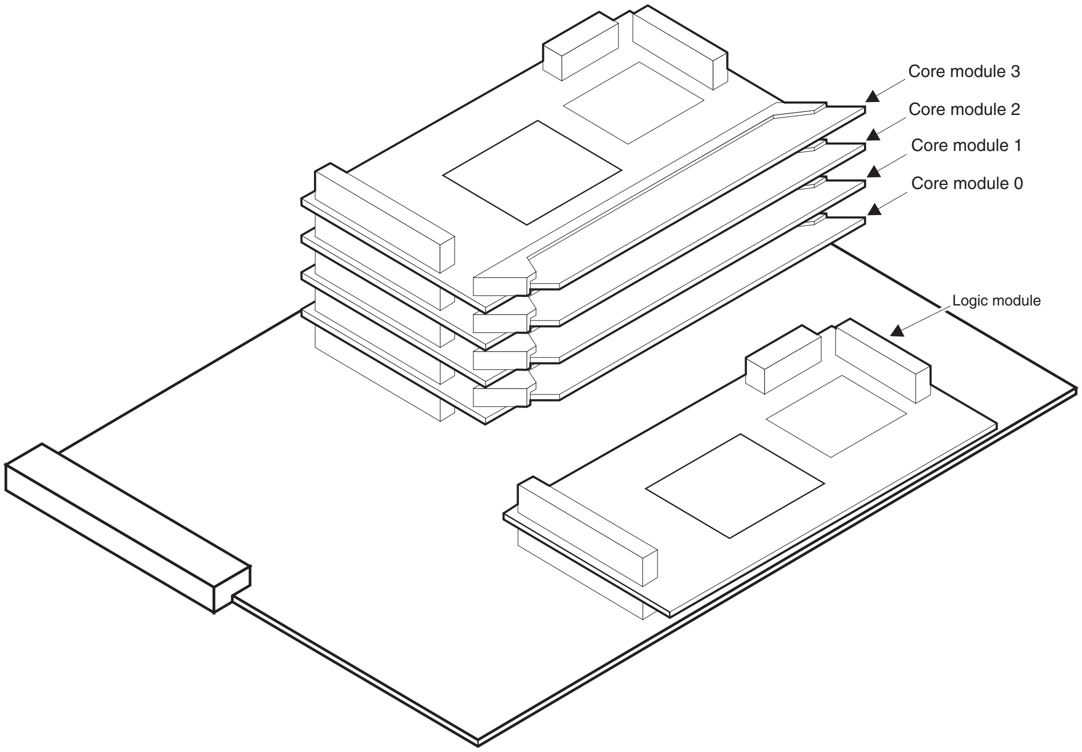


Figure 2-2 Assembled Integrator/AP development system

Note

When fitting core or logic modules to the Integrator/AP:

- do not exceed four core modules or logic modules in one stack
- do not exceed a combined total of five core modules or logic modules on the Integrator/AP
- fit at least one core module.

Fit a core module as follows:

1. Place the Integrator/AP on a firm level surface.
2. Align connectors HDRA and HDRB on core module with the corresponding connectors on the Integrator/AP.
3. Press firmly on both ends of the core module so that both connectors close together at the same time.
4. Repeat steps 2 and 3 for additional core modules.

——— **Caution** ———

To prevent damage to the Integrator/AP:

- Power down before fitting or removing modules.
- Check the connectors on the underside of each module before fitting it. Ensure that there are no blocked holes that would cause damage to the motherboard connectors.
- When removing a module, or when separating modules, take care not to damage the connectors. Do not apply a twisting force to the end of the connectors. Loosen each connector first before pulling on both ends of the module at the same time.
- Protect the Integrator/AP from *ElectroStatic Discharge* (ESD).

2.2.1 Module ID

The ID of each core module or logic module is set automatically when it is mounted on the motherboard (there are no links to set). A number of signals on the HDRB and EXPB connectors are routed to each module according to its mounting location and position in the stack. Core module 0 (zero) and logic module 0 are always nearest to the motherboard.

See the user guide for your core or logic module for a description of how these signals are used.

2.3 Setting the DIP switches

The 4-pole DIP switch (S1), illustrated in Figure 2-3, is used to configure the Integrator platform. The switch settings can be read from the LED and switch register (LED_SWITCHES) and can be assigned any meaning required by the system developer. However, at reset, S1[1] is assigned a special meaning by the hardware.

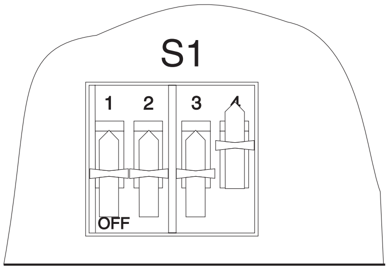


Figure 2-3 DIP switches

At reset S1[1] is used to control whether the boot ROM or flash is located address `0x0` and, therefore, where code execution begins. The switch settings are shown in Table 2-1 (where x = *don't care*).

Table 2-1 DIP switch settings

S1[1]	S1[2]	S1[3]	S1[4]	Function
ON	x	x	x	Code starts execution from boot ROM following reset.
OFF	x	x	x	Code starts execution from flash following reset.

To communicate with the boot monitor, connect a terminal using a null-modem cable to serial port A (J14). See *Using the boot monitor* on page 2-10.

The boot monitor only runs if S1[1] is in the ON position. The boot switcher component of the ARM Firmware Suite then uses S1[4] to select whether to jump to flash or to remain in a loop polling for serial input. If S1[4] is set to OFF, the boot monitor jumps to the boot image in flash.

2.4 Connecting power

There are three options for powering the Integrator/AP:

- from a bench power supply using the screw terminals at J21
- from a standard ATX PC power supply using the connector J3
- from a CompactPCI backplane.

The first two of these options are illustrated in Figure 2-4.

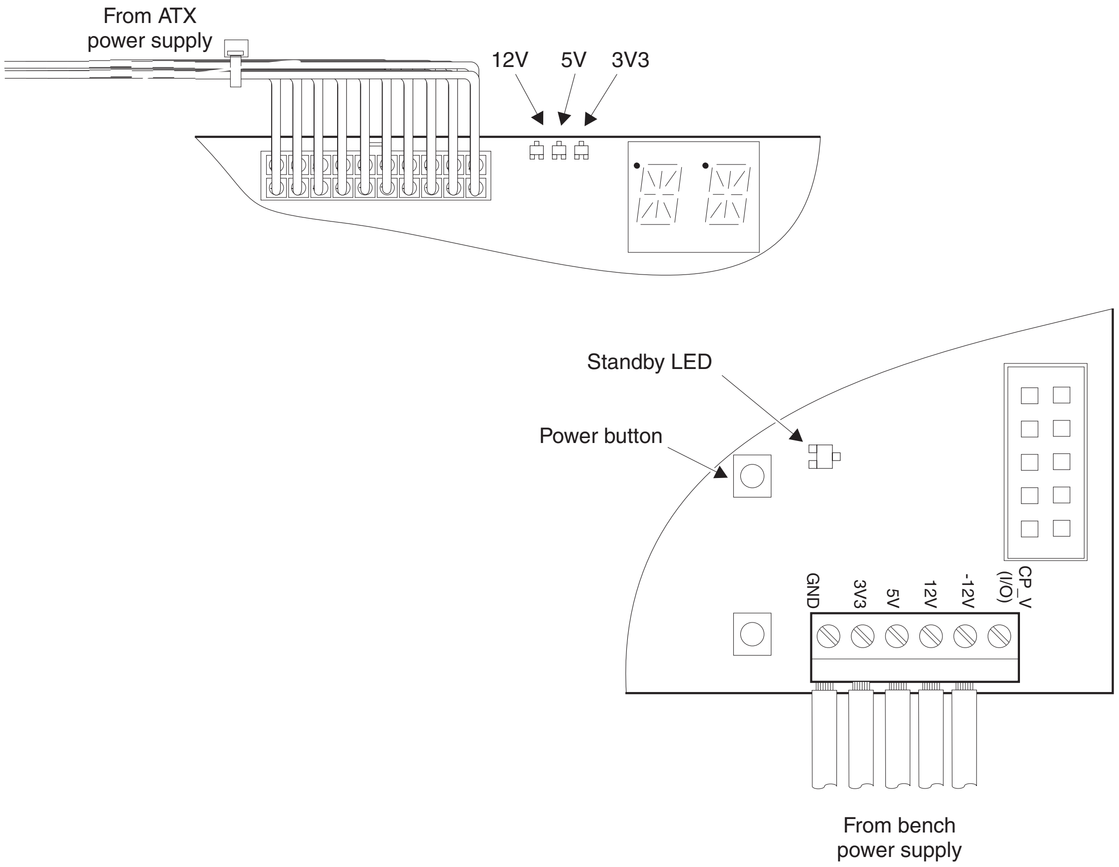


Figure 2-4 Power supply connector

The Integrator/AP and ARM Integrator core modules only require 3.3V and 5V. Connect the 12V and –12V supplies if they are required by a logic module or PCI card.

Power the Integrator/AP as follows:

Caution

Do not connect an external power supply to the **CP_V (I/O)** terminal. The Integrator/AP includes a circuit that provides 4V to **CP_V(I/O)** if the Integrator is not mounted in a CompactPCI card rack.

1. Connect a power supply, as illustrated in Figure 2-4 on page 2-7.
2. Apply the necessary AC supply to the power supply:

   If you are using a bench power supply, the Integrator/AP powers up.

   If you are using an ATX power supply, the standby LED illuminates. Press the power button to power the Integrator/AP development system up.

Caution

Care must be taken to ensure that the power supply is connected correctly because there is no reverse polarity protection provided on the board.

2.5 Installing the Integrator/AP in a CompactPCI card rack

The Integrator/AP is a CompactPCI system controller. When it is installed in a CompactPCI card rack, it must be installed in the system slot. This slot is located either at the left or right end of the backplane and is denoted by a triangle.

When the Integrator/AP is fitted with core modules and logic modules, it occupies up to three card slots (12HP).

——— Caution ———

Do not connect a power supply to J3 or J21 if the Integrator is installed in a CompactPCI card rack.

——— Note ———

The Integrator/AP is longer than a standard CompactPCI card and so protrudes from a card rack.

2.6 Using the boot monitor

The Integrator/AP is shipped with a boot monitor pre-programmed into the boot ROM. This section provides an overview of the boot monitor. A detailed description of the boot monitor is given in the *ARM Firmware Suite Reference Guide*.

This monitor allows the user to:

- load images into RAM and flash memory
- specify the flash image to boot
- run system self tests
- set system clock frequencies
- initialize and interrogate the PCI subsystem.

Use serial port A (J14) to communicate with the boot monitor. Connect a standard null modem cable between the Integrator/AP and a terminal (or a PC running a terminal emulator). The serial port settings are:

- 38400 baud
- no parity
- 8 bits
- 1 stop bit
- Xon/Xoff software handshaking.

2.6.1 System startup

When the Integrator platform is powered up, a message similar to the following is displayed on the terminal:

```
ARM bootPROM [Version 1.2] Rebuilt on Sep 20 2000 at 13:51:50
Running on a Integrator Evaluation Board
Board Revision V1.0, ARM720T Processor
Memory Size is 32MBytes, Flash Size is 32MBytes
Copyright (c) ARM Limited 1999 - 2000. All rights reserved.
Board designed by ARM Limited
Hardware support provided at http://www.arm.com/

For help on the available commands type ? or h
boot Monitor >
```

Type ? to display a list of the available commands. A menu is displayed:

```
ARM bootPROM [Version 1.2] Rebuilt on Sep 20 2000 at 13:51:50
H: Display help
?: Display help
I: Identify this system
B: Set Baud Rate
D: <hex> Display memory
```

```
V: Validate flash contents
BI: Set flash boot image number
E: Erase all of the application flash
T: Run the system self tests
L: Load a Motorola S-Record image into flash
M: Load a Motorola S-Record image into memory and run it

X: enter board specific command mode
```

This list of commands is generic to a range of boards. To display board-specific commands, type the following commands (commands can be entered in both uppercase and lowercase):

```
boot Monitor > x
[Integrator] boot Monitor > ?
```

The following menu is displayed.

```
ARM bootPROM [Version 1.2] Rebuilt on Sep 20 2000 at 13:51:52

?: Display help
H: Display help
I: (Re-)Initialise the PCI sub-system
V: Display V3 chip setup
P: Display PCI topology
DPI: <hex> Display PCI IO space (32 bit reads)
DPM: <hex> Display PCI Memory space (32 bit reads)
DPC: <hex> Display PCI Configuration space (32 bit reads)
SCC: Obsolete Command, use SC
SMC: Obsolete Command, use SC
SSC: Obsolete Command, use SC
SPC: Obsolete Command, use SC
SC: Set Clocks
CC: Set Clocks from SIB
DC: Display Clock Frequencies
DH: Display H/W (FPGA Versions etc.)
G: <hex> Goto address
PEEK: <hex> Display memory
POKE: <hex> <hex-value> Poke memory with value
MEM: Enable on chip memory
X: exit board specific command mode
```

2.6.2 System information block

The *System Information Block* (SIB) occupies the top sector of the flash memory and holds board-specific setup values. On the Integrator/AP the SIB is primarily used to store clock frequencies and program them when images are executed.

By default, the Integrator/AP and core module powers up with a set of frequencies that are suitable for a wide range of processor test chips. Use the `DC` command to list the default frequencies. To change the SIB settings, use the `SC` command to change the SIB settings. To change the board frequencies immediately to the SIB settings, use the `CC` command.

If S1[4] is in the OFF position when the board is reset, the clocks are changed to the SIB settings before the processor jumps to the bootable flash image. The clock frequency settings can be overridden at any time by an application that writes to the appropriate registers.

2.6.3 Built-in self tests

The Integrator/AP firmware provides a number of self-tests that allow you to verify correct operation. To run the self-tests, type `T` at the boot monitor prompt. For example:

```
[Integrator] boot Monitor > x
boot Monitor > t
Generic Tests
Type any character to abort the tests
Timer tests
  Running Timer tests
  ++++++++++
  Timer tests successful
LED flashing test
  Lighting all 4 LEDs in sequence
Did you see the LEDs flash in sequence[Yn]? y
...performed 2 tests, 0 failures
Board Specific Tests
Type any character to abort the tests
Keyboard/mouse tests

Initialising KMI interface
==========================

KMI: wrote FF
KMI: wrote FF
    Port 0: Device unsupported or absent
    Port 1: Device unsupported or absent

...performed 1 tests, 0 failures
boot Monitor >
```

2.6.4 Displaying FPGA version information

To display FPGA version information for the AP, core modules, and logic modules, enter the following command:

```
[Integrator] boot Monitor > dh

Core Modules
============
                                          ------ FPGA ------
CM Core       Arch  SSRAM  SDRAM  Bus   Type     Rev Build
-- ----       ----  -----  -----  ---   ----     --- -----
0  ARM720       4T   256K    32M  ASB   XC4036XL  B    09
1  ARM966    5TExP     1M      0  AHB   XCV600    B    19

System
======
                                          ------ FPGA ------
                    SSRAM  Flash  Bus   Type     Rev Build
                    -----  -----  ---   ----     --- -----
                     512K    32M  AHB   XC4085XL  B    23
```

2.6.5 Possible effect on remap of using a debugger

When the Integrator system is powered up, the boot ROM is mapped at address `0x00000000`. To access RAM at this address, you must remap the memory (see *Accesses to boot ROM and flash* on page 4-9).

However, occasionally when you use a debugger, the boot monitor does not run after power up and so does not remap the memory. If this happens, write to the appropriate core module register before loading code at address `0x00000000`.

Chapter 3
Hardware Description

This chapter describes the Integrator/AP on-board hardware. It contains the following sections:

- *System controller FPGA* on page 3-2
- *System bus* on page 3-3
- *External bus interface* on page 3-11
- *Reset controller* on page 3-12
- *Clock generator* on page 3-15
- *Interrupt controller* on page 3-19
- *Peripherals* on page 3-22.

3.1 System controller FPGA

The system controller FPGA provides comprehensive system control and interface functions, as illustrated in Figure 3-1. These include:

- system bus interface and arbiter for AHB or ASB
- *External Bus Interface* (EBI)
- PCI bridge local interface
- reset controller
- clock rate registers
- interrupt controller
- three counter/timers
- two ARM PrimeCell UARTs (PL010)
- ARM PrimeCell *Real Time Clock* (RTC) (PL030)
- ARM PrimeCell *Keyboard and Mouse Interface* (KMI) (PL050)
- *General Purpose Input/Output* (GPIO) port
- LED driver and boot switch reader.

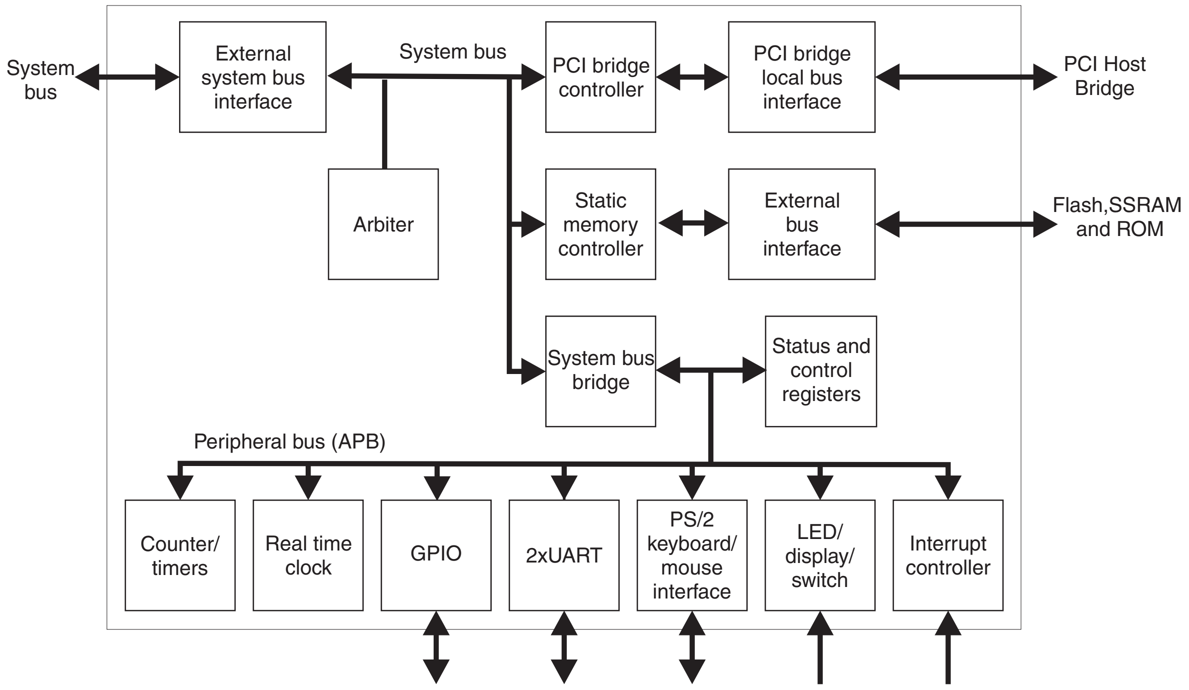


Figure 3-1 System controller FPGA functional block diagram

3.2 System bus

This section describes the system bus as it is implemented on Integrator/AP and contains the following:

- *System bus description*
- *System bus configuration* on page 3-4
- *Module-assigned signal routing* on page 3-5
- *Bus arbitration* on page 3-7
- *JTAG signal routing* on page 3-10.

3.2.1 System bus description

The HDRA/HDRB and EXPA/EXPB connector pairs are used to connect the system bus between the AP and other modules. Normally, the modules connect to the motherboard as follows:

- core modules on the connectors HDRA and HDRB
- logic modules on the connectors EXPA and EXPB.

Note

If you are implementing a processor core or DSP in a logic module and need to connect a debugger through the Multi-ICE server then you are advised to mount the logic module on the core module stack.

Figure 3-2 on page 3-4 shows the main buses on the motherboard.

There are three main system buses (labeled **A[31:0]**, **C[31:0]**, and **D[31:0]**) routed between system controller FPGA on the AP and the FPGAs on core and logic modules. In addition, there is a fourth bus **B[31:0]**) routed between the HDRA and EXPA connectors. These buses have the following functions:

A[31:0] This is the address bus and is connected between the system controller FPGAs on the AP and the FPGAs on each module.

B[31:0] This bus does not connect to the system controller FPGA on the AP, but only connects HDRA to EXPA. Logic modules and core modules that use Virtex FPGAs and also have pins connected to this bus.

The **B[31:0]** signals are reserved for future use.

C[31:0] The upper 10 signals on this bus **C[31:22]** are reserved. The remaining signals are used to implement a system control bus, as described in *Inter-module connectors HDRA and EXPA* on page A-2.

D[31:0] This is the data bus and is connected between the FPGAs on the motherboard and on each module. The AHB on Integrator differs from the AMBA standard in that it uses a bidirectional bus **HDATA** rather than **HWDATA** and **HRDATA** (see *Tristate AHB implementation* on page C-6). Within the FPGAs, the bidirectional bus is split so that standard AHB masters and slaves can be implemented.

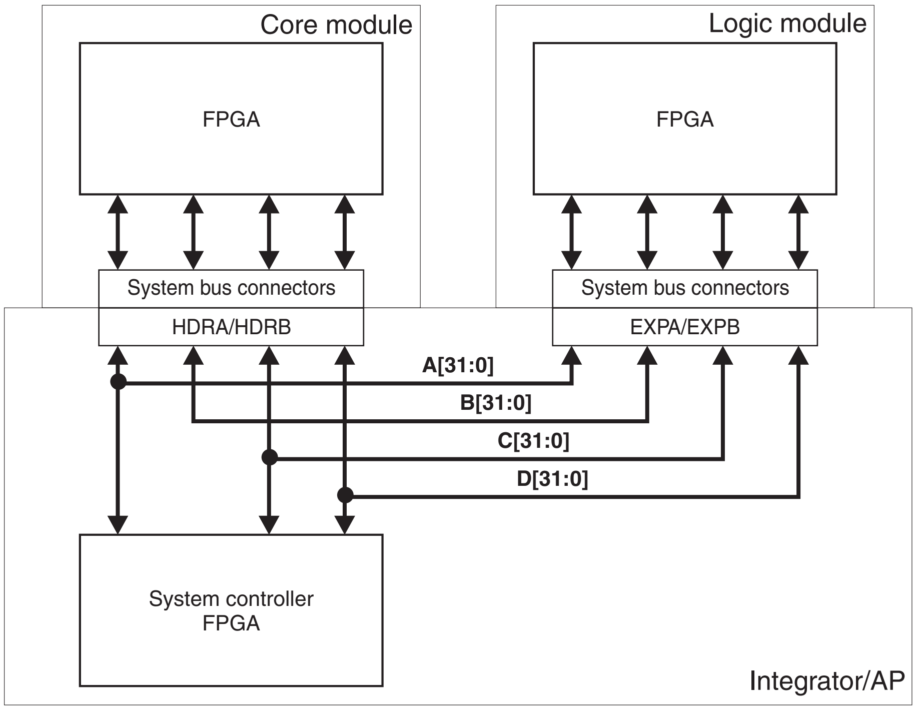


Figure 3-2 System bus architecture

Note

In addition to the buses described in this section the AP provides a 32-bit *General Purpose Input/Output* (GPIO) bus controlled by the GPIO peripheral in the system controller FPGA (see *GPIO* on page 3-29). This is routed to the EXPB connector only.

3.2.2 System bus configuration

The system bus is routed between FPGAs on core and logic modules and the AP. This enables the Integrator to support both of the AHB and ASB bus standards. At reset, the FPGAs are programmed with a configuration image stored in a flash memory device.

On the AP, the flash contains one image that configures the AP for operation with either an AHB or ASB system bus. On core and logic modules, the flash can contain multiple images so that the module can be configured to support either AHB or ASB.

Core and logic modules can be configured automatically using the static configuration select signals **CFGSEL[1:0]** from AP to select the correct image. The encoding of these signals is shown in Table 3-1.

Table 3-1 CFGSEL[1:0] encoding

CFGSEL[1:0]	Description
00	Little endian ASB
01	Reserved
10	Little endian AHB
11	Reserved

The system bus configuration is indicated by a character displayed on the alphanumeric display. This is `S` for ASB or `H` for AHB.

To change the system bus supported by the AP, use Multi-ICE and the progcards utility (which is on the CD-ROM supplied with the AP) to reprogram the FPGA image. However, if you do change the bus, you must also reprogram the CPCI arbiter PLD because it supplies the **CFGSEL[1:0]** signals.

Note

Earlier versions of the system controller FPGA do not display the bus type on the alphanumeric display and the bus type may not be displayed if any connected module does not support the bus used by the AP. It may be possible to update the module or AP to support the required bus (see the ARM website for later versions of the configuration images).

3.2.3 Module-assigned signal routing

Some of the signals on HDRB and EXPB are assigned to specific modules and are routed in a special way between the plug and socket on all core and logic modules to achieve the correct assignment. These signals are rotated in groups of four up through the stack to enable the motherboard to identify each module, to modify how the address decoder and bus arbiter function, and to correctly route interrupts.

The assignment of some of the signals is implemented in slightly different ways for core modules and logic modules. For example, core modules connect to **SREQ[3:0]** with **SREQ0** connected to the core at the bottom of the stack. Logic modules connect to **SREQ[1:4]** with **SREQ4** connected to the module at the bottom of the stack.

The rotated signals are shown in Table 3-2.

Table 3-2 Rotated signal assignment

Generic name	AHB name	ASB name	Description
SREQ[3:0]	**HBUSREQ[3:0]**	**AREQ[3:0]**	System bus requests for core modules
SGNT[3:0]	**HGRANT[3:0]**	**AGNT[3:0]**	System bus grant for core modules
SLOCK[3:0]	**HLOCK[3:0]**	Unused	System bus lock for core modules
SREQ[1:4]	**HBUSREQ[1:4]**	**AREQ[1:4]**	System bus request for logic modules
SGNT[1:4]	**HGRANT[1:4]**	**AGNT[1:4]**	System bus grant for logic modules
SLOCK[1:4]	**HLOCK[1:4]**	Unused	System bus lock for logic modules
ID[3:0]			Module ID
nFIQ[3:0]			Fast interrupt request, one for each core (HDRB only)
nIRQ[3:0]			Interrupt request, one to each core module (HDRB only)
nIRQSRC[3:0]			Interrupt request, one from each logic module (EXPB only)
nPPRES[3:0]			Core module present (HDRB only)
nEPRES[3:0]			Logic module present (EXPB only)
SYSCLK[3:0]	**HCLK[3:0]**	**BCLK[3:0]**	System clocks (HDRB only)
SYSCLK[7:4]	**HCLK[7:4]**	**BCLK[7:4]**	System clocks (EXPB only)

An example of how this signal rotation scheme is implemented is shown in Figure 3-3 on page 3-7. This shows how the bus request signals (**SREQ[3:0]**) are routed through two core modules. Each module connects devices to its version of the signal at **SREQ0** (that is, the signal name used on the schematics for that module), and passes the other signals up the stack onto a different connector pin. This routing ensures that the request from each module is routed onto a specific bus request signal on the AP depending on where the module is in the stack.

In the example shown in Figure 3-3 on page 3-7, the bus request signals are routed as follows:

- Core module 0 connects its own version of **SREQ0** straight to **SREQ0** on the AP.
- Core module 1 connects its own version **SREQ0** to **SREQ1** on core module 0 and core module 0 connects **SREQ1** straight down to **SREQ1** on the AP.

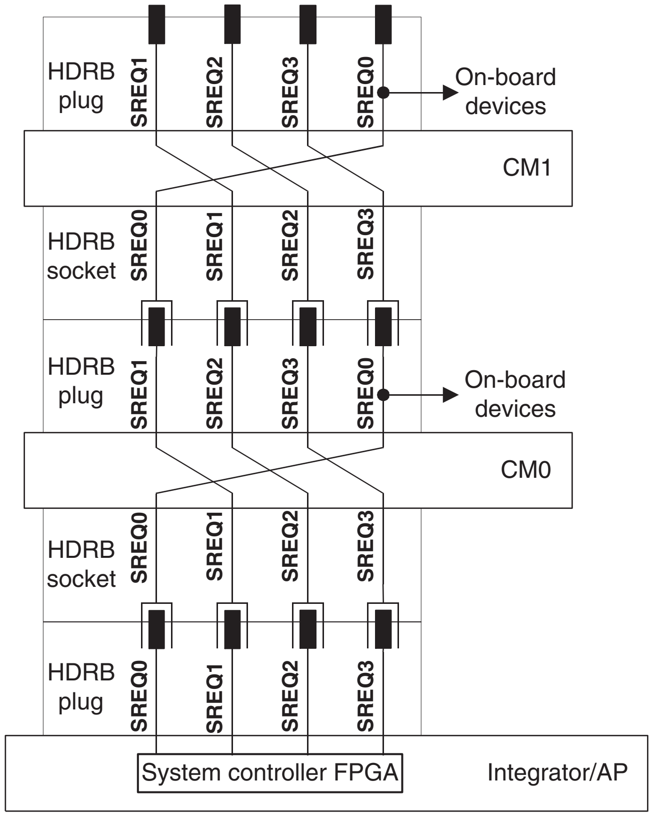


Figure 3-3 Signal rotation scheme

3.2.4 Bus arbitration

The system bus arbiter supports up to six bus masters, five of which can be core and logic modules. The arbitration signals are connected to the modules using the HDRB and EXPB connectors and are rotated up the stack in a similar way to other module-assigned signals as shown in Figure 3-4 on page 3-8 (see *Module-assigned signal routing* on page 3-5). The PCI bridge is always connected to **SREQ5**, **SGNT5**, and **SLOCK5** .

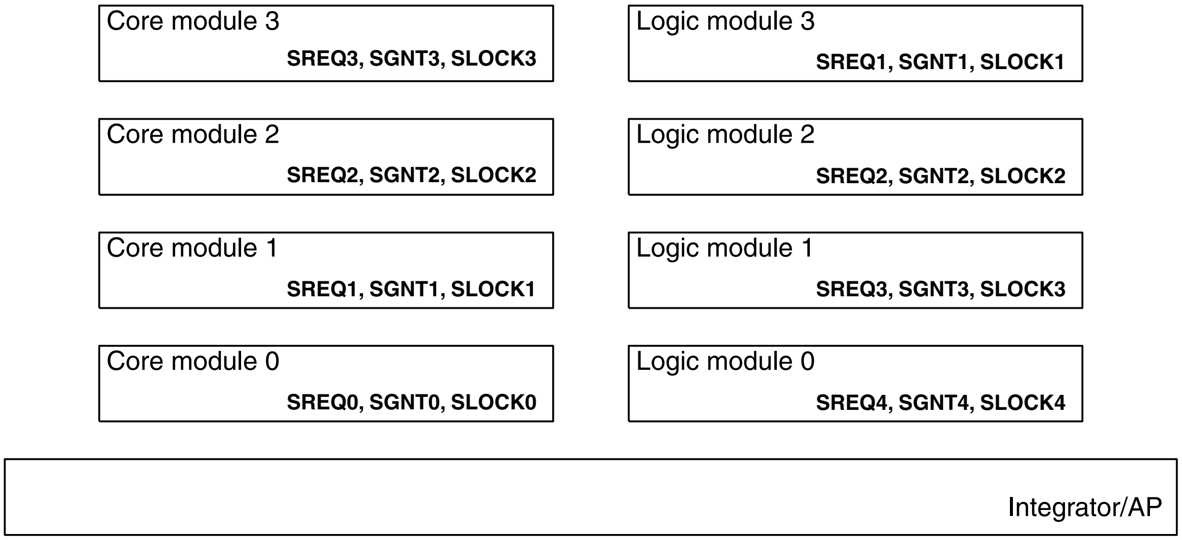


Figure 3-4 Arbitration signal assignment to core and logic modules

Note

The combined total of modules that can be attached to the AP is five. The assignment of signals for core and logic modules assumes that core modules are stacked on the HDRA/HDRB connectors and logic modules are stacked on the EXPA/EXPB connectors (see Appendix A *Connector Pinouts*).

Table 3-3 lists the arbitration signals and their assignment to the bus masters.

Table 3-3 Arbitration signal assignment

Master	Function	Signals
0	Core module 0 (bottom of core module stack)	**SREQ0**, **SGNT0**, **SLOCK0**
1	Core module 1 or logic module 3	**SREQ1**, **SGNT1**, **SLOCK1**
2	Core module 2 or logic module 2	**SREQ2**, **SGNT2**, **SLOCK2**
3	Core module 3 or logic module 1	**SREQ3**, **SGNT3**, **SLOCK3**
4	Logic module 0 (bottom of logic module stack)	**SREQ4**, **SGNT4**, **SLOCK4**
5	PCI bridge	**SREQ5**, **SGNT5**, **SLOCK5**

Arbitration priority scheme

The arbiter grants the PCI bridge the highest priority. For the remaining bus masters, the AP uses a round-robin arbitration scheme so that requesting bus masters can gain access in a balanced way. This is important in a development system, such as the AP, where it is impossible to predict the types of devices that might be added to the system and which of them should have priority.

The round-robin arbitration scheme ensures that all bus masters have equal chance to gain the bus and that a retracted master does not lock up the bus. The arbiter monitors the response signals to detect the end of an access and moves on when an ASB retract, or AHB split or retry occurs.

The arbiter grants the bus to a new master when:

- the current master ceases to request the bus
- a slave issues a retract, split, or retry response
- one of the arbitration counters times out.

Arbitration counters

The arbiter provides two programmable counters that allow bus operation to be tailored to specific system applications. These are:

- The *transaction counter* that defines the maximum number of transactions a bus master is allowed to keep the bus. The default is eight transactions, allowing a master to perform a burst of up to eight transfers before the arbiter reassigns the bus.
- The *cycle counter* that defines the number of bus clock cycles a bus master is allowed to hold the bus.

Both counters are clocked by the system bus clock (**SYSCLK**). See *Clock generator* on page 3-15.

You can program the transaction counter with any value between 0 and 31, where 0 means that the counter is disabled. The cycle counter can be programmed with any value between 0 and 4095 (0 and `0xFFF`) where 0 means the counter is disabled. If both counters are programmed, then the one that times-out first is used.

The transaction counter is used in a system where a bursting master must be allowed to retain the bus to complete a burst.

The cycle counter is used in a system where there is a real-time requirement.For example, to generate a timeout after 1μs when using a 20MHz bus clock (with a 50ns period), load the value 20 (1000/50=20) into the cycle counter.

The transaction counter and cycle counter are programmed using the SC_ARB register. See *Arbiter timeout register* on page 4-15.

3.2.5 JTAG signal routing

The **TDI**, **TDO**, **TMS**, **TCK**, **RTCK**, and **nRTCKEN** JTAG signals on HDRB and EXPB are separate. There is no connection between the two stacks.

The signal **nMBDET** is used by a module to detect when its fitted to a motherboard and to control the routing of the JTAG signals for each stack. This signal is tied LOW by the AP:

- When a module is attached to the motherboard, it detects that **nMBDET** is LOW and routes **TDI** and **TCK** down to the motherboard where they are looped back onto **TD0** and **RTCK**.
- When a module is used standalone, it detects that **nMBDET** is HIGH and provides loop backs for the JTAG signals itself so that the scan chain is intact.

The signal **nRTCKEN** is driven LOW by any module that implements a synthesized processor core, such as ARM7TDMI-S or ARM966E-S. Synthesized cores (on a core module or in FPGA on a logic module) must sample **TCK** and produce a time-delayed version of **TCK** called **RTCK** which is passed to the next device in the scan chain. Core modules that do not sample **TCK**, pass it down to the next module.

When Multi-ICE autoconfigures and detects transitions on **RTCK** it uses adaptive clocking. This means that Multi-ICE adapts to the speed of the target device. For non-synthesized processor cores this is unnecessary and so **RTCK** is driven LOW to ensure that Multi-ICE operates at the maximum **TCK** frequency of 10MHz. However, when connecting to multiple processors it might be necessary to reduce the Multi-ICE **TCK** frequency due to loading on **TMS**. 5MHz is recommended.

3.3 External bus interface

The EBI is a custom design for the AP that provides four fixed-size memory regions with separate chip-select lines. There are four chip selects:

- three are allocated to onboard devices
- one is available for system expansion.

The chip-select assignments are shown in Table 3-4.

Table 3-4 EBI chip-select assignment

Chip select	Size (bits)	Memory space
nMCS0	8	boot ROM.
nMCS1	32	Flash.
nMCS2	32	SSRAM.
nMCS3	8/16/32	Expansion memory space. The bus size and other parameters are set using the EBI_CSR3 register. See *EBI configuration registers* on page 4-19.

You can program theses memory spaces for:

- size (8, 16, or 32-bit)
- number of additional wait states
- synchronous or asynchronous operation
- write protection.

The EBI registers for the boot ROM, flash, and onboard SSRAM are set to default values at power on. However, you might have to modify these values if you change the system bus frequency. Each space is write protected by default. To write to flash or SSRAM, you must program the write enable bit for that region.

For details of how to program the EBI_CSR3 register for the expansion memory space, see *EBI configuration registers* on page 4-19.

Note

The memory expansion interface on the connector EXPM does not support synchronous SRAM.

3.4 Reset controller

A reset controller is incorporated into the system controller FPGA. The AP can be reset from a variety of hardware sources or in software using the CS_CTRL register.

3.4.1 Hardware resets

The hardware reset sources are as follows:

- push-button **PBRST** and CompactPCI signal **CP_PRST**
- ATX PSU power OK signal **nPW_OK** and CompactPCI power fail signal **CP_FAL**
- **FPGADONE** signal (routed through CPCI arbiter to become **nRSTSRC5**)
- logic modules using **nEXPRST**
- core modules (and Multi-ICE) using **nSRST**.

Figure 3-5 shows the architecture of the reset controller.

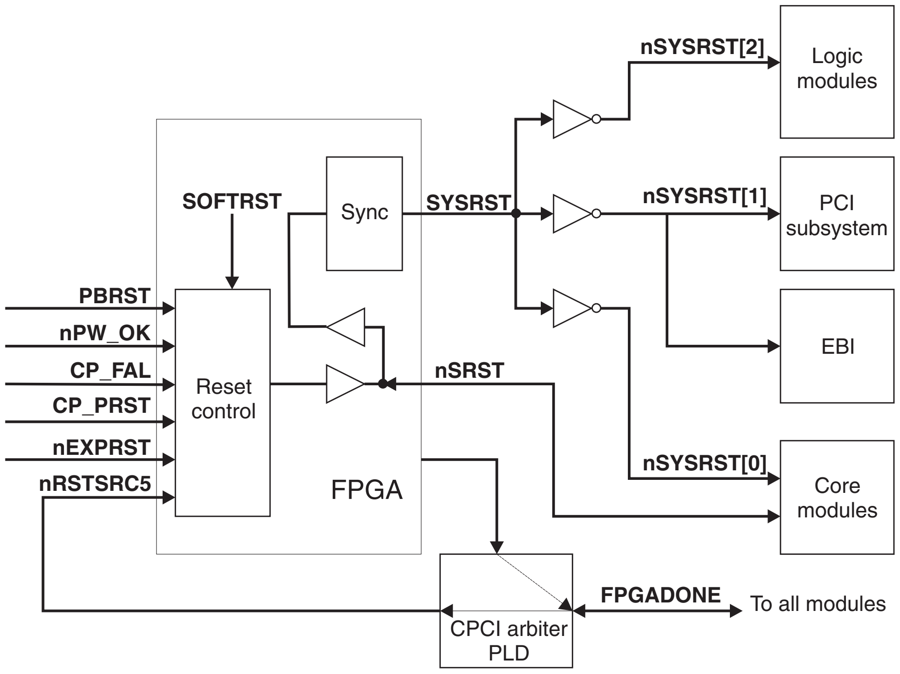


Figure 3-5 Integrator/AP reset control

When any of the reset inputs are asserted, the output **SYSRST** output is driven HIGH. This signal is routed through inverting buffers and drives the following:

- **nSYSRST[2]**, routed to logic modules
- **nSYSRST[1]**, PCI subsystem, EBI, and flash memory
- **nSYSRST[0]**, Core modules.

3.4.2 Reset signals descriptions

Table 3-5 describes the AP reset signals.

Table 3-5 Reset signal descriptions

Name	Description	Function
PBRST	Push-button reset (input)	The **PBRST** signal is generated by pressing the reset button on the AP.
nPW_OK	ATX power supply OK (input)	The **nPW_OK** input is supplied by an ATX power supply, if it is used. It is used to hold the system in reset until the power supply asserts its power OK output.
CP_FAL	CPCI power fail (input)	The **CP_FAL** input is asserted by a CompactPCI rack power supply, if one is in use.
CP_PRST	CPCI reset	The **CP_PRST** signal is generated by a push button on the CompactPCI rack, if one is in use.
nRSTSRC5	System-wide FPGA configured	The **nRSTSRC5** signal is an output from the CompactPCI arbiter PLD. It is used to hold the system in reset until all FPGAs in the system have completed their configuration sequence.
FPGADONE	FPGA configured (wire-AND output)	The **FPGADONE** signal is generated by all FPGAs when they have completed configuration following system power up. It is routed round the system through the HDRB and EXPB connectors from the outputs of all other FPGAs in the system.

Table 3-5 Reset signal descriptions (continued)

Name	Description	Function
nEXPRST	Expansion reset (open collector output)	The **nEXPRST** reset can be driven by a logic module. It must be debounced but does not have to be synchronized by **SYSCLK**.
nSRST	System reset (open collector, bidirectional)	The **nSRST** signal is generated by the core module FPGA when any of the of the reset inputs signals are asserted. It can also be driven by a core module or by Multi-ICE. When driven by a core module, it must be debounced but does not have to be synchronized by **SYSCLK**.
SYSRST	System reset (output)	The **SYSRST** signal is generated by the system controller FPGA and is used to generate the **nSYSRST[2:0]** signals that are routed to various modules within the system. **SYSRST** is asserted asynchronously to **SYSCLK** whenever any reset source is asserted. It is deasserted synchronously to **SYSCLK** when all reset sources are deasserted.

3.4.3 Software reset

The software reset is triggered by writing to the software reset bit in the SC_CTRL register. See *System control register* on page 4-13.

3.4.4 Multi-ICE reset

The reset signal, **nSRST**, can be used by Multi-ICE both to sense and drive the reset on the target system. This signal is routed between the Multi-ICE connector on the uppermost core module and the controller FPGA on each core module and the system controller FPGA on the motherboard.

3.5 Clock generator

The AP generates four clock signals as follows:

- system bus clock **SYSCLK**
- PCI subsystem clock **P_CLK** and CompactPCI **CP_CLK**
- UART clock **UARTCLK**
- KMI and timer clock **CLK24MHZ**.

These clocks are generated by three ICS525 devices, as shown in Figure 3-6.

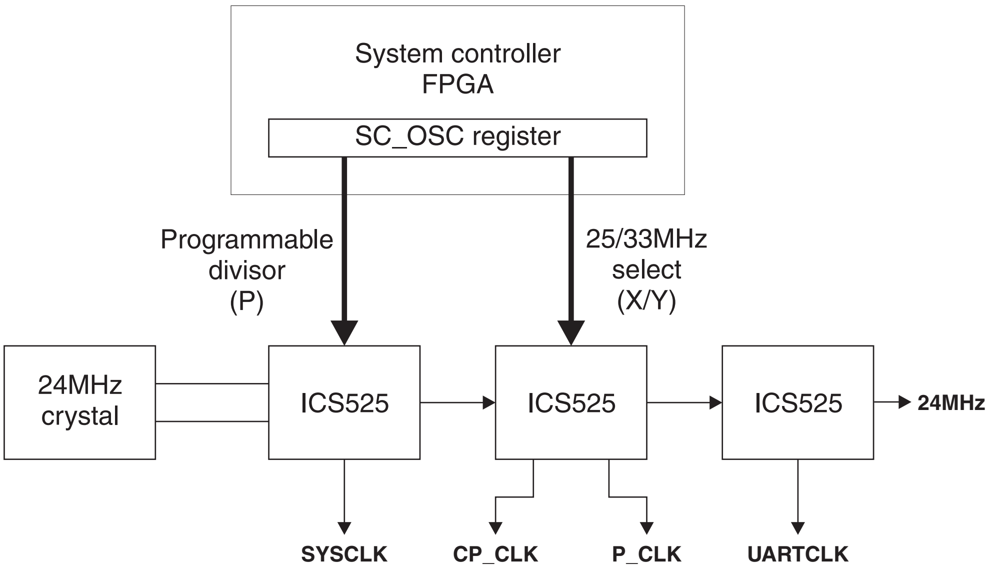


Figure 3-6 Clock generator block diagram

The ICS525 devices are *Phase-Locked Loop* (PLL) frequency generators that you can can configure with *divisor* inputs to produce a wide range of frequencies of between 1 and 160MHz. Each ICS525 contains:

- reference divider
- VCO divider
- output divider.

These contain several bits that are hard wired, and several that can be programmed by accessing the oscillator control register (SC_OSC) to select different output frequencies.

The three ICS525s and FPGA are supplied with a common 24MHz reference clock signal that is daisychained between the ICS525s and the FPGA.

3.5.1 System bus clock (SYSCLK)

The frequency of **SYSCLK** is controllable in 0.25MHz steps in the range 3MHz to 50MHz.

Setting the system bus clock

Table 3-6 shows the divisor settings used to set the frequency of the **SYSCLK** signal. The clock frequency is controlled by programming the VCO and output dividers for the **SYSCLK** generator using the SC_OSC register. The VCO divider is controlled by the S_VDW bits.

The output divider (OPDiv) and Reference Divider Word (S_RDW) are fixed. The bits marked:

- P can be programmed by writing to the S_VDW field in the SC_OSC register
- 1 are tied HIGH
- 0 are tied LOW.

Table 3-6 SYSCLK divider values

Clock speed	S_RDW[6:0]	S_VDW[8:0]	OPDiv [2:0]
3-50MHz in 0.25MHz steps	0 1 0 1 1 1 0	0 P P P P P P P P	0 1 1
	46(dec)	4 to 192(dec) (<4 and >192 not allowed)	4

Note

The bit pattern 011 in the OPDiv field selects an output divider of 4. That is, it is not a binary value.

The clock frequency is given by the formula:

freq(MHz) = (S_VDW+ 8)/4

where S_VDW is the VCO divider word for the system bus clock

The maximum speed of the clock generator exceeds the likely maximum reliable bus speed that can be used (see *Timing specification* on page B-4). For details about how to program S_VDW, see *Oscillator divisor register* on page 4-12.

Note

Values for S_VDW and S_OD can be calculated using the ICS525 calculator on the Microclock website.

System bus clock usage

The **SYSCLK** signal is buffered by a PI49FCT3807 low-skew buffer to drive ten loads as follows:

- **SYSCLK[9]** to the V3 PCI bridge
- **SYSCLK[8]** to the system controller
- **SYSCLK[7:4]** to logic module connector EXPB
- **SYSCLK[3:0]** to core module connector HDRB.

The clock signals to the core and logic modules are rotated up through the stacks to ensure that each module only connects to one clock and so balance the signal loading (see *Module-assigned signal routing* on page 3-5).

3.5.2 PCI clocks (P_CLK and CP_CLK)

The frequency of **P_CLK** and **CP_CLK** is controlled by the DIVX/Y bit in the SC_OSC register and can be set to either 25 or 33MHz. Table 3-7 shows divisor settings supplied to the **P_CLK** clock generator by the system controller FPGA in response to the setting of the DIVX/Y bit.

The bits marked:

- X are 0 for 25MHz and 1 for 33MHz
- Y are 1 for 25MHz and 0 for 33MHz.

Table 3-7 P_CLK divider values

Clock speed	P_RDW[6:0]							P_VDW[8:0]									OPDiv [2:0]		
25MHz	0	0	Y	X	1	1	0	0	0	0	0	Y	X	X	X	Y	0	0	1
	22(dec)							17(dec)									2		
33MHz	0	0	Y	X	1	1	0	0	0	0	0	Y	X	X	X	Y	0	0	1
	14(dec)							14(dec)									2		

Note

The bit pattern 001 in the OPDiv field selects an output divider of 2. That is, it is not a binary value.

The maximum speed of the clock generator exceeds the likely maximum reliable clock speed that can be used (see *Timing specification* on page B-4). For details about how to program the DIVX/Y bit, see *Oscillator divisor register* on page 4-12.

PCI bus clock usage

The **P_CLK** signal is buffered by a PI49FCT3807 low-skew buffer to drive seven loads. These are:

- **P_CLK[6]** to the CP_CLK buffer
- **P_CLK[5]** to the PCI bus arbiter
- **P_CLK[4]** to the PCI-PCI bridge
- **P_CLK[3:1]** to the PCI local bus expansion slots
- **P_CLK[0]** to the PCI-Host bridge controller.

The **CP_CLK** clock signal is buffered by a PI49FCT3807 low-skew buffer to drive nine loads. These are:

- **CP_CLK[8]** to the CompactPCI bus arbiter
- **CP_CLK[7]** to the PCI-PCI bridge
- **CP_CLK[6:0]** to the CompactPCI slots.

3.5.3 UART clock (UARTCLK)

The **UARTCLK** clock is generated at a fixed frequency of 14.7456MHz. It is supplied to the UARTs within the system controller FPGA to provide a 16x baud-rate clock allowing baud rates of up to 460,800 to be selected. Table 3-8 shows divisor settings for the **UARTCLK** generator.

Table 3-8 UARTCLK divider values

Clock speed	U_RDW[6:0]							U_VDW[8:0]									OPDiv [2:0]		
14.7456MHz	0	1	1	1	0	0	1	0	1	0	0	0	1	0	0	1	0	1	0
	57 (dec)							137 (dec)									8 (dec)		

Note

The bit pattern 010 in the OPDiv field selects an output divider of 8. That is, it is not a binary value.

3.5.4 KMI and timer clock (CLK24MHZ)

The **CLK24MHZ** clock is a fixed-frequency reference clock. It is divided down within the system controller FPGA to 8MHz by the KMI, and to 1Hz for the real-time clock.

3.6 Interrupt controller

The system controller FPGA contains four interrupt controllers. These take interrupts from the logic modules and from internal peripherals, and enable you to assign them to the IRQ or FIQ pins of any of up to four processors.

3.6.1 Interrupt architecture

Figure 3-7 shows the architecture of the interrupt controller.

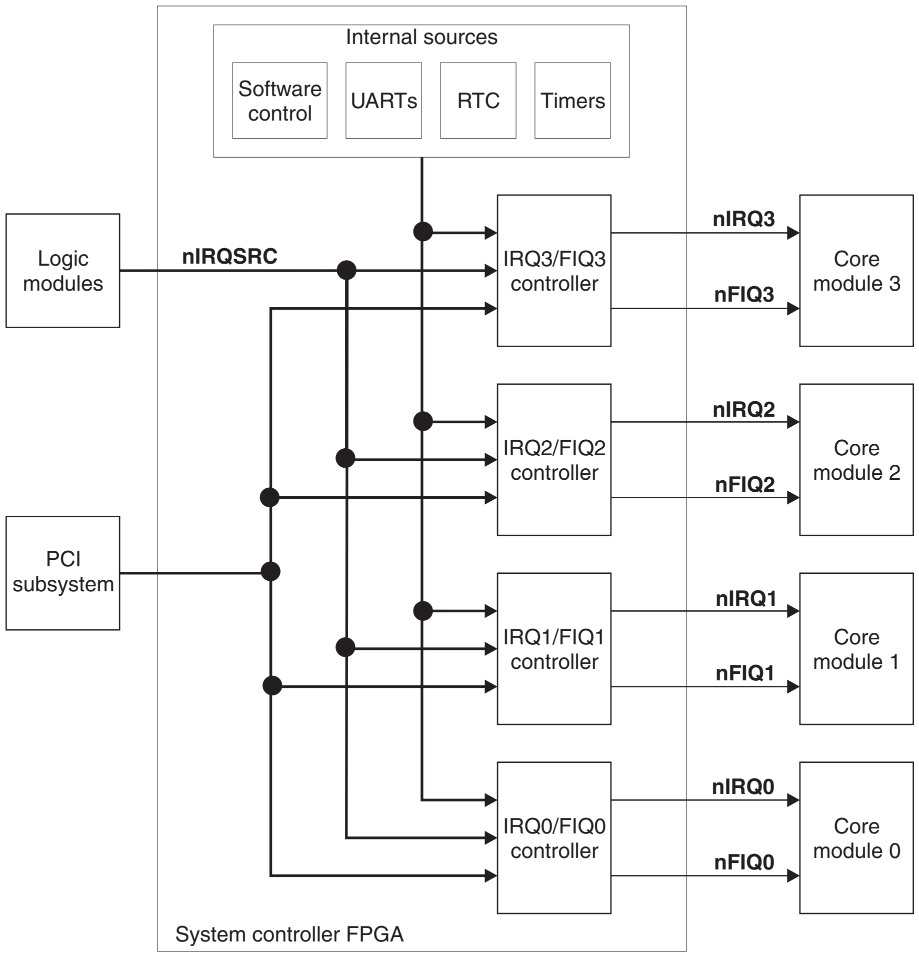


Figure 3-7 Interrupt controller architecture

The system controller incorporates a separate IRQ and FIQ controller for each core module. Each controller has 22 inputs, as shown in Table 4-29 on page 4-31. The four main sources of interrupts are:

- system controller internal peripherals
- peripherals or controllers on logic modules
- PCI subsystem
- software.

In addition to the IRQ and FIQ controllers for the individual processors, the system controller provides a shared software interrupt generator that can be used by the four processors to interrupt one another. See *Software interrupts* on page 4-34.

3.6.2 Interrupt signal routing

The IRQ and FIQ signals are routed to core modules so that each module is assigned to one IRQ and one FIQ. A signal rotation scheme is used on the HDRB and EXPB connectors to ensure that each module receives or issues interrupts on a specific IRQ and FIQ signal, depending on its position in the stack (that is, according to the ID of the module). This scheme is described in *Module-assigned signal routing* on page 3-5.

The routing of the interrupts means that core and logic modules treat interrupts in different ways. The differences are:

- core modules:
  - receive interrupts and must be mounted on the HDRA/HDRB connectors
  - receive interrupts on one IRQ signal and one FIQ signal each, as determined by their position in the stack.
- logic modules:
  - issue interrupts and must be mounted on the EXPA/EXPB connectors
  - issue interrupts on one **nIRQSRC** signal each, as determined by their position in the stack.

    ——— Note ———

    You can use logic modules to implement a synthesized core. In this case they are connected to the HDRB connector and are treated in the same way as any other core module.

    ————————

Figure 3-3 on page 3-7 shows how the interrupt request signals are routed upwards through the core module connectors and downwards through logic module connectors. Each module utilizes only its own version of **nIRQ0** (as identified by the schematic diagram for the module) and passes the **IRQ[3:0]** up or down to the next module in the stack. This means that CM0 is assigned to the signal **nIRQ0** from the AP, CM1 is assigned to **nIRQ1**, CM2 is assigned to **nIRQ2**, and CM3 is assigned to **nIRQ3** from the AP. The **nFIQ[3:0]** signals are routed in a similar way.

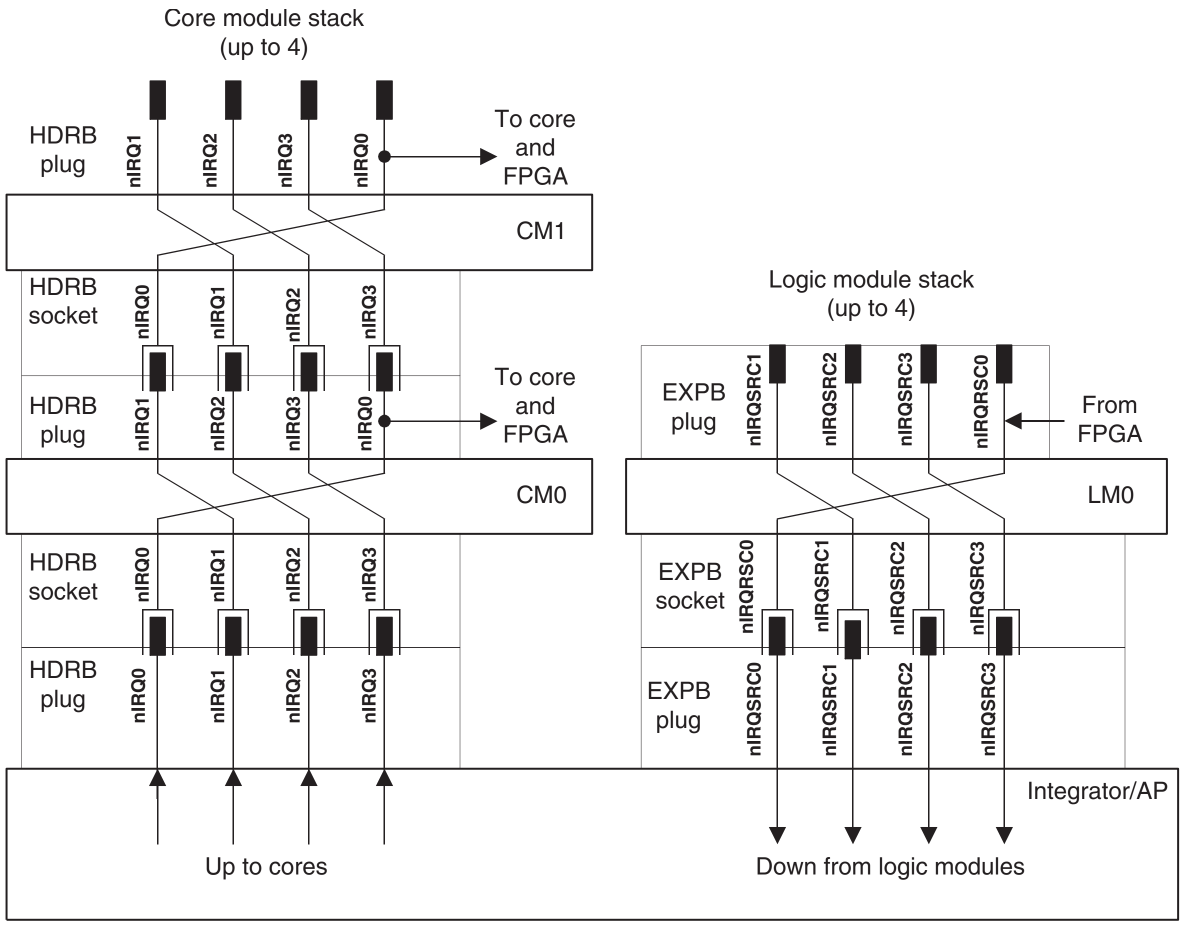


Figure 3-8 nIRQ[3:0] signal routing

3.7 Peripherals

This section describes the peripheral devices incorporated into the system controller FPGA. These include:

- *Counter/timers* on page 3-22
- *Real-time clock* on page 3-24
- *UARTs* on page 3-25
- *Keyboard and mouse interface* on page 3-28
- *GPIO* on page 3-29.

3.7.1 Counter/timers

Each counter/timer comprises:

- a 16-bit down counter with selectable prescale
- a load register
- a control register.

This is illustrated in Figure 3-9

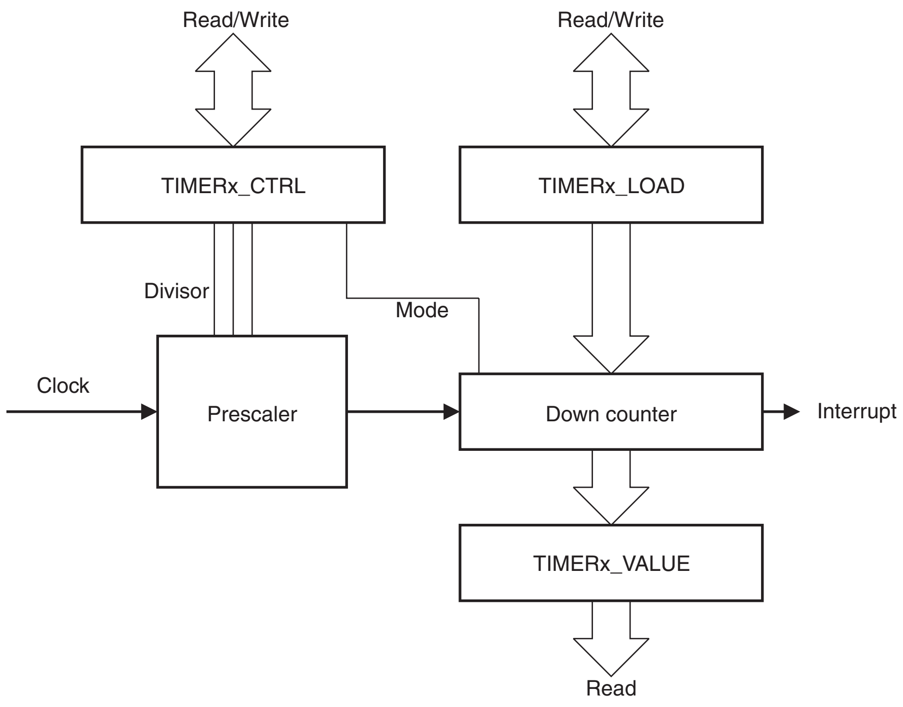


Figure 3-9 Counter/timer block diagram

Timer 0 is clocked by the system bus clock. Timer 1 and Timer 2 are clocked at a fixed frequency of 24MHz. The counters can be clocked directly or with a divide by 16 or 256 clock. The timers provide two operating modes:

- free-running
- periodic.

The prescale divisor and operating modes are selected by programming the control register TIMERx_CTRL.

A timer is loaded by writing to the load register TIMERx_LOAD. If the timer is enabled it begins a down count. When it reaches zero it generates an interrupt request. Interrupts are cleared by writing to the TIMERx_CLR register. The current value can be read at any time from the TIMERx_VALUE register.

After reaching a zero count:

- If the timer is operating in free-running mode, it wraps round and continues to decrement from the maximum register value.
- If the timer is operating in periodic mode, it reloads the value held in the load register and continues to decrement. In this mode the timer is able to generate a periodic interrupt.

The timer is enabled by setting a bit in the TIMERx_CTRL register. This register also contains the prescale selection bits and mode control bit.

See *Counter/timer registers* on page 4-23 for information about the timer registers.

3.7.2 Real-time clock

This section provides a functional overview of the real-time clock. However:

- for programming information, see *RTC control register* on page 4-40
- for detailed information about the real-time clock see the *RTC (PL030) Technical Reference Manual*.

RTC functional overview

The RTC comprises the following elements:

- a 32-bit counter
- a 32-bit match register
- a 32-bit comparator.

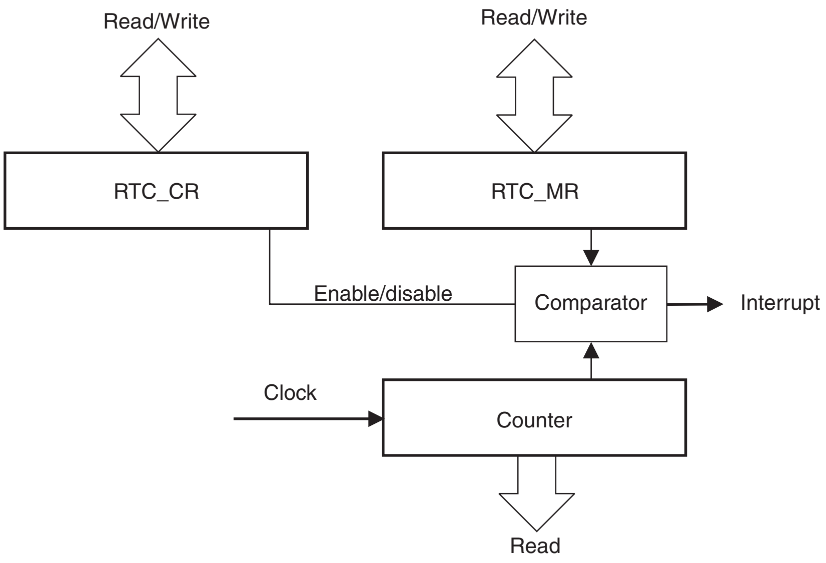


Figure 3-10 RTC block diagram

The 32-bit counter increments on successive rising edges of a 1Hz clock generated by the system controller FPGA. The counter is loaded with a start value by writing to the load register RTC_LR. The value of the counter can be read from the data register RTC_DR. When the counter reaches the maximum value, `0xFFFFFFFF`, it wraps to zero and continues incrementing.

The match register is programmed by writing to the match register RTC_MR. The value in the match register can be read at any time. The counter and match values are compared, and when the values are equal the RTC interrupt is asserted. After reset, values must be written to the load register RTC_LR and match register RTC_MR.

RTC interrupts

When the counter and match register contents are identical, the interrupt is asserted.

RTC interrupts are controlled as follows:

- to enable the interrupt, set the match interrupt enable bit MIE in the control register RTC_CR
- clear the interrupt by writing any value to the interrupt clear register RTC_EOI
- mask the interrupt **RTCINT** by writing to the control register RTC_CR
- read the status of the interrupt from RTC_STAT.

Synchronization logic is implemented to prevent propagation of metastable values when reading RTC_DR. This ensures the stability of the data, even at the point that the counter is incrementing.

The RTC interrupt can be used to implement a basic time alarm function. By using a 1Hz clock signal, the counter increments in one second intervals. This can be used to implement a real-time clock function in software as well as a basic alarm function.

3.7.3 UARTs

This section provides a functional overview of the UARTs. However:

- for programming information, see *UART registers* on page 4-41
- for detailed information about the UART, see the *UART (PL010) Technical Reference Manual*
- for serial interface connector pinout details, see *Serial interface connectors* on page A-14.

The serial interface is implemented with two PrimeCell UARTs are incorporated into the system controller FPGA. This is illustrated in Figure 3-11 on page 3-26.

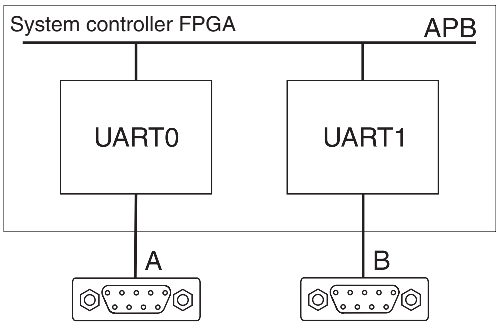


Figure 3-11 Serial interface

The UARTs are functionally similar to standard 16C550 devices. Each UART provides the following features:

- modem control inputs CTS, DCD, DSR, and RI
- modem output control signals RTS and DTR
- programmable baud rates of up to 460,800
- 16-byte transmit FIFO
- 16-byte receive FIFO
- four interrupts.

UART functional overview

Data for transmission is written into a 16-byte transmit FIFO. This causes the UART to start transmitting data frames with the parameters defined in the UART line control register. Transmission continues until the FIFO is emptied. On the receive side, the UART begins sampling after it receives a start bit (LOW level input). When a complete word has been received, it is stored in the receive FIFO together with any error bits associated with that word. See *UART registers* on page 4-41 for details of the read FIFO bits.

The FIFOs can be disabled. In this case, the UART provides a 1 byte holding register for each of the transmit and receive channels. The overrun bit in the UART_RSR register is set and an interrupt is generated if a word is received before the previous one was read. As a feature of the UART, the FIFOs are not physically disabled but are bypassed. This means that if an overrun error occurs, the excess data is still stored in the FIFO and must be read out to clear the FIFO.

The baud rate of the UART is set by programming the UART_LCRM and UART_LCRL bit rate divisor registers. See *UART registers* on page 4-41.

UART interrupts

Each UART generates four interrupts. These are:

- Modem status which is asserted when any of the status lines (DCD, DSR, and CTS) change. It is cleared by writing to the UART_ICR register.
- UART disabled which is asserted when the UART is disabled and a start bit (low level) is detected on the receive line. It is cleared if the UART is enabled or if the receive line goes HIGH.
- Rx interrupt which is asserted when one of the following events occur:
  - the receive FIFO is enabled and the FIFO is half or more than half full (contains 8 or more bytes)
  - the receive FIFO is not empty and there has been no data for more than a 32-bit period
  - the receive FIFO is disabled and data is received.

  The Rx interrupt is cleared by reading contents of the FIFO.
- Tx interrupt which is asserted when one of the following events occur:
  - the transmit FIFO is enabled and the FIFO is half or less than half full
  - the transmit FIFO is disabled and the holding buffer is empty.

  The Tx interrupt is cleared by filling the FIFO to more than half full or writing to the holding register.

Baud rate selection

The baud rate generator in each UART uses a 16-bit divisor stored in the UART_LCRM and UART_LCRL registers to determine the bit period. The baud rate generator contains a 16-bit down counter that is clocked at 14.7456MHz by the **UARTCLK** signal. See *UART registers* on page 4-41 for information about the UART internal registers.

3.7.4 Keyboard and mouse interface

This section provides a functional overview of the *Keyboard and Mouse Interface* (KMI). However:

- for programming information, see *Keyboard and mouse interfaces* on page 4-51
- for detailed information about the KMI, see *KMI (PL050) Technical Reference Manual*
- for KMI connector pinout details, see *Keyboard and mouse connectors* on page A-15.

The keyboard and mouse controllers are implemented with two PrimeCell KMIs that are incorporated into the system controller FPGA. This is shown in Figure 3-12.

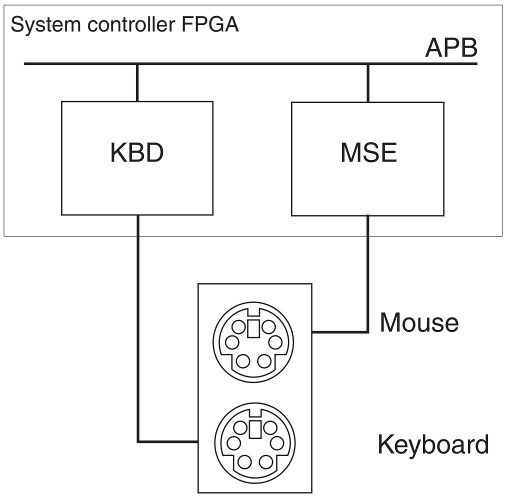


Figure 3-12 KMI block diagram

Each KMI contains the following functional blocks:

- APB interface and register block
- transmit block
- receive block
- controller block
- timer/clock divider blocks
- synchronization logic.

Functional overview

The APB interface and register block provides the interface with the APB and control registers. All control register bits are synchronized to the main clock for KMI logic before they are used in the KMI core. This block also generates individual transmit and receive interrupts.

The transmit block converts the parallel transmit data into a serial bit stream with a rate dependent on the incoming KMI clock signal. This block performs odd parity generation, data framing, and parallel-to-serial conversion. This block operates on **KMIREFCLK** (that on the AP is supplied by **CLK24MHz**) with the incoming clock signal **KMICLKIN** providing the bit-rate information. The data is shifted out on the falling edge of the **KMICLKIN** input.

The receive block performs serial-to-parallel conversion on the serial data stream received on the **KMIDATAIN** input pin. The falling edge on the synchronized and sampled **KMICLKIN** input signal is used to sample the **KMIDATAIN** input line. The **KMIDATAIN** input is synchronized to the **KMIREFCLK** clock.

The controller controls transmit and receive operations. If simultaneous requests for transmission and reception occur, the transmit request is given priority.

KMI interrupts

The transmit interrupt is asserted to indicate that a byte can be written to the data register KMIDATA for transmission. The receive interrupt is asserted to indicate that a byte has been received and can be read from the data register KMIDATA. A combined interrupt is also asserted if either the transmit or receive interrupt is asserted. This combined interrupt is used by the system interrupt controller that provides masking for the outputs from each peripheral.

3.7.5 GPIO

The system controller FPGA incorporates a 32-bit *General Purpose Input/Output Controller* (GPIO) port. These connect to pins on the EXPB connector (see *Logic module connector EXPB* on page A-8). This can be used to implement a bus between the system controller FPGA on the AP and the FPGA(s) on the logic module.

Each bit can be individually programmed as an input or an output using the GPIO_DIRN register. Bits in the data register are set and cleared using the GPIO_DATASET and GPIO_DATACLR registers. It is read from and written to using the GPIO_DATAIN and GPIO_DATAOUT locations (see *GPIO* on page 4-36).

Chapter 4
Programmer's Reference

This chapter describes the Integrator/AP memory map and local registers. It contains the following sections:

- *About the Integrator memory map* on page 4-2
- *System memory map regions* on page 4-4
- *Accesses to boot ROM and flash* on page 4-9
- *System control registers* on page 4-10
- *EBI configuration registers* on page 4-19
- *Counter/timer registers* on page 4-23
- *Alphanumeric display, LED, and DIP switch registers* on page 4-25
- *Interrupt controller registers* on page 4-29
- *Peripheral registers* on page 4-36.

4.1 About the Integrator memory map

Core modules and system bus masters on logic modules or the PCI have a different view of memory map for the lowest 272MB of the system address space. Figure 4-1 shows the memory map for core modules. This contains resources on core module and resources on the motherboard and other modules. The bottom 272MB contains the SSRAM, SDRAM, and control registers on the core module. Accesses by the core to these areas are not normally passed to the motherboard.

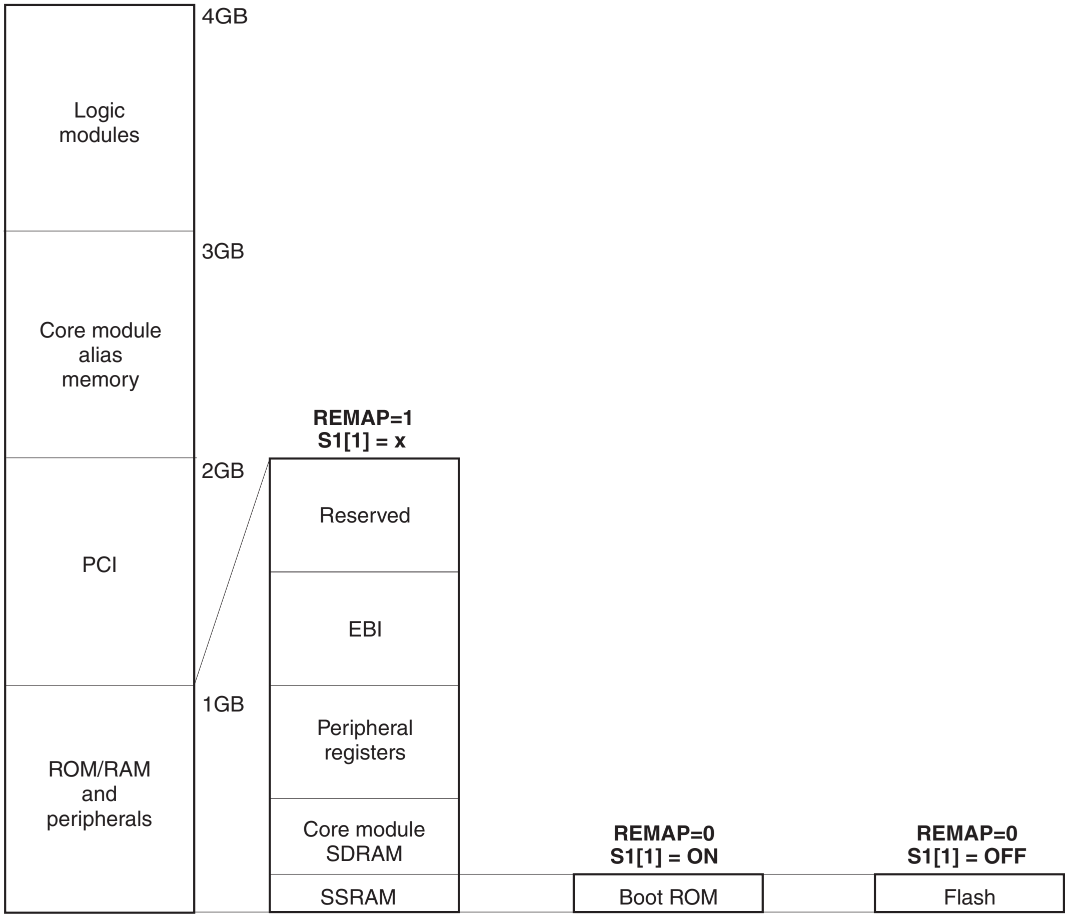


Figure 4-1 Integrator memory map for core modules

The EBI region contains the boot ROM and flash. However, during system startup either the boot ROM or the flash is aliased to the bottom of the memory map. The SSRAM on the core shares the same address as this alias. The switch S1[1] and the REMAP bit in the CM_CTRL register on the core module control whether the SSRAM, boot ROM, or flash are accessible at this location. See *Accesses to boot ROM and flash* on page 4-9.

Figure 4-2 shows the memory map for all other system bus masters. In this case, the lowest 64MB of the system address space contains an alias of the EBI (see *External bus interface* on page 4-6). The mapping of the EBI alias is also affected by the setting of S1[1] (see *Accesses to boot ROM and flash* on page 4-9).

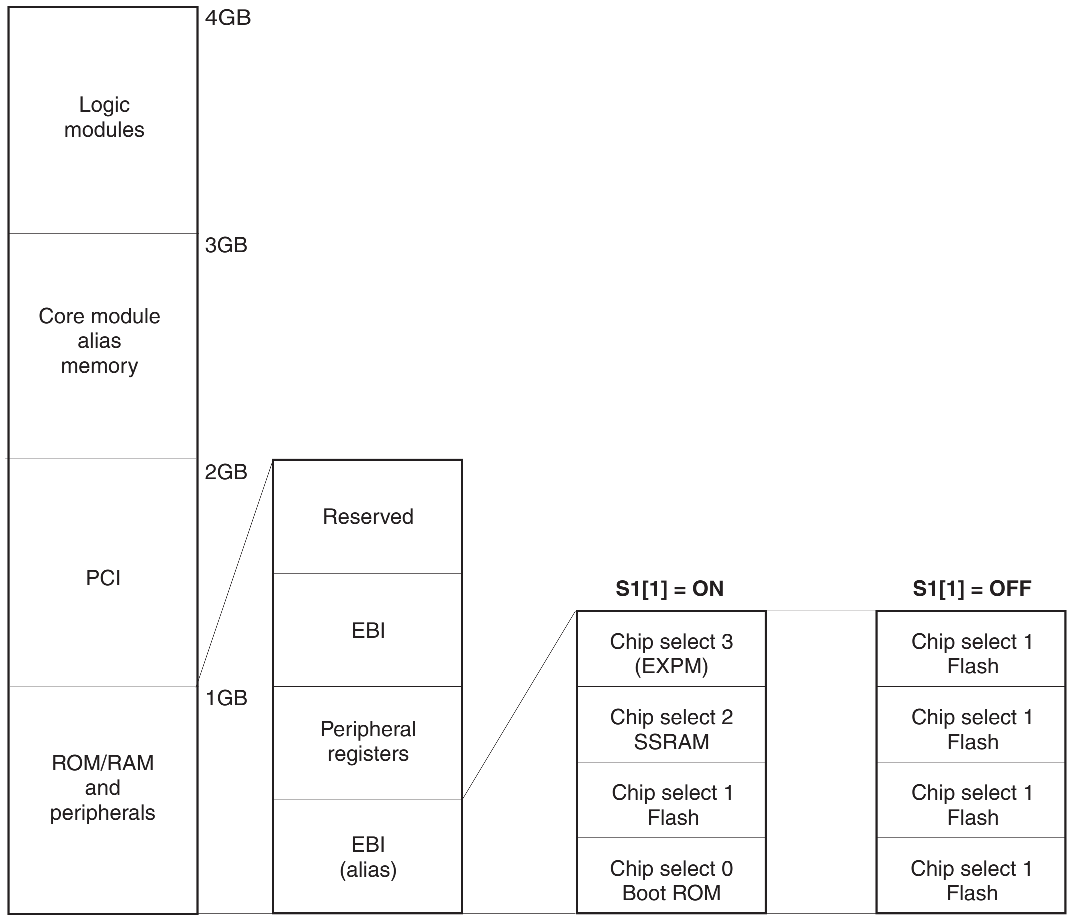


Figure 4-2 Integrator memory map for logic modules

4.2 System memory map regions

The system memory map contains four main regions. These are:

- *Logic module region* on page 4-4
- *Core module alias memory region* on page 4-5
- *PCI region* on page 4-6
- *ROM, RAM, and peripherals region* on page 4-6.

The top-level memory map is shown in Table 4-1.

Table 4-1 Top level Integrator/AP memory map

Base address	Size	Description
`0xC0000000`	1GB	Logic modules
`0x80000000`	1GB	Core module alias memory
`0x40000000`	1GB	PCI space
`0x00000000`	1GB	Local RAM, ROM, and peripherals

4.2.1 Logic module region

This region contains any memory and device registers located on logic modules stacked on the EXPA/EXPB connectors.

Each logic module decodes its own address space and provide responses to all accesses within its assigned address space. You can design your own logic modules with fixed, link selectable, or position-selected base addresses. However, to ensure compatibility with other ARM Integrator modules, it is recommended that your logic module uses the **ID[3:0]** signals from the EXPB connector to determine its base addresses as shown in Table 4-2.

Table 4-2 Recommended logic module address decoding

Base address	Size	Description	ID[3:0]
`0xF0000000`	256MB	Logic module 3 memory	1101
`0xE0000000`	256MB	Logic module 2 memory	1011
`0xD0000000`	256MB	Logic module 1 memory	0111
`0xC0000000`	256MB	Logic module 0 memory	1110

4.2.2 Core module alias memory region

This region provides access to the SDRAM located on the core modules at an *alias* address on the system bus. The address of the aliased SDRAM on a core module is automatically configured by its position in the stack and by the signals **ID[3:0]** signals from the HDR connector, as shown in Table 4-3.

Table 4-3 Aliased core module SDRAM mapping

Base address	Size	Description	ID[3:0]
`0xB0000000`	256MB	Core module 3 alias memory	1101
`0xA0000000`	256MB	Core module 2alias memory	1011
`0x90000000`	256MB	Core module 1 alias memory	0111
`0x80000000`	256MB	Core module 0 alias memory	1110

Figure 4-3 illustrates the local and alias memory mapping of the SDRAM on four core modules. See the user guide for your particular core module for more information.

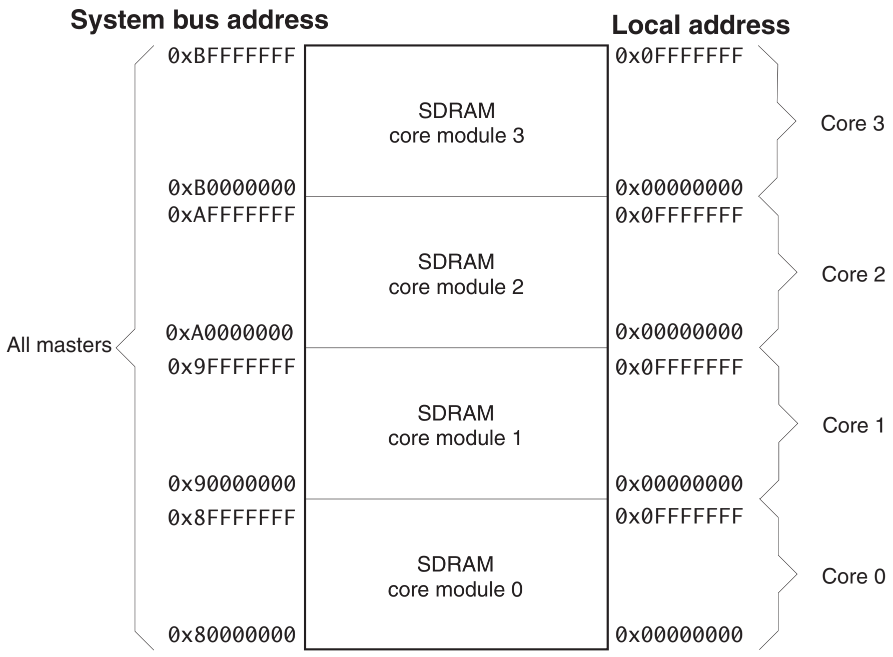


Figure 4-3 Core module alias memory mapping

4.2.3 PCI region

This region provides access to the PCI bus through the V360EPC PCI host bridge controller. The address range `0x40000000` to `0x7FFFFFFF` is mapped to PCI memory space by the host bridge controller (see Chapter 5 *PCI Subsystem*).

4.2.4 ROM, RAM, and peripherals region

This region contains the:

- *External bus interface*
- *System control and peripheral registers area* on page 4-7
- *Core module local SSRAM and SDRAM* on page 4-8.

Table 4-4 shows the memory map for this region.

Table 4-4 ROM, RAM, and peripherals region

Base address	Size	Core module	Logic module
`0x30000000`	256MB	Reserved for future expansion	Reserved for future expansion
`0x20000000`	256MB	EBI	EBI
`0x10000000`	256MB	System control and peripheral registers	System control and peripheral registers
`0x00000000`	256MB	Core module local SSRAM and SDRAM	EBI (alias)

External bus interface

The EBI space provides access to four smaller areas of memory. These contain the boot ROM, flash and SSRAM. An expansion memory can be connected to the EXPM connector and accessed within this region. Table 4-5 shows the memory map of the EBI space.

Table 4-5 External bus interface chip selects

Base address	Bus width	Description
`0x2C000000`	8, 16 or 32bits	Chip select 3 (for expansion)
`0x28000000`	32bits	SSRAM
`0x24000000`	32bits	Flash
`0x20000000`	8bits	Boot ROM

These devices are also mapped into the bottom the system address space in different ways for core and logic modules, as described in *Accesses to boot ROM and flash* on page 4-9.

See *EBI configuration registers* on page 4-19.

System control and peripheral registers area

Within the this region are the Integrator/AP peripheral control, interrupt and system control registers. The naming convention used for the registers is based on the device and register function. For example the real-time clock match register is named RTC_MR.

Table 4-6 shows the base addresses for the registers for each device. Read and write accesses to locations shown as spare do not generate bus errors but any data read is undefined.

Table 4-6 Peripheral registers

Base address	Size	Description
`0x1F000000`	16MB	Spare
`0x1E000000`	16MB	Spare
`0x1D000000`	16MB	Spare
`0x1C000000`	16MB	Spare
`0x1B000000`	16MB	GPIO
`0x1A000000`	16MB	LED display and boot switch
`0x19000000`	16MB	Mouse
`0x18000000`	16MB	Keyboard
`0x17000000`	16MB	UART1
`0x16000000`	16MB	UART0
`0x15000000`	16MB	RTC
`0x14000000`	16MB	Interrupt controller
`0x13000000`	16MB	Counter/timers

Table 4-6 Peripheral registers (continued)

Base address	Size	Description
`0x12000000`	16MB	EBI configuration registers
`0x11000000`	16MB	System controller registers
`0x10000000`	16MB	Core module registers (for core modules) Spare (for logic modules)

Core module local SSRAM and SDRAM

This region contains SSRAM and SDRAM on the core modules. Each core module has sole access to its own SSRAM and SDRAM within this address range. However, the SDRAM also appears within the alias memory region where it can be accessed by any system bus master in the system (see *Core module alias memory region* on page 4-5).

The boot ROM or flash on the AP can be masked by the lowest portion of the SSRAM if the REMAP bit is set to 1 (see *Core module accesses to boot ROM or flash* on page 4-9).

4.3 Accesses to boot ROM and flash

The mapping of this area of the address map differs for core modules and logic modules.

4.3.1 Core module accesses to boot ROM or flash

Accesses by a core module to the SSRAM on the core module and boot ROM or flash on the motherboard are controlled by the REMAP bit in the CM_CTRL register on the core module and S1[1] on the AP:

REMAP = 0 Default following reset. Accesses to addresses `0x00000000` to `0x0003FFFF` (or a larger space on later core modules) are routed to the boot ROM or flash on the AP depending on the setting of S1[1]:

S1[1] = ON
the access is to boot ROM

S1[1] = OFF
the access is to flash.

REMAP = 1 Accesses to addresses `0x00000000` to `0x0003FFFF` (or a larger space on later core modules) are routed to the local SSRAM on the core module.

For information about the DIP switch, see *Setting the DIP switches* on page 2-6.

4.3.2 Logic module accesses to boot ROM or flash

The boot ROM and flash on the AP are attached to the EBI (see *External bus interface* on page 4-6) and are also aliased at the bottom the memory map that a logic module can see (as shown in Figure 4-2 on page 4-3). Logic modules can access the boot ROM or flash in this area depending on the setting of S1[1]:

S1[1] = ON the EBI resources are mapped into the bottom 256MB of the system memory map.

S1[1] = OFF the flash is mapped repeatedly into the bottom 256MB of the system memory map.

4.4 System control registers

The system control registers are used to configure the system controller. They are usually controlled by firmware in the boot ROM. Applications programs can also access the registers, but with care. Writing inappropriate values into the oscillator register can prevent the system from operating. The oscillator register is protected against accidental writes by a lock register (see *Oscillator lock register* on page 4-16).

Note

The control registers do not support byte writes and must be accessed as 32-bit words. Preserve the values of bits that are not detailed or are marked as *reserved* using read-modify-write operations.

Table 4-7 shows the system controller registers.

Table 4-7 System controller status and control registers

Address	Name	Type	Function	Reset by
`0x11000000`	SC_ID	Read	System controller identification	Reset
`0x11000004`	SC_OSC	Read/write	Oscillator divisors	Reset
`0x11000008`	SC_CTRLS	Read/write	Control register set	Reset
`0x1100000C`	SC_CTRLC	Read/write	Control register clear	Reset
`0x11000010`	SC_DEC	Read	Decoder status	Reset
`0x11000014`	SC_ARB	Read/write	Arbiter time-out values	Reset
`0x11000018`	SC_PCI	Read/write	PCI control	Reset
`0x1100001C`	SC_LOCK	Read/write	Lock	Reset
`0x11000020`	SC_LBFADDR	Read	PCI local bus fault address	Reset
`0x11000024`	SC_LBFCODE	Read	PCI local bus fault code	Reset
`0x11000030`	SC_FLAGS	Read	Flag register	Reset
`0x11000030`	SC_FLAGSET	Write	Flag set register	Reset
`0x11000034`	SC_FLAGSCLR	Write	Flag clear register	Reset

Table 4-7 System controller status and control registers

Address	Name	Type	Function	Reset by
`0x11000038`	SC_NVFLAGS	Read	Nonvolatile flag register	POR
`0x11000038`	SC_NVFLAGSSET	Write	Nonvolatile flag set register	POR
`0x1100003C`	SC_NVFLAGSCLR	Write	Nonvolatile flag clear register	POR

4.4.1 System controller ID register

The system controller ID register (SC_ID) is a 32-bit read-only register that identifies the board manufacturer, board type, and revision.

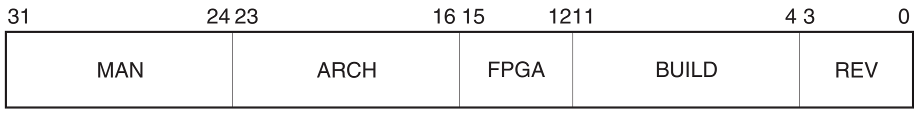


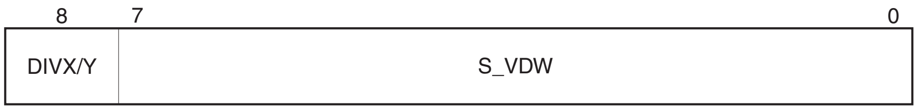
Table 4-8 describes the system controller ID register bits.

Table 4-8 SC_ID register

Bits	Name	Type	Function
31:24	MAN	Read	Manufacturer ID (`0x41`=ARM).
23:16	ARCH	Read	Architecture: `0x00` = ASB little-endian `0x01` = AHB little-endian
15:12	FPGA	Read	System controller FPGA type: 1 = XC4062 2 = XC4085
11:4	BUILD	Read	Build value
3:0	REV	Read	Revision: 0 = Revision A 1 = Revision B

4.4.2 Oscillator divisor register

The oscillator divisor register (SC_OSC) contains nine active bits that are used to set the frequency of the system bus clock and PCI bus clock signals (see *Clock generator* on page 3-15).



Note

Before writing to the SC_OSC register, you must unlock it by writing the value `0x0000A05F` to the SC_LOCK register. When you have finished writing to the SC_OSC register lock it again by writing any value *other* than `0x0000A05F` to the SC_LOCK register.

Table 4-9 describes the oscillator divisor register bits.

Table 4-9 SC_OSC register

Bits	Name	Type	Function
8	DIVX/Y	Read/write	This bit is used to set the frequency of the PCI bus clock signal **CP_CLK**. It controls the values of bits in the VCO divider within the clock generator chip. The settings for this bit are as follows: 0 = 33MHz (default) 1 = 25MHz.
7:0	S_VDW	Read/write	These bits are used to set the frequency of the system bus clock signal **SYSCLK**. They control the value of the lower 8 VCO divider bits within the clock generator chip. The range of values for this bit-field is 4-192(dec) (giving 3-50MHz in 0.25MHz steps). Values below 4 and above 192 are not permitted.

4.4.3 System control register

The system control register (SC_CTRL) is accessed using the SC_CTRLS and SC_CTRLC locations. The system control register is an 8-bit register that is used to control the UART modem lines, flash memory protection, and software reset.

7	6	5	4	3	2	1	0
UART1DTR	UART1RTS	UART0DTR	UART0RTS	RSVD	FLASHWP	FLASHVPP	SOFTRST

Set, read, and clear bits in the system controller register as follows:

- Set bits in the control register by writing to the SC_CTRLS location:

  1 = SET the associated bit in the control register

  0 = leave the associated bit in the control register unchanged.

- Read the current state of the control register bits from the SC_CTRLS location.

- Clear bits in the control register by writing to the SC_CTRLC location:

  1 = CLEAR the associated bit in the control register

  0 = leave the associated bit in the control register unchanged.

Table 4-10 describes the bits in the system control register.

Table 4-10 System control register

Bits	Name	Function
7	UART1DTR	Controls the UART1 DTR line (active LOW): 0 = HIGH (default) 1 = LOW.
6	UART1RTS	Controls the UART1 RTS line (active LOW): 0 = HIGH (default) 1 = LOW.
5	UART0DTR	Controls the UART0 DTR line (active LOW): 0 = HIGH (default) 1 = LOW.
4	UART0RTS	Controls the UART0 RTS line (active LOW): 0 = HIGH (default) 1 = LOW.
3	-	Reserved

Table 4-10 System control register (continued)

Bits	Name	Function
2	FLASHWP	Flash write-protection: 0 = protected (default) 1 = unprotected.
1	FLASHVPP	Flash Vpp enable: 0 = disabled (default) 1 = enabled.
0	SOFTRESET	Software reset. A 1 written to this bit resets the system.

4.4.4 Decoder status register

The decoder status register (SC_DEC) is an 8-bit read-only register that provides information about any modules that are stacked on the EXP and HDR connectors.

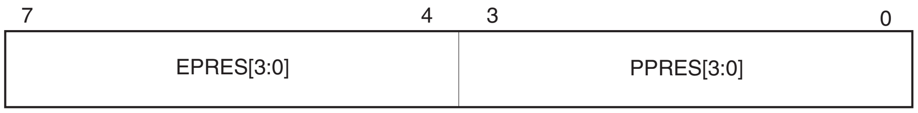


Table 4-11 describes the bits in the status decoder register.

Table 4-11 SC_DEC register

Bits	Name	Function
7:4	EPRES[3:0]	Module present on EXP connectors, 1 bit for each module: 0000 = no modules fitted 0001 = module 0 fitted 0011 = module 1 and 0 fitted 0111 = module 2, 1, and 0 fitted 1111 = module 3, 2, 1, and 0 fitted.
3:0	PPRES[3:0]	Module present on HDR connectors 1 bit for each module: 0000 = no modules fitted 0001 = module 0 fitted 0011 = module 1 and 0 fitted 0111 = module 2, 1, and 0 fitted 1111 = module 3, 2, 1, and 0 fitted.

4.4.5 Arbiter timeout register

The arbiter timeout register (SC_ARB) is a 32-bit read-write register that contains a transaction counter and cycle counter. Use the counters to tailor the operation of the arbiter for your application. See *Arbitration counters* on page 3-9.

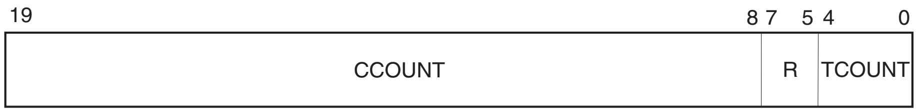


Table 4-12 describes the arbiter timeout register bits.

Table 4-12 SC_ARB register

Bits	Name	Function
19:8	CCOUNT	Cycle counter load value, `0x000` to `0xFFF`(default 0x000).
7:5	Reserved	Reserved.
4:0	TCOUNT	Transaction counter load value, 0x00 to 0x1F (default 0x08).

4.4.6 PCI control register

The PCI control register is used to enable or disable the PCI interface and to clear a PCILBINT interrupt. Table 4-13 describes the PCI control register bits. For information about using the PCILBINT interrupt, see *PCI subsystem interrupts* on page 5-17.

Table 4-13 SC_PCI register

Bits	Name	Function
1	PCILBINT_CLR	Clear PCILBINT: Always reads as 0. Write 1 to clear an interrupt.
0	PCIEN	PCI interface enable: 0 = disabled (default) 1 = enabled.

4.4.7 Oscillator lock register

The lock register (SC_LOCK) is used to control access to the SC_OSC register allowing it to be locked and unlocked. Use this mechanism to protect SC_OSC registers from being accidently overwritten with values that could render the Integrator/AP inoperable.

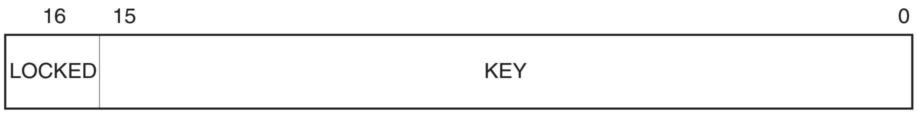


Table 4-14 describes the lock register bits.

Table 4-14 SC_LOCK register

Bits	Name	Function
16	LOCKED	Lock bit. Read this bit to determine if the SC_OSC register is locked: 0 = unlocked 1 = locked.
15:0	KEY	Lock key. Write `0xA05F` to unlock the SC_OSC register. Write any other value to lock the SC_OSC register.

4.4.8 PCI local bus fault address register

The PCI local bus fault address register (SC_LBFADDR) is read when a PCILBINT interrupt occurs. Read this register to determine the address at which the fault occurred during an access on the PCI local bus.

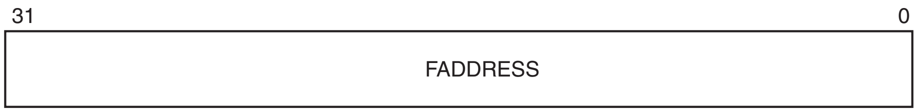


Table 4-15 describes the bits in the PCI local bus fault address register.

Table 4-15 SC_LBFADDR register

Bits	Name	Function
31:0	FADDRESS	Fault address. The word address at which a fault occurred on the PCI local bus. Read the byte enable BEN[3:0] bits in the fault code register for the byte or half-word address.

4.4.9 PCI local bus fault code register

The PCI local bus fault code register (SC_LBFCODE) is read when a PCILBINT interrupt occurs. Read this register for information about the access in progress when the fault occurred.

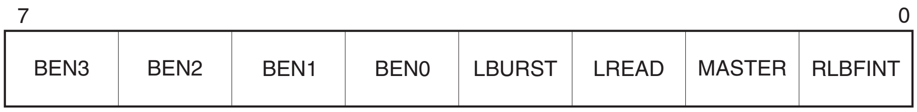


Table 4-16 describes the bits in the PCI local bus fault code register.

Table 4-16 SC_LBFCODE register

Bits	Name	Function
7	BEN3	Byte enable 3. Indicates that data bits 31:24 were being accessed.
6	BEN2	Byte enable 2. Indicates that data bits 23:16 were being accessed.
5	BEN1	Byte enable 1. Indicates that data bits 15:8 were being accessed.
4	BEN0	Byte enable 0. Indicates that data bits 7:0 were being accessed.
3	LBURST	Indicates that a burst was in progress. That is: 1 = this word was not the last of the burst.
2	LREAD	These bits indicate the direction of transfer as viewed by the current master. The encoding of bits 2:1 is: 00 = data in, PCI (or V3 DMA) write to AMBA 01 = data out, AMBA write to PCI 10 = data out, PCI (or V3 DMA) read from AMBA 11 = data in, AMBA read from PCI.
1	MASTER	
0	RLBFINT	Raw PCI local bus fault interrupt (PCILBNT) status.

For information about how to use the SC_LBFADDR and SC_LBFCODE registers (see *PCI subsystem interrupts* on page 5-17).

4.4.10 System controller flag registers

The flag registers provide you with two 32-bit register locations containing general purpose flags that you can assign any meaning to.

Table 4-17 System controller flag registers

Register Name	Address	Access	Reset by	Description
SC_FLAGS	`0x11000030`	R	Reset	Flag register
SC_FLAGSSET	`0x11000030`	W	Reset	Flag set register
SC_FLAGSCLR	`0x11000034`	W	Reset	Flag clear register
SC_NVFLAGS	`0x11000038`	R	POR	Nonvolatile flag register
SC_NVFLAGSSET	`0x11000038`	W	POR	Nonvolatile flag set register
SC_NVFLAGSCLR	`0x1100003C`	W	POR	Nonvolatile flag clear register

The core module provides two distinct types of flag registers:

- the FLAGS registers are cleared by a normal reset, such as a reset caused by pressing the reset button
- the NVFLAGS registers retain their contains after a normal reset and are only cleared by a *Power-On Reset* (POR).

Flag and nonvolatile flag register

The status register contains the current state of the flags.

Flag and nonvolatile flag set register

The flag set locations are used to set bits in the flag registers as follows:

- write 1 to SET the associated flag.
- write 0 to leave the associated flag unchanged.

Flag and nonvolatile flag clear register

The clear locations are used to clear bits in the flag registers as follows:

- write 1 to CLEAR the associated flag
- write 0 to leave the associated flag unchanged

4.5 EBI configuration registers

The EBI configuration registers are 8-bit read/write registers that are used to define the operating parameters for four EBI regions. The EBI provides four fixed-size memory regions each with its own chip select signal. Additional address decoding must be handled by addresses within the region selected by the chip select.

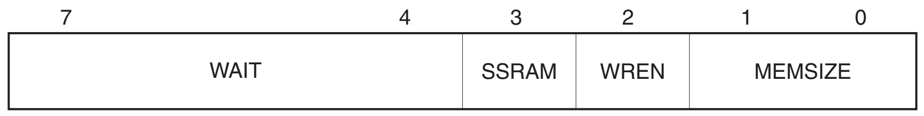
The locations of the EBI registers are shown in Table 4-18.

Table 4-18 EBI configuration register locations

Address	Name	Type	Function
`0x12000000`	EBI_CSR0	Read/write	Chip select 0 configuration (boot ROM)
`0x12000004`	EBI_CSR1	Read/write	Chip select 1 configuration (flash)
`0x12000008`	EBI_CSR2	Read/write	Chip select 2 configuration (SSRAM)
`0x1200000C`	EBI_CSR3	Read/write	Chip select 3 configuration (EXPM)
`0x12000020`	EBI_LOCK	Read/write	EBI lock register

4.5.1 EBI configuration registers

The four registers have a similar format.



These parameters are shown in Table 4-19 on page 4-20 and are described in:

- *Wait states* on page 4-21
- *SSRAM* on page 4-21
- *Write enable* on page 4-21
- *Memory size* on page 4-21.

Table 4-19 EBI_CSRx register

Bits	Name	Function
7:4	WAIT	Wait states: `0x0` = 0 cycles `0x1` = 1 cycle `0x2` = 2 cycles ... `0xC` = 12 cycles `0xD` = 13 cycles `0xE` = 13 cycles `0xF` = 13 cycles.
3	SSRAM	Synchronous SRAM: 0 = asynchronous memory 1 = synchronous memory.
2	WREN	Write enable: 0 = writes disabled 1 = writes enabled.
1:0	MEMSIZE	Memory size: 00 = 8 bit 01 = 16 bit 10 = 32 bit 11 = reserved.

The default values in the EBI_CSRx registers are shown in Table 4-20.

Table 4-20 EBI_CSRx default values

Register	Default value	WAIT	Memory type	WREN	MEMSIZE	Device
EBI_CSR0	`0x20`	2 cycles	Asynchronous	Disabled	8 bit	Boot ROM
EBI_CSR1	`0x22`	2 cycles	Asynchronous	Disabled	16 bit	Flash
EBI_CSR2	`0x0E`	N/A	Synchronous	Enabled	32 bit	SSRAM
EBI_CSR3	Undefined	-	-	-	-	User defined

Wait states

The basic transfer takes one decode cycle and two active cycles to complete. A 4-bit counter is used to add wait states. This is programmed using the WAIT field of the associated EBI_CSRx register. Values of 0 - 13 (decimal) are valid. Values of 14 and 15 are treated as 13.

The access time is given by the formula:

Cycles = 3 + wait states

where cycles is less than or equal to 16.

For example, in a system with a 25MHz system bus the cycle time is 40ns, giving a minimum read cycle time of 3 x 40 = 120ns. If the wait-state register is programmed with 3, the read cycle time is 6 x 40 = 240ns.

SSRAM

When the SSRAM bit is 1, the wait state field is ignored. Accesses always take one decode and two active cycles. All control signals are sampled on the rising edge of **MEMCLK**. The address strobe signals **nMADSP** and **nMADSC** are address strobes for the synchronous SRAM and not are available on the expansion connector EXPM.

Write enable

For asynchronous memories the write enable pulse is driven LOW half a clock cycle after **MA** and **nMCS** become active. This allows for address and chip-select setup before write enable, as required by most asynchronous memories. The write enable pulse is driven HIGH half a clock cycle before **MA** and **nMCS** are driven inactive to ensure sufficient data hold time.

For synchronous SRAM, the write enable pulse is always 1 cycle and driven in the second active cycle.

The write enable bit in the register is 0 by default to avoid accidental writes to the boot ROM and flash.

Memory size

The memory size bits in the registers define the data width of the external memory. This is 8 bits for boot ROM and 32 bits for the flash and onboard SSRAM. The EBI compares the memory size with the access width (indicated by **HSIZE** or **BSIZE**) and generates the appropriate number of cycles. For example, a 32 bit access to an 8 bit memory requires one decode cycle followed by 8 access cycles.

4.5.2 EBI lock register

The lock register (EBI_LOCK) is used to control access to the EBI configuration registers allowing it to be locked and unlocked. Use this mechanism to protect EBI_CSRx registers from being accidently overwritten with values that could render the Integrator/AP inoperable.

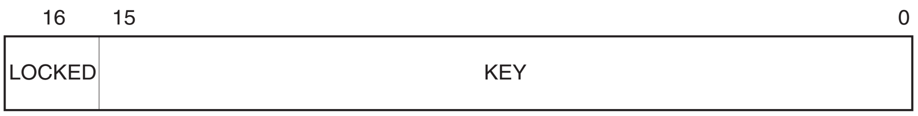


Table 4-14 describes the lock register bits.

Table 4-21 EBI_LOCK register

Bits	Name	Function
16	LOCKED	Lock bit. Read this bit to determine if the EBI_CSRx registers are locked: 0 = unlocked 1 = locked.
15:0	KEY	Lock key. Write `0xA05F` to unlock the EBI_CSRx registers. Write any other value to lock the EBI_CSRx register.

4.6 Counter/timer registers

The counter/timer registers control the three counter/timers. See *Counter/timers* on page 3-22. There are four registers for each of the three counter/timers, as shown in Table 4-22.

Table 4-22 Counter/timer registers

Address	Name	Type	Size	Function
`0x13000000`	TIMER0_LOAD	Read/write	16	Timer 0 load register
`0x13000004`	TIMER0_VALUE	Read	16	Timer 0 current value register
`0x13000008`	TIMER0_CTRL	Read/write	16	Timer 0 control register
`0x1300000C`	TIMER0_CLR	Write	1	Timer 0 clear register
`0x13000100`	TIMER1_LOAD	Read/write	16	Timer 1 load register
`0x13000104`	TIMER1_VALUE	Read	16	Timer 1 current value register
`0x13000108`	TIMER1_CTRL	Read/write	16	Timer 1 control register
`0x1300010C`	TIMER1_CLR	Write	1	Timer 1 clear register
`0x13000200`	TIMER2_LOAD	Read/write	16	Timer 2 load register
`0x13000204`	TIMER2_VALUE	Read	16	Timer 2 current value register
`0x13000208`	TIMER2_CTRL	Read/write	16	Timer 2 control register
`0x1300020C`	TIMER2_CLR	Write	1	Timer 2 clear register

4.6.1 Timer x load register

The timer load register is a 16-bit read/write register that contains a 16-bit initial count value that is reloaded each time the counter reaches zero, if the timer is operating in periodic mode.

Write the upper 16 bits (of 32 bits) as 0s. They are undefined when read.

4.6.2 Timer x current value register

The timer value register contains the current count value for the timer. The upper 16bits (of 32bits) are undefined.

4.6.3 Timer x control register

The timer control registers are 8-bit read/write registers that control the operation of their associated counter/timers. The format of these three registers is similar.

7	6	5	4	3	2	1	0
ENABLE	MODE	Unused		PRESCALE		Unused	

Table 4-23 describes the timer control register bits.

Table 4-23 TIMERx_CTL register

Bits	Name	Function
7	ENABLE	Timer enable: 0 = disabled 1 = enabled.
6	MODE	Timer mode: 0 = free running, counts once and then wraps to `0xFFFF` 1 = periodic, reloads from load register at the end of each count.
5:4	Unused	Unused, always write as 0s.
3:2	PRESCALE	Prescale divisor: 00 = none 01 = divide by 16 10 = divide by 256 11 = undefined.
1:0	Unused	Unused, always write as 0s.

4.6.4 Timer x clear register

The timer clear register is a write-only location that does not have a storage element. Writing any value to this location clears the interrupt for the associated counter/timer.

4.7 Alphanumeric display, LED, and DIP switch registers

These registers are used to control the alphanumeric display and LEDs, and to read the 4-input DIP switch.

Table 4-24 LED control and switch registers

Address	Name	Type	Size	Function
`0x1A000000`	LED_ALPHA	Read/write	32	Alphanumeric characters register
`0x1A000004`	LED_LIGHTS	Read/write	4	LED control register
`0x1A000008`	LED_SWITCHES	Read	4	DIP switch register

4.7.1 Alphanumeric characters register

This register is used to write characters to the alphanumeric display. The system controller FPGA manages the transfer of character data into a shift register within the alphanumeric display. You must allow each transfer to complete before writing new characters to the display using the following sequence:

1. Check that the display status is IDLE (bit 0 = 0) in the LED_ALPHA register.
2. Write 2 characters LED_ALPHA register (bits 31:1).

Note

The data from the LED_ALPHA register is transferred as a serial bit-stream into the alphanumeric display at the same time as the LED control bits in LED_LIGHTS register.

Table 4-25 describes LED_ALPHA register bits.

Table 4-25 LED_ALPHA bit assignment

Bit	Name	Function
30:1	CHARACTERS	Bit patterns that form characters on the alphanumeric display. See Table 4-26 for segment and bit assignments. Ensure that the display is idle before writing each new character.
0	STATUS	This is a read-only bit that returns the status of the alphanumeric display: 0 = idle 1 = busy

The digits and segments are identified in Figure 4-4.

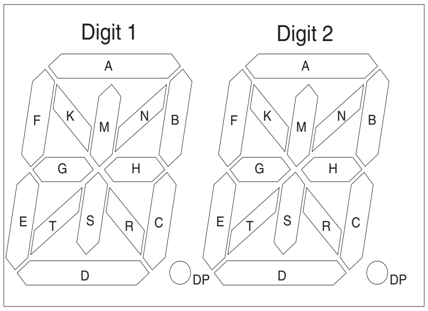


Figure 4-4 Alphanumeric display segment designation

The bit assignments for the alphanumeric display segments in the LED_ALPHA register are shown in Table 4-26.

Table 4-26 LED_ALPHA bit-to-segment mapping

Digit 1	Segment	Digit 2
29	DP	30
28	T	14
27	S	13
26	R	12

Table 4-26 LED_ALPHA bit-to-segment mapping (continued)

Digit 1	Segment	Digit 2
25	N	11
24	M	10
23	K	9
22	H	8
21	G	7
20	F	6
19	E	5
18	D	4
17	C	3
16	B	2
15	A	1

4.7.2 LED control register

The LED_LIGHTS register controls four of the LEDs on the Integrator/AP. The LEDs are attached to bit-ports provided by the alphanumeric display. Accesses to these bit ports are managed by the FPGA. Application programs write appropriate values into the LED_LIGHTS register, and the FPGA carries out the necessary transfer to turn the LEDs ON or OFF.

You must allow each transfer the alphanumeric display to complete before writing new data to the display using the following sequence:

1. Check the display status is IDLE (bit 0 = 0) in the LED_ALPHA register.
2. Write a value to the LED_LIGHTS register.

Note

The data from the LED_LIGHTS register is transferred as a serial bit-stream into the alphanumeric display at the same time as the character data in LED_ALPHA.

The bit assignments for the LED_LIGHTS register are shown in Table 4-27.

Table 4-27 LED_LIGHTS register

Bit	Type	Function
3	Read/write	LED3: 0 = OFF 1 = ON.
2	Read/write	LED2: 0 = OFF 1 = ON.
1	Read/write	LED1: 0 = OFF 1 = ON.
0	Read/write	LED0: 0 = OFF 1 = ON.

See *LEDs* on page 1-10 for the location and function of the LEDs.

4.7.3 DIP switch register

The DIP switch register is used to read the 4-pole DIP switch S1. The bit assignments for this register are shown in Table 4-28. A bit reads as 1 when the associated switch is ON and 0 when the switch is OFF.

Table 4-28 LED_SWITCH register

Bit	Type	Function
3	Read-only	Switch pole 4 (S1-4)
2	Read-only	Switch pole 3 (S1-3)
1	Read-only	Switch pole 2 (S1-2)
0	Read-only	Switch pole 1 (S1-1)

See *Setting the DIP switches* on page 2-6 for the location and function of the switches.

4.8 Interrupt controller registers

The system controller FPGA provides interrupt handling for up to four processors. For each processor there is a 22-bit IRQ and 22-bit FIQ controller with the IRQ and FIQ outputs from the FPGA being assigned to the processors as follows:

- IRQ0 and FIQ0 to the processor on core module 0
- IRQ1 and FIQ1 to the processor on core module 1
- IRQ2 and FIQ2 to the processor on core module 2
- IRQ3 and FIQ3 to the processor on core module 3.

See *Interrupt controller* on page 3-19 for a description of the interrupt controller hardware.

4.8.1 About the IRQ and FIQ control registers

The IRQ and FIQ controllers each provide three registers for controlling and handling interrupts. These are:

- status register
- raw status register
- enable register, accessed using the enable set and enable clear locations.

The way that the interrupt enable, clear, and status bits function for each interrupt is illustrated in Figure 4-5.

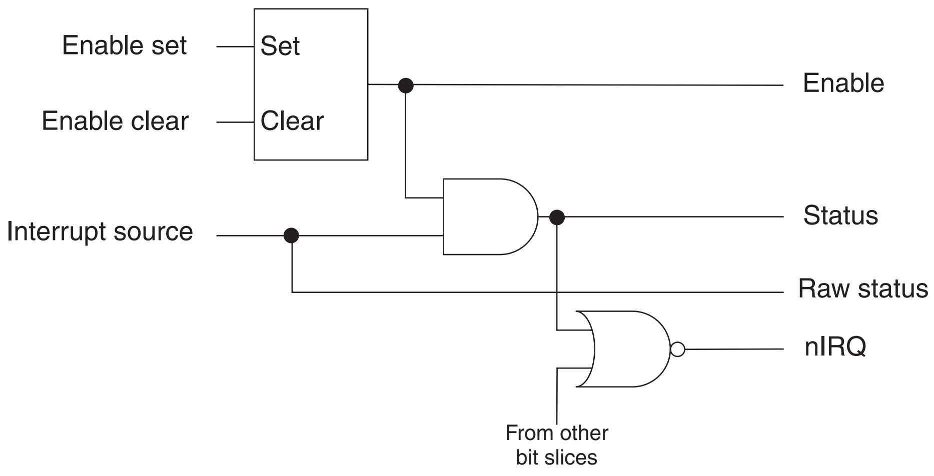


Figure 4-5 Interrupt control

The following subsections describe each of these registers. The headings for the descriptions contain the generic register name for both IRQ and FIQ types, and address offset from the base address of each interrupt controller. The full addresses are shown in Table 4-30 on page 4-32 and Table 4-31 on page 4-33.

IRQx_STATUS/FIQx_STATUS (+0x0)

The status register contains the logical AND of the bits in the raw status register and the enable register.

IRQx_RAWSTAT/FIQx_RAWSTAT (+0x4)

The raw status register indicates the signal levels on the interrupt request inputs. A bit set to 1 indicates that the corresponding interrupt request is active.

IRQx_ENABLESET/FIQx_ENABLESET (+0x8)

The enable set location is used to set bits in the enable register as follows:

- Set bits in the enable register by writing to the ENABLESET location for the required IRQ or FIQ controller:

  1 = SET the associated bit in the enable register

  0 = leave the associated bit in the enable register unchanged.

- Read the current state of the enable register bits from the ENABLESET location.

IRQx_ENABLECLR/FIQx_ENABLECLR (+0xC)

The enable clear location is used to clear bits in the enable register as follows:

- Clear bits in the enable register by writing to the ENABLECLR location

  1 = CLEAR the associated bit in the enable register for the required IRQ or FIQ controller

  0 = leave the associated bit in the enable register unchanged.

4.8.2 IRQ and FIQ register bit assignments

The bit assignment for interrupts in the status, raw status, and enable registers for each of the interrupt controllers is similar and is shown in Table 4-29.

Table 4-29 IRQ register bit assignments

Bit	Name	Function
21	EXTINT	External interrupt reserved for external sources
20	PCILBINT	PCI local bus fault interrupt
19	ENUMINT	CompactPCI auxiliary interrupt (ENUM#)
18	DEGINT	CompactPCI auxiliary interrupt (DEG#)
17	LINT	V3 PCI bridge interrupt
16	PCIINT3	PCI bus (INTD#)
15	PCIINT2	PCI bus (INTC#)
14	PCIINT1	PCI bus (INTB#)
13	PCIINT0	PCI bus (INTA#)
12	EXPINT3	Logic module 3 interrupt
11	EXPINT2	Logic module 2 interrupt
10	EXPINT1	Logic module 1 interrupt
9	EXPINT0	Logic module 0 interrupt
8	RTCINT	Real time clock interrupt
7	TIMERINT2	Counter-timer 2 interrupt
6	TIMERINT1	Counter-timer 1 interrupt
5	TIMERINT0	Counter-timer 0 interrupt
4	MOUSEINT	Mouse interrupt
3	KBDINT	Keyboard interrupt
2	UARTINT1	UART 1 interrupt
1	UARTINT0	UART 0 interrupt
0	SOFTINT	Software interrupt (see *Software interrupts* on page 4-34)

4.8.3 IRQ registers

The register set of each IRQ controller is shown in Table 4-30. Not shown in this table are the software interrupt registers that are described in *Software interrupts* on page 4-34.

Table 4-30 IRQ register addresses

Address	Name	Type	Size	Function
`0x14000000`	IRQ0_STATUS	Read	22	IRQ0 status
`0x14000004`	IRQ0_RAWSTAT	Read	22	IRQ0 interrupt request status
`0x14000008`	IRQ0_ENABLESET	Read/write	22	IRQ0 enable set
`0x1400000C`	IRQ0_ENABLECLR	Write	22	IRQ0 enable clear
`0x14000040`	IRQ1_STATUS	Read	22	IRQ1 status register
`0x14000044`	IRQ1_RAWSTAT	Read	22	IRQ1 raw status
`0x14000048`	IRQ1_ENABLESET	Read/write	22	IRQ1 enable set
`0x1400004C`	IRQ1_ENABLECLR	Write	22	IRQ1 enable clear
`0x14000080`	IRQ2_STATUS	Read	22	IRQ2 status register
`0x14000084`	IRQ2_RAWSTAT	Read	22	IRQ2 raw status
`0x14000088`	IRQ2_ENABLESET	Read/write	22	IRQ2 enable set
`0x1400008C`	IRQ2_ENABLECLR	Write	22	IRQ2 enable clear
`0x140000C0`	IRQ3_STATUS	Read	22	IRQ3 status register
`0x140000C4`	IRQ3_RAWSTAT	Read	22	IRQ3 raw status
`0x140000C8`	IRQ3_ENABLESET	Read/write	22	IRQ3 enable set
`0x140000CC`	IRQ3_ENABLECLR	Write	22	IRQ3 enable clear

4.8.4 FIQ registers

The register set of each FIQ controller is shown in Table 4-31.

Table 4-31 FIQ registers addresses

Address	Name	Type	Size	Function
`0x14000020`	FIQ0_STATUS	Read-only	22	FIQ0 status
`0x14000024`	FIQ0_RAWSTAT	Read-only	22	FIQ0 raw status
`0x14000028`	FIQ0_ENABLESET	Read/write	22	FIQ0 enable set
`0x1400002C`	FIQ0_ENABLECLR	Write-only	22	FIQ0 enable clear
`0x14000060`	FIQ1_STATUS	Read-only	22	FIQ1 status register
`0x14000064`	FIQ1_RAWSTAT	Read-only	22	FIQ1 raw status
`0x14000068`	FIQ1_ENABLESET	Read/write	22	FIQ1 enable set
`0x1400006C`	FIQ1_ENABLECLR	Write-only	22	FIQ1 enable clear
`0x140000A0`	FIQ2_STATUS	Read-only	22	FIQ2 status register
`0x140000A4`	FIQ2_RAWSTAT	Read-only	22	FIQ2 raw status
`0x140000A8`	FIQ2_ENABLESET	Read/write	22	FIQ2 enable set
`0x140000AC`	FIQ2_ENABLECLR	Write-only	22	FIQ2 enable clear
`0x140000E0`	FIQ3_STATUS	Read-only	22	FIQ3 status register
`0x140000E4`	FIQ3_RAWSTAT	Read-only	22	FIQ3 raw status
`0x140000E8`	FIQ3_ENABLESET	Read/write	22	FIQ3 enable set
`0x140000EC`	FIQ3_ENABLECLR	Write-only	22	FIQ3 enable clear

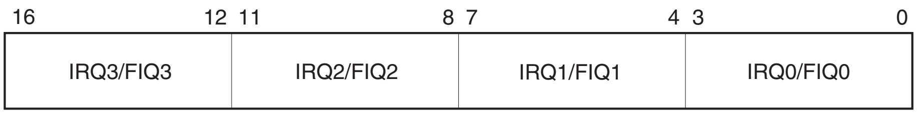
4.8.5 Software interrupts

The interrupt controller provides a 16-bit software interrupt register that is multiply mapped to appear at an offset of `0x10` from the four IRQ register-set base addresses. The software interrupt registers are accessed at the locations shown in Table 4-32.

Table 4-32 Software interrupt register

Address	Name	Type	Size	Function
`0x14000010`	INT_SOFTSET	Read/write	16	Software interrupt set
`0x14000050`				
`0x14000090`				
`0x140000D0`				
`0x14000014`	INT_SOFTCLEAR	Write	16	Software interrupt clear
`0x14000054`				
`0x14000094`				
`0x140000D4`				

The software interrupt register contains four fields, each containing 4 bits. The bits within each field are logically ORed so that, if the software interrupt is enabled, an interrupt is generated for a processor if any bit within the associated field is set.



For example, if any of bits [11-8] are set then the SOFTINT bit in both IRQ2_RAWSTAT and FIQ2_RAWSTAT are set.

The following subsections describe the function of each of the soft interrupt register locations. The headings for the descriptions contain the generic register name. There is one set and one clear location. These can be accessed at the address offset shown from the base address of each IRQ interrupt controller. The full addresses for the these locations are shown in Table 4-32.

INT_SOFTSET (+0x10)

The software interrupt set locations are used to set bits in the software interrupt (SOFT_INT) register as follows:

- Set bits in the SOFT_INT register by writing to the SOFTSET location:

  1 = SET the associated bit in the SOFT_INT register

  0 = leave the associated bit in the SOFT_INT register unchanged.

- Read the current state of the SOFT_INT register bits from the SOFTSET location.

INT_SOFTCLEAR (+0x14)

The software interrupt clear locations are used to clear bits in the software interrupt (SOFT_INT) register as follows:

- Clear bits in the SOFT_INT register by writing to the SOFTCLR location:

  1 = CLEAR the associated bit in the SOFT_INT register.

  0 = leave the associated bit in the SOFT_INT register unchanged.

Using the software interrupt register

Using the SOFT_INT register, it is possible for one processor to interrupt any other, or all four processors simultaneously, with a single write. Also, because there are 4 bits for each processor, it is possible to devise a software protocol to determine the source of the interrupt.

For example, in order to pass messages between processors, you would want to know which processor generated an interrupt. To do this, assign the software interrupts as follows:

- bits 0, 4, 8, and 12 assigned to interrupts *from* processor 0
- bits 1, 5, 9, and 13 assigned to interrupts *from* processor 1
- bits 2, 6, 10, and 14 assigned to interrupts *from* processor 2
- bits 3, 7, 11, and 13 assigned to interrupts *from* processor 3.

Using this protocol, the interrupted processor is able to determine the source of the interrupt by reading the software interrupt register and then take the appropriate action. For example, reading a shared data structure in memory.

The bit assignment in the software interrupt register is arbitrary enabling you to devise a protocol to suit your application.

4.9 Peripheral registers

This section provides an overview of the peripheral registers implemented in the system controller FPGA. These are:

- *GPIO*
- *Real time clock* on page 4-38
- *UART registers* on page 4-41
- *Keyboard and mouse interfaces* on page 4-51.

Table 4-33 shows the base address for each of these devices.

Table 4-33 Peripheral register locations

Base address	Size	Description
`0x1B000000`	16MB	GPIO
`0x19000000`	16MB	Mouse
`0x18000000`	16MB	Keyboard
`0x17000000`	16MB	UART1
`0x16000000`	16MB	UART0
`0x15000000`	16MB	RTC

4.9.1 GPIO

The GPIO provides 32 general-purpose input and output signals that are connected to the EXPB logic module connector. See *Logic module connector EXPB* on page A-8. The registers used to control these pins are shown in Table 4-34.

Table 4-34 GPIO register summary

Address	Name	Type	Size	Function
`0x1B000000`	GPIO_DATASET	Write	32	Data output set
	GPIO_DATAIN	Read	32	Read data input pins
`0x1B000004`	GPIO_DATACLR	Write	32	Data register output clear
	GPIO_DATAOUT	Read	32	Read data output pins
`0x1B000008`	GPIO_DIRN	Read/write	32	Data direction

Data output set register

The GPIO_DATASET location is used to set individual GPIO output bits as follows:

1 = SET the associated GPIO output bit

0 = leave the associated GPIO bit unchanged.

Read data input register

Read the current state of the GPIO input bits from this location.

Data register output clear

The GPIO_DATACLR location is used to clear individual GPIO output bits as follows:

1 = CLEAR the associated GPIO output bit

0 = leave the associated GPIO bit unchanged.

Read data output pins

Read the current state of the GPIO output bits from this location.

Data direction

The GPIO_DIRN location is used to set the direction of each GPIO pin as follows:

1 = pin is an output

0 = pin is an input (default).

Figure 4-6 on page 4-37 shows the data direction control for one GPIO bit.

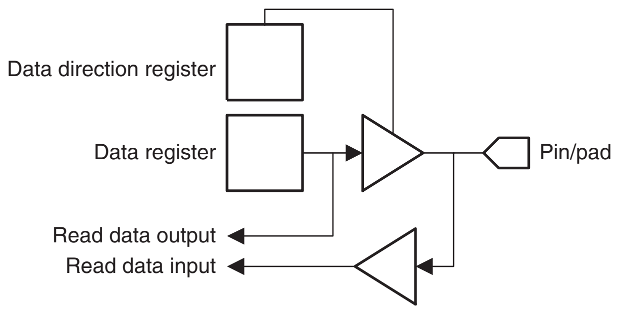


Figure 4-6 GPIO direction control(1 bit)

4.9.2 Real time clock

This section provides a memory map and functional overview of the RTC registers:

- *RTC data register* on page 4-38
- *RTC match register* on page 4-39
- *RTC interrupt status register and Interrupt clear register* on page 4-39
- *RTC load register* on page 4-39
- *RTC control register* on page 4-40.

For more detailed information, please refer to the *ARM PrimeCell RTC (PL030) Technical Reference Manual*. The RTC registers are summarized in Table 4-35.

Table 4-35 RTC register summary

Address	Name	Type	Size	Function
`0x15000000`	RTC_DR	Read	32	Data register
`0x15000004`	RTC_MR	Read/write	32	Match register
`0x15000008`	RTC_STAT	Read	1	Interrupt status register
	RTC_EOI	Write	0	Interrupt clear register
`0x1500000C`	RTC_LR	Read/write	32	Load register
`0x15000010`	RTC_CR	Read/write	1	Control register

RTC data register

The RTC data register is a 32-bit read-only data register. Reads from this register return the current counter value. Table 4-36 describes the RTC data register bits.

Table 4-36 RTC_DR register

Bits	Name	Access	Function
31:0	RTC data register	Read	Current counter value.

RTC match register

The RTC match register is a 32-bit read/write register. Writes to this register load the match register. Reads return the last written value. Table 4-37 describes the RTC match register bits.

Table 4-37 RTC_MR register

Bits	Name	Type	Function
31:0	RTC match register	Read/write	Match register.

RTC interrupt status register and Interrupt clear register

The RTC status and RTC interrupt clear registers share the same location. The write location has no physical storage element but clears the **RTCINT** interrupt line and the corresponding status bit. A read from bit 0 returns the value of **RTCINT**.

Table 4-38 RTC_STAT/RTC_EOI registers

Bits	Name	Access	Function
31:0	RTC_EOI	Write	A write to this register clears **RTCINT**, regardless of data value written.
31:1	Reserved	Read	Reserved, unpredictable when read.
0	Interrupt status (**RTCINT**)	Read	This bit is set to 1 if **RTCINT** interrupt is asserted.

RTC load register

The RTC load register is a 32-bit read/write load register. Writes to this register load the counter (the loads do not occur immediately). The contents of this register are loaded into an intermediate register before updating the free-running counter on the rising edge of a 1MHz clock generated internally by the system controller FPGA. Reads return the last written value.

Table 4-39 RTC_LR register

Bits	Name	Type	Function
31:0	RTC load register	Read/write	Counter load register.

RTC control register

The RTC control register is a 1-bit read/write register that controls the masking of the RTC interrupt. Writing 1 to bit position 0 enables the interrupt. Writing 0 disables the interrupt. Reads from this register return the last value written at bit position 0.

Table 4-40 RTC_CR register

Bits	Name	Type	Function
31:1	Reserved	Read/write	Reserved, read unpredictable. Preserve using read-modify-writes.
0	Match interrupt enable (MIE)	Read/write	If this bit is set to 1, the match interrupt is enabled.

4.9.3 UART registers

This section provides an memory map and functional overview of the UART registers:

- *UARTx receive data register* on page 4-42
- *UARTx receive status register* on page 4-42
- *UARTx error clear register* on page 4-44
- *UARTx line control register, high byte* on page 4-44
- *UARTx line control register, middle byte* on page 4-45
- *UARTx line control register, low byte* on page 4-46
- *UARTx control register* on page 4-47
- *UARTx flag register* on page 4-48
- *UARTx interrupt identification and clear registers* on page 4-50

For more detailed information, refer to the *ARM PrimeCell UART (PL010) Technical Reference Manual*.

The UART registers are summarized in Table 4-41. The following subsections describe each of these registers. The headings for the descriptions contain the generic register name and address offset from the base address of each UART.

Table 4-41 UART register summary

UART0 address	UART1 address	Name	Access	Function
`0x16000000`	`0x17000000`	UARTx_DR	Read	Receive data
			Write	Transmit data
`0x16000004`	`0x17000004`	UARTx_RSR	Read	Receive status register
		UARTx_ECR	Write	Error clear register
`0x16000008`	`0x17000008`	UARTx_LCRH	Read/write	Line control register, high byte
`0x1600000C`	`0x1700000C`	UARTx_LCRM	Read/write	Line control register, middle byte
`0x16000010`	`0x17000010`	UARTx_LCRL	Read/write	Line control register, low byte
`0x16000014`	`0x17000014`	UARTx_CR	Read/write	Control register
`0x16000018`	`0x17000018`	UARTx_FR	Read	Flag register
`0x1600001C`	`0x1700001C`	UARTx_IIR	Read	Interrupt identification register
		UARTx_ICR	write	Interrupt clear register

UARTx receive data register

The UART receive data register stores the last byte received by the UART as follows:

- For bytes to be transmitted:
  - FIFOs enabled. Data written to this location is pushed into the transmit FIFO.
  - FIFOs disabled. Data is stored in the transmitter holding register (the bottom byte of the transmit FIFO).
- For received byte:
  - FIFOs enabled. The data byte is extracted, and a 3-bit status (break, frame and parity) is pushed into the 11-bit wide receive FIFO.
  - FIFOs disabled. The data byte and status are stored in the receiving holding register (the bottom byte of the receive FIFO).

The received data byte is read by performing reads from the UARTx_DR register and then reading the corresponding status information from the UARTx_RSR register.

Table 4-42 UARTx_DR register

Bits	Name	Type	Function
7:0	DATA	Read/write	Receive (read) data character Transmit (write) data character

Note

Read a received character from UARTx_DR first followed by the status error associated with that character from the UARTx_RSR. This read sequence cannot be reversed.

UARTx receive status register

The UART receive status register is a 4-bit read-only register containing status information associated with the data character read from UARTx_DR immediately prior to reading this register.

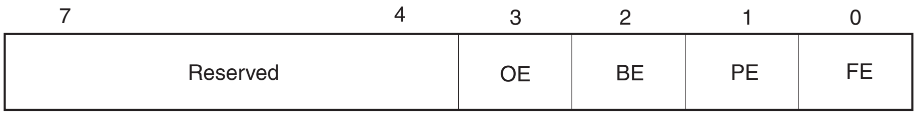


Table 4-43 describes the UART receive status register bits. All the bits are cleared to 0 on reset.

Table 4-43 UARTx_RSR

Bit	Name	Type	Function
7:4	Reserved	Read	Reserved, unpredictable when read.
3	OE	Read	Overrun Error. This bit is set to 1 when data is received and the FIFO is already full. This bit is cleared to 0 by a write to UARTx_ECR. The FIFO contents are still valid because data cannot be written when the FIFO is full. Only the contents of the shift register are overwritten. To empty the FIFO, the CPU must read the data.
2	BE	Read	Break Error. This bit is set to 1 if a break condition is detected. This bit is cleared to 0 by a write to UARTx_ECR.
1	PE	Read	Parity Error. This bit is set to 1 to indicate that the parity of the received data character does not match the parity selected in UARTx_LCRH (bit 2). This bit is cleared to 0 by a write to UARTx_ECR.
0	FE	Read	Framing Error. This bit is set to 1 to indicate that the received character did not have a valid stop bit (a valid stop bit is 1). This bit is cleared to 0 by a write to UARTx_ECR.

Note

The received data character must be read first from UARTx_DR and the error status associated with that data character can be read from UARTx_RSR. This read sequence cannot be reversed.

UARTx error clear register

The UART error clear register is a write-only location. Write any value to this location to clear the framing, parity, break, and overrun errors bits in the UARTx_RSR register.

UARTx line control register, high byte

Note

The UARTx_LCRH, UARTx_LCRM, and UARTx_LCRL locations together access the 23-bit wide register (UARTx_LCR). This register is updated on the write strobe generated by writing to UARTx_LCRH. Therefore, writes to UARTx_LCRM or UARTx_BLCRL, must always be followed by a write to UARTx_LCRH.

The UART line control high byte location accesses bits 23 to 16 of the UART bit rate and line control register, UARTx_LCR. All the bits are cleared to 0 after a reset.

7	6	5	4	3	2	1	0
Reserved	WLEN		FEN	STP2	EPS	PEN	BRK

Table 4-44 describes the UART line control register high byte bits.

Table 4-44 UARTx_LCRH register

Bits	Name	Type	Function
7	Reserved	Read/write	Reserved, do not modify, read as 0.
6:5	WLEN	Read/write	Word length [1:0] The select bits indicate the number of data bits transmitted or received in a frame as follows: 11 = 8 bits 10 = 7 bits 01 = 6 bits 00 = 5 bits.
4	FEN	Read/write	Enable FIFOs Set to 1 to enable the transmit and receive FIFO buffers (FIFO mode). Cleared to 0 to disable the FIFOs (character mode).

Table 4-44 UARTx_LCRH register (continued)

Bits	Name	Type	Function
3	STP2	Read/write	Two Stop Bits Select Set to 1 to transmit two stop bits at the end of each frame. The receive logic does not check for two stop bits being received.
2	EPS	Read/write	Even Parity Select Set to 1 to select even parity generation and checking. Cleared to 0 to select odd parity generation and checking. This bit has no effect if parity is disabled by the Parity Enable (bit 1) being cleared to 0.
1	PEN	Read/write	Parity Enable Set to 1 to enable parity checking and generation.
0	BRK	Read/write	Send Break Set to 1 to drive a continuous low level output on the UARTx_TXD. This bit must be asserted for at least one complete frame transmission time in order to generate a break condition. For normal use, this bit must be cleared to 0.

UARTx line control register, middle byte

The UART line control middle byte location accesses bits 15 to 8 of the UARTx_LCR register, as shown in Table 4-45. All the bits are cleared to 0 on reset.

Table 4-45 UARTx_LCRM register

Bit	Name	Type	Function
7:0	Baud Rate [15:8]	Read/write	Most significant byte of baud rate divisor. These bits are cleared on reset.

UARTx line control register, low byte

The UART line control register low byte location accesses bits 7 to 0 of the UARTx_LCR register, as shown in Table 4-46. All the bits are cleared to 0 on reset.

Table 4-46 UARTx_LCML register

Bit	Name	Type	Function
7:0	Baud Rate [7:0]	Read/write	Least significant byte of baud rate divisor. These bits are cleared to 0 on reset.

The baud rate divisor is calculated as follows:

Baud rate divisor = ($F_{UARTCLK}$/ (16 * Baud rate)) –1

where $F_{UARTCLK}$ is the UART reference clock frequency.

Table 4-47 shows some typical bit rates and their corresponding divisors (in hexadecimal) for a UART clock frequency of 14.7456 MHz. A divisor value of zero is illegal and prevents transmission or reception.

Table 4-47 Typical baud rates and divisors

Baud rate	Divisor	UARTx_LCRM	UARTx_LCRL
460800	`0x0001`	`0x00`	`0x01`
230400	`0x0003`	`0x00`	`0x03`
115200	`0x0007`	`0x00`	`0x07`
76800	`0x000B`	`0x00`	`0x0B`
57600	`0x000F`	`0x00`	`0x0F`
38400	`0x0017`	`0x00`	`0x17`
19200	`0x002F`	`0x00`	`0x2F`
14400	`0x003F`	`0x00`	`0x3F`
9600	`0x005F`	`0x00`	`0x5F`
2400	`0x017F`	`0x01`	`0x7F`
1200	`0x02FF`	`0x02`	`0xFF`

Note

The UART operates at up to 460800 baud but the line driver is only guaranteed up to 250Kbaud.

UARTx control register

The UART control register is an 8-bit read/write register that is used to set a number of operating parameters.

7	6	5	4	3	2	1	0
LBE	RTIE	TIE	RIE	MSIE	Reserved	Reserved	UARTEN

Table 4-48 describes the bits in the UART control register. All the bits are cleared to 0 on reset.

Table 4-48 UARTx_CR register

Bit	Name	Type	Function
7	LBE	Read/write	Loop Back Enable. When this bit is set to 1 and the SIR Enable bit is 0, the internal transmit path is fed through to the receive path. This bit is cleared to 0 on reset, disabling loopback mode.
6	RTIE	Read/write	Receive Timeout Interrupt Enable. If this bit is set to 1, the receive timeout interrupt is enabled.
5	TIE	Read/write	Transmit Interrupt Enable. If this bit is set to 1, the transmit interrupt is enabled.
4	RIE	Read/write	Receive Interrupt Enable. If this bit is set to 1, the receive interrupt is enabled.
3	MSIE	Read/write	Modem Status Interrupt Enable. If this bit is set to 1, the modem status interrupt is enabled.

Table 4-48 UARTx_CR register (continued)

Bit	Name	Type	Function
2	Unused	Read/write	This bit selects the IrDA encoding mode (unused by the Integrator). Always write with 0.
1	Unused	Read/write	This bit is the SIR enable bit (unused by the Integrator). Always write with 0.
0	UARTEN	Read/write	UART Enable. If this bit is set to 1, the UART is enabled. Data transmission and reception occurs for either UART signals or SIR signals according to the setting of SIR Enable (bit 1).

UARTx flag register

UARTx_FR register is an 8-bit read-only register that contains status flags.

7	6	5	4	3	2	1	0
TXFE	RXFF	TXFF	RXFE	BUSY	DCD	DSR	CTS

Table 4-49 describes the UART status flags. After reset **TXFF**, **RXFF** and **BUSY** are 0, **TXFE** and **RXFE** are1.

Table 4-49 UARTx_FR register

Bits	Name	Type	Function
7	TXFE	Read	Transmit FIFO Empty. If the FIFO is disabled, this bit is SET when the transmit holding register is empty. If the FIFO is enabled, this bit is SET when the transmit FIFO is empty.
6	RXFF	Read	Receive FIFO Full. If the FIFO is disabled, this bit is SET when the receive holding register is full. If the FIFO is enabled, this bit is SET when the receive FIFO is full.
5	TXFF	Read	Transmit FIFO Full. If the FIFO is disabled, this bit is set when the transmit holding register is full. If the FIFO is enabled, the TXFF bit is set when the transmit FIFO is full.
4	RXFE	Read	Receive FIFO Empty. If the FIFO is disabled, this bit is set when the receive holding register is empty. If the FIFO is enabled, the RXFE bit is set when the receive FIFO is empty.
3	BUSY	Read	UART Busy. If this bit is set to 1, the UART is busy transmitting data. This bit remains set until the complete byte, including all the stop bits, has been sent from the shift register. This bit is set as soon as the transmit FIFO becomes non-empty (regardless of whether the UART is enabled or not).

Table 4-49 UARTx_FR register (continued)

Bits	Name	Type	Function
2	DCD	Read	Data Carrier Detect This bit is the complement of the UART data carrier detect modem status input. That is, the bit is 1 when the modem status input is 0.
1	DSR	Read	Data Set Ready. This bit is the complement of the UART data set ready modem status input. That is, the bit is 1 when the modem status input is 0.
0	CTS	Read	Clear To Send. This bit is the complement of the UART clear to send modem status input. That is, the bit is 1 when the modem status input is 0.

UARTx interrupt identification and clear registers

This location provides access to the read-only UART interrupt identification and the write-only interrupt clear register. Reading this location returns the UART interrupt flags. A write of any value to UARTx_ICR clears the interrupts.

7 4	3	2	1	0
Reserved	RTIS	TIS	RIS	MIS

The interrupt identification bits are as shown in Table 4-50. All the bits are cleared to 0 when reset.

Table 4-50 UARTx_IIR register

Bit	Name	Type	Function
7:4	Reserved	Read	Reserved, unpredictable when read.
3	RTIS	Read	Receive Timeout Interrupt Status. This bit is set to 1 if the **UARTRTINT** receive interrupt is asserted.

Table 4-50 UARTx_IIR register (continued)

Bit	Name	Type	Function
2	TIS	Read	Transmit Interrupt Status. This bit is set to 1 if the **UARTTXINT** transmit interrupt is asserted.
1	RIS	Read	Receive Interrupt Status. This bit is set to 1 if the **UARTRXINT** receive interrupt is asserted.
0	MIS	Read	Modem Interrupt Status. This bit is set to 1 if the **UARTMSINT** modem status interrupt is asserted.

4.9.4 Keyboard and mouse interfaces

This section provides a memory map and functional overview of the KMI registers:

- *KMI control register* on page 4-52
- *KMI status register.* on page 4-53
- *KMI data register* on page 4-54
- *KMI clock divisor register* on page 4-54
- *KMI interrupt identification register* on page 4-54.

For more detailed information, please refer to the *KMI (PL050) Technical Reference Manual*. The KMI registers are summarized in Table 4-51.

Table 4-51 KMI register summary

Keyboard Address	Mouse Address	Name	Type	Description
`0x18000000`	`0x19000000`	KMI_CR	Read/write	Control register
`0x18000004`	`0x19000004`	KMI_STAT	Read	Status register
`0x18000008`	`0x19000008`	KMI_DATA	Read	Received data register
			Write	Transmit data register
`0x1800000C`	`0x1900000C`	KMI_CLKDIV	Read/write	Clock divisor register
`0x18000010`	`0x19000010`	KMI_IR	Read	Interrupt identification register

KMI control register

The KMI control register is an 8-bit read/write register that control various functions within the KMI.

7	6	5	4	3	2	1	0
Reserved		TYPE	RIEN	TIEN	KMIEN	KMID	FKMIC

The KMI control register bits are described in Table 4-52. All bits are cleared to 0 on reset.

Table 4-52 KMI_CR register

Bits	Name	Access	Function
7:6	Reserved	read/write	Reserved, read unpredictable, write as 0.
5	TYPE	read/write	Keyboard type: 0 = PS2/AT mode (default) 1 = No line control bit mode.
4	RIEN	read/write	Receiver interrupt enable. Set to 1 to enable the KMI receiver interrupt.
3	TIEN	read/write	Transmitter interrupt enable. Set to 1 to enable the KMI transmitter interrupt.
2	KMIEN	read/write	KMI enable. Set to 1 to enable the KMI.
1	FKMID	read/write	Force KMI data LOW. Set to 1 to force PS2 the KMI data LOW.
0	FKMIC	read/write	Force KMI clock LOW. Set to 1 to force the KMI clock LOW.

KMI status register.

The KMI status register is a 8-bit, read-only register that indicates the status of the different lines in the KMI.

7	6	5	4	3	2	1	0
Reserved	TXEMPTY	TXBUSY	RXFULL	RXBUSY	RXPARITY	KMICKIN	KMIDIN

The KMI status register bits are described in Table 4-53.

Table 4-53 KMI_STAT register

Bits	Name	Type	Function
7	Reserved	Read	Reserved, read unpredictable.
6	TXEMPTY	Read	Transmit register status: 0 = Transmit register full 1 = Transmit register empty, ready to be written.
5	TXBUSY	Read	KMI data transmitter status: 0 = Idle 1 = Busy, sending data.
4	RXFULL	Read	Receiver register status: 0 = Receive register empty. 1 = Receive register full, ready to be read.
3	RXBUSY	Read	KMI receiver status: 0 = Idle 1 = Busy, receiving data.
2	RXPARITY	Read	Parity bit for the last received data byte (odd parity).
1	KMIC	Read	Status of the KMI clock input line after synchronizing and sampling.
0	KMID	Read	Status of the KMI data input line after synchronizing.

KMI data register

The KMI transmit/receive data register is 8-bits wide. When KMI_DATA is read, the received data is accessed. When KMI_DATA is written to, it is loaded into the transmit register and then serially shifted out onto the KMI DATA pin.

Table 4-54 KMI_DATA transmit/receive data register read/write bits

Bits	Name	Type	Function
7:0	KMIDATA	Read/write	Receive/transmit register: Read - Receive register. Write - Transmit register.

KMI clock divisor register

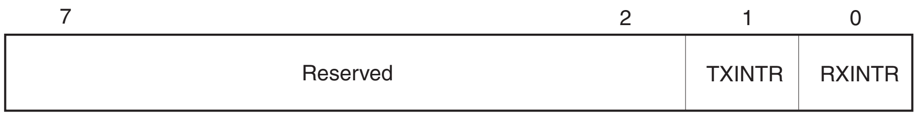
The KMI clock divisor register is used to specify the division factor by which the input **KMIREFCLK** is divided to provide an 8MHZ internal clock signal. The frequency of the clock signal supplied to this input is fixed at 24MHz. This register must be programmed with a divisor value of 2 order to generate the internal 8MHz signal.

Table 4-55 KMI_CLKDIV register read/write bits

Bits	Name	Type	Function
3:0	KMICLKDIV	Read/write	Clock divisor register. The KMI divides the input clock by (1+KMICLKDIV) to generate the 8MHZ internal clock.

KMI interrupt identification register

The KMI interrupt identification register is an 8-bit read-only register that contains interrupt status information for the KMI.



The KMI interrupt identification register bits are described in Table 4-56.

Table 4-56 KMI_IR register

Bits	Name	Type	Function
7:2	Reserved	Read	Reserved, read unpredictable.
1	TXINTR	Read	This bit is set to 1 if the KMI transmit interrupt is asserted.
0	RXINTR	Read	This bit is set to 1 if the KMI receive interrupt is asserted.

Chapter 5
PCI Subsystem

This chapter describes the PCI subsystem of the Integrator/AP. It contains the following sections:

- *About the PCI subsystem* on page 5-2
- *System to local bus bridge operation* on page 5-6
- *V360EPC PCI to Host Bridge operation* on page 5-9
- *PCI to PCI bridge operation* on page 5-15
- *Initializing the PCI subsystem* on page 5-16
- *PCI subsystem interrupts* on page 5-17.

5.1 About the PCI subsystem

The PCI subsystem enables you to add expansion cards to the Integrator/AP or to connect it to a CompactPCI backplane. The Integrator/AP is a slot 1 CompactPCI card.

The basic operations of the PCI subsystem can be described in terms of the:

- topology of the PCI subsystem
- cycle types supported
- function of the PCI bridges
- arbitration on various bus segments within the subsystem
- PCI subsystem interrupts.

The remaining sections in this chapter describe these operations in more detail.

5.1.1 PCI subsystem topology

The major parts of the PCI subsystem are:

- System bus to local bus bridge (part of the system controller FPGA)
- V-cubed V360EPC PCI Host Interface controller
- PCI expansion bus with 3 expansion slots and 5-slot arbiter
- Intel 21152 PCI-PCI Bridge
- CompactPCI system controller providing:
  - CompactPCI bus arbiters
  - Clock generator.

The architecture of the PCI subsystem is illustrated in Figure 5-1.

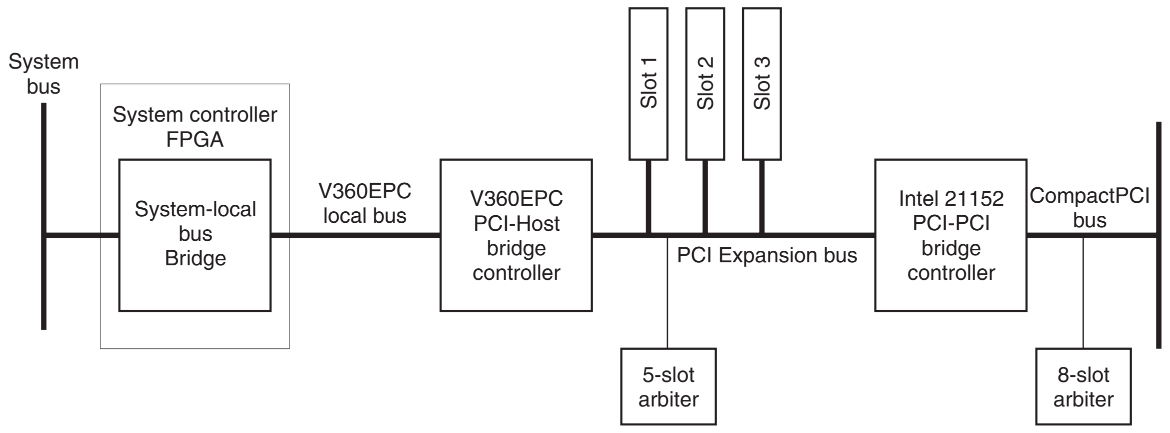


Figure 5-1 PCI subsystem

5.1.2 PCI subsystem access types

The PCI subsystem supports several access types:

- reads and writes by system bus initiators to local PCI bus targets
- reads and writes by system bus initiators to CompactPCI bus targets
- reads and writes by PCI expansion bus initiators to system bus targets
- reads and writes by PCI expansion bus initiators to CompactPCI bus targets
- reads and writes by PCI expansion bus initiators to PCI expansion bus targets
- reads and writes by CompactPCI bus initiators to system bus targets
- reads and writes by CompactPCI bus initiators to PCI expansion bus targets
- reads and writes by CompactPCI bus initiators to CompactPCI bus targets.

5.1.3 PCI bridges

The function of a bridge is pass valid accesses in both directions between the bus nearest to the initiator and the bus nearest to the target. The system to local bus bridge (within the system controller FPGA) provides the host (or primary) bus for the V360EPC bridge. The system-local bus bridge recognizes addresses in the range `0x40000000` to `0x7FFFFFFF` as being intended for a target within the PCI address space of the Integrator memory map, and passes accesses within this region through to the host-PCI bridge without address translation.

The V360EPC host-to-PCI bridge passes valid accesses from the local bus through to its secondary bus, in this case the PCI expansion bus. The PCI to PCI bridge, in turn, passes accesses intended for its secondary bus side through to the CompactPCI. This process is repeated as required on other CompactPCI boards until the target device is finally reached, and the process operates in reverse for accesses from PCI initiators.

Some bridges are capable of being programmed to perform address translation or to restrict the types of PCI access types available.

The mapping of system bus addresses to PCI address space is defined within registers in the V360EPC by the resident firmware during system initialization. This mapping is shown in Figure 5-4 (see *Local bus to PCI windows* on page 5-10).

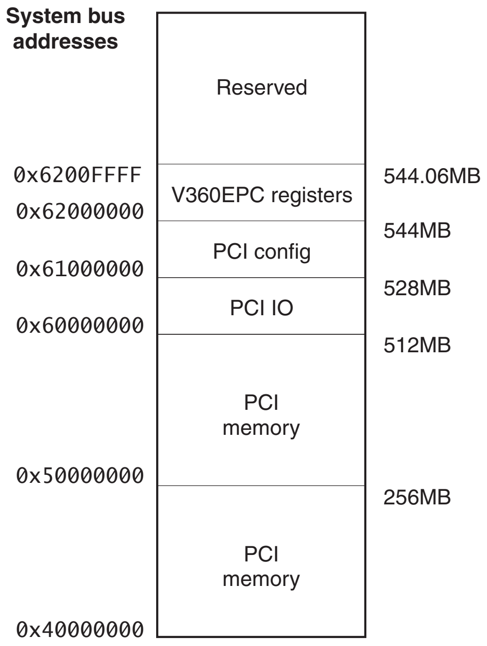


Figure 5-2 System bus to PCI space mapping

5.1.4 PCI bus arbitration

The PCI subsystem provides three separate arbiters. These are for the:

- local bus
- PCI expansion bus
- CompactPCI bus.

The local bus arbiter is part of the system controller FPGA.

The PCI expansion bus has a 5-slot arbiter that controls accesses by the two bridges at either end of the PCI expansion bus and devices plugged into the PCI expansion slots.

The CompactPCI interface has a separate 8-slot arbiter to provide arbitration for the CompactPCI bus.

5.2 System to local bus bridge operation

The system bus to local bus bridge supports read and write accesses in both directions, as shown in Figure 5-3.

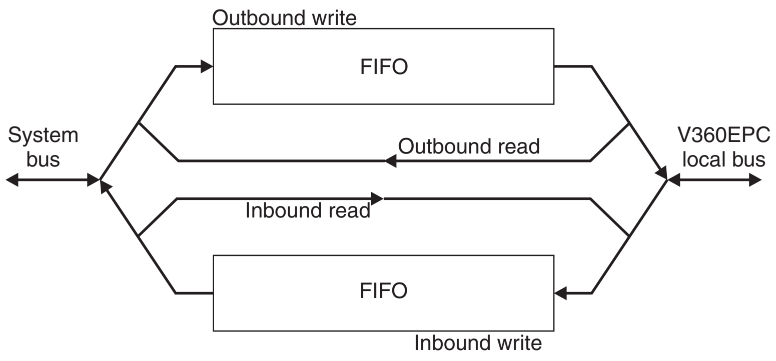


Figure 5-3 System to local bus bridge transactions

Depending on the direction of transfer, these fall into two general types:

Inbound transactions

Inbound transactions are those performed by the V360EPC on behalf of initiators on its PCI expansion bus side. The system to local bus bridge captures addresses generated by the V360EPC and passes them through to the system bus with no address translation.

Outbound transactions

Outbound transactions are those performed by initiators on the system bus to targets on the PCI expansion bus or CompactPCI bus. Accesses in the range `0x40000000` to `0x7FFFFFFF` from the system bus are passed through to the local bus without translation.

Caution

The V360EPC must not be programmed to generate local bus addresses in the range `0x40000000` - `0x7FFFFFFF` because the operation of the system is unpredictable.

Write transactions in both directions are supported by write FIFOs. These are described in detail below.

5.2.1 Inbound transactions

The V360EPC is configured to operate its local bus interface in Am29K bus mode. This uses word-aligned addresses and four byte lane strobes, allowing easy internal translation to and from PCI cycle types. The AMBA system bus specifies transfer sizes of byte, halfword or word and uses byte addresses that are transfer-size aligned.

The system to local bus bridge handles inbound transactions as follows:

- A local bus write or read that can be translated to a single AMBA data transfer is performed as a single system bus transfer of the selected size.
- A local bus write that cannot be performed as a single AMBA data transfer is translated into two write cycles. For example, a 3-byte transfer is performed as two data transfers, of 1 byte followed by 1 halfword. These are made indivisible on the system bus by using the AMBA bus locking mechanism.
- A local bus read that specifies a pattern of byte lanes that cannot be translated to a single AMBA data transfer is performed as a word transfer and the V360EPC discards the unwanted data.

5.2.2 Outbound transactions

All normal AMBA transactions (as used on the system bus) can be translated to corresponding single local bus cycles.

The local bus does not provide a bus locking facility. Locked transactions, such as the data read and data write of an ARM SWAP instruction, are only indivisible on the system bus. They can be separated on the PCI bus by transactions performed by other PCI bus masters. This can affect how a system with multiple PCI masters performs inter-processor communication. To avoid the effects of this limitation on Integrator system with multiple core modules, do not store inter-processor communication semaphores in PCI memory space.

5.2.3 Write FIFOs

The local bus bridge incorporates FIFOs to support write transactions in both directions. Both FIFOs operate in a similar way as follows:

Writes If there is space in the FIFO for the write transaction, the write is stored in the FIFO that supports transfers in that direction. The write transaction is completed on the initiating bus with no additional wait states.

When there are writes stored in the FIFO, the bridge issues a request to perform the write on the target bus. Writes are carried out in the order they are stored.

Reads Read transactions are not buffered in the FIFOs. For each direction, a read transaction is suspended until the write transactions already stored in the corresponding write FIFO have been completed. This ensures that the order of transactions on the initiating bus is maintained on the target bus.

5.2.4 Local bus arbitration

The local bus bridge provides arbitration between inbound transactions and outbound transactions. It prevents deadlocks by using wait and retract signals and by assigning a higher priority to inbound transactions.

Inbound transaction responses are as follows:

- Write transactions receive a WAIT response until the FIFO has space available.
- Read transactions receive a WAIT response until the FIFO is empty, and until the corresponding read has completed on the system bus and the data is available.

Outbound transaction responses are as follows:

- Write transactions complete on the system bus with no WAIT states, providing the FIFO has space available. If the FIFO is full, they receive a RETRACT response and the cycle must be retried.
- Read transactions receive a RETRACT response unless the FIFO is empty. In this case they receive a WAIT until the corresponding read transfer has completed on the local bus and the data is available.

After a RETRACT response, if no other bus masters require the system bus, then the original master can immediately retry the transaction. The transaction can complete as soon as the FIFO and conditions on the local bus permit.

5.3 V360EPC PCI to Host Bridge operation

This sections provides an overview of the operation of the V360EPC. For a more detailed description, see the *V360EPC User Manual*.

The basic function of the V360EPC is to forward memory accesses in both directions between the local bus interface and the on-board PCI bus, as shown Figure 5-4.

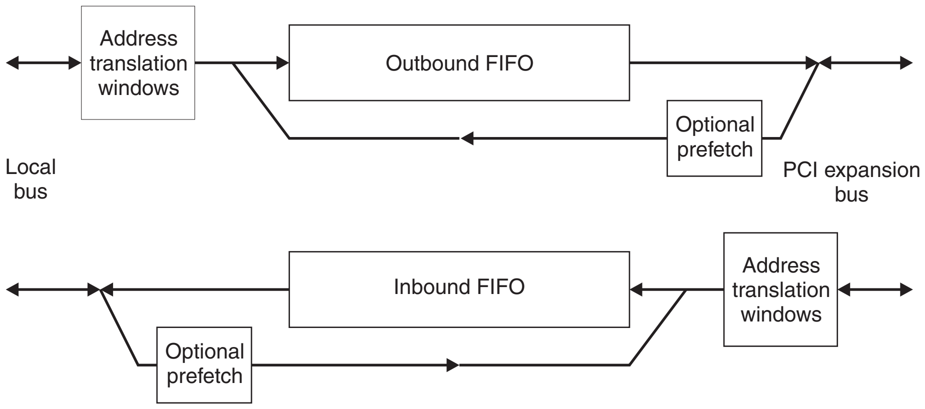


Figure 5-4 PCI to host bridge

The V360EPC provides windows through which accesses are made from one side of the bridge to the other. This is supported by the following features:

FIFOs These buffer write transactions in both directions, avoiding delays due to arbitration or traffic on the destination bus. Read cycles can be programmed to take precedence over queued writes.

Speculative prefetch

The V360EPC can use speculative read transfers on the target bus in anticipation that subsequent cycles on the initiating bus are likely to request data from the same target bus locations. This feature should only be used with a thorough understanding of the effects on the system.

DMA controller

The DMA controller can be programmed to move blocks of data from one bus to the other. It supports chaining, automatically loading sequences of DMA descriptors.

Mailboxes Mailboxes can be programmed to generate interrupts when accessed. This can be used to for handshaking between processors on Integrator core modules and processors on the PCI bus.

5.3.1 Local bus to PCI windows

The local bus side of the V360EPC provides three windows through which accesses can be made by system bus masters to the PCI expansion bus and CompactPCI. Each window is defined by its base address and its size in the host address space. A map register is used to define the starting location of each window in the PCI address space. A fourth window defines where the V360EPC internal registers appear in the system bus memory map but does not generate accesses on the PCI expansion bus side.

In the Integrator/AP, the default values are programmed by the ARM Firmware Suite as shown in Table 5-1.

Table 5-1 PCI outbound access windows

Window	PCI space	Size	PCI address	System bus address
0	Memory	256MB	`0x40000000` to `0x4FFFFFFF`	`0x40000000` to `0x4FFFFFFF`
1	Memory	256MB	`0x50000000` to `0x5FFFFFFF`	`0x50000000` to `0x5FFFFFFF`
2	I/O	16MB	`0x000000` to `0xFFFFFF`	`0x60000000` to `0x60FFFFFF`
Registers	-	64KB	-	`0x62000000` to `0x6200FFFF`

Note

The mapping of windows 0 and 1 must be changed to allow access to device registers when PCI configuration cycles are required (see *Configuration cycles* on page 5-12).

In the following, the contents of some of the V360EPC control registers are shown as they would be displayed by the boot monitor, with additional comment describing the effects of the programmed value.

Window 0

Window 0 provides a large window in the Integrator memory map to PCI memory address space. Addresses from `0x40000000` to `0x4FFFFFFF` (or `0x5FFFFFFF`) are mapped linearly with no translation between system bus and PCI bus. Accesses through the window generate PCI memory read (type 6) and memory write (type 7) cycles.

```
[0x62000054]  LB_BASE0  : 0x40000081
                        ; Local Bus start address 0x40000000,
                        ; size 256MB
[0x6200005E]  LB_MAP0   : 0x4006
                        ; maps to PCI address of 0x40000000,
                        ; normal cycles
```

Window 1

Window 1 provides a second large window from system bus address space to PCI memory address space. Addresses from 0x50000000 to 0x5FFFFFFF are mapped linearly with no translation between system bus and PCI bus. Accesses through the window generate standard PCI memory read (type 6) and memory write (type 7) cycles. Prefetches can be used in this window from PCI addresses.

```
[0x62000058]  LB_BASE1  : 0x50000081
                        ; Local Bus start address 0x50000000,
                        ; size 256MB
[0x62000062]  LB_MAP1   : 0x5006
                        ; maps to PCI address of 0x540000000,
                        ; normal cycles
```

Window 2

Window 2 provides a smaller window from system bus memory space to PCI I/O space. Addresses from 0x60000000 to 0x60FFFFFF are mapped to PCI I/O addresses 0x000000 to 0xFFFFFF. Accesses through the window generate PCI I/O read (type 2) and I/O write (type 3) cycles.

```
[0x62000064]  LB_BASE2  : 0x6001
                         ; Local bus start address 0x60000000,
                         ; size 16MB
[0x62000066]  LB_MAP2    : 0x0000
                         ; maps to PCI address of 0x00000000,
                         ; I/O cycles
```

Register window

This window is used to identify where the V360EPC internal registers appear in the system bus address map. Addresses from 0x62000000 to 0x6200FFFF are mapped to the internal register space. These do not generate any cycles on the PCI bus.

```
[0x6200006E]  LB_IO_BASE  : 0x6200
                          ; Local bus start address 0x62000000,
                          ; size 64K
```

Caution

When initializing the system, the first access to the V360EPC after reset must be a write to the LB_IO_BASE register.

5.3.2 Configuration cycles

Configuration cycles are used to access the registers in a particular device, such as an Ethernet controller, in order to initialize the device or to change its operational characteristics or state.

Access to device registers is gained by re-using window 1. The window is remapped to address `0x61000000` to `0x61FFFFFF` and to use of PCI Configuration read (type 10) and write (type 11) cycles. However, before remapping window 1, you must remap window 0 to ensure that application software can access the address space normally assigned to window 1 while window 1 is re-used for configuration cycles.

In other words, you must follow a strict sequence of steps to remap windows 0 and 1 before configuration cycles begin and then map them after configuration cycles are completed.

Note

The ARM Firmware suite provides a set of functions for performing PCI configuration cycles. See the *ARM Firmware Suite Reference Guide*.

The sequence of steps is:

1. Map window 0 to access the address space `0x40000000` to `0x5FFFFFFF`.
2. Map window 1 to provide access to the required registers.
3. Perform the required register accesses.
4. Map window 1 back to `0x50000000` to `0x5FFFFFFF`.
5. Map window 0 back to `0x40000000` to `0x4FFFFFFF`.

For example:

```
[0x62000054] LB_BASE0  : 0x40000091
                       ; Local Bus start address 0x40000000,
                       ; size 512MB
[0x6200005E] LB_MAP0   : 0x4006
                       ; maps to PCI address of 0x40000000,
                       ; normal cycles
[0x62000058] LB_BASE   : 0x61000041
                       ; Local Bus start address 0x61000000,
                       ; size 16MB
[0x62000062] LB_MAP1   : 0x800A
                       ; maps to PCI type 0 config cycle
                       ; to device 20
```

5.3.3 PCI to local bus windows

The PCI bus side of the V360EPC provides two windows through which accesses can be made from PCI bus initiators to the Integrator system bus. Each window is assigned a base address to define where it starts in PCI address space and a size to define how much space it occupies. Map registers define the corresponding starting location in the Integrator memory map.

A third window provides access to the V360EPC internal registers from the PCI bus.

In the Integrator system, the default values programmed by the ARM Firmware Suite are shown in Table 5-2.

Table 5-2 PCI inbound access windows

Window	PCI space	PCI address	Local bus address	Integrator resource
0	Memory	`0x20000000` to `0x3FFFFFFF`	`0x20000000` to `0x3FFFFFFF`	External bus interface
1	Memory	`0x80000000` to `0xBFFFFFFF`	`0x80000000` to `0xBFFFFFFF`	Core module alias memory
Registers	-	disabled by default		

Window 0

Window 0 provides a large window from PCI memory address space to system bus address space. Addresses from `0x20000000` to `0x3FFFFFFF` are mapped linearly with no translation between PCI bus and system bus. These addresses correspond to the EBI space in the Integrator memory map. The window captures PCI memory cycles (types 6, 7, C, E, F) but not PCI I/O cycles (types 10 and 11).

```
[0x62000014]  PCI_BASE0    : 0x20000000
                           ; PCI Bus start address 0x20000000,
                           ; memory cycles
[0x62000040]  PCI_MAP0     : 0x20000093
                           ; maps to System Bus at 0x20000000,
                           ; 512MB, enabled
```

Window 1

Window 1 provides a second large window from PCI memory address space to system bus address space. Addresses from `0x80000000` to `0xBFFFFFFF` are mapped linearly, with no translation between the PCI bus and system bus. This address range corresponds to the core module alias memory space. The window captures PCI memory cycles (types 6, 7, C, E, F) but not PCI I/O cycles (types 10 and 11).

```
[0x62000018]  PCI_BASE1    : 0x80000000
                           ; PCI Bus start address 0x80000000,
                           ; memory cycles
[0x62000044] PCI_MAP1      : 0x800000A3
                           ; maps to System Bus at 0x80000000,
                           ; 1024MB, enabled
```

Register window

This window can be used to allow access from the PCI bus to the V360EPC internal registers. This is disabled by default.

```
[0x62000010]  PCI_IO_BASE    : 0x00000000
                             ; internal registers at PCI memory
                             ; address 0 but
[0x6200007C]  PCI_CFG        : 0x3166
                             ; PCI_IO_BASE disabled
                             ; (and other functions)
```

Note

The ARM Firmware Suite provides initialization routines that program registers within the V360EPC. See the *ARM Firmware Suite Reference Guide*.

5.4 PCI to PCI bridge operation

The Intel 21152 on the Integrator/AP is a standard part. For a detailed description of its operation see the *PCI-PCI Bridge Specification* and the *Intel 21152 User Manual*.

The ARM Firmware Suite provides initialization routines that program registers within the bridge according to the topology of a particular system. For an example of a program that initializes the PCI system see the `scanpci` program in the PCI component of the ARM Firmware Suite.

The basic operation of the device is to forward memory and I/O transactions in either direction between the PCI expansion bus and CompactPCI bus when the addresses are within ranges that exist on the target side of the bridge.

PCI Configuration transactions are more complex. Certain configuration reads and writes propagate through bridges from primary to secondary side until they reach their destination bus where they are translated to other Configuration cycles and perform their function. This is very much a system dependent operation and is discussed in detail in the *ARM Firmware Suite Reference Guide*.

5.5 Initializing the PCI subsystem

After reset, the state of the PCI subsystem is as follows:

- the system to local bus bridge is disabled
- the V360EPC is reset so that its registers are not visible
- the PCI expansion bus is held in reset by the V360EPC
- the CompactPCI bus is held in reset by the Intel 21152 PCI-PCI Bridge

The ARM Firmware Suite provides initialization routines for the PCI subsystem and utilities for working with PCI. It is strongly recommended that the user start with these routines and, if changes are needed, use them as the basis for their own code.

The following is a summary of the important actions that the initialization must carry out. For further information see the *ARM Firmware Suite Reference Guide*.

1. Enable the system to local bus bridge in the system controller by setting the PCIEnable bit in the SC_PCI register.

   System bus accesses to the PCI bridge address space (`0x40000000` to `0x7FFFFFFF`), when this bit is 0, terminate with a Bus Error response. On typical core modules, a Bus Error response to a read transaction results in the processor taking a data abort exception. However, write transactions complete on the core module memory bus when that data is posted in the bridge FIFO so the data is just discarded.

2. Wait until 230ms after the end of the reset period before accessing the V360EPC internal registers. The V360EPC supports the use of a serial configuration PROM and the software must wait for the device to detect the absence of this PROM before accessing any registers. The required delay is a function of the PCI Clock speed but at the lower frequency (25MHz) is 230ms.

3. Your first access to the V360EPC must be to write the address decode register (LB_IO_BASE). Typically this is:
   ```
   (unsigned short *) 0x6200006E = 0x6200;
   ```

4. After the other configuration registers are written, release the PCI bus from reset by writing to the V360EPC SYSTEM register.

5.6 PCI subsystem interrupts

The Integrator interrupt controller includes 8 bits in its source, mask, set and clear registers that relate to the PCI sub-system. These are listed in Table 5-3.

Table 5-3 PCI interrupts

Bit	Name	Description
20	PCILBINT	System bus-to-V360EPC local bus interface error
19	ENUMINT	CompactPCI auxiliary interrupt
18	DEGINT	CompactPCI auxiliary interrupt
17	PCILINT	PCI host bridge interrupt
16	PCIINT3	Main PCI/CompactPCI interrupts
15	PCIINT2	Main PCI/CompactPCI interrupts
14	PCIINT1	Main PCI/CompactPCI interrupts
13	PCIINT0	Main PCI/CompactPCI interrupts

5.6.1 PCI Host Bridge interrupts

The V360EPC host bridge contains a number of potential interrupt sources including:

- PCI bus master abort
- PCI bus parity error
- PCI bus system error
- DMA finished
- Spare INTB#, INTC# inputs.

The V360EPC can be programmed so that these conditions generate an interrupt on PCILINT. For details of how to program these interrupt sources, see the *V360EPC User Manual*.

The spare INTD# input of the V360EPC is tied to 1 (inactive) on the Integrator/AP.

5.6.2 CompactPCI auxiliary interrupts

The CompactPCI bus includes two status signals and two legacy IDE signals that can be used to interrupt the host processor. These are:

DEG#	This is a signal from the system power supply unit that gives an early warning that the power may soon fail. This is connected to the DEGINT interrupt.
ENUM#	This is normally used for CompactPCI hot swap systems to signal board extraction or insertion. This is connected to the ENUMINT interrupt.
INTP	This is provided on the CompactPCI bus for backward compatibility with ISA IDE systems, where it was the *primary ISA interrupt*. It can be used as a general purpose interrupt. **INTP** is active high so it is first inverted. It is then connected to the **INTB#** input of the V360EPC Host Bridge, that can be programmed to cause an interrupt on its **LINT** pin.
INTS	This is provided on the CompactPCI bus for backward compatibility with ISA IDE systems, where it was the *secondary ISA interrupt*. It can be used as a general purpose interrupt. **INTS** is active high so it is first inverted. It is then connected to the **INTC#** input of the V360EPC Host Bridge, that can be programmed to cause an interrupt on its **LINT** pin.

For more information about the use of these signals, refer to the CompactPCI specification.

5.6.3 System bus to local bus interface timeout interrupt

Under some fault conditions accesses from the system bus to the V360EPC can result in an infinite wait state condition. The typical cause of this condition is a data access into a part of the address range `0x40000000` to `0x7FFFFFFF` that the V360EPC has not been programmed to capture. To simplify debugging of such systems the bridge to this local bus in the system controller FPGA includes a timeout controller.

If a bus transaction fails to complete on the local bus for more than approximately 1ms without a data transfer taking place, the timeout controller aborts the transaction. The action taken depends upon the type of transaction that caused the timeout.

In all cases the timeout controller sets the local bus fault interrupt PCILBINT of the interrupt controller. The address for the transaction is stored in the SC_LBFADDR register and a fault code is stored in the SC_LBFCODE register. The interrupt is cleared by writing a 1 to bit 1 of the SC_PCI register.

Accesses that cause faults frequently occur in bursts. However, the most useful debugging information is a record of the first access. Therefore, the local bus fault code and local bus fault address registers are only updated if the PCILBINT bit is clear when a fault occurs.

The fault codes and action taken are summarized in Table 5-4, and described below.

Table 5-4 System bus fault codes

Fault code	Initiator	Transaction type	Action
xxxx x00x	V360EPC	PCI write to system bus	Data discarded
xxxx x01x	System bus master	Write to PCI	Data discarded
xxxx x10x	V360EPC	PCI read from system bus	Read cycle terminated
xxxx x11x	System bus master	Read from PCI	System Bus Error (and hence Data Abort exception)

PCI write to system bus

During PCI write transactions, the data is buffered by one or more FIFOs before it reaches the local bus. Therefore, it is no longer possible to issue a Target Abort to the PCI initiator because it would have completed the cycle. The data is discarded but the fault condition is recorded in the local bus fault registers.

See also registers SC_PCI, SC_LBFADDR and SC_LBFCODE.

System bus write to PCI

System bus write cycles are completed when the data is posted into the bridge FIFO. Therefore, it is not useful to issue a Bus Error response to the system bus master because the processor will have advanced in its program. In this case the data is discarded and the next transaction in the FIFO commences on the local bus.

PCI read from system bus

V360EPC read cycles are terminated by forcing the **RDY** signal active. This may only be passed back to the PCI bus as a cycle termination, without being flagged as an error condition to the PCI bus initiator receiving the data. The local bus fault interrupt is available as the indication that this error condition has occurred. The system bus arbitration algorithm assigns higher priority to inbound transactions from PCI so the

V360EPC local bus time-outs are only caused by these transactions under exceptional fault conditions. For example, if a faulty system bus slave causes bus congestion by issuing thousands of WAIT states without a RETRACT response.

System bus read from PCI

System bus read cycles are terminated with a Bus Error response to the bus master. On typical core modules, this causes the processor to take a Data Abort exception.

5.6.4 PCI IDSEL and interrupt assignments

PCI Configuration transactions require a physical address. This address identifies the bus, device, and function. In a bus with expansion slots, the device number identifies the physical connector position. The device in a given slot is selected by activating its **IDSEL** signal. On both the CompactPCI bus and the PCI expansion bus, the **IDSEL** signals are driven by PCI address lines (AD31 - AD11).

PCI devices assert interrupts on any of the four lines INTA, INTB, INTC and INTD. The PCI specification describes how these lines are rotated from one slot to the next in order to share the interrupt load between the four PCI interrupt channels of the system interrupt controller.

PCI expansion bus

The PCI expansion bus the Integrator/AP on contains two bridges and up to three additional devices plugged into the expansion slots. The bridge devices can be the targets of configuration cycles but do not generate PCI interrupts. The **IDSEL** and interrupt assignments for the PCI expansion bus are shown in Table 5-5.

Table 5-5 PCI IDSEL and interrupt assignments for the PCI expansion bus

PCI Device		Interrupt			
IDSEL	**Name**	**INTA**	**INTB**	**INTC**	**INTD**
AD24	V360EPC host bridge				
AD23	PCI slot 1	PCIINT3	PCIINT0	PCIINT1	PCIINT2
AD22	PCI slot 2	PCIINT2	PCIINT3	PCIINT0	PCIINT1
AD21	PCI slot 3	PCIINT1	PCIINT2	PCIINT3	PCIINT0
AD20	Intel 21152 PCI-PCI bridge, primary side				

CompactPCI bus

The Integrator is a CompactPCI system controller board so is plugged into the system slot (slot 1) of a CompactPCI bus. The mapping of CompactPCI slot to **IDSEL** numbers are shown in Table 5-6.

Table 5-6 PCI IDSEL and interrupt assignments for the CompactPCI bus

PCI Device		Interrupt			
IDSEL	**Name**	**INTA**	**INTB**	**INTC**	**INTD**
AD31	CompactPCI slot 2	PCIINT3	PCIINT0	PCIINT1	PCIINT2
AD30	CompactPCI slot 3	PCIINT2	PCIINT3	PCIINT0	PCIINT1
AD29	CompactPCI slot 4	PCIINT1	PCIINT2	PCIINT3	PCIINT0
AD28	CompactPCI slot 5	PCIINT0	PCIINT1	PCIINT2	PCIINT3
AD27	CompactPCI slot 6	PCIINT3	PCIINT0	PCIINT1	PCIINT2
AD26	CompactPCI slot 7	PCIINT2	PCIINT3	PCIINT0	PCIINT1
AD25	CompactPCI slot 8	PCIINT1	PCIINT2	PCIINT3	PCIINT0
AD24	No device present because the Intel21152 PCI-PCI bridge has no secondary IDSEL				

The Integrator/AP features open collector buffers that wire-OR the CompactPCI interrupt signals onto the PCI expansion bus interrupt signals. The **IDSEL** of the Intel21152 PCI-PCI Bridge avoids any rotation in the mapping between the interrupt signals on these two buses.

Interrupt signal assignments

Table 5-7 shows the assignment of the CompactPCI and PCI expansion bus interrupt signals and the system interrupt controller.

Table 5-7 PCI bus interrupt signal to interrupt controller assignment

CompactPCI	PCI expansion bus	Interrupt controller
CP_nIntA	**P_nIntA**	PCIINT0
CP_nIntB	**P_nIntB**	PCIINT1
CP_nIntC	**P_nIntC**	PCIINT2
CP_nIntD	**P_nIntD**	PCIINT3

PCI interrupt connector pin assignments

Devices in PCI Expansion and CompactPCI slots connect their local interrupts to the connector pins as shown in Table 5-8.

Table 5-8 Interrupt to connector pin assignment

Interrupt	Connector pin	
	Expansion slot	**CompactPCI**
INTA#	A6	A3
INTB#	B7	B3
INTC#	A7	C3
INTD#	B8	E3

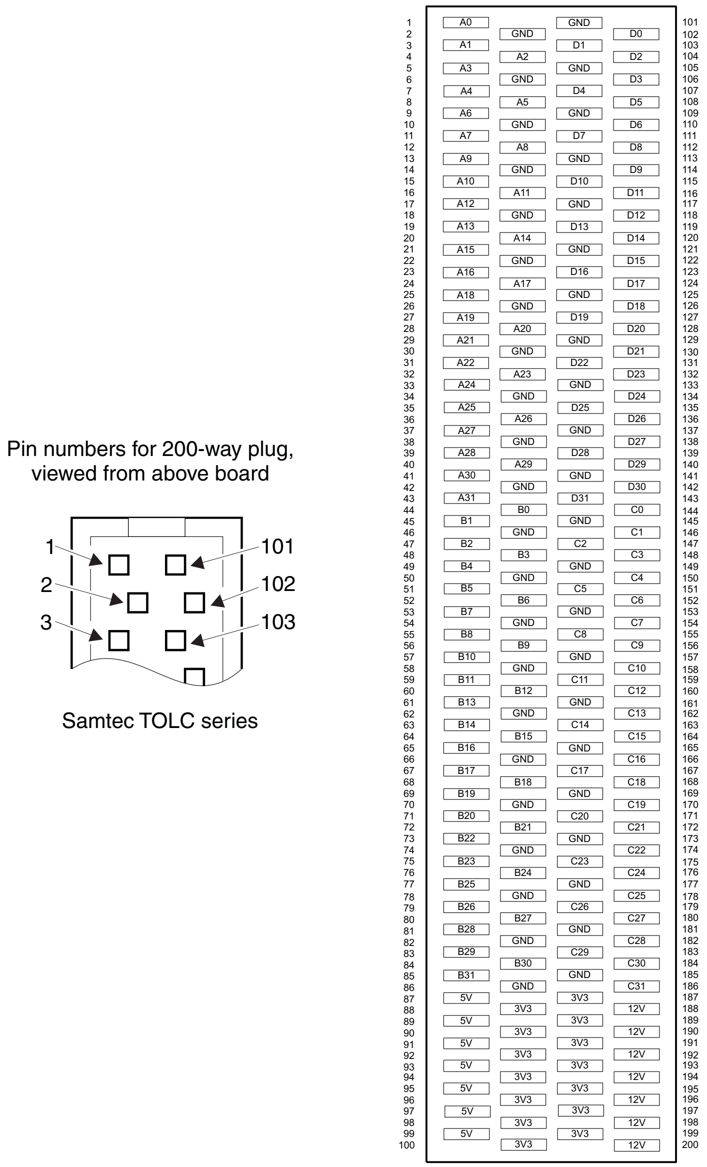
Appendix A
Connector Pinouts

This appendix describes the Integrator/AP interface connectors and signal connections. It contains the following sections:

- *Inter-module connectors HDRA and EXPA* on page A-2
- *Core module connector HDRB* on page A-5
- *Logic module connector EXPB* on page A-8
- *Expansion module connector EXPM* on page A-11
- *Serial interface connectors* on page A-14
- *Keyboard and mouse connectors* on page A-15.

A.1 Inter-module connectors HDRA and EXPA

Figure A-1 shows the pin numbering of the connector HDRA and EXPA. Both connectors have a similar pinout.



Pin	Signal	Pin	Signal
1	A0	101	GND
2	GND	102	D0
3	A1	103	D1
4	A2	104	D2
5	A3	105	GND
6	GND	106	D3
7	A4	107	D4
8	A5	108	D5
9	A6	109	GND
10	GND	110	D6
11	A7	111	D7
12	A8	112	D8
13	A9	113	GND
14	GND	114	D9
15	A10	115	D10
16	A11	116	D11
17	A12	117	GND
18	GND	118	D12
19	A13	119	D13
20	A14	120	D14
21	A15	121	GND
22	GND	122	D15
23	A16	123	D16
24	A17	124	D17
25	A18	125	GND
26	GND	126	D18
27	A19	127	D19
28	A20	128	D20
29	A21	129	GND
30	GND	130	D21
31	A22	131	D22
32	A23	132	D23
33	A24	133	GND
34	GND	134	D24
35	A25	135	D25
36	A26	136	D26
37	A27	137	GND
38	GND	138	D27
39	A28	139	D28
40	A29	140	D29
41	A30	141	GND
42	GND	142	D30
43	A31	143	D31
44	B0	144	C0
45	B1	145	GND
46	GND	146	C1
47	B2	147	C2
48	B3	148	C3
49	B4	149	GND
50	GND	150	C4
51	B5	151	C5
52	B6	152	C6
53	B7	153	GND
54	GND	154	C7
55	B8	155	C8
56	B9	156	C9
57	B10	157	GND
58	GND	158	C10
59	B11	159	C11
60	B12	160	C12
61	B13	161	GND
62	GND	162	C13
63	B14	163	C14
64	B15	164	C15
65	B16	165	GND
66	GND	166	C16
67	B17	167	C17
68	B18	168	C18
69	B19	169	GND
70	GND	170	C19
71	B20	171	C20
72	B21	172	C21
73	B22	173	GND
74	GND	174	C22
75	B23	175	C23
76	B24	176	C24
77	B25	177	GND
78	GND	178	C25
79	B26	179	C26
80	B27	180	C27
81	B28	181	GND
82	GND	182	C28
83	B29	183	C29
84	B30	184	C30
85	B31	185	GND
86	GND	186	C31
87	5V	187	3V3
88	3V3	188	12V
89	5V	189	3V3
90	3V3	190	12V
91	5V	191	3V3
92	3V3	192	12V
93	5V	193	3V3
94	3V3	194	12V
95	5V	195	3V3
96	3V3	196	12V
97	5V	197	3V3
98	3V3	198	12V
99	5V	199	3V3
100	3V3	200	12V

Figure A-1 Connector pin numbering

The signals on the pins labeled A[31:0], B[31:0], C[31:0], and D[31:0] for are listed in Table A-1. A more in depth description of these buses in provided in *System bus* on page 3-3.

Table A-1 Bus bit assignment

Pin label	AHB signal name	ASB signal name	Description
A[31:0]	**HADDR[31:0]**	**BA[31:0]**	System address bus
B[31:0]	Not used	Not used	-
C[31:22]	Not used	Not used	-
C[21:16]	**HSPLIT[5:0]**	Not used	Split transaction. Split transactions are a feature of AHB, but not of ASB. For full details of the split and retry responses refer to the *AMBA Specification*. When a slave issues a split response the arbiter notes the current master and does not grant it again until the slave drives one of the signals **HSPLIT[5:0]** HIGH to indicate that the transaction can continue. Core modules treat a split response as a retry response and do not generate split responses themselves. For this reason these pins on a core module are currently not used.
C15	**HMASTLOCK**	**BLOK**	Locked transaction. **BLOK** on ASB is a shared signal driven by the current master to indicate a locked transaction. **HMASTLOCK** on AHB is driven by the arbiter to indicate that a locked transaction is in progress. Masters indicate a locked transaction using the **HLOCK** signals. These are connected point-to-point between the master and the arbiter. The **HLOCK** signals are implemented on the HDRB/EXPB connectors to provide one per master (that is, there is one master per module).
C14	**HRESP1**	**BLAST**	Slave response
C13	**HRESP0**	**BERROR**	Slave response
C12	**HREADY**	**BWAIT**	Slave wait response
C11	**HWRITE**	**BWRITE**	Write transaction
C10	**HPROT2**	Not used	Transaction protection type

Table A-1 Bus bit assignment (continued)

Pin label	AHB signal name	ASB signal name	Description
C[9:8]	**HPROT[1:0]**	**BPROT[1:0]**	Transaction protection type
C[7:5]	**HBURST[2:0]**	Not used	Transaction burst size
C4	**HPROT[3]**	Not used	Transaction protection type
C[3:2]	**HSIZE[1:0]**	**BSIZE[1:0]**	Transaction width The AHB specification defines a 3-bit bus for HSIZE, but 2 bits is sufficient to describe transfers of up to 64-bits wide which is why a 2-bit bus is sufficient on Integrator.
C[1:0]	**HTRAN[1:0]**	**BTRAN[1:0]**	Transaction type
D[31:0]	**HDATA[31:0]**	**BD[31:0]**	System data bus

A.2 Core module connector HDRB

Figure A-2 shows the pin numbers of the connector HDRB.

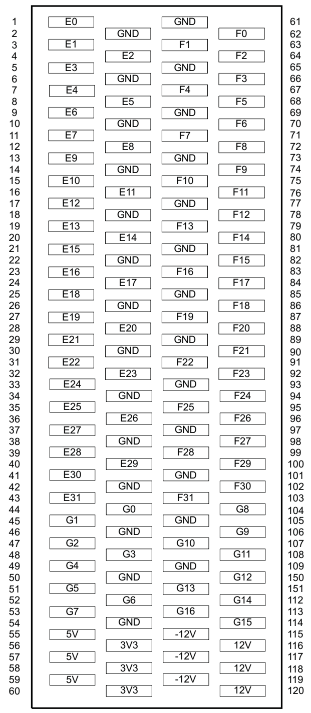


Figure A-2 HDRB pin numbering

Table A-3 describes the signals on the pins labeled E[31:0], F[31:0], and G[16:0] for AMBA AHB system bus.

Table A-2 HDRB signal description

Pin label	AHB signal Name	ASB signal name	Description
E[31:28]	**HCLK[3:0]**	**BCLK[3:0]**	System clock to the core module.
E[27:24]	**nPPRES[3:0]**	**nPPRES[3:0]**	Core module present. Each core module ties **nPPRES[0]** LOW and leaves **nPPRES[3:1]** open circuit. These signals rotate as they move up or down the stack so that there is a connection between each module and one of these signals at the system controller on the motherboard.
E[23:20]	**nIRQ[3:0]**	**nIRQ[3:0]**	Interrupt request to processor.
E[19:16]	**nFIQ[3:0]**	**nFIQ[3:0]**	Fast interrupt requests to processor.
E[15:12]	**ID[3:0]**	**ID[3:0]**	Core module stack position indicator.
E[11:8]	**HLOCK[3:0]**	Reserved	System bus lock from processor.
E[7:4]	**HGRANT[3:0]**	**AGNT[3:0]**	System bus grant to processor.
E[3:0]	**HBUSREQ[3:0]**	**AREQ[3:0]**	System bus request from processor.
F[31:0]	-	-	Not connected.
G16	**nRTCKEN**	**nRTCKEN**	**RTCK** AND gate enable.
G[15:14]	**CFGSEL[1:0]**	**CFGSEL[1:0]**	FPGA configuration select.
G13	**nCFGEN**	**nCFGEN**	Sets motherboard into configuration mode.
G12	**nSRST**	**nSRST**	Multi-ICE reset (open collector).
G11	**FPGADONE**	**FPGADONE**	Indicates when FPGA configuration is complete.
G10	**RTCK**	**RTCK**	Returned JTAG test clock.
G9	**nSYSRST**	**nSYSRST**	Buffered system reset.
G8	**nTRST**	**nTRST**	JTAG reset.
G7	**TDO**	**TDO**	JTAG test data out.
G6	**TDI**	**TDI**	JTAG test data in.
G5	**TMS**	**TMS**	JTAG test mode select.

Table A-2 HDRB signal description (continued)

Pin label	AHB signal Name	ASB signal name	Description
G4	**TCK**	**TCK**	JTAG test clock.
G[3:1]	**HMAST[2:0]**	**MASTER[2:0]** (implemented though not strictly ASB)	Master ID. Binary encoding of the master currently performing a transfer on the bus. Corresponds to the module ID and to the **HBUSREQ** and **HGRANT** line numbers.
G0	**nMBDET**	**nMBDET**	Motherboard detect. This signal is tied LOW on the AP. When a module is attached to the motherboard, it detects that **nMBDET** is LOW and routes **TDI** and **TCK** down to the motherboard where they are looped back onto **TD0** and **RTCK**. Also, core modules pass addresses above `0x11000000` on to the system bus where they are decoded by the motherboard or by other modules. When a module is used standalone, it detects that **nMBDET** is HIGH and provides loop backs for the JTAG signals itself so that the scan chain is intact. Accesses to addresses above `0x11000000` produce a bus error or abort.

A.3 Logic module connector EXPB

Figure A-3 shows the pin numbers of the EXPB plug.

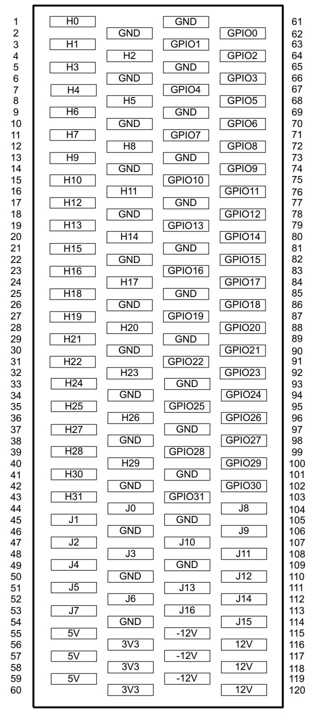


Figure A-3 EXPB socket pin numbering

Table A-3 describes the signals on the pins labeled F[31:0], H[31:0], J[15:0], and GPIO[31:0].

Table A-3 EXPB signal description

Pin label	AHB signal name	ASB signal name	Description
H[31:28]	**HCLK[7:4]**	**BCLK[7:4]**	System clock to each logic module.
H[27:24]	**nEPRES[3:0]**	**nEPRES[3:0]**	Logic module present. Each logic module ties **nEPRES[0]** LOW and leaves **nEPRES[3:1]** open circuit. These signals rotate as they move up or down the stack so that there is a connection between each module and one of these signals at the system controller on the motherboard.
H[23:20]	**nIRQSRC[3:0]**	**nIRQSRC[3:0]**	Interrupt request from logic module 3, 2, 1, and 0 respectively.
H[19:16]	-	-	Not connected
H[15:12]	**ID[3:0]**	**ID[3:0]**	Logic module stack position indicator.
H[11:8]	**HLOCK[1:4]**	-	System bus lock from processor 3, 2, 1, and 0 respectively (not used in ASB). These signals rotate in the opposite direction to those on the HDRB connector. That is, H11 connects to **SLOCK1** and H8 connects to **SLOCK4**.
H[7:4]	**HGNT[1:4]**	**AGNT[1:4]**	System bus grant.
H[3:0]	**HBUSREQ[1:4]**	**AREQ[1:4]**	System bus request.
J16	**nRTCKEN**	**nRTCKEN**	**RTCK** AND gate enable.
J[15:14]	**CFGSEL[1:0]**	**CFGSEL[1:0]**	FPGA configuration select.
J13	**nCFGEN**	**nCFGEN**	Sets motherboard into configuration mode.
J12	**nSRST**	**nSRST**	Multi-ICE reset (open collector).
J11	**FPGADONE**	**FPGADONE**	Indicates when FPGA configuration is complete (open collector).
J10	**RTCK**	**RTCK**	Returned JTAG test clock.
J9	**nSYSRST**	**nSYSRST**	Buffered system reset.
J8	**nTRST**	**nTRST**	JTAG reset.
J7	**TDO**	**TDO**	JTAG test data out.
J6	**TDI**	**TDI**	JTAG test data in.

Table A-3 EXPB signal description (continued)

Pin label	AHB signal name	ASB signal name	Description
J5	**TMS**	**TMS**	JTAG test mode select.
J4	**TCK**	**TCK**	JTAG test clock.
J[3:1]	**HMASTER[2:0]**	**MASTER[2:0]** (implemented though not strictly ASB)	Master ID. Binary encoding of the master currently performing a transfer on the bus. Corresponds to the module ID and to the **HBUSREQ** and **HGRANT** line numbers.
J0	**nMBDET**	**nMBDET**	Motherboard detect pin. See Table A-2 on page A-6
GPIO[31:0]	-	-	These connect to the GPIO pins on the FPGA and are available for your own applications.

A.4 Expansion module connector EXPM

Figure A-4 shows the pin numbers of the connector EXPM.

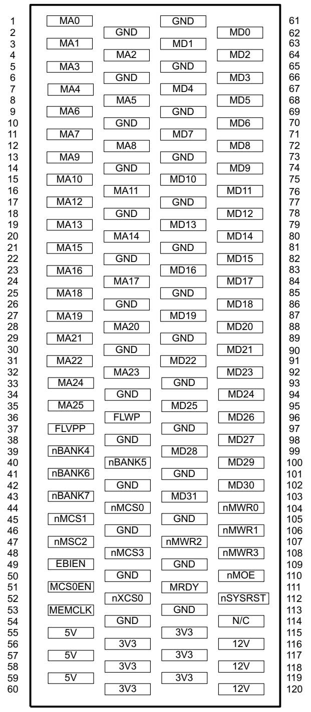


Figure A-4 EXPM pin numbering

Table A-4 describes the signals on the EXPM pins.

Table A-4 EXPM signal description

Name	Description
MA[25:0]	Memory address bus
MD[31:0]	Memory data bus
nMCS[3:0]	Memory chip select: **nMCS0** = ROM **nMCS1** = flash **nMCS2** = SSRAM **nMCS3** = spare

Table A-4 EXPM signal description (continued)

Name	Description
nMOE	Memory output enable (active LOW)
nMWR[3:0]	Memory write strobe (active LOW): **nMWR0 = MD[7:0]** **nMWR1 = MD[15:8]** **nMWR2 = MD[23:16]** **nMWR3 = MD[31:24]**
MCS0EN	**nMCS0** enable If **MCS0EN** is HIGH **nMCS0** is driven as usual. If **MCS0EN** is LOW **nXCS0** is driven instead. This allows the Integrator/AP to boot from an alternative device. For, example if you build your own memory board you can boot from your own 8-bit wide EPROM or flash device rather than the normal boot ROM. A 10K pullup resistor on the AP defaults this signal HIGH. There is a 32-pin DIL socket provided on the Integrator/AM Analyzer Module for this purpose. It also allows an EPROM emulator to be used. **Caution** Make sure the emulator drives 3V3 signal levels as 5V may damage the other memory devices on the AP.
nXCS0	Expansion chip select 0. This active LOW signal is used to boot from an alternative device on an expansion card. It can only be used when **MCS0EN** is LOW.
MEMCLK	Memory clock. This signal is provided for synchronous memory devices or peripherals with a static memory interface that require a clock.
MRDY	Memory ready. This signal allows expansion peripherals to wait the system bus. If any of **nMCS[3:0]** signals is LOW and the interface has been programmed for asynchronous operation, the EBI waits the system bus when **MRDY** is driven LOW. MRDY is read on the rising edge of **HCLK** or the falling edge of **BCLK**. There is a 1K pullup resistor on the AP which defaults this signal high.

Table A-4 EXPM signal description (continued)

Name	Description
nFLWP	Flash write protect.
FLVPP	Flash Vpp enable.
nBANK[7:4]	Memory bank selects for **nMCS1**. Can be used to expand flash.
EBIEN	EBI enable. Drive LOW to tristate the following signals: **MA[25:0]** **nMCS[3:0]** **nMWR[3:0]** **nMOE** **nVCS0** **nFLWP** **MEMCLK** **MD[3:0]**.
nSYSRST2	Buffered system reset

A.5 Serial interface connectors

The pinout of the serial ports is shown in Figure A-5 and Table A-5.

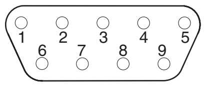


Figure A-5 Serial interface connector pinout

Table A-5 Serial interface signal descriptions

Pin	Signal	Type	Function
1	**DCD**	Input	Data carrier detect
2	**Rx**	Input	Receive
3	**Tx**	Output	Transmit
4	**DTR**	Output	Data terminal ready
5	**GND**	-	Ground
6	**DSR**	Input	Data set ready
7	**RTS**	Output	Ready to send
8	**CTS**	Input	Clear to send
9	**RI**	-	not connected on Integrator

Note

The serial interfaces signals operate at RS232 signal levels.

A.6 Keyboard and mouse connectors

The pinout of the KMI connectors is shown in Figure A-6.

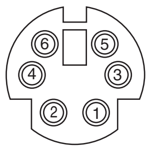


Figure A-6 KMI connector pinouts

Table A-6 shows signals on the KMI connectors.

Table A-6 Mouse and keyboard port signal descriptions

Pin	Keyboard (Lower)		Mouse (Top)	
	Signal	**Function**	**Signal**	**Function**
1	**KDATA**	Keyboard data	**MDATA**	Mouse Data
2	N/C	Not connected	N/C	Not connected
3	**GND**	Ground	**GND**	Ground
4	**5V**	5V	**5V**	5V
5	**KCLCK**	Keyboard clock	**MCLCK**	Mouse clock
6	N/C	Not connected	N/C	Not connected

Appendix B
Specifications

This appendix contains the specifications for the Integrator/AP. It contains the following sections:

- *Mechanical details* on page B-2
- *Electrical specification* on page B-3
- *Timing specification* on page B-4.

B.1 Mechanical details

The Integrator/AP is a PC ATX motherboard designed to be to be installed in a PC ATX housing. It can also be installed in a CompactPCI card cage or used as a bench-top system. Figure B-1shows the mechanical outline of the board.

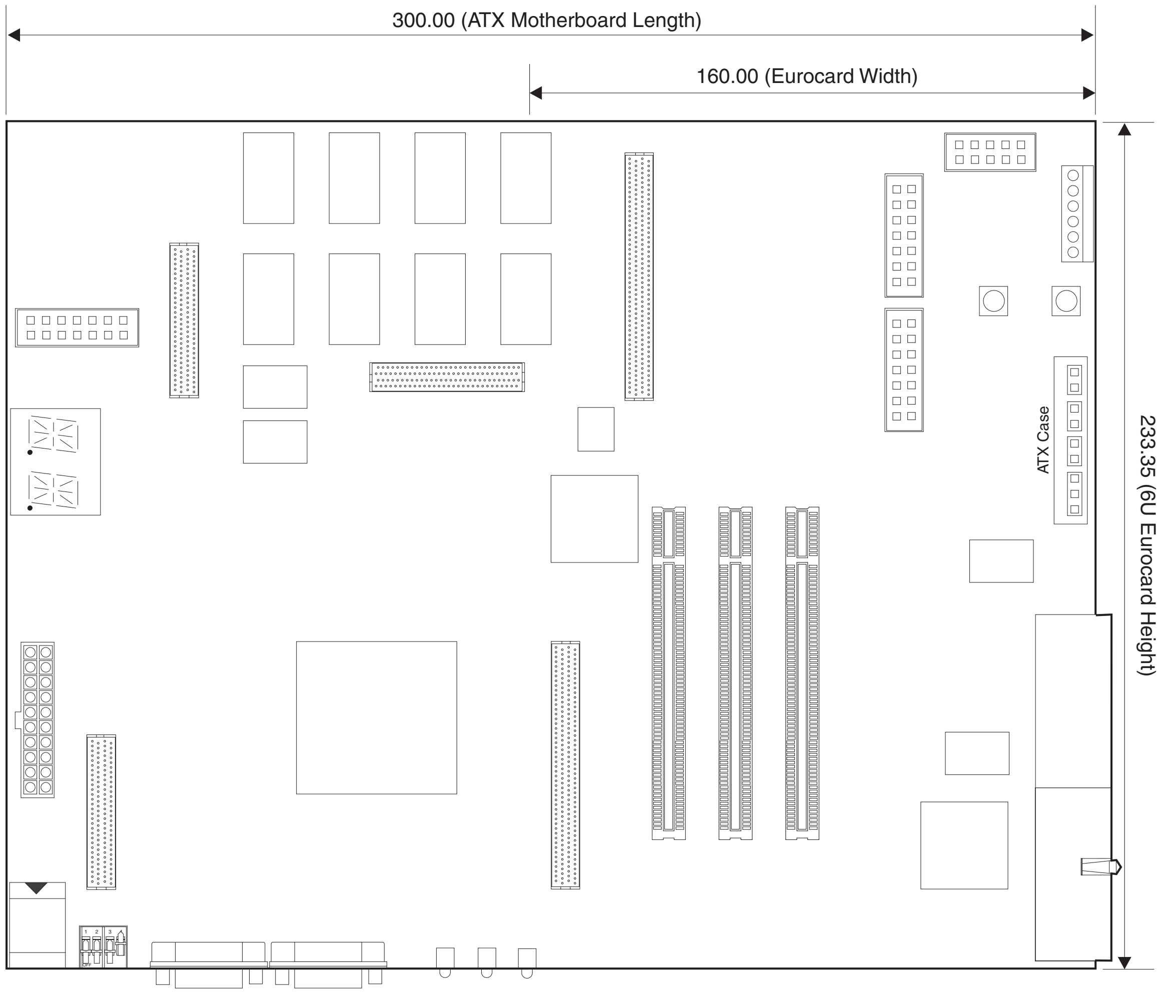


Figure B-1 Board outline

B.2 Electrical specification

Table B-1 shows the core module electrical characteristics for the system bus interface.

The Integrator/AP and core modules uses 3.3V and 5V. The +12V and –12V inputs can be supplied if required by a logic or expansion module.

The measurements are typical for LVCMOS inputs and LVTTL outputs.

Table B-1 Electrical characteristics

Symbol	Description	Min	Max	Unit
3V3	Supply voltage (interface signals)	3.1	3.5	V
5V	Supply voltage	4.75	5.25	V
+12V	Supply voltage	11.5	12.5	V
– 12V	Supply voltage	–11.5	–12.5	V
VIH	High level input voltage	2.0	3.6	V
VIL	Low level input voltage	0	0.8	V
VOH	High level output voltage	2.4	-	V
VOL	Low level output voltage	-	0.4	V
CIN	Input capacitance	-	20	pF

B.3 Timing specification

This section is a reference for designers adding modules on to an Integrator system. The timing information presented here is representative only. Specific modules and FPGA revisions will deviate from these numbers, but they provide some guidance when constraining FPGA designs.

The following sections detail the timing parameters for a typical ASB and AHB modules and motherboards.

B.3.1 Integrator timing parameters and the AMBA Specification

The parameters listed are those specified in the AMBA Specification with the following important differences:

- only output valid and input setup times are quoted
- the required input hold time (Tih) is always less than or equal to 0ns
- the output hold time (Toh) is always greater than 2ns.

Each version and revision of the FPGA has subtly different timing. The figures are those you can expect under nominal conditions and should be used as a guideline when designing you own motherboards and modules. The figures have been rounded to simplify timing analysis and constraints.

B.3.2 AHB system bus timing parameters

Table B-2 shows the clock and reset timing parameters.

Table B-2 Clock and reset parameters

Parameter	Description	Time (ns)	Notes
T_{clk}	**HCLK** minimum clock period	30	Representative of worst case maximum frequency
T_{isrst}	**HRESETn** deasserted setup time before **HCLK**	15	Applies to modules only
T_{ovrst}	**HRESETn** deasserted valid time before **HCLK**	15	Applies only to the module or motherboard implementing the reset source

Table B-3 shows the AHB slave input parameters.

Table B-3 AHB slave input parameters

Parameter	Description	Time (ns)	Notes
T_{issel}	**HSELx** setup time before **HCLK**	n/a	**HSELx** are internally generated, not visible at the pins
T_{istr}	Transfer type setup time before **HCLK**	5	-
T_{isa}	**HADDR[31:0]** setup time before **HCLK**	10	-
T_{isctl}	**HWRITE**, **HSIZE[2:0]** and **HBURST[2:0]** control signal setup time before **HCLK**	5	-
T_{iswd}	Write data setup time before **HCLK**	5	-
T_{isrdy}	Ready setup time before **HCLK**	5	-
T_{ismst}	Master number setup time before **HCLK** (SPLIT-capable only)	n/a	Applies to modules with split capable slaves only
T_{ismlck}	Master locked setup time before **HCLK** (SPLIT-capable only)	n/a	Applies to modules with split capable slaves only

Table B-4 shows the AHB slave output parameters.

Table B-4 AHB slave output parameters

Parameter	Description	Time (ns)	Notes
T_{ovrsp}	Response valid time after **HCLK**	15	-
T_{ovrd}	Data valid time after **HCLK**	15	-
T_{ovrdy}	Ready valid time after **HCLK**	15	-
T_{ovsplt}	Split valid time after **HCLK** (SPLIT-capable only)	n/a	Applies to modules with split capable slaves only

Table B-5 shows the bus master input timing parameters.

Table B-5 Bus master input timing parameters

Parameter	Description	Time (ns)	Notes
T_{isgnt}	**HGRANTx** setup time before **HCLK**	5	Modules implementing masters only
T_{isrdy}	Ready setup time before **HCLK**	5	-
T_{isrsp}	Response setup time before **HCLK**	5	-
T_{isrd}	Read data setup time before **HCLK**	5	

Table B-6 shows bus master output timing parameters.

Table B-6 Bus master output timing parameters

Parameter	Description	Time (ns)	Notes
T_{ovtr}	Transfer type valid time after **HCLK**	15	-
T_{ova}	Address valid time after **HCLK**	15	-
T_{ovctl}	Control signal valid time after **HCLK**	15	-
T_{ovwd}	Write data valid time after **HCLK**	15	-
T_{ovreq}	Request valid time after **HCLK**	15	Modules implementing masters only
T_{ovlck}	Lock valid time after **HCLK**	15	Modules implementing masters only

Table B-7 shows the AHB arbiter input parameters. Applies only to the module or motherboard implementing the arbiter

Table B-7 AHB arbiter input parameters

Parameter	Description	Time (ns)	Notes
T_{isrdy}	Ready setup time before **HCLK**	5	-
T_{isrsp}	Response setup time before **HCLK**	5	-
T_{isreq}	Request setup time before **HCLK**	10	-
T_{islck}	Lock setup time before **HCLK**	10	-

Table B-7 AHB arbiter input parameters

Parameter	Description	Time (ns)	Notes
T_{issplt}	Split setup time before **HCLK**	10	-
T_{istr}	Transfer type setup time before **HCLK**	5	-
T_{isctl}	Control signal setup time before **HCLK**	5	-

Table B-8 shows the AHB arbiter output parameters. Applies only to the module or motherboard implementing the arbiter

Table B-8 AHB arbiter output parameters

Parameter	Description	Time (ns)	Notes
T_{ovgnt}	Grant valid time after **HCLK**	15	-
T_{ovmst}	Master number valid time after **HCLK**	15	-
T_{ovmlck}	Master locked valid time after **HCLK**	15	-

B.3.3 ASB system bus timing parameters

Table B-9 shows the clock and reset parameters.

Table B-9 Clock and reset parameters

Parameter	Description	Time (ns)	Notes
T_{clk}	**BCLK** minimum clock period	40	Representative of worst case maximum frequency
T_{clkl}	**BCLK** LOW time	20	-
T_{clkh}	**BCLK** HIGH time	20	-
T_{isnres}	**BnRES** deasserted setup to rising **BCLK**	15	Applies to modules only
T_{ovnres}	**BnRES** deasserted valid after rising **BCLK**	15	Applies only to the module or motherboard implementing the reset source

Table B-10 shows the ASB slave input parameters. Applies only to the module or motherboard implementing the arbiter.

Table B-10 ASB slave input parameters

Parameter	Description	Time (ns)	Notes
T_{isdsel}	**DSEL** setup to falling **BCLK**	n/a	**DSEL** is internally generated, not visible at the pins
T_{isa}	**BA[31:0]** setup to falling **BCLK**	10	Path through decoder is up to 30ns
T_{isctl}	**BWRITE** and **BSIZE[1:0]** setup to falling **BCLK**	10	-
T_{isdw}	For write transfers, **BD[31:0]** setup to falling **BCLK**	10	-

Table B-11 shows the ASB slave output parameters.

Table B-11 ASB slave output parameters

Parameter	Description	Time (ns)	Notes
T_{ovresp}	**BWAIT**, **BERROR** and **BLAST** valid after falling **BCLK**	10	-
T_{ovdr}	For read transfers, **BD[31:0]** valid after rising **BCLK**	30	-

Table B-12 shows the bus master input timing parameters

Table B-12 Bus master input parameters

Parameter	Description	Time (ns)	
T_{isresp}	**BWAIT**, **BERROR** and **BLAST** setup to rising **BCLK**	15	-
T_{isdr}	For read transfers, **BD[31:0]** setup to falling **BCLK**	10	-
Tisagnt	**AGNT** setup to rising **BCLK**	10	Modules implementing masters only

Table B-13 shows the bus master output timing parameters.

Table B-13 Bus master output parameters

Parameter	Description	Time (ns)	Notes
T_{ovtr}	**BTRAN** valid after rising **BCLK**	10	-
T_{ova}	**BA[31:0]** valid after rising **BCLK**, all transfer types	10	-
T_{ovctl}	**BWRITE**, **BSIZE[1:0]** and **BPROT[1:0]** valid after rising **BCLK**, all transfer types	10	-
T_{ovdw}	**BD[31:0]** valid after rising **BCLK**, all transfer types	30	-
T_{ovlok}	**BLOK** valid after rising **BCLK**	10	-
T_{ovareq}	**AREQ** valid after rising **BCLK**	10	-

Table B-14 shows the ASB decoder input parameters.

Table B-14 ASB decoder input parameters

Parameter	Description	Time (ns)	Notes
T_{istr}	**BTRAN** setup to falling **BCLK**	10	-
T_{isresp}	**BWAIT**, **BERROR** and **BLAST** setup to rising **BCLK**	15	-

Table B-15 ASB decoder output parameters.

Table B-15 ASB decoder output parameters

Parameter	Description	Time (ns)	Notes
T_{ovresp}	**BWAIT**, **BERROR** and **BLAST** valid after falling **BCLK**	10	-
T_{ovdsel}	**DSEL** valid after rising **BCLK**	n/a	**DSEL** is internally generated, not visible at the pins

Table B-16 shows the ASB arbiter input parameters. Applies only to the module or motherboard implementing the arbiter.

Table B-16 ASB arbiter input parameters

Parameter	Description	Time (ns)	Notes
T_{isareq}	**AREQ** setup to falling **BCLK**	10	-
T_{isresp}	**BWAIT** setup to rising **BCLK**	10	-

Table B-17 ASB arbiter output parameters. Applies only to the module or motherboard implementing the arbiter.

Table B-17 ASB arbiter output parameters

Parameter	Description	Time (ns)	Notes
T_{ovagnt}	**AGNT** valid after falling **BCLK**	10	-

Table B-18 shows the ASB arbiter combinatorial parameters.

Table B-18 ASB arbiter combinatorial parameters

Parameter	Description	Time (ns)	Notes
$T_{lokagnt}$	Delay from valid **BLOK** to valid **AGNT**	n/a	Not applicable, arbiter samples all inputs

B.3.4 Notes on FPGA timing analysis

The system bus on all Integrator boards is routed between FPGAs. These FPGAs are routed with timing constraints shown in *AHB system bus timing parameters* on page B-4 and *ASB system bus timing parameters* on page B-7. The exact performance of a system depends on the timing parameters of the motherboard and all modules in the system. Some allowance is needed for clock skew, routing delays and number of modules (that is, loading).

Not all FPGAs meet the ideal timing parameters, because of the complexity of the design or routing congestion within the device. For this reason the PLL clock generators on Integrator default to a safe low value that all modules can achieve.

A detailed timing analysis involves calculating the input/output delays between modules for all parameters. In general, a simpler approach is to increase the operating frequency until the system becomes unstable. The maximum stable operating frequency for your board combination is likely to be a few MHz lower.

ARM processors and core module FPGAs do not dissipate large amounts of heat. However, to be sure of stable operation, run the test program for a few minutes. Experiments show that the FPGAs, when operating at maximum system bus frequency, slowly increase in temperature, but that the maximum is typically less than 35°C.

Appendix C
Interfacing to the System Bus

This appendix contains information about interfacing to the system bus on the Integrator/AP. It contains the following sections:

- *About the system bus* on page C-2
- *Interfacing with the ASB system bus* on page C-3
- *Interfacing to the AHB system bus* on page C-5.

C.1 About the system bus

All bus interfaces on Integrator are implemented in PLDs and FPGAs. This means that the bus type can be any one of a number of different standards. To date ASB and AHB versions of the Integrator system bus have been produced. Any piece of Integrator hardware may conform to one or more system bus standards, and boards can be upgraded in the field without any component or board modification using the progcards utility.

C.1.1 ASB and AHB

The system bus connects modules to the motherboard and can be either ASB or AHB, as dictated by the configuration of the system controller FPGA on the motherboard. The FPGA can have only one configuration at a time. The motherboard signals the correct bus protocol to core and logic modules using two static signals **CFGSEL[1:0]**. These signals are driven by the CPCI arbiter PLD with the encoding shown in Table C-1.

Table C-1 CFGSEL[1:0] encoding

CFGSEL[1:0]	Description
00	Little endian ASB
01	Reserved
10	Little endian AHB
11	Reserved

Note

If you change the system controller configuration, you must reprogram the CPCI arbiter at the same time so that the correct bus protocol is signaled. System controller configurations later than Revision A Build 49 (ASB) or Revision B Build 7 (AHB) indicate bus type by displaying an S or H on the alphanumeric display at power on.

C.1.2 Address decoding

The Integrator implements a distributed address decoding scheme. This means that each core or logic module must decode its own area of memory. A central decoder in the system controller FPGA responds with a bus error response on all areas of the address space that are not occupied by peripherals. This response is disabled when a core or logic module is fitted. It is important, when adding modules, to ensure that the expansion logic implements a decoder and responds to all memory accesses in the appropriate memory region.

C.2 Interfacing with the ASB system bus

The section contains the information to design modules or boards that interface with an Integrator using an ASB system bus.

C.2.1 DSEL generation

ASB slaves receive their select signal **DSEL** from the ASB decoder. However, because the Integrator uses a distributed address decoding scheme, there are no **DSEL** lines from the system controller to core or logic modules. Local **DSEL** signals must be produced on each logic module. For position independent modules (modules that can be fitted in any position in a stack) the ID bits must be factored into the decoder logic. This is fully described in the *Logic Module User Guide*. Examples are supplied with the logic modules.

The **DSEL** signal is only driven when there is an active transaction and the address is correctly decoded. This means that there is no need for ASB slaves to take account of the **BTRAN[1:0]** signals. This is in contrast to AHB where the select line **HSEL** is a combinatorial decode of address **HADDR** only.

C.2.2 Timing analysis and critical paths

Static timing analysis of ASB systems is often difficult because of the tristate implementation and the existence of combinatorial and false paths. In addition, both edges of the clock are used to sample signals and timing on certain signals changes during bus handover cycles. A thorough understanding of the ASB specification is required to do a detailed timing analysis. However there are usually some critical paths that can be optimized for best performance.

The general approach on Integrator has been to provide programmable clock generators so that maximum operating frequency can be achieved.

It is not always easy to establish the maximum frequency by looking at the individual timing parameters. Different modules are implemented with different FPGAs and so absolute maximum system bus frequency may vary depending on the type and number of modules fitted.

Critical path 1: BA decode and DSEL generation

ASB masters generate the address **BA** from the rising edge of the clock **BCLK**. The decoder must generate **DSEL**. This is sampled by slaves on the falling edge of **BCLK**. so that there is a single clock phase for address generation and decode. In a board implementation this includes 2 ON/OFF chip transitions.

For fastest operation, the clock to output time on the master and the address path through the decoder must be carefully controlled. The analysis is complicated by the fact that **BTRAN** is only valid about the falling edge of **BCLK** and so must be passed through a transparent latch. Static timing tools are generally not very good at analyzing paths through transparent latches, and there is a combinatorial path from **BTRAN** to **DSEL**.

During arbitration handover cycles, the address **BA** is generated from the falling edge of **BCLK** rather than the rising edge. It is, therefore, not possible to generate **BA** from a register in an FPGA input/output block (IOB) because there is always a multiplexor between registers and the device output. Minimizing the delay through this multiplexor is key to a fast system.

Critical path 2: BWAIT/BERROR/BLAST generation and setup to master

The decoder and all ASB slaves generate the response signals **BWAIT**, **BERROR** and **BLAST**. The responses from each slave and the decoder inside a FPGA must be multiplexed together to generate the external signal that is routed to other FPGAs on the system bus. ASB slaves generate their response after detecting **DSEL** on the falling edge of **BCLK**. The response is sampled by a master on the rising edge of the clock. Therefore, there is a single clock phase for response generation and setup to the master. Minimizing the delay through the multiplexor, and reducing the input setup time is key to a fast system.

Critical path 3: Tristate enables

The motherboard and modules communicate using a a tristate bus. This means that it is important to be able to enable and disable output buffers quickly. The logic controlling the tristate enable is often on the critical path, particularly for the transaction **BTRAN** and response (**BWAIT**, **BERROR**, and **BLAST**) signals. When designing systems, ensure that the tristate enable signal is driven directly from a register (some FPGAs have a tristate enable register in the IOB), although for ASB this is not always possible because some signals must only be turned on for a clock phase.

ASB prevents bus clash or contention by ensuring a phase of handover between masters. To improve speed it might be possible to accept some degree of contention to get more setup time or simplify the output enable logic. This would be a violation of the ASB specification so care must be taken. The consequence of contention is an increase in power consumption (therefore temperature), but not physical damage. This is because *Field-Effect Transistors* (FETs) have a gain that reduces with temperature, which means they do not suffer the thermal-runaway destruction of bipolar devices.

C.3 Interfacing to the AHB system bus

The section contains the information to design modules or boards that interface with an Integrator using an AHB system bus.

C.3.1 HSEL generation

AHB slaves receive their select signal **HSEL** from the AHB decoder. However, because the Integrator uses a distributed address decoding scheme, there are no **HSEL** lines from the system controller to core or logic modules. Local **HSEL** signals must be produced on each logic module. For position independent modules (modules that can be fitted in any position in a stack) the ID bits need to be factored into the decoder logic. This is fully described in the *Logic Module User Guide*.

The **HSEL** signal is a combinatorial decode of address **HADDR** only. This means that each slave must also take account of the transaction type **HTRANS[1:0]**. This is in contrast to ASB where the decoder manages the transaction type and generates an appropriate select line **DSEL**.

C.3.2 Timing analysis and critical paths

Static timing analysis of AHB systems is easier than ASB due to the fact that it is a single clock edge design and no transparent latches are required. The AHB specification favours an interconnection scheme based on unidirectional busses and multiplexors rather than tristate busses. Within a single FPGA this means that there are fewer false paths and timing constraints can often be reduced to a simple period constraint on the clock.

There is a full clock cycle to generate and decode all signals, which means that critical paths based do not occur on AHB. This means that, typically, an AHB implementation runs faster than the equivalent ASB implementation, although there is increased complexity due to the increased level of pipelining on the bus.

The Integrator system implements a version of AHB based on tristate busses at the FPGA interconnection level.

Critical path 1: Tristate enables and contention

The motherboard and modules communicate using a a tristate bus. This means that it is important to be able to enable and disable FPGA output buffers quickly. The logic controlling the tristate enable is often on the critical path, particularly for the address **HADDR**, transaction **HTRANS[1:0]** and response **HREADY**, **HRESP[1:0]** signals. When designing systems ensure that the tristate enable signal is driven directly from a register (some FPGAs have a tristate enable register in the IOB).

ASB prevents bus clashes or contention by ensuring a phase of handover between masters, but AHB does not. This results in some degree of bus contention because different modules turn their tristate enables ON and OFF at different speeds. The consequence of contention is an increase in power consumption (and, therefore, temperature), but not physical damage. This is because FETs have a gain that reduces with temperature, which means they do not suffer the thermal-runaway destruction of bipolar devices. It is possible to reduce the contention by ensuring that modules turn ON slowly and OFF quickly, but this then reduces the maximum speed, which might not be desirable

C.3.3 Tristate AHB implementation

A unidirectional bus topology based on multiplexors works for subsystems contained in a single device. However, FPGAs with very high numbers of input/output pins are needed to implement unidirectional buses at the board level. It difficult to implement a system that allows additional modules to be attached because the size of the multiplexors is not known. For this reason the Integrator system implements a bidirectional tristate version of AHB buses at the FPGA interconnection level.

C.3.4 Tristate enable control

The VHDL code segments shown in Example C-1 describe how to drive the AHB signals tristate for use on Integrator platform boards.

Example C-1 Tristate enable control

```
Integrator System Bus uses tristate multiplexing

  -- Combined HGRANT signal for all internal masters
  HGRANTi <= HGRANT(0);  -- OR HGRANT(1) OR HGRANT(2) ...

  -- Registered versions of the HGRANT signals are used to control the master address
  -- multiplexors because the HGRANT signals are valid in the ARBITRATION phase
  -- and the multiplexed outputs are needed in the ADDRESS phase (which follows
  -- it one HREADY later).

  p_MasterEnable : process(HCLK, HRESETn, HGRANTi)
  begin
    if ((HRESETn = '0') and (HGRANTi = '0')) then
      AhbMasterEn <= '0';
    elsif rising_edge(HCLK) then
      if ((HREADY = '1') or (HRESETn = '0')) then
        AhbMasterEn <= HGRANTi;
      end if;
    end if;
end process;

-- Master outputs
-- Internal signals from the master to slave multiplexor are suffixed i
HTRANS <= HTRANSi when (AhbMasterEn = '1') else (others => 'Z');
HBURST <= HBURSTi when (AhbMasterEn = '1') else (others => 'Z');
HWRITE <= HWRITEi when (AhbMasterEn = '1') else 'Z';
HADDR  <= HADDRi  when (AhbMasterEn = '1') else (others => 'Z');
HSIZE  <= HSIZEi  when (AhbMasterEn = '1') else (others => 'Z');
HPROT  <= HPROTi  when (AhbMasterEn = '1') else (others => 'Z');

p_SlaveEnable : process(HCLK, HRESETn, HSELall)
begin
  if ((HRESETn = '0') and (HSELall = '0')) then
    AhbSlaveEn <='0';
  elsif rising_edge(HCLK) then
    if ((HREADY = '1') or (HRESETn = '0')) then
      -- HSELall is the logical OR of all internal HSEL signals
      AhbSlaveEn <= HSELall;
    end if;
  end if;
end process;

-- Slave responses
-- Internal signals from the slave to master multiplexor are suffixed i
HRESP  <= HRESPi  when (AhbSlaveEn = '1') else (others => 'Z');
HREADY <= HREADYi when (AhbSlaveEn = '1') else 'Z';

p_HDataEnable : process(HCLK, HRESETn)
begin
  if (HRESETn = '0') then
    HDMasterEn <= '0';
    HDSlaveEn  <='0';
  elsif rising_edge(HCLK) then
    if (HREADY = '1') then
      HDMasterEn <= AhbMasterEn and HWRITE;
      HDSlaveEn  <= HSELall and not HWRITE;
    end if;
  end if;
end process;

-- Master and slave data
-- HRDATAslave is the internal slave read data bus
-- HRDATAmaster is the internal master write data bus
HDATAOUT <= HWDATAmaster when (HDMasterEn = '1') else HRDATAslave;
HDEn     <= HDMasterEn or HDSlaveEn;
HDATA    <= HDATAOUT when (HDEn = '1') else (others => 'Z');
```

Index

The items in this index are listed in alphabetical order, with symbols and numerics appearing at the end. The references given are to page numbers.

A

Access types, PCI subsystem 5-3
Address decode C-2
Address decode, logic modules 4-4
Advanced peripheral bus 1-6
AHB timing parameters B-4
AHSB interfacing C-5
Alias memory region 4-5
Alphanumeric characters 4-25, 4-27
Alphanumeric display 1-6
  control 4-25
  segment designation 4-26
AMBA transactions 5-7
Am29K bus mode 5-7
Arbitration
  counters 3-9
  local bus 3-7, 5-8
  PCI bus 5-4
  system bus 3-2
  timeout register 4-15
Architecture, interrupt controller 3-19
Architecture,PCI subsystem 5-2
ARM email address xviii
ARM Firmware Suite 5-16
ARM SWAP instruction 5-7
ASB bridge 1-6
ASB interfacing C-3
ASB timing parameters B-7
Assembled Integrator system 2-3
ATX PC power supply 2-7
ATX power OK 3-13

B

Baud rate divisor 4-45, 4-46
Baud rates divisors, typical 4-46
Baud rate, programming 3-26, 3-27, 4-46
Bench power supply 2-7
Block diagrams
  counter/timer 3-22
  interrupt controller 3-19
  PCI subsystem 5-3
  RTC 3-24
  serial interface 3-26
  system 1-5
Boot monitor, using 2-10
Boot ROM 1-4, 1-7, 2-6, 2-13, 4-9
Boot ROM access 4-9
Boot switch 3-2
Boot switch reader 4-25
Built-in self tests 2-12
Burst indication 4-17
Bus arbitration 3-7
Bus arbitration counters 3-9
Bus arbitration scheme 3-9
Bus arbitration signal assignment 3-8
Bus architecture 3-3
Bus clock, system 3-15, 3-16
Bus interface
  expansion 1-8
  PCI 1-8
  system 1-8
Bus interfaces
  external 3-11
Bus master abort, PCI 5-17
Byte enable bits 4-17

C

Calculating the baud rate divisor 4-46
Calculating the system bus clock
frequency 3-16
CE Declaration of Conformity iii
CFGSEL[1:0] signals 3-5
Chip selects, EBI 3-5, 3-11
Clear to send 4-50
CLK24MHZ 3-15
Clock
  output divider 3-15
  VCO divider 3-15
Clock dividers (ICS525) 3-15
Clock divisor register, KMI 4-54
Clock generator 1-4, 1-7, 3-15
Clock generator, PCI 5-2
Clock rate registers 3-2
Clock speed, control 1-7
Clocks
  KMI and timer 3-15
  UART 3-15, 3-18
Clock, reference divider 3-15
Code start locations 2-6
CompactPCI auxiliary interrupt 5-17
CompactPCI auxiliary interrupts 5-18
CompactPCI bus 5-4, 5-15, 5-21
CompactPCI bus arbiter 5-2
CompactPCI clock 3-15
CompactPCI clock divider values 3-17
CompactPCI reset 3-12
CompactPCI subsystem clock 3-15
Configurable clock datasheet xvii
Configuration cycles, PCI 5-12
Connectors
  EXPB 4-36, A-8
  EXPM 2-4, 4-6
  HDRA 2-4
  HDRA/ EXPA A-2
  HDRB 2-4, A-5
  summary 1-9
Controllers
  clock 3-15
  interrupt 3-19
  keyboard 3-28
  mouse 3-28
Core module alias memory 4-4, 4-5
Core module alias memory, PCI address
5-13
Core module ID 2-5
Core module interface 3-3
Core module memory mapping 4-5
Core module reset 3-12
Core modules 2-3
Core modules, attaching 2-4
Core modules, maximum number 2-4
Counter/timer 1-4
  block diagram 3-22
Counter/timer modes
  free running 3-23
  periodic 3-23
Counter/timer registers 4-23
Counter/timers 1-6, 3-2, 3-22
CPCI power fail 3-13
CPCI reset 3-13
CP_CLK 3-15
CP_FAL signal 3-12
CP_RST signal 3-12
CP_V (I/O) terminal 2-8
CS_CTRL 3-14
Current count value 4-23
Current counter value 4-38
Cycle counter 3-9
Cycle counter programming 3-9
Cycle counter value 4-15

D

Damage prevention 2-5
Data carrier detect 4-50
Data direction, GPIO 4-36
Data ouptut set, GPIO 4-36
Data register output clear, GPIO 4-36
Data set ready 4-50
Data transmitter status, KMI 4-53
Decoder status register 4-14
DEG# signal 5-18
DIP switch register 4-28
DIP switches, setting 2-6
Direction of transfer bits 4-17
Distributed address decode C-2
DIVX/Y bit 3-17, 4-12
DMA controller, PCI 5-9
DMA finished, PCI 5-17
Document confidentiality status iii
DONE signal 3-13
Down counter 3-22
DSEL generation C-3
DTR control 4-13

E

EBI 1-6, 1-7
  wait states 4-20
EBI chip selects 3-5, 3-11, 4-6
EBI configuration registers 4-19
EBI memory map 4-6
EBI memory size 4-20
EBI write enable 4-20
EBI_CSR 3-11
Electrical specification B-3
Enable register, flag 4-18
Enabling UART FIFOs 4-44
ENUM# signal 5-18
EXPA connector A-2
Expansion bus interface 4-6
Expansion card present bit 4-14
Expansion memory space 3-11
Expansion reset 3-14
EXPB connector 4-36, A-8
EXPM connector 4-6
EXPM signal description A-11
External bus interface 1-4, 1-8, 3-2,
3-11

F

Feedback xviii
FIFOs, PCI host bridge 5-9
FIQ 1-6
  controllers 3-20
  enable register 4-29
  raw status register 4-29
  register bit assignments 4-31
  register mapping 4-33
  status register 4-29
FIQ-to-processor assignment 4-29
Flag registers 4-18
Flash access 4-9
Flash memory 1-4, 1-7, 2-6
Flash memory write protection 1-7
Flash Vpp bit 4-14
Flash write protect bit 4-14
Flash write-protection 1-7

FPGA configured 3-13
FPGA DONE signal 3-12
FPGA register map 4-7
FPGA timing analysis B-10
Free running mode, timer 3-23
Frequency of UART clock 3-18
Front panel reset button 1-7
Functional block diagram, FPGA 3-2

G

General purpose input output 3-2
GPIO 4-36
GPIO controller 1-6
GPIO registers
  GPIO_DATACLR 4-37
  GPIO_DATAIN 4-37
  GPIO_DATAOUT 4-37
  GPIO_DATASET 4-37
  GPIO_DIRN 4-37

H

Hardware resets 3-12
HDRA connector 2-4, 2-5
HDRA/EXPB connector A-2
HDRB connector 2-4, 2-5, A-5
HDRB signals A-5
HSEL generation C-5

I

ID, core module 2-5
Inbound transaction 5-7
Inbound transactions 5-6
Initial count value 4-23
Initialization sequence, PCI subsystem 5-16
Initializing the PCI subsystem 5-16
Integrator, assembled 2-3
Intel 21152 5-15
Intel 21152 PCI-PCI Bridge 5-2
Interrupt control 4-29
Interrupt controller 1-4, 1-6, 3-2, 3-19
Interrupt controller block diagram 3-19
Interrupt controller registers 4-29
Interrupt identification register, KMI 4-54
Interrupts
  KMI 3-29
  module assigned 3-20
  PCI auxilliary 5-17
  PCI host bridge 5-17
  PCI timeout 5-18
  RTC 3-25
  UART 3-27
  UART modem 4-47
  UART receive 4-47
  UART receive timeout 4-47
  UART transmit 4-47
Interrupts PCI DMA finished 5-17
INTP signal 5-18
INTS signal 5-18
IRQ 1-6
  controllers 3-20
  enable register 4-29
  raw status register 4-29
  register bit assignments 4-31
  register mapping 4-32
  status registers 4-29
IRQ-to-processor assignment 4-29

J

JTAG signals 3-10

K

Keyboard and mouse interface 1-6, 3-2, 3-28
Keyboard controller 3-28
Keyboard type, selecting 4-52
KMI
  enable bit 4-52
  interrupts 3-29
  overview 3-29
KMI and timer clock 3-15, 3-18
KMI registers 4-51
  KMICLKDIV 4-54
  KMI_CR 4-52
  KMI_IIR 4-54
  KMI_STAT 4-53
KMICLKIN signal 3-29
KMIREFCLK signal 3-29

L

LB_IO_BASE 5-16
LB_IO_BASE register 5-11
LED control 1-6, 4-25
LED control register 4-27
LED driver 3-2
LED switch register 2-6
LEDs
  locations 1-10
LEDs, functional summary 1-11
LED_ALPHA bit-to-segment mapping 4-26
LED_ALPHA register 4-25
LED_SWITCHES register 2-6
Line control register, UART 4-44
Local bus arbitration 5-8
Local bus bridge, operation 5-6
Local bus bridge, transactions 5-6
Local bus to PCI windows 5-10
Location of LEDs 1-10
Location of test points 1-12
Lock bit 4-16, 4-22
Lock key 4-16, 4-22
Lock register 4-16, 4-22
Locked transactions A-3
Logic module address decoding 4-4
Logic module interface 3-3
Logic module interrupts 3-20
Logic module reset 3-12
Logic modules 2-3
Logic modules region 4-4
Logic modules, attaching 2-4
Logic modules, maximum number 2-4
Loopback enable, UART 4-47

M

Mailboxes, PCI 5-9
Match register 4-39
Maximum number of modules 2-4
Mechanical details B-2
Memory 1-7
  flash 2-6
Memory map

EBI 4-6
Logic module 4-4
Memory map, system 4-2
Memory size, EBI 4-20
Messages passing between processors, example 4-35
Modem interrupt status 4-51
Modem status 3-27
Modem status interrupt enable 4-47
Module-assigned signals 3-5
Modules, combined maximium 2-4
Motherboard bus interface 1-8
Motherboard detect 3-10
Mouse controller 3-28
Multi-ICE 3-14
Multi-ICE reset 3-12, 3-14

N

nPWOK signal 3-12
nRSTSRC5 signal 3-12
nSRST signal 3-14

O

OPDiv 3-16
Organization of manual xiv
Oscillator divisor register 4-12
Outbound transactions 5-6, 5-7
Output divider 3-15, 3-16

P

Parity bit, KMI 4-53
Parity enable, UART 4-45
Parity select, UART 4-45
PBRST 3-13
PBRST signal 3-12
PCI address, of system bus resources 5-13
PCI address, system bus access 5-10
PCI backplane 1-7
PCI bridge 3-2
PCI bridges 5-3
PCI bus 1-6
PCI bus arbitration 5-4
PCI bus clock usage 3-18
PCI bus interface 1-4, 1-8
PCI clock generator 5-2
PCI configuration cycles 5-12, 5-15
PCI expansion bus 5-4, 5-5, 5-15, 5-20
PCI host bridge interrupt 5-17
PCI host bridge operation 5-9
PCI IdSel 5-20
PCI local bus fault code register 4-17
PCI local bus fault registers 4-16
PCI read from system bus 5-19
PCI region 4-6
PCI space 5-10
PCI space mapping 5-4
PCI subsystem
access types 5-3
initialization 5-16
overview 5-2
topology 5-2
PCI subsystem interrupts 3-20
PCI to host bridge 5-9
PCI to local bus windows 5-13
PCI to PCI bridge operation 5-15
PCI write to system bus 5-19
Periodic mode, timers 3-23
Peripheral controllers 1-4
Peripheral input/output controllers 1-6
Peripheral register region 4-7
Peripheral registers 4-36
Pinouts
EXPB A-8
EXPM A-11
HDRA/EXPA A-2
HDRB A-5
keyboard A-15
mouse A-15
serial interface A-14
Power button 1-11
Powering the Integrator/AP 2-8
Preventing damage 2-5
PrimeCell UART 3-2, 3-26
Processor module present bit 4-14
Product status iii
Programing the transaction counter 3-9
Push-button reset 3-13
P_CLK signal 3-15

R

Read data input pins, GPIO 4-36
Read data ouput pins, GPIO 4-36
Reading the DIP switches 4-28
Real time clock 1-6, 3-2
Real time clock registers 4-38
Real-time clock 3-24
Receive FIFO empty, UART 4-49
Receive FIFO full, UART 4-49
Receive FIFO, UART 3-27
Receive interrupt enable, UART 4-47
Receive interrupt status 4-51
Receive timeout interrupt enable, UART 4-47
Receive timeout interrupt status 4-50
Receiver interrupt enable, KMI 4-52
Receiver register status, KMI 4-53
Receiver status, KMI 4-53
Reference divider 3-15
Register naming convention 4-7
Register window, PCI 5-11
Registers
EBI_CSRx 4-20
FIQx_RAWSTAT 4-33
FIQx_STATUS 4-33
GPIO_DATACLR 4-37
GPIO_DATAIN 4-37
GPIO_DATAOUT 4-37
GPIO_DATASET 4-37
GPIO_DIRN 4-37
INT_SOFTCLEAR 4-35
INT_SOFTSET 4-35
IRQx_RAWSTAT 4-32
IRQx_STATUS 4-32
KMI_CLKDIV 4-54
KMI_CR 4-52
KMI_IIR 4-54
KMI_STAT 4-53
LED_ALPHA 4-25
LED_LIGHTS 4-27
LED_SWITCHES 2-6, 4-28
RTC_CR 3-25, 4-40
RTC_DR 3-24, 4-38
RTC_EOI 3-25, 4-39
RTC_LR 3-24, 4-39
RTC_MR 3-25, 4-39
RTC_STAT 3-25, 4-39
SC_ARB 4-15

SC_CTRLCLR 4-13
SC_CTRLS 4-13
SC_DEC 4-14
SC_ID 4-11
SC_LBFADDR 4-16
SC_LBFCODE 4-17
SC_LOCK 4-16, 4-22
SC_OSC 3-16, 3-17, 4-12
TIMERx_CLR 3-23, 4-24
TIMERx_CTRL 3-23, 4-24
TIMERx_LOAD 3-23, 4-23
TIMERx_VALUE 3-23, 4-23
UART_CR 4-47
UART_DR 4-42
UART_ECR 4-44
UART_FR 4-48
UART_IIR 4-50
UART_LCRH 4-44
UART_LCRL 3-26, 4-46
UART_LCRM 3-26, 4-45
UART_RSR 3-26, 4-42
Remap 2-13, 4-9
Reset
Core module 3-12
logic module 3-12
signal descriptions 3-13
Reset button 1-11
Reset controller 1-4, 1-7, 3-2, 3-12
Resets 3-14
software 3-14
Resets, hardware 3-12
RETRACT response 5-8
ROM, RAM, and peripheral region 4-49
Round-robin arbitration 3-9
RTC block diagram 3-24
RTC interrupts 3-25
RTC overview 3-24
RTC registers
RTC_CR 3-25, 4-40
RTC_DR 3-24, 4-38
RTC_EOI 3-25, 4-39
RTC_LR 4-39
RTC_MR 3-25, 4-39
RTC_STAT 3-25, 4-39
RTC_LR register 3-24
RTS control 4-13
Rx interrupt, UART 3-27

S

SC 4-13
SC_LBFADDR register 5-18
SC_LBFCODE register 5-18
SC_OSC 3-15, 3-16
SC_OSC register 3-16, 3-17
SC_PCI register 5-16, 5-18
Serial interface pinout A-14
Serial interface word length 4-44
Serial interface, block diagram 3-26
Setting the DIL switches 2-6
Setting the system bus clock 3-16
Setting the UART baud rate 3-26, 3-27
Setting up the ARM Integrator/AP 2-2
Signal assignment
bus arbitration 3-8
Signal descriptions
serial interface A-14
Signal rotation 3-5, 3-20
Software interrupt set register 4-35
Software interrupts 3-20, 4-34
Software reset 1-7, 3-14
Software reset control 4-14
Speculative prefetch, PCI 5-9
Split transactions A-3
SRAM 1-7
Standby LED 2-8
Status register, flag 4-18
Stop bit select, UART 4-45
Supplying power
ATX PSU 2-7
bench PSU 2-7
Switch reader 1-6
Switch setting, DIL 2-6
SYSCLK divider values 3-16
SYSCLK signal 3-15
System block diagram 1-5
System bus 3-3
System bus address, PCI space 5-10
System bus arbiter 1-4, 1-6, 3-2
System bus architecture 3-3
System bus clock 3-9, 3-15, 3-16
System bus clock frequency, calculating 3-16
System bus clock usage 3-17
System bus configuration C-2
System bus fault codes 5-19
System bus interface 1-4, 1-6, 1-8, 3-2
System bus to local bus bridge 5-2
System bus write to PCI 5-19
System controller FPGA 3-2, 3-3, 3-7
System controller FPGA, overview 1-6
System controller interrupts 3-20
System controller registers 4-10
SC_ARB 4-15
SC_CTRLCLR 4-13
SC_CTRLS 4-13
SC_DEC 4-14
SC_ID 4-11
SC_LBFADDR 4-16
SC_LBFCODE 4-17
SC_LOCK 4-16, 4-22
SC_OSC 4-12
System information block 2-12
System memory map 4-2
System reset 3-14
System startup 2-10
System status and control registers 1-4, 1-7
System-wide FPGA configured 3-13
S_RDW 3-16
S_VDW 3-16, 4-12

T

Test point functions 1-12
Test points 1-12
Timeout interrupt, PCI 5-18
Timer control register 3-22
Timer enable bit 4-24
Timer load register 3-22
Timer mode bit 4-24
Timer modes 3-23
free-running 3-23
periodic 3-23
Timer prescale divisor 4-24
Timer registers
TIMERx_CLR 3-23, 4-24
TIMERx_CTRL 3-23, 4-24
TIMERx_LOAD 3-23, 4-23
TIMERx_VALUE 3-23, 4-23
Timing analysis, AHB C-5
Timing analysis, ASB C-3
Timing specification B-4
Topology of the PCI subsystem 5-2
Transaction counter 3-9

Transaction counter value 4-15
Transaction width A-4
Transmit FIFO empty, UART 4-49
Transmit FIFO full, UART 4-49
Transmit FIFO, UART 3-27
Transmit interrupt enable, UART 4-47
Transmit interrupt status, UART 4-51
Transmit register status, KMI 4-53
Transmitter interrupt enable, KMI 4-52
Tristate AHB implementation C-6
Tristate enable control C-6
Tx interrupt, UART 3-27

U

UART 3-2, 3-24, 3-25, 3-26
UART busy, status bit 4-49
UART clock 3-15, 3-18
UART enable 4-48
UART FIFO, enabling 4-44
UART interrupts 3-27
UART receive FIFO 3-27
UART registers 4-41
  UART_CR 4-47
  UART_DR 4-42
  UART_ECR 4-44
  UART_FR 4-48
  UART_IIR 4-50
  UART_LCRH 4-44
  UART_LCRL 4-46
  UART_LCRM 4-45
  UART_RSR 4-42
UART transmit FIFO 3-27
UARTCLK 3-18
UARTCLK divider values 3-18
UARTCLK signal 3-15
UARTs 1-6
UARTx DTR control 4-13
UARTx RTS control 4-13
UART, overview 3-26
UART_IIR 4-50
UART_LCRL 4-46
UART_LCRL register 3-26, 3-27
UART_LCRM register 3-26, 3-27
UART_RSR register 3-26
Using Integrator/AP in a CompactPCI rack 2-9
Using the software interrupt register 4-35

V

VCO divider 3-15, 3-16
V360EPC 5-9
V360EPC host bridge controller 4-6
V360EPC internal registers 5-13
V360EPC PCI Host Interface controller 5-2
V360EPC reset 5-16

W

WAIT response 5-8
Wait states, EBI 4-20
Windows, local bus-PCI 5-10
Write enable
  EBI 4-20
Write FIFOs, PCI local bridge 5-7
Write protection, EBI 3-11
Writing characters to alphanumeric display 4-25, 4-27