Integrator™/CM922T-XA10

Core Module (HBI-0100)

User Guide

ARM

ARM DUI 0184B

Integrator/CM922T-XA10
User Guide

Release Information

Date	Release	Comments
September 2002	A	First release
November 2003	B	Second release, updated register information

Proprietary Notice

Confidentiality Status

This document is Non-Confidential. The right to use, copy and disclose this document may be subject to license restrictions in accordance with the terms of the agreement entered into by ARM and the party that ARM delivered this document to.

Product Status

The information in this document is final, that is for a developed product.

Web Address

`http://www.arm.com`

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

CE

The system should be powered down when not in use.

The Integrator generates, uses, and can radiate radio frequency energy and may cause harmful interference to radio communications. However, there is no guarantee that interference will not occur in a particular installation. If this equipment causes harmful interference to radio or television reception, which can be determined by turning the equipment off or on, you are encouraged to try to correct the interference by one or more of the following measures:

- ensure attached cables do not lie across the card
- reorient the receiving antenna
- increase the distance between the equipment and the receiver
- connect the equipment into an outlet on a circuit different from that to which the receiver is connected
- consult the dealer or an experienced radio/TV technician for help

Note

It is recommended that wherever possible Shielded interface cables be used.

Contents
Integrator/CM922T-XA10 User Guide

List of Tables
Integrator/CM922T-XA10 User Guide

List of Figures
Integrator/CM922T-XA10 User Guide

Preface

This preface introduces the *Integrator/CM922T-XA10 User Guide*. It contains the following sections:

About this book

This book describes how to set up and use your Integrator/CM922T-XA10 core module.

Intended audience

This document has been written for hardware and software developers to aid the development of ARM-based products using the core module as part of a development system.

Using this book

This book is organized into the following parts and chapters:

Chapter 1 ***Introduction***

Read this chapter for an introduction to the core module. This chapter describes the main features and identifies the main components on the core module.

Chapter 2 ***Getting Started***

Read this chapter for a description of how to set up the core module ready for use.

Chapter 3 ***Using Multi-ICE, ByteBlaster, and Trace***

Read this chapter for a description of the debug support provided by the core module.

Chapter 4 ***Integrator/CM922T-XA10 System Architecture***

Read this chapter for a description of the platform architecture. This includes the main system bus, memory, and other physical components on the core module.

Chapter 5 ***Basic Example Image for Simple Standalone Operation***

Read this chapter for a description of the *Base Example* (BE) image that can be programmed into the PLD. This includes how it implements interfaces to the system buses, memory, and control registers.

Chapter 6 ***CM Image for Operation Standalone or with an Integrator/AP***

Read this chapter for a description of the *Core Module* (CM) functional image that can be programmed into the PLD. This includes how it implements interfaces to the system buses, memory controllers, and control registers.

Chapter 7 ***CP Image for Operation with an Integrator/CP Baseboard***

Read this chapter for a description of the Integrator/CP922T system. This includes how it implements interfaces to the system buses, memory controllers, and control registers.

Chapter 8 ***IM-PD1 Image for Operation with an Integrator/IM-PD1***

Read this chapter for a description of the Integrator/CM922T-XA10 and IM-PD1 system. This includes control registers.

Appendix A ***Signal Descriptions***

Read this appendix for pinouts of the core module connectors.

Appendix B ***Excalibur PLD Pinout***

Read this appendix for signal pinout details of the Altera Excalibur PLD.

Appendix C ***Mechanical Specification***

Read this appendix for dimensional information about the Integrator/CM922T-XA10.

Typographical conventions

The following typographical conventions are used in this book:

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

Further reading

This section lists publications from both ARM Limited and third parties that provide additional information on developing code for the ARM family of processors.

ARM periodically provides updates and corrections to its documentation. See `http://www.arm.com` for current errata sheets and addenda.

See also the ARM Frequently Asked Questions list at:
`http://www.arm.com/DevSupp/Sales+Support/faq.html`

ARM publications

The following publication provides information about related ARM Integrator products:

- *ARM Integrator/AP User Guide* (ARM DUI 0098)
- *ARM Integrator/IM-PD1 User Guide* (ARM DUI 0152)
- *ARM Integrator/CP User Guide* (ARM DUI 0159).

The following publication provides information about the related ARM core:

- *ARM922T (Rev 0) Technical Reference Manual* (ARM DDI 0184).

The following publications provide information about ARM PrimeCell® devices that can be used to control the interfaces described in this manual:

- *ARM PrimeCell UART (PL011) Technical Reference Manual* (ARM DDI 0183*)*
- *ARM PrimeCell Synchronous Serial Port Master and Slave (PL022) Technical Reference Manual* (ARM DDI 0171)
- *ARM PrimeCell Advanced Audio CODEC Interface (PL041) Technical Reference Manual* (ARM DDI 0173)
- *ARM PrimeCell GPIO (PL061) Technical Reference Manual* (ARM DDI 0190)
- *ARM PrimeCell Color LCD Controller (PL110) Technical Reference Manual* (ARM DDI 0161)
- *ARM PrimeCell Smart Card Interface (PL130) Technical Reference Manual* (ARM DDI 0148)
- *ARM PrimeCell Multimedia Card Interface (PL181) Technical Reference Manual* (ARM DDI 0205)
- *ARM PrimeCell Vectored Interrupt Controller (PL190) Technical Reference Manual* (ARM DDI 0181).

The following publications provide reference information about the ARM architecture:

- *AMBA Specification* (ARM IHI 0011)
- *ARM Architecture Reference Manual* (ARM DDI 0100).

The following publications provide information about related ARM development tools:

- *ARM Multi-ICE User Guide* (ARM DUI 0048)
- *ARM Firmware Suite Reference Guide* (ARM DUI 0102)
- *ARM Software Development Toolkit User Guide* (ARM DUI 0040)
- *ARM Software Development Toolkit Reference Guide* (ARM DUI 0041)
- *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).

Other publications

The following publications provide information about third-party devices used on the module:

- *Excalibur ARM-Based Embedded Processor PLDs Hardware Reference Manual* (A-DS-EXCARMD), Altera Corporation
- *MicroClock OSCaR User Configurable Clock Data Sheet* (ICS307), MicroClock Division of ICS, San Jose, CA.

Feedback

ARM Limited welcomes feedback on all Integrator products and their documentation.

Feedback on the core module

If you have any problems with the core module, contact your supplier. To help us provide a rapid and accurate response, give:

- Details of the boards you are using. Quote the exact name screen printed on the board(s) and any HBI number(s) that are shown on label(s) on the board(s).
- Details of the host platform you are using, operating system type, and version.
- A small standalone sample of code that reproduces the problem.
- A clear explanation of what you expected to happen, and what actually happened.
- The commands you used, including any command-line options.
- Sample output illustrating the problem.
- The version string of the tool, including the version number and date.

Feedback on this book

If you have any comments on this book, send email to `errata@arm.com` giving:

- the document title
- the document number
- the page number(s) to which your comments apply
- a concise explanation of your comments.

General suggestions for additions and improvements are also welcome.

Chapter 1
Introduction

This chapter introduces the Integrator/CM922T-XA10 core module. It contains the following:

1.1 About the Integrator/CM922T-XA10

The Integrator/CM922T-XA10 core module is a compact development platform that enables you to develop ARM-based products by using an Altera Excalibur Embedded ARM Processor PLD.

The Excalibur device contains an ARM922T core and a user programmable *Programmable Logic Device* (PLD). The PLD is programmed when the module is powered ON with one of four images that are stored in flash memory:

Basic Example image

This design configures the PLD with a basic example image. It contains registers to set the on-board clocks and to read the user switches. It is designed to enable you to develop and add your own IP for use with the ARM core. This image is signified by the code b01010101 being displayed by the user LEDs (0=OFF, 1=ON) when the system is powered ON.

CM image This design configures the PLD with a design that functions as an Integrator ARM922T with an AHB interface. This represents a standard Integrator core module that can be used standalone or with an Integrator/AP motherboard. The core module can be used on its own (standalone) and is capable of running boot monitor or GDBstub over a serial line. This image is signified by the code b11100111 being displayed by the user LEDs (0=OFF, 1=ON) when the system is powered ON.

CP image This design enables you to use the core module mounted onto an Integrator/CP baseboard. The PLD is programmed with the appropriate image containing PrimeCell peripherals and interface controls for the interfaces provided by the baseboard. This image is signified by the code b11001100 being displayed by the user LEDs (0=OFF, 1=ON) when the system is powered ON.

IM-PD1 image

This design configures the core module to operate with an Integrator/IM-PD1 mounted onto it. The image contains the appropriate PrimeCell peripherals and controls to drive the interfaces provided by the IM-PD1. This image is signified by the code b11110000 being displayed by the user LEDs (0=OFF, 1=ON) when the system is powered ON.

Note

All four images are programmed into flash during board manufacture. However, the source code for the basic example and the CM images only are supplied with the Integrator/CM922T-XA10. Source code for the CP and IM-PD1 images are supplied with the Integrator/CP and Integrator/IM-PD1 products respectively.

Figure 1-1 shows the physical layout of the Integrator/CM922T-XA10.

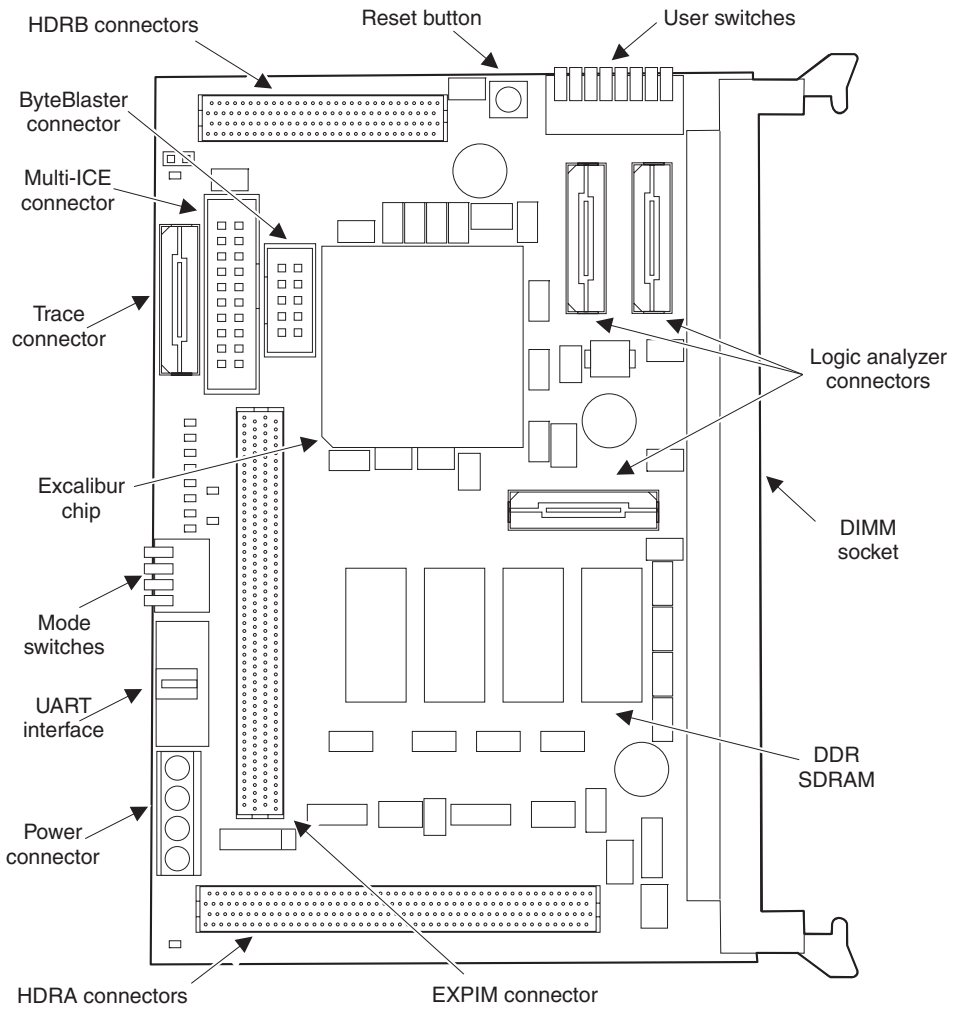


Figure 1-1 Integrator/CM922T-XA10 core module layout

1.2 Main features

This section describes the main features of the core module as follows:

- *Core module components*
- *Altera Excalibur EPXA10 features*.

1.2.1 Core module components

The major components of the core module are as follows:

- Altera Excalibur EPXA10 ARM-based embedded processor PLD
- 24MB flash (8MB for configuration, 16MB user)
- 128MB of DDR SDRAM fitted to the board
- 2MB ZBT SSRAM (controller must be implemented in the PLD)
- up to 256MB of SDRAM in a DIMM socket (optional, controller must be implemented in the PLD and some signal traces are shared with SSRAM)
- clock generators
- Integrator system bus connectors
- Multi-ICE and ByteBlaster connectors
- general-purpose logic analyzer connectors
- trace port
- serial port.

1.2.2 Altera Excalibur EPXA10 features

The Altera Excalibur EPXA10 ARM-based embedded processor device contains:

- embedded processor stripe that incorporates:
  - ARM922T embedded microprocessor core
  - *Embedded Trace Macrocell* (ETM)
  - 256KB of single-ported SRAM and 128KB of dual-ported SRAM
  - SDRAM controller for off-chip SDRAM
  - *External Bus Interface* (EBI) that supports off-chip flash, ROM, and SRAM
  - PLL
  - UART
  - general-purpose timers and watchdog timer
  - interrupt controller
  - JTAG port
  - master and slave AHB ports for the PLD.
- user-programmable PLD.

1.3 Core module connectors

Table 1-1 lists the core module connectors that are shown in Figure 1-1 on page 1-3.

Table 1-1 Core module connectors

Connector	Name	Function
J1	UART	Serial interface connector to the UART within the processor stripe.
J2	-	DIMM socket.
J3	POWER	Power input connector. Use this connector to supply power for standalone operation or operation with an IM-PD1. Do not use if fitted to a baseboard.
J4	TRACE	Trace debug connector.
J5	LA3	Logic analyzer connector.
J6	M-ICE	Multi-ICE JTAG hardware debug connector
J7	LA2	Logic analyzer connector.
J8	CONFIG	Config enable. Only fit to enable reprogramming of system FGAs and PLDs.
J9	B-BLAST	ByteBlaster connector for use with Altera development tools.
J10	LA1	Logic analyzer connector.
J11	EXPIM	Interface signals connector on the underside of the core module (not fitted).
J12	HDRB/EXPB	System bus and interface signals socket to baseboard (on underside).
J13	HDRA/EXPA	System bus socket to baseboard (on underside).
J14	EXPIM	Interface signals connector to an IM-PD1, if fitted.
J15	HDRA/EXPA	System bus connector to the next module in the stack.
J16	HDRB /EXPB	System bus and interface signals connector to the next module in the stack.

1.4 Core module LEDs, switches, and links

The core module LEDs, switches, and links are shown in Figure 1-2.

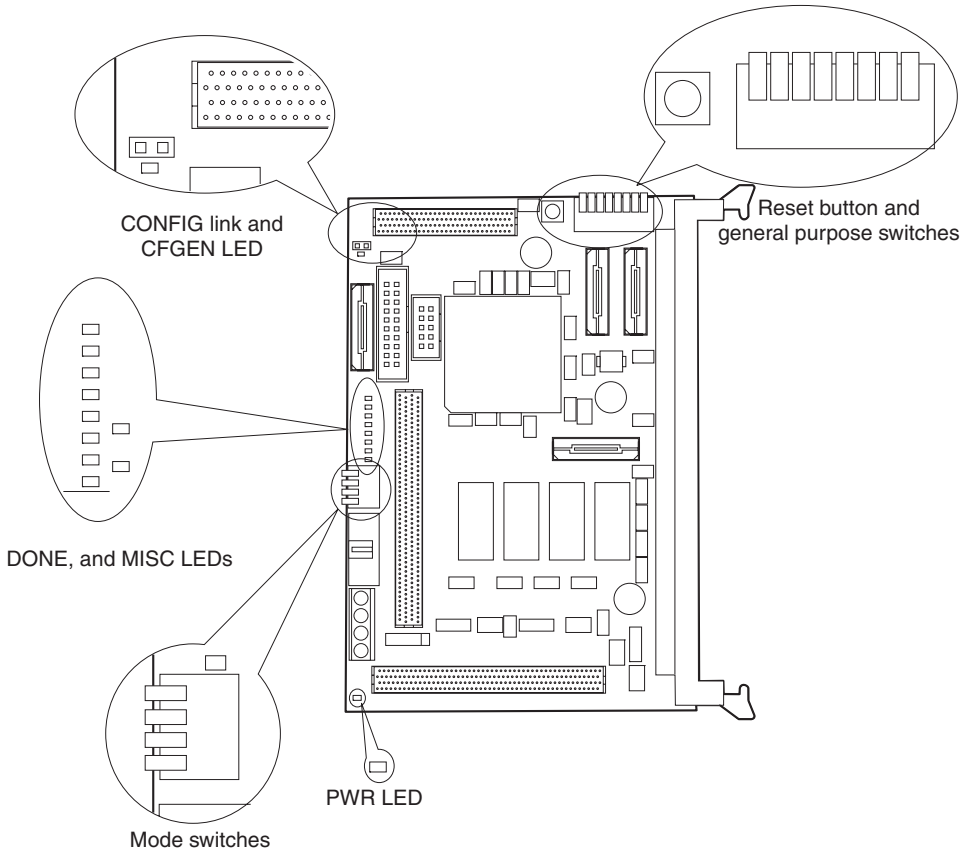


Figure 1-2 LEDs, links, and switches

1.4.1 LEDs

The functions of the LEDs are shown in Table 1-2.

Table 1-2 LED functional summary

Name	Color	Function
DONE	Green	This LED illuminates when the PLD has been successfully programmed following power-on.
PWR	Green	This LED illuminates to indicate that a 3.3V supply is present.
CFGEN	Orange	This LED illuminates to indicate that CONFIG mode is enabled. CONFIG mode is enabled by fitting the CONFIG link on this module or another module in the same stack.
USER[7:0]	Green	User LEDs controlled by a register or by user-implemented logic programmed into the PLD. Immediately after power ON and when using the supplied PLD images, these LEDs display a code that indicates which image has been loaded into the PLD. At other times they are used as general purpose indicators.
MISC	Green	Miscellaneous LED that is controlled by a register or by user-implemented logic programmed into the PLD.

1.4.2 Switches

The core module provides two DIP switches and a push button as follows:

Mode switches, S1

This is a four-pole DIP switch (S1) that is used to select the image that is programmed into the PLD when the module is powered ON. The image that you select must be compatible with the system setup.

Caution

To ensure correct operation of your Integrator platform, ensure that you select the correct PLD image for your platform configuration.

General-purpose/boot code switches, S2

This is an eight-pole DIP switch (S2) that can be used for any purpose. During system startup, however, switches S2[8] and S2[7] are used to control the boot mode, see *Setting the general purpose/boot code switch* on page 2-8. At all other times these switches can be used for any purpose.

The PLD images supplied with the core module provide a register that you can use to read the switch settings.

Reset button

This is a general purpose push button that is connected through a debounce network to an input/output pin of the PLD. It is used by the supplied PLD images to trigger a reset.

1.4.3 CONFIG link

The CONFIG link, shown in Figure 1-2 on page 1-6, selects between debug and configuration mode:

Debug mode This is the normal operating mode selected when the CONFIG jumper is not fitted. The processor core and debuggable devices on other modules are accessible on the scan chain, as shown in Figure 3-2 on page 3-4. You can use debug mode to download and run programs.

Config mode This mode is selected when the CONFIG jumper is fitted. The debuggable devices are still accessible and, in addition, all FPGAs and PLDs in the system are added into the scan chain. This enables the PLD image to be reconfigured or upgraded in the field.

For more details on configuration see Chapter 3 *Using Multi-ICE, ByteBlaster, and Trace*.

1.5 Precautions

This section contains safety information and advice on how to avoid damage to the core module.

1.5.1 Ensuring safety

The core module is powered from 3.3V and 5V DC supplies. It also uses a 12V supply when used in conjunction with an Integrator/IM-PD1 with color LCD controller.

Warning

To avoid a safety hazard, only connect *Safety Extra Low Voltage* (SELV) equipment to the JTAG interface.

1.5.2 Preventing damage

The core module is intended for use within a laboratory or engineering development environment. It is supplied without an enclosure which leaves the board sensitive to electrostatic discharges and allows electromagnetic emissions.

Caution

To avoid static damage to the board, observe the following precautions:

- never subject the board to high electrostatic potentials
- always wear a grounding strap when handling the board
- only hold the board by the edges
- avoid touching the component pins or any other metallic element.

Do not use the board near equipment that is:

- sensitive to electromagnetic emissions (such as medical equipment)
- a transmitter of electromagnetic emissions.

Caution

To prevent damage to your Integrator system connectors, observe the following precautions:

- When removing a core or logic module from a motherboard, or when separating modules, take care not to damage the connectors. Do not apply a twisting force to the ends of the connectors. Loosen each connector first before pulling on both ends of the module at the same time.

- Use the system in a clean environment and avoid debris fouling the connectors on the underside of the PCB. Blocked holes can cause damage to connectors on the motherboard or module below. Visually inspect the module to ensure that connector holes are clear before mounting it onto another board.

1.5.3 Ensuring correct operation

The Integrator/CM922T-XA10 is supplied with multiple PLD configuration images programmed into flash. Is important that you select the correct one for your platform configuration.

Caution

To ensure correct operation of your Integrator platform, ensure that you select the correct PLD image for your platform configuration.

Chapter 2
Getting Started

This chapter describes how to set up the core module ready for use. It contains the following sections:

2.1 Preparing the system hardware

The core module can be used as a compact standalone development platform or, depending on PLD configuration, with an Integrator motherboard and other modules. If you are using the core module with other Integrator products, first assemble the system as described in one of the following sections:

- *Fitting an Integrator/IM-PD1 onto the core module* on page 2-3
- *Mounting the core module onto an Integrator/CP* on page 2-4
- *Mounting the core module onto an Integrator/AP* on page 2-5.

The steps required to complete the set up of the core module are described in the following sections:

- *Setting the mode switch* on page 2-6
- *Setting the general purpose/boot code switch* on page 2-8
- *Connecting power* on page 2-9
- *Connecting programming and debug equipment to the Integrator/CM-XA10* on page 2-10.

2.1.1 Fitting an Integrator/IM-PD1 onto the core module

Figure 2-1 shows the core module with an IM-PD1 mounted onto it.

Ensure that the core is powered down and then attach the IM-PD1 to the core module using the stacking connectors HDRA and HDRB, and interface expansion connector EXPIM. It requires considerable force to fully engage the connector, so ensure that all three connectors are in perfect alignment before pressing the boards together.

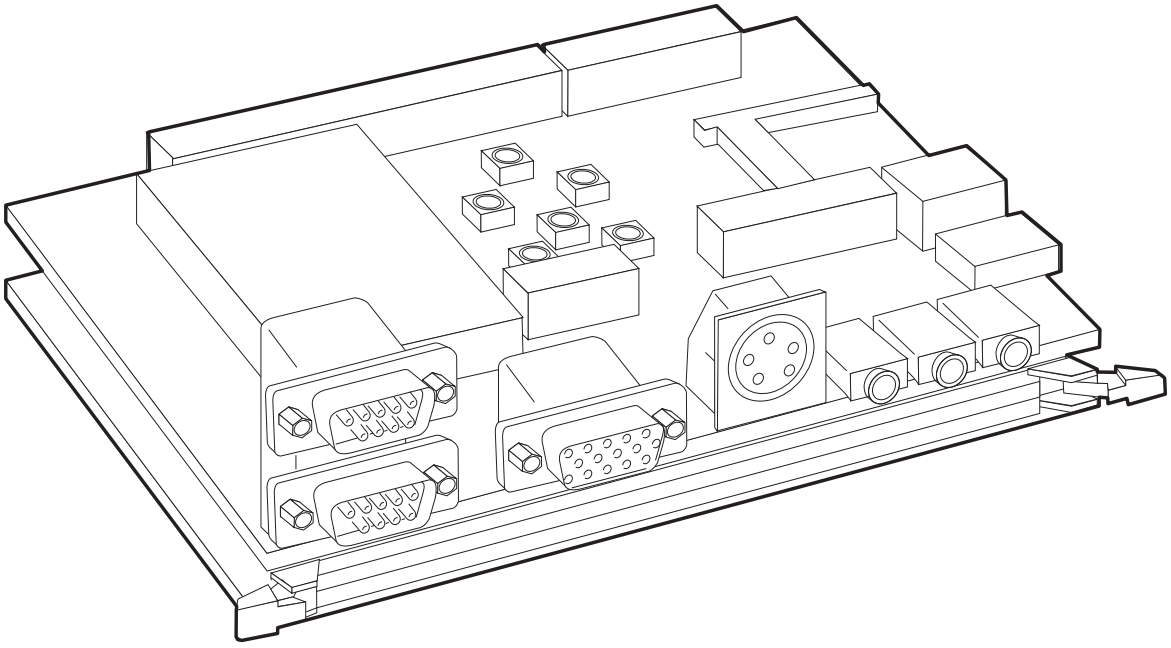


Figure 2-1 IM-PD1 mounted on the core module

2.1.2 Mounting the core module onto an Integrator/CP

Figure 2-2 shows the core module mounted onto an Integrator/CP baseboard.

Ensure that the core is powered down and then attach the core module to the baseboard using the stacking connectors HDRA and HDRB. It requires considerable force to fully engage the connector, so ensure that the two connectors are in perfect alignment before pressing the boards together.

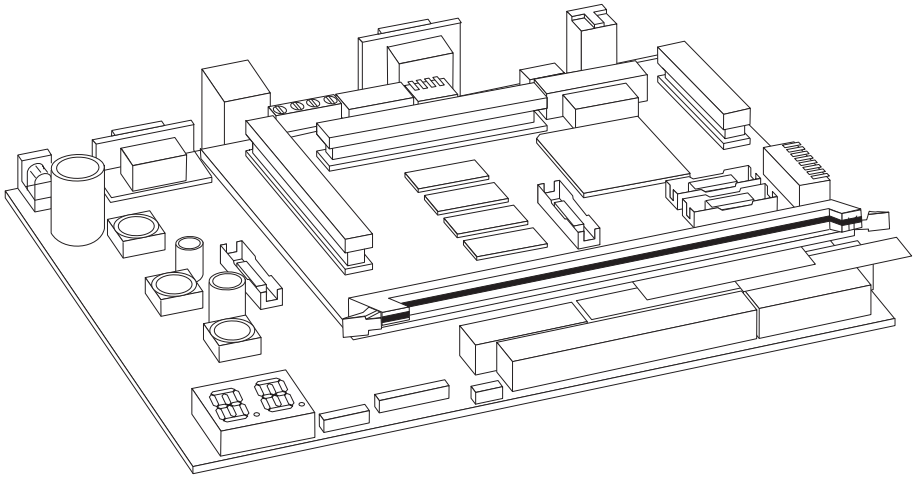


Figure 2-2 Core module mounted on an Integrator/CP baseboard

2.1.3 Mounting the core module onto an Integrator/AP

Figure 2-3 shows an Integrator/AP motherboard fitted with four core modules and a logic module.

Ensure that the system is powered down and then attach the core module to the Integrator/AP using the stacking connectors HDRA and HDRB on the motherboard. It requires considerable force to fully engage the connectors, so ensure that the two connectors are in perfect alignment before pressing the boards together.

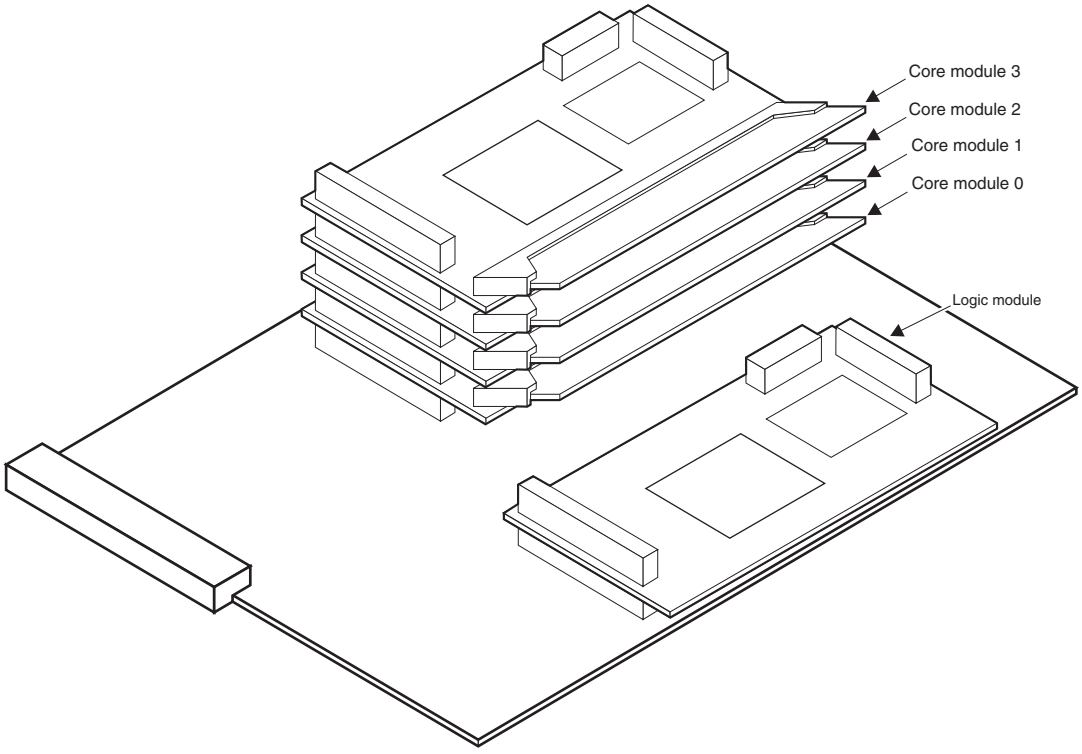


Figure 2-3 Assembled Integrator system

Note

To ensure correct operation, use the HDRA/HDRB connector position for core modules, and the EXPA/EXPB position for logic modules.

The Integrator/AP can have up to four core modules or four logic modules mounted onto it, although the combined total is limited to five. The module IDs are determined by their position in the stack (there are no links to set). For example, the module installed first is module 0 and is always closest to the motherboard.

The relative stack position of the core module defines, for example:

- which of interrupt request signals the core receives from the motherboard (see *Module-assigned signals* on page 4-13)
- the public address range of the core module (see *Module ID selection* on page 4-15).

2.1.4 Setting the mode switch

Select the PLD image using the mode switch (S1) or the two signals **CFGSEL[1:0]** supplied by the motherboard. The default setting of these signals ensure that the CM image is selected automatically when the core module is used standalone.

The mode switch, shown in Figure 2-4, is a four-pole DIP switch that is used at power ON and hard reset to control the selection of the PLD images stored in flash.

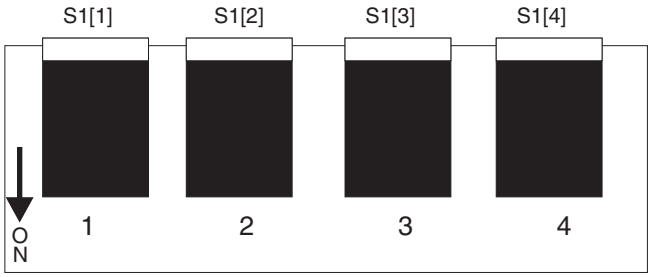


Figure 2-4 Mode switch

Use the switches as follows:

- If you are using the supplied images stored in flash, and are using the core module by itself or with an Integrator motherboard, set all switches to OFF. The core module uses **CFGSEL[1:0]** signals to select the PLD design that is appropriate for the system configuration.
- If you want to use the basic example image or you have loaded your own PLD designs into the flash, set S1[3] to ON and use S1[1] and S1[2] to select the correct image.

The settings for the four switches are shown in Table 2-1.

Table 2-1 Mode switch settings

S1[4]	S1[3]	S1[2]	S1[1]	Image	Function
OFF	OFF	x	x	-	Image selected by motherboard using the signals **CFGSEL[1:0]**.
OFF	ON	OFF	OFF	11	The core module can be used with an Integrator/CP.
OFF	ON	OFF	ON	10	The core module can be used standalone or with an Integrator/AP.
OFF	ON	ON	OFF	01	The core module can be used with an Integrator/IM-PD1.
OFF	ON	ON	ON	00	The core module can be used standalone.
ON	x	x	x	x	This setting selects the whole of the flash device. This setting is only used when reprogramming the flash, see *Loading new PLD configurations* on page 3-18. The contents are protected from accidental erasure by a write-protect bit in the CM_CTRL register. See *Core module control register, CM_CTRL* on page 6-10.

For more information about PLD image selection, see *PLD image selection* on page 4-8.

Caution

To ensure correct operation of your Integrator platform, ensure that you select the correct PLD image for your platform configuration.

2.1.5 Setting the general purpose/boot code switch

When the board is reset, the core boots from address `0x0`. The initialization code at this location performs several system configuration functions and then reads the switches S2[8] and S2[7]. These control the next stage of the boot sequence. The meaning of the switch settings are shown in Table 2-2.

Table 2-2 General purpose/boot code switch settings

S2[8]	S2[7]	Function
OFF	OFF	Remains in a loop in RAM code.
OFF	ON	Jump to the flash in EBI2.This can be used to store user programs.
ON	x	Jump to the flash in EBI1. The ARM boot monitor normally resides in EBI1

The ARM boot monitor is factory programmed into the flash in memory region EBI1. However, you can program your own boot code into the flash in this region and EBI2 and jump to your code using switches S2[8] and S2[7].

When the ARM boot monitor begins execution, the first routine to be run is the boot switcher. This detects whether or not the core module is mounted onto a motherboard, and then reads S2[2] and S[1] to determine where code execution jumps. See *Boot switcher* on page 2-12.

2.1.6 Connecting power

When using the core module standalone, connect a bench power supply with 3.3V and 5V outputs to the power connector, as illustrated in Figure 2-5.

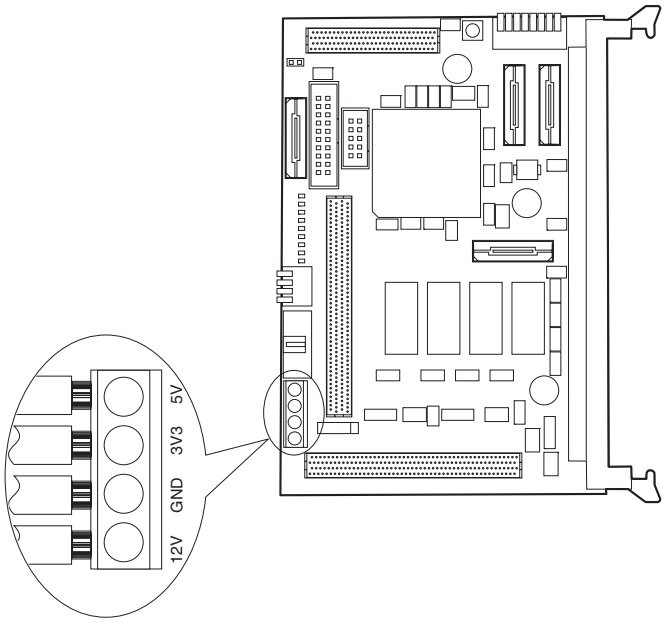


Figure 2-5 Power connector

Caution

Take care to connect the power correctly by observing that the GND connection is in between the 12V and 3.3V connections.

Connect the 12V supply if you are using the core module with an Integrator/IM-PD1 or other module that requires 12V on HDRA/HDRB.

Note

Do not connect power to the core module if it is mounted on an Integrator/AP motherboard or an Integrator/CP baseboard. Connect power to the AP or CP instead.

2.1.7 Connecting programming and debug equipment to the Integrator/CM-XA10

The core module provides interfaces that supports connection to Multi-ICE, trace, or ByteBlaster equipment. This section provides the following:

- *Connecting Multi-ICE*
- *Connecting trace* on page 2-11
- *Connecting ByteBlaster* on page 2-11.

Connecting Multi-ICE

You can use Multi-ICE equipment to download programs into memory or to download new PLD designs. Figure 2-6 shows how to connect Multi-ICE to a standalone core module.

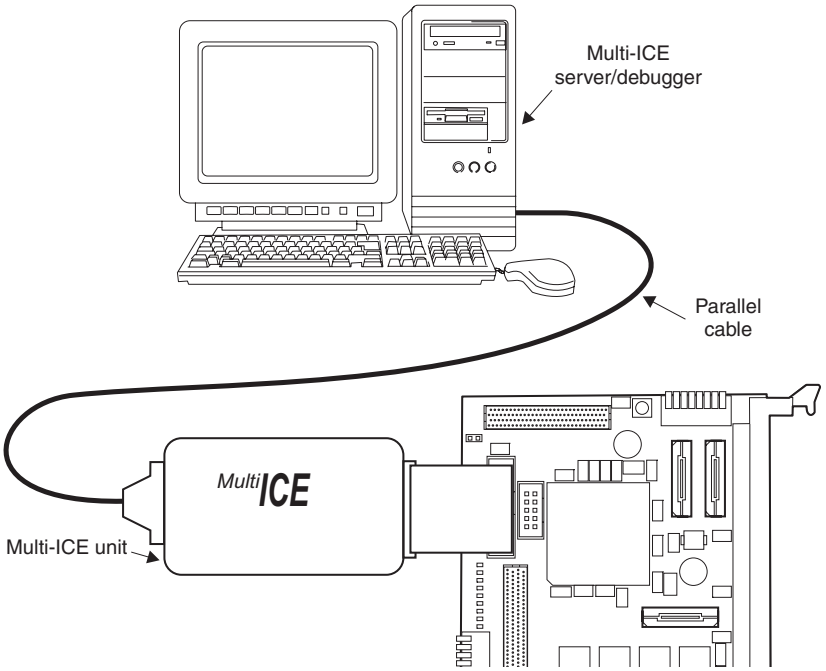


Figure 2-6 Multi-ICE connection to a core module

You can use Multi-ICE when a core module is attached to a motherboard. If more than one module is attached, then connect the Multi-ICE to the module at the top of the stack.

Connecting trace

Trace equipment can be used to access the EmbeddedICE logic incorporated into the core. To use trace, connect the trace port adapter board to the core module using the target buffer board supplied with the MultiTrace unit.

Connecting ByteBlaster

Caution

Do not connect Multi-ICE and ByteBlaster equipment to the core module at the same time. Connecting multiple devices to these connectors causes unreliable operation.

The ByteBlaster connector provides access to the scan chain and can be used with Altera programming tools. ByteBlaster can be used in debug mode or in config mode. In debug mode, it is used in conjunction with the ARM tool chain but you must use `altera-rdi.dll` in place of `multiice.dll`. In config mode, you must use the Altera programming device in place of Multi-ICE and the `progcards` utility.

2.2 Getting started with Boot monitor

The core module is shipped with a boot monitor preprogrammed into the flash. 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*.

The monitor enables you to:

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

2.2.1 Boot switcher

The boot switcher routine is embedded in the boot monitor that is factory programmed into EBI1. When the board is powered ON with S2[8] set to ON, the boot switcher is the first part of the boot monitor to be run. The boot switcher detects whether or not the core module is fitted to a motherboard, and then reads the settings of S2[2] and S2[1] to determine what action to take next. The options are shown in Table 2-3.

Table 2-3 Boot switcher actions

Mounted on AP or CP	S2[2]	S2[1]	Action
YES	OFF	x	Execution jumps to the flash on the AP or CP at `0x20000000`. The flash at this address on the AP and CP also contain versions of the boot monitor.
YES	ON	OFF	The boot switcher selects a user image in the core module flash in EBI2.
YES	ON	ON	The boot switcher selects the core module boot monitor. A welcome message is output on the serial port.
NO	x	OFF	The boot switcher selects a user image in the core module flash in EBI2.
NO	x	ON	The boot switcher selects the core module boot monitor. A welcome message is output on the serial port.

To run the boot monitor, S2[8] must be set to ON.

2.2.2 System startup for a standalone core module

Connect a standard null modem cable and the supplied ribbon cable adaptor between J1 on the core module and a PC running a terminal emulator. The serial port settings are:

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

When the core module platform is powered ON, a message similar to the following is displayed on the terminal:

```
ARM bootPROM [Version 1.4] Rebuilt on Aug 12 2002 at 10:43:00
Running on a Excalibur922T Evaluation Board
Board Revision V1.0, ARM922T Processor
Memory Size is 128MBytes, Flash Size is 16MBytes
Copyright (c) ARM Limited 1999 - 2002. 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 > ?
ARM bootPROM [Version 1.4] Rebuilt on Aug 12 2002 at 10:43:00
```

The list of generic commands is displayed. To display commands specific to the Integrator/CM922T-XA10 enter the following commands (commands can be entered in both uppercase and lowercase):

```
boot Monitor > x
[Integrator] boot Monitor > ?
ARM bootPROM [Version 1.4] Rebuilt on Feb  1 2002 at 11:44:43

G hex            Go to address
PEEK address     Display memory at address (use hex format)
POKE hex data    Poke data at address (use hex format for both values)
ESIB             Erase the boot monitor SIB
R i              Run image number i from flash
X                Exit board-specific command mode
X_command        Exit board-specific mode or execute single non board-specific
                 command
H or ?           Display help
```

2.2.3 System startup for other system configurations

When the core module is mounted onto an Integrator/AP or and Integrator/CP, you can use the boot monitor in the core module flash or use boot monitor programmed into the motherboard flash. This selection is controlled by the boot switcher.

The serial interface that you use depends on which boot monitor you select and the system configuration:

- Integrator/AP:
  - If you select the boot monitor on the core module, connect your terminal to J1 on the core module.
  - If you select the boot monitor on the AP, connect your terminal to J14 on the AP.
- Integrator/CP:
  - If you select the boot monitor on the core module, connect your terminal to J1 on the core module.
  - If you select the boot monitor on the CP, connect your terminal to J17-top on the CP.
- Integrator/IM-PD1. The terminal input/output can be routed to the serial interface on the IM-PD1. Connect your terminal to J12A on the IM-PD1.

For a detailed description of the ARM boot monitor, refer to the *ARM Firmware Suite Reference Guide*.

2.3 Building and downloading an executable image

The *ARM Firmware Suite* (AFS) (supplied separately) can be used to build and download executable images onto the Integrator/CM922T-XA10. The CD supplied with the core module or with associated modules provides a suite of register level software tests.

The directory *install_directory*\platforms\software\selftest\build\ADS1.1 contains files for these. Depending on the system configuration, the self tests require the use of loopbacks, interface cables and display.

For details of the self tests refer to the readme.txt file on the CDs supplied with the system components.

Chapter 3
Using Multi-ICE, ByteBlaster, and Trace

This chapter describes the debug facilities provided by the core module. It contains the following sections:

3.1 Configuring JTAG

The core module provides you with access to a JTAG scan path by using the Multi-ICE and ByteBlaster connectors. The CONFIG link is used to configure the scan path to operate in one of two modes:

- *Debug mode* on page 3-3
- *Config mode* on page 3-3.

The connectors and CONFIG link are shown in Figure 3-1.

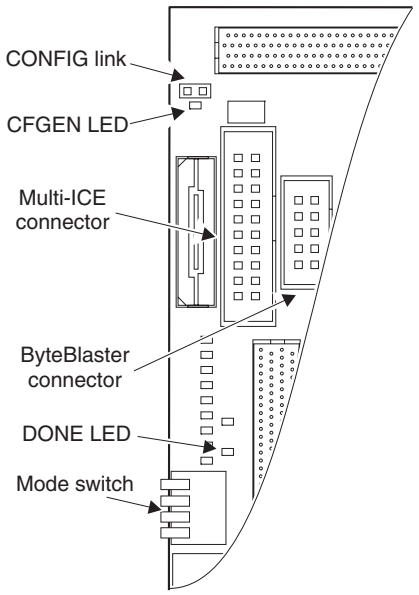


Figure 3-1 JTAG connectors, CONFIG link, and CFGEN LED

3.1.1 Debug mode

Debug mode is the normal mode of operation and is used for general system development and debug, including using trace (see *Embedded trace support* on page 3-11).

In this mode, the processor core and debuggable devices on other modules are accessible on the scan chain, as shown in Figure 3-2 on page 3-4.

This mode is selected by default (when the CONFIG link is left open, see Figure 3-1 on page 3-2).

Note

After auto configuring the CM922T-XA10 using Multi-ICE, it is recommended that the board power cycled before code is loaded. The effect of auto configuration can put the core module into an unknown state. Loading a manual configuration file overcomes this issue.

3.1.2 Config mode

In config mode, all FPGAs and PLDs in the system are added into the scan chain. In this mode, you can reprogram the programmable devices in the system in the field using Multi-ICE or ByteBlaster.

To select configuration mode, fit a jumper to the CONFIG link on the core module at the top of the stack (see Figure 3-1 on page 3-2). This has the effect of pulling the **nCFGEN** signal LOW which illuminates the CFGEN LED (orange) on each module in the stack. The scan path is routed on all modules in the stack. The LED provides an indication that the development system is in config mode.

Note

Configuration mode is guaranteed for a single core module attached to a motherboard but might be unreliable if more than one module is attached.

Before the image becomes active after configuration or code updates, you must:

1. Remove the CONFIG link.
2. Power the system OFF and then ON again.

3.2 JTAG signal routing

This section describes the routing of the JTAG scan chain on the core module and other modules when they are present. It discusses data and clock signal paths, and module stacking options in the following sections:

- *Data path*
- *Clock path* on page 3-5
- *Module stacking options* on page 3-6
- *JTAG signal descriptions* on page 3-8.

3.2.1 Data path

Figure 3-2 shows a simplified diagram of the data path. The diagram shows the routing with config mode enabled (**nCFGEN**=0) and with the core module standalone (**nMBDET**=1).

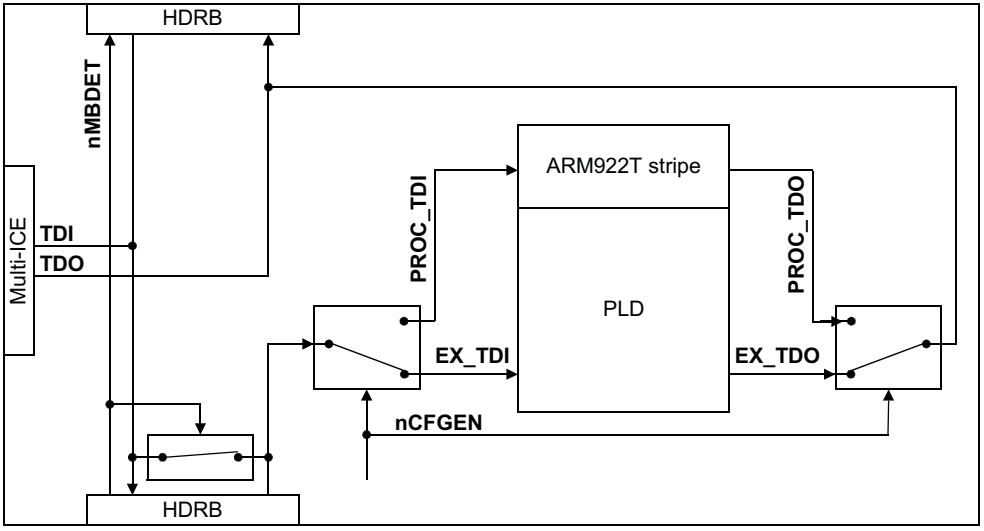


Figure 3-2 JTAG data path

The JTAG signals on the core module are switched by the **nCFGEN** signal to select the JTAG pins on either the core or the PLD.

The **nCFGEN** signal also drives the JSELECT pin. JSELECT equal to 0 causes the Altera device to internally route the JTAG data signals such that the data goes to the PLD TAP controller first and then to the core TAP controller as shown in Figure 3-3.

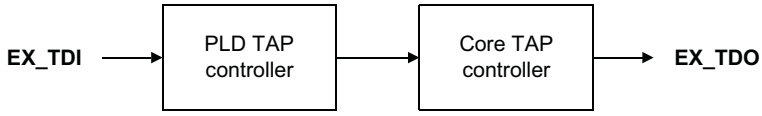


Figure 3-3 Excalibur internal JTAG data path routing (JSELECT=0)

When you use the core module as a standalone development system, the data path can be routed through the processor core only or through the PLD and the core.

If the core module is attached to an Integrator motherboard, the **TDI** signal from the module at the top of the stack is routed through the HDRB connectors of all other modules in the stack down to the motherboard. From there the path is routed back up the stack through components on the modules, before being returned to the Multi-ICE connector as **TDO**. The motherboard detect signal **nMBDET** controls a switching circuit on the core module and, therefore, the routing of **TDI**.

When the core module is in config mode, the PLD on the core module is added to the core in the scan chain. FPGAs on other boards in the system are also added into the scan chain, as described in *Configuring JTAG* on page 3-2.

3.2.2 Clock path

The clock path is routed in a similar way to the data path, although in the opposite direction. Figure 3-4 on page 3-6 shows a simplified diagram of the clock path.

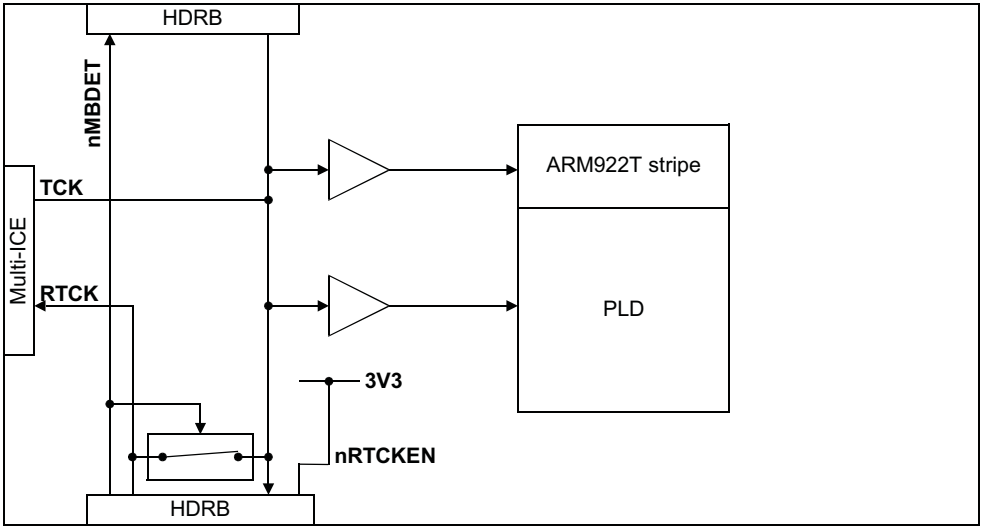


Figure 3-4 JTAG clock path

The routing of the **TCK/RTCK** signals through the stack is controlled by switches, in a similar way to the data path. In this case, however, the loopback is controlled by the signal **nRTCKEN** and a circuit on the motherboard. The core module also provides a similar **RTCK** routing scheme to the motherboard (omitted from Figure 3-4 for clarity) that is used when the core module is configured for operation without a motherboard. See *Module stacking options*.

Cores that do not use adaptive clocking route the **TCK** signal straight through to the next board down the stack. Core modules that do use adaptive clocking drive **RTCK** instead and assert **nRTCKEN**, you must ensure that the board at the bottom of the stack provides a return path for **RTCK**. All Integrator motherboards and baseboards do so.

3.2.3 Module stacking options

The Integrator system provides stacking options that can be selected by moving a surface-mount link (LK1). The link ensures that the **TDI/TDO** and **TCK/RTCK** signals are correctly routed through the stack for each configuration.

Table 3-1 shows the link position used to select the different stacking options.

Table 3-1 Link positions

Position	Function
A-C	Normal (default)
B-C	CM functions as motherboard

The stacking options are:

Normal With this option enabled, you can use the core module standalone or with an Integrator motherboard or baseboard and other modules.

Core module functions as motherboard

With this option enabled, you can use the core module at the bottom of a stack of one or more other modules.

Note

The supplied PLD images do not support stacking without a motherboard. One of the modules in the stack must provide the system controller functions normally provided by the Integrator/AP or CP.

3.2.4 JTAG signal descriptions

Table 3-2 provides a description of the JTAG signals. For pinout details of the Multi-ICE connector, see *Multi-ICE connector* on page A-13.

Note

In the description in Table 3-2, the term *JTAG equipment* refers to any hardware that can drive the JTAG signals to devices in the scan chain. Typically, Multi-ICE is used, although you can also use hardware from third-party suppliers to debug ARM processors.

Table 3-2 JTAG and associated signal description

Name	Description	Function
DBGRQ	Debug request (from JTAG equipment)	**DBGRQ** is a request for the processor core to enter the debug state. It is provided for compatibility with third-party JTAG equipment.
DBGACK	Debug acknowledge (to JTAG equipment)	**DBGACK** indicates to the debugger that the processor core has entered debug mode. It is provided for compatibility with third-party JTAG equipment.
DONE	FPGA configured	**DONE** is an open-collector signal that indicates when FPGA configuration is complete. Although this signal is not a JTAG signal, it does affect **nSRST**. The **DONE** signal is routed between all FPGAs in the system through the HDRB connectors. The master reset controller on the motherboard senses this signal and holds all the boards in reset (by driving **nSYSRST** LOW) until all FPGAs are configured.
nCFGEN	Configuration enable (from jumper on module at the top of the stack)	**nCFGEN** is an active LOW signal used to put the boards into configuration mode. In configuration mode all FPGAs and PLDs are connected to the scan chain so that they can be configured by the JTAG equipment.
nRTCKEN	Return **TCK** enable (from core module to motherboard)	**nRTCKEN** is an active LOW signal driven by any core module that requires **RTCK** to be routed back to the JTAG equipment. If **nRTCKEN** is HIGH, the motherboard drives **RTCK** LOW. If **nRTCKEN** is LOW, the motherboard drives the **RTCK** signal back up the stack to the JTAG equipment.

Table 3-2 JTAG and associated signal description (continued)

Name	Description	Function
nSRST	System reset (bidirectional)	**nSRST** is an active LOW open-collector signal that can be driven by the JTAG equipment to reset the target board. Some JTAG equipment senses this line to determine when a board has been reset by the user. The open collector **nSRST** reset signal can be driven LOW by the reset controller on the core module to cause the motherboard to reset the whole system by driving **nSYSRST** LOW. This is also used in configuration mode to control the initialization pin (**nINIT**) on the FPGAs. Though not a JTAG signal, **nSRST** is described because it can be controlled by JTAG equipment.
nTRST	Test reset (from JTAG equipment)	This active LOW open-collector signal is used to reset the JTAG port and the associated debug circuitry on the processor. It is asserted at power-up by each module, and can be driven by the JTAG equipment. This signal is also used in configuration mode to control the programming pin (**nPROG**) on FPGAs.
RTCK	Return TCK (to JTAG equipment)	**RTCK** is a mechanism for returning the sampled clock to the JTAG equipment, so that the clock is not advanced until the synchronizing device captured the data. In *adaptive clocking mode*, Multi-ICE is required to detect an edge on **RTCK** before changing **TCK**. In a multiple device JTAG chain, the **RTCK** output from a component connects to the **TCK** input of the down-stream device. The **RTCK** signal on the module connectors HDRB returns **TCK** to the JTAG equipment. If there are no synchronizing components in the scan chain then it is unnecessary to use the **RTCK** signal and it is connected to ground on the motherboard.
TCK	Test clock (from JTAG equipment)	**TCK** synchronizes all JTAG transactions. **TCK** connects to all JTAG components in the scan chain. Series termination resistors are used to reduce reflections and maintain good signal integrity. **TCK** flows down the stack of modules and connects to each JTAG component. However, if there is a device in the scan chain that synchronizes **TCK** to some other clock, then all down-stream devices are connected to the **RTCK** signal on that component (see **RTCK**).

Table 3-2 JTAG and associated signal description (continued)

Name	Description	Function
TDI	Test data in (from JTAG equipment)	**TDI** is the test data in signal that is routed down the stack of modules to the motherboard and then back up the stack, labeled **TDO**, where it connects to each component in the scan chain.
TDO	Test data out (to JTAG equipment)	**TDO** is the return path of the data input signal **TDI**. The module connectors HDRB have two pins labeled **TDI** and **TDO**. **TDI** refers to data flowing down the stack and **TDO** to data flowing up the stack. The JTAG components are connected in the return path so that the length of track driven by the last component in the chain is kept as short as possible.
TMS	Test mode select (from JTAG equipment)	**TMS** controls transitions in the tap controller state machine. **TMS** connects to all JTAG components in the scan chain as the signal flows down the module stack.

3.3 Embedded trace support

The ARM922T processor within the stripe incorporates an *ARM9 Embedded Trace Macrocell* (ETM9). This enables you to carry out real-time debugging by connecting external trace equipment to the core module. To trace program flow, the ETM broadcasts branch addresses, data accesses, and status information through the trace port. Later in the debug process, the complete instruction flow can be reconstructed by the ARM *Trace Debug Tools* (TDT).

3.3.1 About using trace

Figure 3-5 illustrates a trace debugging setup with the core module.

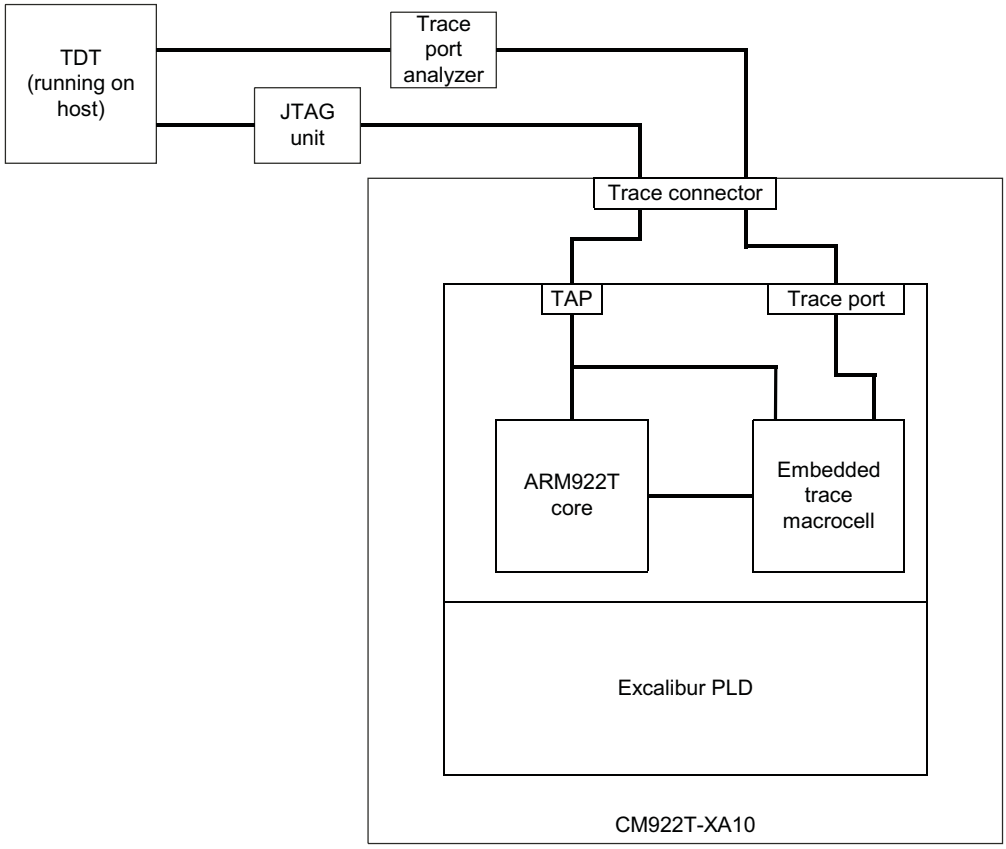


Figure 3-5 Trace connection

The components in the trace debug setup shown in Figure 3-5 on page 3-11 are as follows:

Embedded trace macrocell

The ETM monitors the ARM core buses and outputs compressed information through the trace port to a *Trace Port Analyzer* (TPA). The on-chip ETM contains trigger and filter logic to control what is traced. The ETM implemented in the Excalibur provides signals (**DEBUG_EXTIN[3:0]** and **DEBUG_EXTOUT[3:0]**) that provide you with access to logic within the PLD. However, the supplied PLD images do not make use of these signals. See the *Excalibur ARM-Based Embedded Processor PLD Hardware Reference Manual*.

Trace port analyzer

The TPA is an external device that stores information from the trace port.

JTAG unit This is a protocol converter that converts debug commands from the debugger into JTAG messages for the ETM. The JTAG unit might be a separate device or might be incorporated within the TPA.

Trace debug tools

The *Trace Debug Tools* (TDT) is an optional component of the *ARM Developer Suite* (ADS) that runs on a host system. It is used to set up the filter logic, retrieve data from the analyzer, and reconstruct an historical view of processor activity. For further information, see the *ADS Trace Debug Tools User Guide*.

3.3.2 Debug communications interrupts

The processor core incorporates EmbeddedICE logic which contains a communications channel used for passing information between the core and the JTAG equipment. The debug communications channel is implemented as coprocessor 14.

The processor accesses the debug communications channel registers using `MCR` and `MRC` instructions. The JTAG equipment reads and writes the register using the scan chain. For a description of the debug communications channel, see the *ARM922T (Rev 0) Technical Reference Manual*.

You can use interrupts to signal when data has been written into one side of the register and is available for reading from the other side. These interrupts are supported by the interrupt controller within the core module FPGA, and can be enabled and cleared by accessing the interrupt registers (see *Comms interrupts* on page 6-23).

3.3.3 Trace interface description

The logic analyzer connection is a high-density AMP Mictor connector. The pinout for this connector is provided in *Trace connector pinout* on page A-14.

3.4 Using ByteBlaster

The core module provides a connector and support for using the Altera ByteBlasterMV parallel port download cable and QuartusII development software. The ByteBlaster connector provides access to the JTAG scan chain through the Integrator system.

Caution

The ByteBlaster and Multi-ICE connectors cannot both be used at the same time.

ByteBlaster can be used in debug mode or in config mode:

- in debug mode, it is used in conjunction with the ARM tool chain but you must use `altera-rdi.dll` in place of `multiice.dll`
- in config mode, you must use the Altera programming tools in place of Multi-ICE and the `progcards` utility.

For more information about using ByteBlaster, see the Altera documentation.

3.5 Altera tool flow

To prepare FPGA configuration files, you must carry out the following steps:

1. *Synthesis* on page 3-16
2. *Place and route* on page 3-16
3. *Compilation* on page 3-17.

Figure 3-6 shows the basic tool flow process.

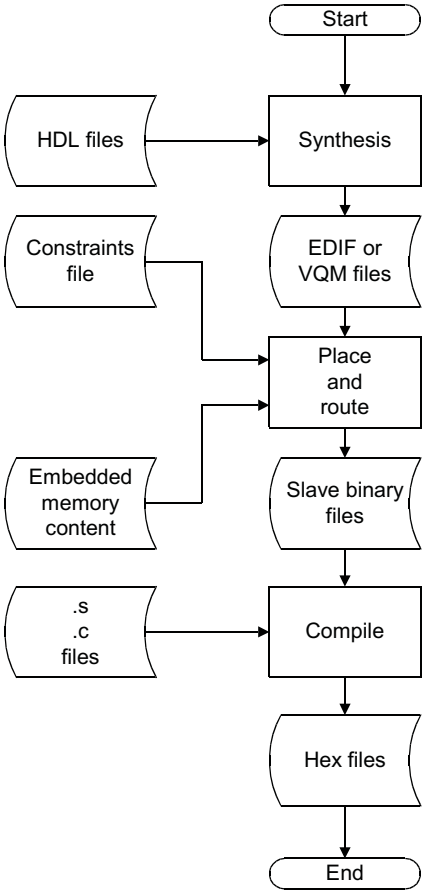


Figure 3-6 Basic tool flow

3.5.1 Synthesis

The synthesis stage of the tool flow takes the HDL files (either VHDL, Verilog, or a combination) and compiles them into a netlist targeted at a particular technology. For this core module, the target technology is Altera Excalibur. There are several synthesis tools available for both Windows and UNIX platforms, that provide support for a variety of programmable logic vendors. Synthesis information is supplied either through a GUI front end or command-line script. The information typically includes:

- a list of HDL files
- the target technology
- required optimization, such as area or delay
- timing and frequency requirements.

Refer to the documentation for your particular software tool for further information.

Common netlist file formats produced by synthesis are VQM and EDIF files (for example, `filename.vqm`). The netlist file is used by the next stage of the tool flow, which is place and route.

3.5.2 Place and route

Place and route for this module is performed using the Altera Quartus place and route tool. This produces a `.hex` file that is used to program the flash.

The Altera targeted `.vqm` is aimed at a particular device, and takes into account the device size, package type, and speed grade. However, to ensure that Quartus generates a file that operates correctly with the core module, use the following settings:

1. From the **Processing** menu, select the **Compiler settings** option.
2. Select the **Chips & Devices** tab, and then click the **Device & Pin Options** button.The **Device & Pin Options** dialog is displayed. Do the following (in any order).
   - Select the **General** tab and check the **Enable INIT_DONE output** box.
   - Select the **Configuration** tab, set **Configuration scheme** to **Boot from flash**.
   - If you are using Multi-ICE to program in flash programming mode, select **Programming** files tab and check the **Slave Binary Image File** box.
   - Select the **Unused pins** tab and set all unused pins as inputs, tristated.

   Use the default for all other settings.

Signal names from the top-level HDL are mapped onto actual device pins by a user compiler setting file `.csf`. You can also specify the timing requirements within this file.

3.5.3 Compilation

Compilation of the hex file for PLD configuration is carried out using the Altera Quartus tools. This step takes as input the slave binary files from the place and route stage and appends it to device initialization code supplied by Altera. External `.s` or `.c` files can also be linked in to define what the ARM core does after the PLD has been configured. Typically, this is a branch to one of the user code areas in flash memory.

In the supplied images for the CM922T-XA10, the slave binary file and a routine to check the boot code switches and select the appropriate jump are appended to the device initialization code, see *Setting the general purpose/boot code switch* on page 2-8.

It is recommended that you use the ARM development tools to generate the application code that runs on the core module.

3.6 Loading new PLD configurations

You can reprogram the PLD configuration data by writing new PLD images into the image flash memory using Multi-ICE or ByteBlaster.

To enable flash program mode, the CONFIG link must be fitted or S1[4] must be ON:

- Setting switch S1[4] enables the image flash to be programmed from Multi-ICE, without reconfiguring Multi-ICE.

  Using S1[4] does not put the XA10 into the scan chain and therefor the FPGA cannot be directly reprogrammed. Reprogramming of the FPGA can only be done indirectly by reprogramming the image flash.

- Fitting the CONFIG link (J8) puts the system into configure mode. The CFGLED is lit as an indication that configure mode is selected.

  Multi-ICE can be used in configure mode to program the image flash and the FPGA. Using configure mode, however, requires reconfiguring Multi-ICE.

  Note

  Programming by setting the CONFIG link is provided for compatibility and where multiple boards are to be programmed. If you are programming only the CM922T-XA10, use switch S1[4].

3.6.1 Downloading new PLD configurations into the image flash

The ARM core uses data in flash to configure the PLD during power-up (if the CONFIG link is not fitted).

The `progcards` utility is used to program the flash.

Note

Progcards version 2.40 or later is required for use with the CM922T-XA10.

To load a new configuration into the PLD:

1. Produce a `<filename>.hex` file.
2. Produce a `<filename>.brd` for your design. This is a configuration file for `progcards.exe`.
3. Configure the Multi-ICE server using an auto config.
4. Put the system in flash program mode by fitting the CONFIG link or setting S1[4] to ON.

Note

If you set the CONFIG link, you must reconfigure Multi-ICE before continuing.

5. Run the `progcards` utility. All `.brd` files present in the current directory that match the TAP configuration are offered as options.
6. Remove the CONFIG link or set S1[4] to OFF.
7. Power the system OFF.
8. Set switch S1[3:1] to select the appropriate block (see *PLD image selection* on page 4-8).
9. Power the system ON again.

3.6.2 Reconfiguring the PLD directly with JTAG

The PLD can be reconfigured directly using the DOWNLOAD connector (J9) with the Altera ByteBlaster cable and the Altera Quartus tools. Refer to Altera documentation for the use and operation of this tool.

Caution

A PLD configuration downloaded using this method is lost when the core module is power cycled or reset and is replaced by an image from the flash memory.

Chapter 4
Integrator/CM922T-XA10 System Architecture

This chapter describes the basic hardware architecture of the core module. It contains the following sections:

4.1 About the hardware architecture

Figure 4-1 shows a block diagram of the core module.

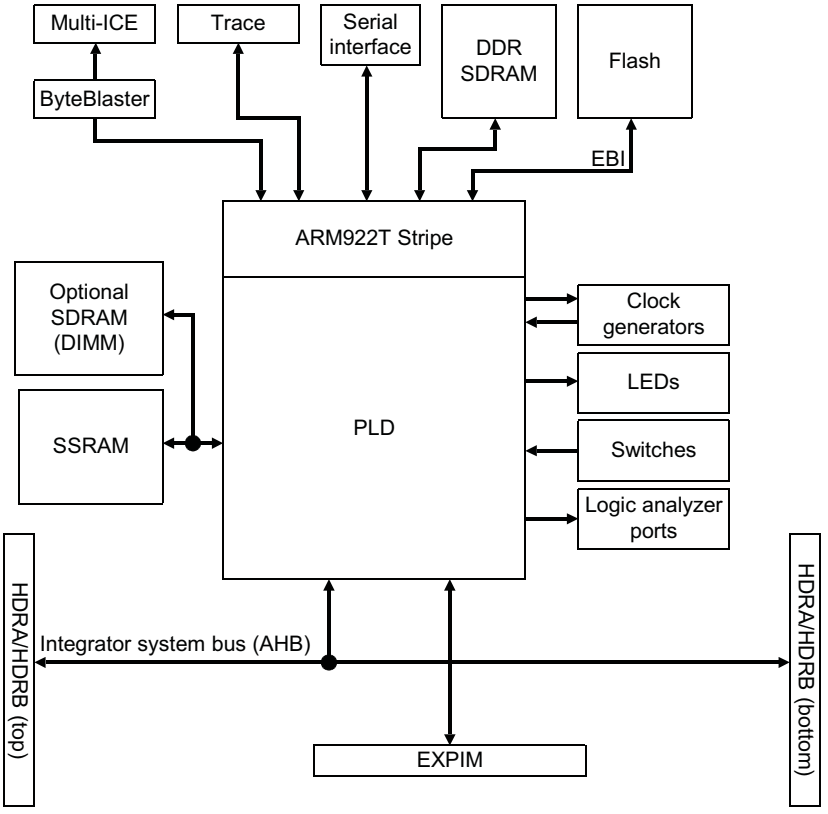


Figure 4-1 Integrator/CM922T-XA10 board architecture

The block diagram shows the main hardware components and buses on the core module and how they connect to either the ARM922T stripe or to the user programmable PLD.

The components connected to the stripe include the on-board SDRAM and flash, a serial interface, and the debug interfaces.

The PLD connected components include memory, clock generators, LEDS, switches, and connections to the Integrator system buses. Control logic for these must be provided in the PLD design.

The core module is supplied with four PLD designs stored in the config flash. These are:

Basic Example Image

This design configures the PLD with a basic example image. The image contains registers to set the on-board clocks and to read the user switches. It is designed to allow you to develop and add your own IP for use with the ARM core.

For a description of this image, see Chapter 5 *Basic Example Image for Simple Standalone Operation*.

CM Image This design configures the PLD with a design that functions as an Integrator core module with an AHB interface. The core module can be used its own (standalone) or with an Integrator/AP motherboard, and is capable of running boot monitor or GDBstub over a serial line.

For a description of this image, see Chapter 6 *CM Image for Operation Standalone or with an Integrator/AP*.

CP Image

This design configures the core module when mounted onto an Integrator/CP baseboard. The PLD is programmed with the appropriate image containing PrimeCell peripherals and interface controls for the interfaces provided by the baseboard.

For a description of this image, see Chapter 7 *CP Image for Operation with an Integrator/CP Baseboard*.

IM-PD1 Image

This design configures the core module to operate with an Integrator/IM-PD1 mounted onto it. The design contains the appropriate PrimeCell peripherals and interface controls to drive the interfaces provided by the IM-PD1.

For a description of this image, see Chapter 8 *IM-PD1 Image for Operation with an Integrator/IM-PD1*

Note

All four images are programmed into flash during board manufacture. However, the source code for the basic example and the CM images only are supplied with the IntegratorCM922T-XA10. Source code for the CP and IM-PD1 images are supplied with the Integrator/CP and Integrator/IM-PD1 products respectively.

4.2 About the Altera Excalibur EPXA10

Figure 4-2 shows the internal architecture of the Altera Excalibur embedded processor PLD. A brief description of the device is given in the following sections:

- *Embedded core*
- *Stripe memory and peripherals* on page 4-5
- *Excalibur EPXA10 memory map* on page 4-6
- *PLD image selection* on page 4-8
- *PLD signal assignment* on page 4-10.

For detailed information about the device, refer to the *Excalibur ARM-Based Embedded Processor PLD Hardware Reference Manual* from Altera.

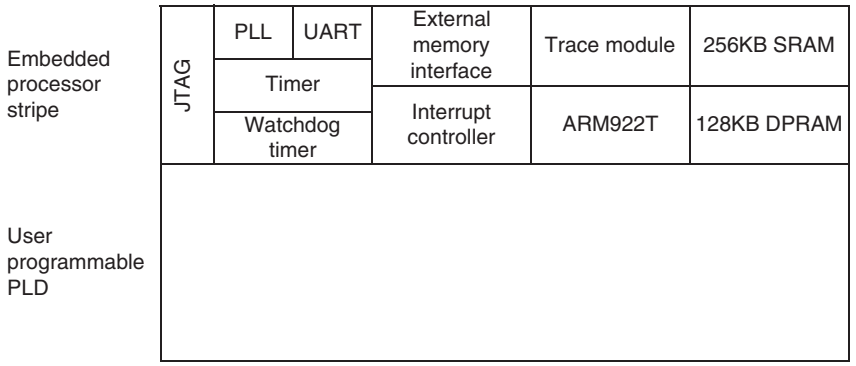


Figure 4-2 Altera Excalibur embedded processor PLD

4.2.1 Embedded core

The embedded core is the ARM922T. This is a member of the ARM9 Thumb family of Harvard architecture cores. It uses a five-stage pipeline and supports 32-bit ARM and 16-bit Thumb instruction sets, that you can use to balance between high code density and performance. The core contains:

- data and instruction MMUs
- 8KB instruction and data caches
- write-back page address TAG RAM
- write data buffer
- embedded Trace module
- AMBA bus interface to the AHB.

For more information about this core, see *ARM922T (Rev 0) Technical Reference Manual*.

4.2.2 Stripe memory and peripherals

The stripe internal components are shown in Figure 4-3 and include:

- memory:
  - 256KB SRAM
  - 128KB Dual port RAM.
- peripherals:
  - Interrupt controller
  - UART
  - timer
  - watchdog timer.
- SDRAM controller for off-chip SDRAM
- *Expansion Bus Interface* (EBI)
- reset controller
- *Phase Locked Loop* (PLL)
- AHB master and slave bridges to the PLD
- configuration logic.

These devices and their control registers are described in the *Excalibur ARM-Based Embedded Processor PLD Hardware Reference Manual*.

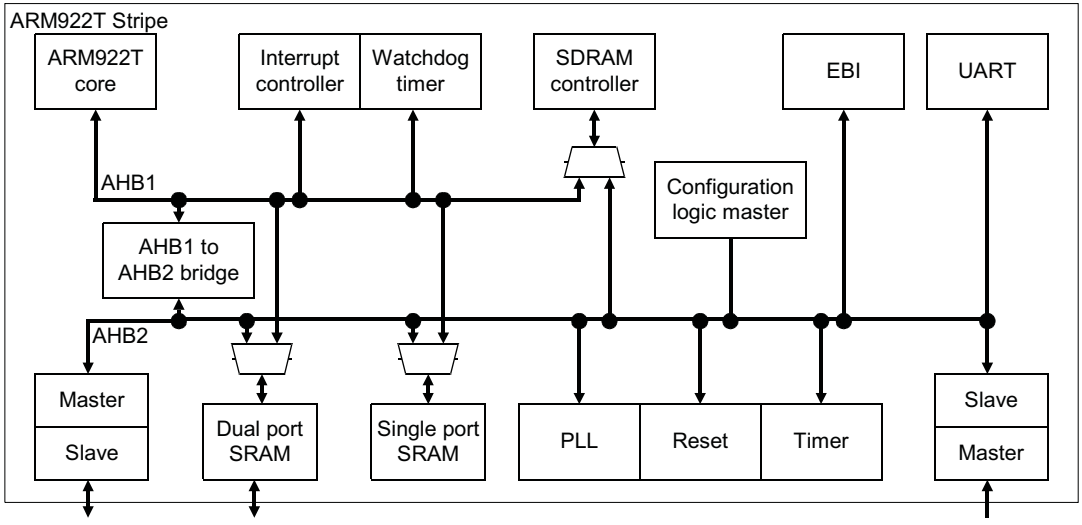


Figure 4-3 Embedded processor stripe internal architecture

These functional blocks within the stripe enable the core, independently of the PLD configuration, to carry out the following functions:

- access external boot memory
- boot and run a program
- program the PLD
- run interactive debug
- detect errors and restart, reboot, or reprogram the system as required
- communicate with a terminal
- run a real-time operating system.

4.2.3 Excalibur EPXA10 memory map

The memory map provided by the Excalibur chip contains 15 memory regions that must be configured for size and start address using the range definition registers provided by the stripe. For the Integrator, 14 regions are used and the size and start address parameters must be programmed as shown in Table 4-1.

Table 4-1 Excalibur memory map for the CM image

Excalibur defined		Integrator defined		
Memory region	**Size limits**	**Required size**	**Base address**	**End address**
EBI0 (only at boot)	16KB - 32MB	2MB	0x00000000	0x001FFFFF
SDRAM0	16KB - 256MB	64MB	0x00000000	0x03FFFFFF
SDRAM1	16KB - 256MB	64MB	0x04000000	0x07FFFFFF
Embedded SRAM0	128KB	128KB	0x08000000	0x0801FFFF
Embedded SRAM1	128KB	128KB	0x08020000	0x0803FFFF
DPSRAM0	64KB	64KB	0x08100000	0x0810FFFF
DPSRAM1	64KB	64KB	0x08110000	0x0811FFFF
Stripe registers	16KB	16KB	0x0B000000	0x0B003FFF
EBI1	16KB - 32MB	8MB	0x0F000000	0x0F7FFFFF
EBI2	16KB - 32MB	8MB	0x0F800000	0x0FFFFFFF
EBI3	The chip select EBI3 from the Excalibur is unconnected. This region must always be disabled.			

Table 4-1 Excalibur memory map for the CM image

Excalibur defined		Integrator defined		
Memory region	**Size limits**	**Required size**	**Base address**	**End address**
PLD0	16KB - 2GB	256MB	0x10000000	0x1FFFFFFF
PLD1	16KB - 2GB	512MB	0x20000000	0x3FFFFFFF
PLD2	16KB - 2GB	1GB	0x40000000	0x7FFFFFFF
PLD3	16KB - 2GB	2GB	0x80000000	0xFFFFFFFF

The range definition registers are described in the *Excalibur ARM-Based Embedded Processor PLD Hardware Reference Manual*.

4.2.4 PLD image selection

When the core module is powered ON or reset, the ARM core loads a configuration image into the PLD from the flash memory at EBI0 (see *Flash memory* on page 4-17). This operation is controlled by some initialization code that resides in the bottom of each configuration image stored in flash.

The core module is supplied with four images already programmed into the flash. You can also download your own images into the flash by setting S1[4] to ON or by fitting the CONFIG link, see *Loading new PLD configurations* on page 3-18.

To select an image, set the mode switch S1 or use the signals **CFGSEL[1:0]** from a motherboard. These generate the address bits **CFG_EBI_A[22:21]** that are used to select the required configuration image. Figure 4-4 shows the PLD configuration architecture on the core module.

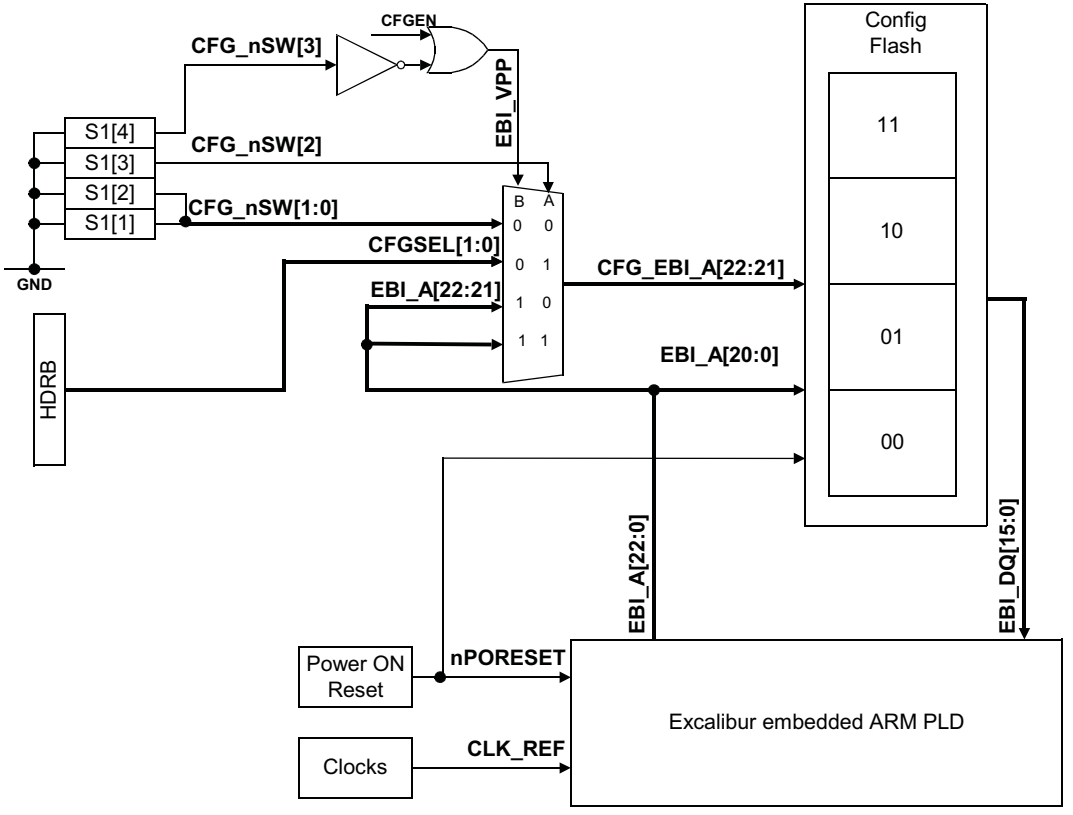


Figure 4-4 PLD configuration architecture

Note

The switch elements on S1 are engraved 1, 2, 3, and 4. These align with the signals **CFG_nSW0**, **CFG_nSW1**, **CFG_nSW2**, and **CFG_nSW3** respectively.

When to use the mode switch for image selection

Use the switches if you have written your own configuration image into the flash to ensure that the correct image is selected for your design. Table 4-2 shows the mode switch settings required to select the images. To use S1[1] and S1[2] for image selection, set S1[3] to ON.

Table 4-2 Image selection using the mode switch

S1[4]	S1[3]	S1[2]	S1[1]	Image ID	Meaning with the supplied images
OFF	ON	OFF	OFF	11	The core module can be used with an Integrator/CP baseboard.
		OFF	ON	10	The core module can be used standalone or with an Integrator/AP.
		ON	OFF	01	The core module can be used with an Integrator/IM-PD1.
		ON	ON	00	This selects the basic example image.

When to use CFGSEL[1:0] for image selection

The signals **CFGSEL[1:0]** are static signals used by different Integrator motherboards as a type identifier. They are used to select the image that enables the core module to be used with that motherboard. Use this option, if you mount the core module on an Integrator/AP or CP and are using the PLD images in the locations supplied.

The meaning of the **CFGSEL[1:0]** values settings is given in Table 4-3. To use **CFGSEL[1:0]** signals for image selection, set S1[3] to OFF.

Table 4-3 Image selection using CFGSEL[1:0]

S1[4]	S1[3]	S1[2]	S1[1]	CFGSEL[1:0]	Meaning with supplied images
OFF	OFF	x	x	00	Image 00 is selected. This code is not generated by any of the currently available Integrator boards.
				01	Image 01 is selected. This code is not generated by any currently available Integrator boards.
				10	Image 10 is selected. This is the default. This code is generated by pullup and pulldown resistors on **CFGSEL[1:0]** on the core module or by an Integrator/AP motherboard. Use an image that enables the core module to be used standalone or with an Integrator/AP.
				11	Image 11 is selected. This code is generated by an Integrator/CP baseboard. Use an image that enables the core module to be used with an Integrator/CP baseboard.

4.2.5 PLD signal assignment

Many of the interface signals from the user PLD are assigned to specific hardware blocks that are included on the core module. This includes the switches, LEDs, clock generators, and system buses. The functional blocks instantiated into the PLD are described in the sections relevant to their function. For example, the clock signals are described in *Core module clocks* on page 4-23.

Appendix B *Excalibur PLD Pinout* provides a listing of the signal to pin connections.

4.3 Integrator system bus

The Integrator family of modules and platform boards share a large number of signal traces that together from the system bus. These are described in:

- *System bus signal routing*
- *Signal assignments for connectors HDRA and HDRB* on page 4-12
- *Module-assigned signals* on page 4-13
- *Module ID selection* on page 4-15.

4.3.1 System bus signal routing

The system bus signal connections shared between the Integrator modules are shown in Figure 4-5. This diagram shows an example system comprising a core module mounted onto an Integrator/AP, and an IM-AD1 mounted on top of the core module.

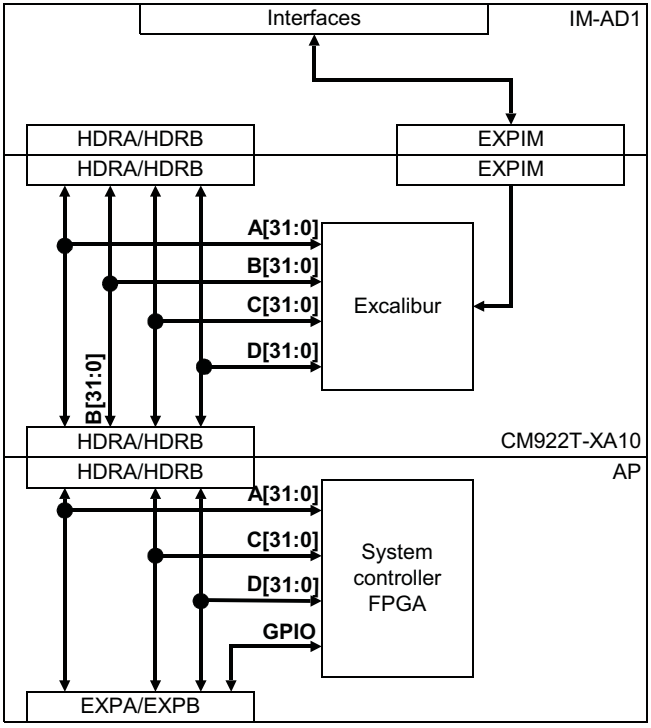


Figure 4-5 Integrator system bus routing

The diagram shows the bus routing between the Excalibur PLD on the core module and system controller FPGA on the Integrator/AP motherboard. You normally mount core modules on the motherboard at the HDRA/HDRB connector position and logic modules, if required, at the EXPA/EXPB connector position.

The B bus is routed between the HDRA connectors on the core module, but is not connected on the AP. You can use this bus to carry interface signals between the Excalibur PLD on the core module and, for example, an FPGA on a logic module.

The interface signal traces from the Excalibur PLD connect with the EXPIM connector on the upper side of the core module. These can be used to carry interface signals up to an interface module, such as the Integrator/IM-AD1. This requires that you instantiate any interface controllers into the PLD.

Caution

To avoid damage to the core module and other boards, take care that there are no signal clashes when connecting various modules together.

Consult the signal descriptions for each of your modules and for your PLD designs.

4.3.2 Signal assignments for connectors HDRA and HDRB

The system bus is connected between the baseboard, core module card, and additional logic modules using the HDRA and HDRB connectors.

The signals carried by these connectors are described in Table 4-4 and the pinouts for the core module are shown in Appendix A *Signal Descriptions*.

Table 4-4 System bus connector signal assignments for AHB

Connector	Pins	Function
HDRA	**A[31:0]**	This is the AHB address bus.
	B[31:0]	These are used to carry the display interface signals in designs that implement a display controller. They are unused by the Integrator/AP.
	C[31:0]	These carry AHB bus control signals and various interface signals.
	D[31:0]	This is the AHB data bus. This uses a bidirectional bus **HDATA** rather than the separate unidirectional buses **HWDATA** and **HRDATA** described in the AMBA specification.

Table 4-4 System bus connector signal assignments for AHB

Connector	Pins	Function
HDRB	**E[31:0]**	This bus carries the system bus control signals. These are mainly the signals associated with the Integrator system, such as interrupt requests, clocks, and arbitration signals. There are no arbitration signal connections on the Integrator/CP baseboard. The clock signals, interrupt signals, module ID, and module presence signals are routed so that they rotate up through the stack (see *Module-assigned signals*). These pins correspond with the pins labeled **H[31:0]** on logic modules.
	GPIO/F[31:0]	This set of pins is used to implement a variety of interface connections. The Integrator/AP motherboard uses these to implement GPIO signals.
	G[16:0]	This set of pins is used to implement the Multi-ICE/JTAG signals and FPGA configuration control signals. These pins correspond with the **J[16:0]** pins on logic modules.

4.3.3 Module-assigned signals

Some of the signals on the HDRA connectors are assigned to specific modules and are routed between the plug and socket on all core modules so that they are rotated in groups of four up through the stack. This enables a motherboard to identify each module based on its position in the stack, to modify how the address decoder and bus arbiter behave, and to correctly route interrupts.

The signals that are rotated are:

nIRQ[3:0] These are the interrupt request signals from devices on the baseboard and the logic modules.

nFIQ[3:0] These are the fast interrupt request signals from the baseboard.

nPPRES[3:0]

These are the module presence signals from the core modules. They indicate to the address decoder that a module has been added to the system and is responsible for generating bus responses for its own address space. On the logic modules these signals are called **nEPRES[3:0]**. On the Integrator/CP, up to four modules can be stacked, but only one core module is permitted.

SYSCLK[3:0]

These are the system bus clock signals rotated up through the stack to ensure even distribution and signal loading.

ID[3:0] These signals rotate up through the stack to indicate the position of a board in the stack of modules. See *Module ID selection* on page 4-15.

SREQ[3:0] These bus request signals from core multiple core modules. They are used on a multi-master Integrator/AP system by the bus arbiter. The Integrator/CP baseboard supports only one bus master and **SREQ[3:1]** are reserved.

An example of how this signal rotation scheme is implemented by an Integrator/AP is shown in Figure 4-6.

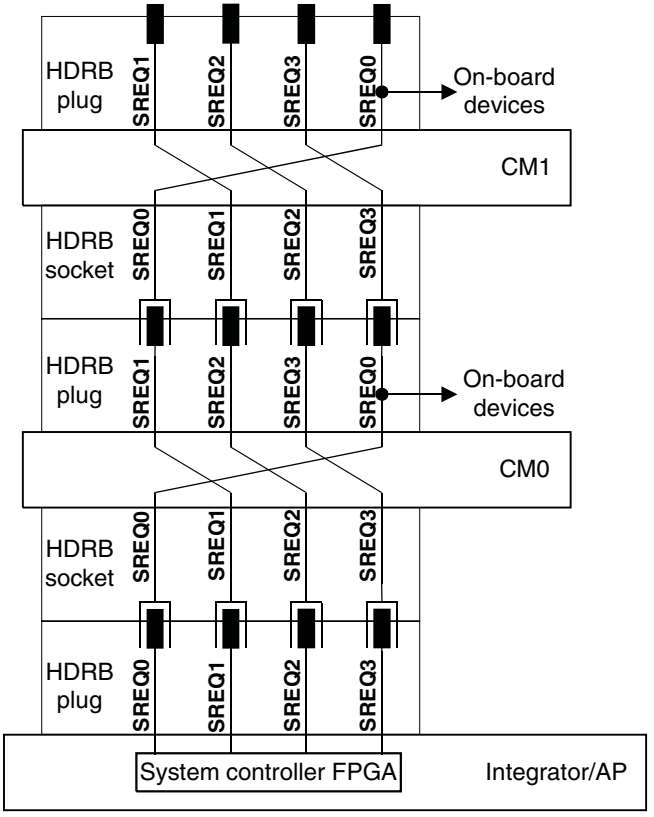


Figure 4-6 Signal rotation scheme

This example shows how a group of four signals **SREQ[3:0]** are routed through two core modules stacked on an Integrator/AP. Each module requests the bus using only its own version of **SREQ0** (that is, the signal name used on the schematics for that

module), and passes the unused signals down from the modules above it in the stack to the motherboard below. The signals are routed between the plug and socket on each module so that, as in this example:

- 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.

4.3.4 Module ID selection

The position of modules in the HDRA/HDRB stack is used to determine their ID and, from this, their slave addresses (see the user guide for you motherboard), the interrupts they issue or respond to.

Note

The core module cannot be damaged by connecting it onto the EXPA/EXPB position on the Integrator/AP motherboard, but fitting it in this position prevents correct operation.

The signals **nPPRES[3:0]** (core module present) are used to signal the presence of modules to the central decoder. The signals **ID[3:0]** indicate to the module its position in the stack and the address range that its own decoder must respond to. On the motherboard the **ID[3:0]** signals are tied to give the bit pattern 1110 and rotate as they pass up the stack, as described in *System bus signal routing* on page 4-11.

The slave address of a core module is determined in hardware, although a module can determine its own position by reading the state of **ID[3:0]** from the CM_STAT register (see *Core module status register, CM_STAT* on page 6-13). Table 4-5 shows alias addresses for a core module fitted to a the motherboard on the HDRA/HDRB stack.

Table 4-5 Core module address decode

ID[3:0]	Module ID	Address range	Size
1101	3 (top)	`0xB0000000` to `0xBFFFFFFF`	256MB
1011	2	`0xA0000000` to `0xAFFFFFFF`	256MB
0111	1	`0x90000000` to `0x9FFFFFFF`	256MB
1110	0 (bottom)	`0x80000000` to `0x8FFFFFFF`	256MB

Note

The Integrator/CM922T-XA10 cannot access its own slave address.

4.4 Expansion module interface EXPIM

The EXPIM can be used to carry signals for interfaces implemented into the PLD on the core module and interfaces on an interface module, such as an Integrator/IM-PD1.

4.5 Core module memory

The core module has up to four types of memory fitted to it:

- *Flash memory*
- *Double Data Rate SDRAM* on page 4-20
- *SSRAM* on page 4-21
- *Optional SDRAM DIMM* on page 4-21.

4.5.1 Flash memory

The core module provides three 8MB 16-bit wide flash chips. These are connected to the stripe EBI data and address buses and occupy the area of the system memory map selected by EBI0, EBI1, and EBI2.

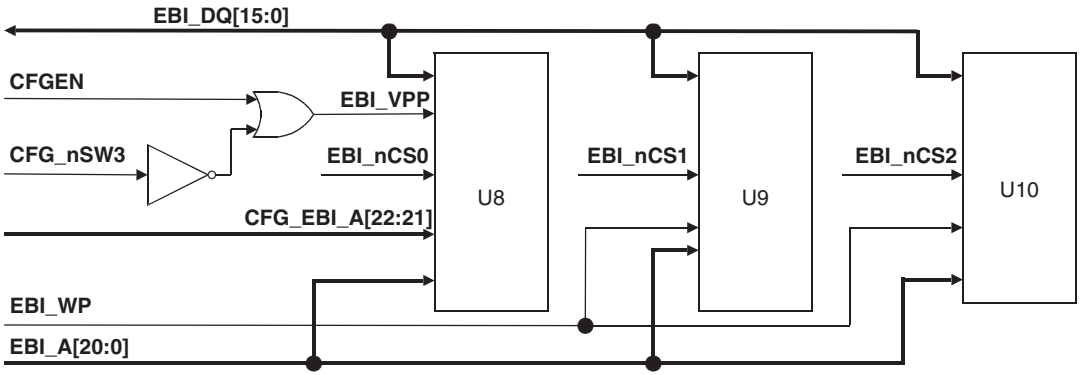


Figure 4-7 Flash memory architecture

The flash memory is organized into the following areas:

- PLD configurations and initialization code (U8)
- ARM boot monitor in (U9)
- User data in (U10).

PLD configurations

The PLD configuration is loaded into the PLD at power ON from the flash chip U8. The PLD configuration flash contains four images that are selected by the signals **CFGSEL[1:0]** from the motherboard or by using the image select switch. See *PLD image selection* on page 4-8.

The flash chip U8 is selected by the EBI0 chip select signal **EBI_nCS0**. During power ON, the first 32KB of this chip select region is mapped to begin address `0x0`. The flash contains four PLD images, and image select switches or **CFGSEL[1:0]** signals are used to control the upper 2 address bits **CFG_EBI_A[22:21]**. This allows you to select which of the four images to appears at `0x0`.

During normal operation the EBI0 area cannot be accessed. To reprogram this flash, you must put the core module in config mode or set S1[4] to ON. See *Loading new PLD configurations* on page 3-18.

Figure 4-8 Flash memory EBI0 usage

ARM boot monitor

The boot monitor is stored in flash in EBI1 (`0x0F000000-0x0F7FFFFF`). It is copied into the SRAM at startup. The boot configuration then jumps to the SRAM when boot monitor is run. To run the boot monitor you must set the switch S2[8] to ON, see Table 4-6 on page 4-19.

User data

The user area of the core module flash is in EBI2 (0x0F800000-0x0FFFFFFF). You can use this area to store your own data. The user area is write-protected by default. Before you can write to the flash you must drive **EBI_WP** HIGH. This signal must be controlled by logic in the PLD image.

The supplied PLD images provide a register location that allows you to control the **EBI_WP** signal. For more information about controlling this signal:

- for the BE image, see *Basic example image registers* on page 5-7
- for the CM, CP, and IM-PD1 images, see *Core module control register, CM_CTRL* on page 6-10.

You can run the code in EBI1 or EBI2 at startup by setting switch S2[8:7] as shown in Table 4-6.

Table 4-6 General purpose/startup code switch settings

S2[8]	S2[7]	Function
OFF	OFF	Remains in a loop in EBI0.
OFF	ON	Jump to the flash in EBI2.This can be used to store user programs.
ON	x	Jump to the flash in EBI1. The ARM boot monitor normally resides in EBI1

4.5.2 Double Data Rate SDRAM

Four 32MB 16-bit *Double Data Rate* (DDR) SDRAM chips are fitted permanently on the core module. The data, address, refresh, and control signals for these are supplied by the SDRAM controller incorporated into the stripe. They are organized into two 32-bit banks each of 64MB. The base address of each memory bank is determined by configuration data loaded when the core module is powered ON. The supplied PLD images always map the DDR SRAM to `0x0`.

Figure 4-9 shows a simplified block diagram of the DDR SDRAM.

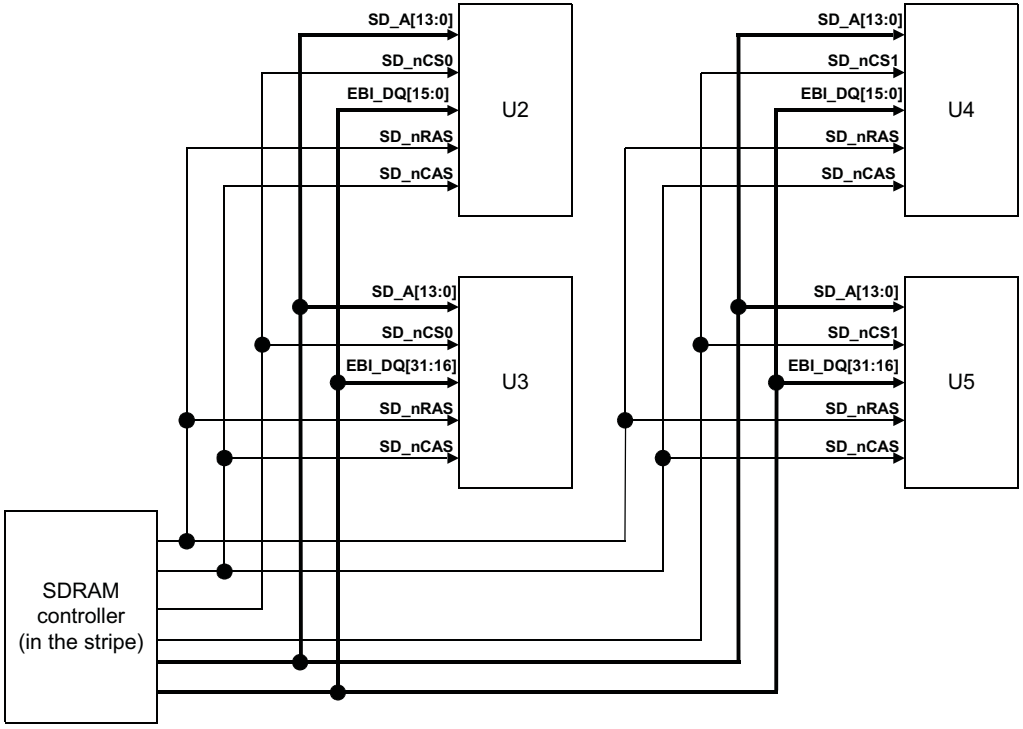


Figure 4-9 DDR SDRAM block diagram

4.5.3 SSRAM

A 2MB 32-bit SSRAM chip is fitted permanently to the core module. This uses signals from the user programmable PLD for data, address, and control signals. Some of the signals used are shared with the optional SDRAM DIMM, which means that they cannot both be used at the same time, see *SSRAM and SDRAM DIMM interface signals* on page 4-22.

The CP and IM-PD1 images implement an SSRAM controller. The SSRAM is used by the CP and IM-PD1 images as a frame buffer for the *Color LCD Controller* (CLDC) implemented in these images. However, the CM PLD image does not provide an SSRAM controller.

4.5.4 Optional SDRAM DIMM

You can fit an SDRAM DIMM to the core module as an option. The following type of DIMM is supported:

- JEDEC r9 compliant
- 3V3 supply
- 168 pin PL100.

The design of the core module imposes limitations on the use of a DIMM. These are:

- the supplied PLD images do not provide a controller for the optional SDRAM DIMM you must implement your own
- the SSRAM and optional SDRAM DIMM share many of the same interface signals and cannot both be used at the same time
- the supplied CP and IM-PD1 images use the on-board SSRAM to store a display frame buffer and this precludes the option of fitting an SDRAM DIMM, see *SSRAM and SDRAM DIMM interface signals* on page 4-22.

4.5.5 SSRAM and SDRAM DIMM interface signals

Figure 4-10 shows the memory interface signal connections from the PLD to the SSRAM and DIMM socket.

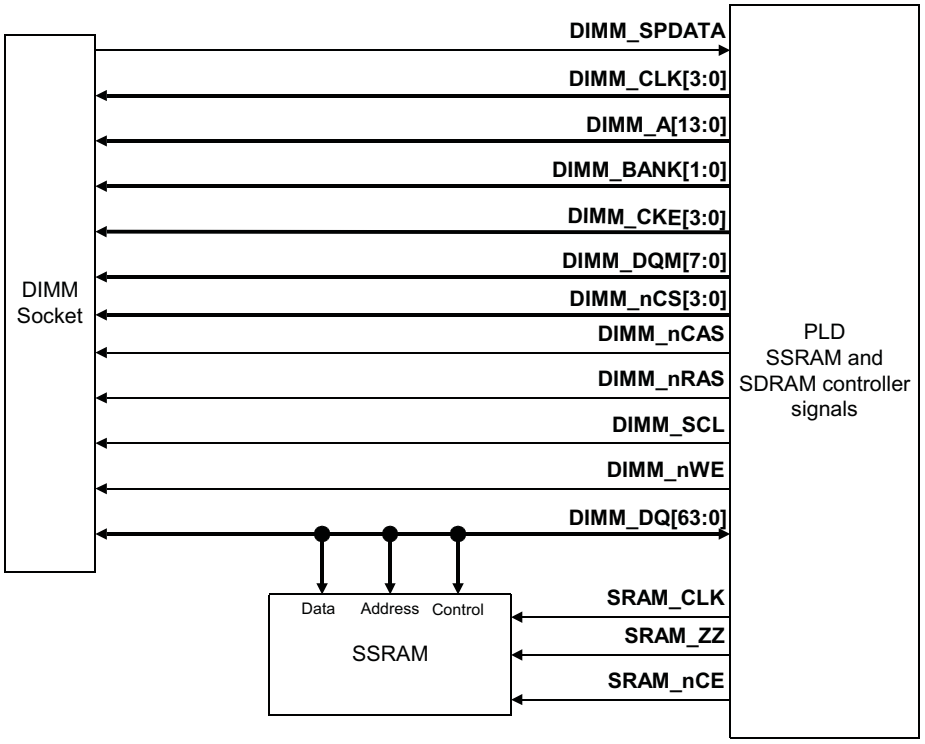


Figure 4-10 PLD/memory interface signals

The **DIMM_DQ[63:0]** signals are shared by the SSRAM and socket. The the SSRAM controllers in the CM and IM-PD1 images use the shared signals as shown in Table 4-7.

Table 4-7 Assignment of DIMM_DQ signals to SSRAM

Signals	SSRAM assignment
DIMM_DQ[58:50]	Control
DIMM_DQ[49:32]	Address
DIMM_DQ[31:0]	Data

4.6 Core module clocks

The core module clocks are described in:

- *Clock signal description* on page 4-24
- *Clock control parameters* on page 4-26
- *Clock programming interface* on page 4-27.

Figure 4-11 on page 4-24 shows the clock generation architecture of the core module. The programmable clock outputs are generated by the two ICS307 devices, OSC1 (U13) and OSC2 (U15). U13 is supplied with a reference clock from a 24MHz crystal oscillator and provides a programmable reference clock to U15 (this is set to 24MHz by default). A second crystal supplies a 33MHz fixed frequency clock to the stripe.

The clock signals are buffered and then supplied to other modules in the system, to the PLD, and to the logic analyzer connectors as described in Table 4-1 on page 4-25. The output enable of the buffer U12B is controlled by the signal **MBDET** so that these outputs are only enabled when the core module is used without a motherboard.

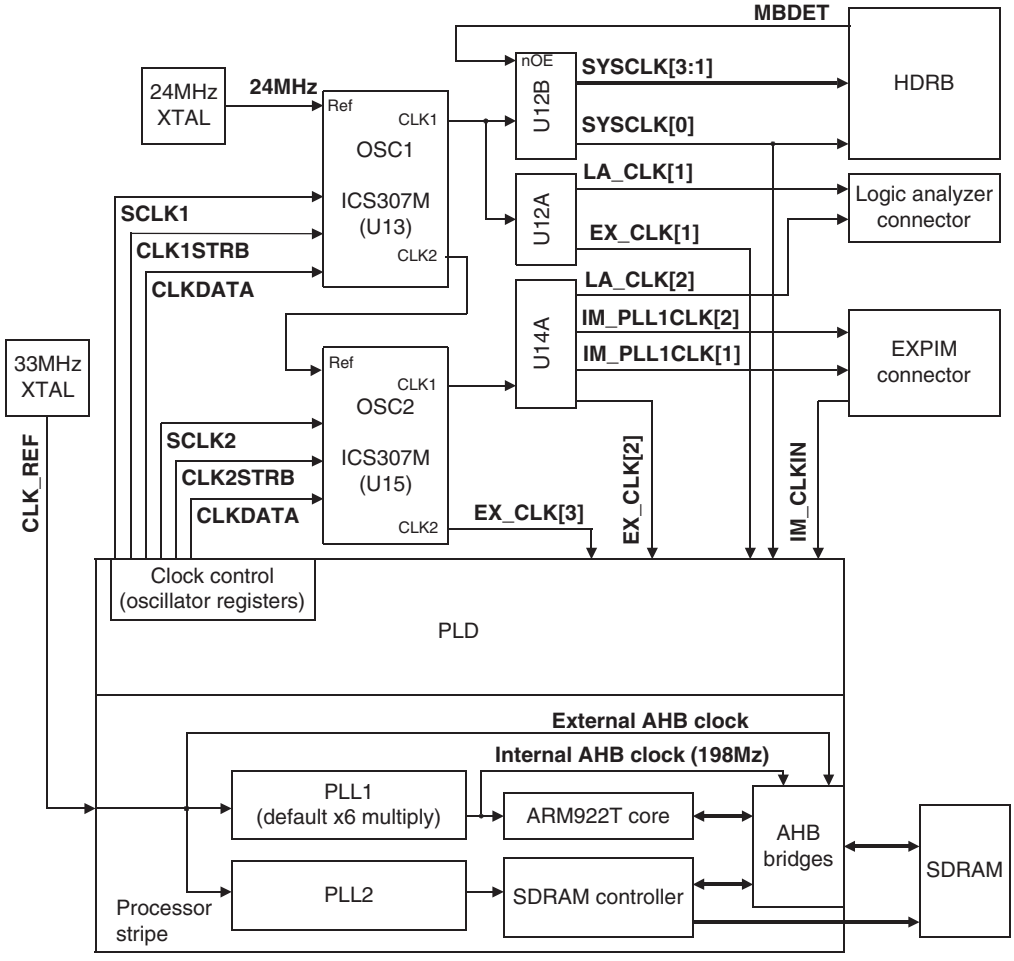


Figure 4-11 Clocking architecture

4.6.1 Clock signal description

The first clock generator OSC1 (U13) is supplied with a 24MHz reference clock by a crystal oscillator. CLK1 from U13 is buffered to produce the signals:

- **SYSCLK[3:0]** (when there is no motherboard present)
- **EX_CLK[1]**
- **LA_CLK[1]**

CLK2 from U13 supplies the reference clock for U15.

The buffered clock signals **SYSCLK[3:0]** are disabled when the core module is mounted on an Integrator/AP motherboard by the signal **MBDET**. This is because the Integrator/AP supplies these clock signals itself.

CLK1 from the second clock generator OSC2 (U15) is buffered to supply the signals:

- **IM_PLL1CLK[2:1]**
- **EX_CLK[2]**
- **LA_CLK[2]**

CLK2 from U15 supplies the clock signal **EX_CLK[3]**.

The output from the crystal oscillator U24 supplies the clock signal **CLK_33MHz**.

Table 4-1 shows the uses to which the generated clock signals are put.

Table 4-1 Clock signal usage

Signal	Function
CLK_33MHz	This clock signal is driven by a fixed-frequency 33MHz crystal oscillator and is used to drive the stripe side PLLs that clock all of the stripe components. PLL1 drives the processor core, and the internal part of the AHB1 and AHB2 bridges. By default, the PLL1 divider is set to 6. This gives a core frequency of 198MHz. PLL2 drives the SDRAM controller.
SYSCLK[3:0]	These signals are generated by the serial programmable clock source, OSC1 (U13), when the core module is used without a motherboard. If the core module is mounted on a motherboard, the buffer U12A is disabled and the signals are generated by the motherboard. These clocks are normally generated on a motherboard and distributed to any modules mounted onto it. Four signal traces are provided to ensure even load balancing. **SYSCLK0** always feeds CLK1p pin of the PLD.
EX_CLK[3:1]	**EX_CLK1** feeds the CLK3p pin of the PLD, **EX_CLK2** feeds CLK4p pin of the PLD, and **EX_CLK3** feeds CLK2p pin of the PLD.

Table 4-1 Clock signal usage (continued)

Signal	Function
LA_CLK[2:1]	These signals are buffered versions of the outputs from the two clock generators and are routed to the logic analyzer connectors. These signals are always driven.
IM_PLL1CLK[2:1]	These signals are buffered versions of CLK1 from the second serial programmable clock source, OSC2 (U15), and routed to the EXPIM connector. These signals are always driven.
IM_CLKIN	This signal from the EXPIM connector is used to drive the FAST4 input of the PLD. This provides access to a low-delay path that you can use to clock devices within the PLD.

4.6.2 Clock control parameters

The clock generators each produce two programmable clock outputs:

CLK1 This is a programmable clock that is generated by passing the reference clock through a reference divider, *Voltage Controlled Oscillator* (VCO), and output divider. The frequency of the output is controlled by three parameters:

- *Reference Divider Word* (RDW)
- *VCO Divider Word* (VDW)
- *Output Divider* (OD).

The general formula used to calculate the output clock frequency of CLK1 from an ICS307 on this core module is:

freq = 48*((VDW+ 8)/(RDW+2)*OD)

Where:

S[2:0] OD selection.

V[8:1] VDW (4 to 511)

R[6:1] RDW (1 to 127).

The default setting at power up, with the control parameters unprogrammed, is CLK1= 24MHz.

CLK2 This is a programmable clock controlled by the function parameter F[[1:0]. The default setting at power up is CLK2=24MHz.

4.6.3 Clock programming interface

The ICS307 is a serially programmed device. It provides a control interface comprising three signals:

CLKDATA This signal is used to pass the control parameters as a 24-bit word into a shift register within the ICS307. This signal is common to both chips.

SCLKn This signal is used to clock the control parameter bits into the shift register. Each bit is transferred in 1 clock cycle.

CLKnSTRB

This is a strobe signal that is asserted after the 24th bit of the parameter word has been transferred into the shift register. It causes the data to be transferred into the internal registers of the ICS307.

Figure 4-12 shows the timing for the serial programming interface signals.

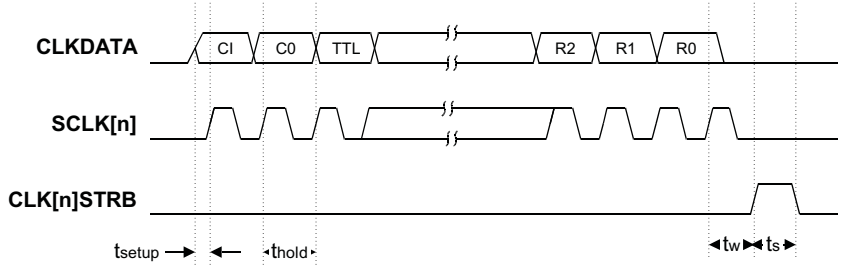


Figure 4-12 Clock generator control interface timing

The control parameters are transferred using the 24-bit control word shown in Figure 4-13.

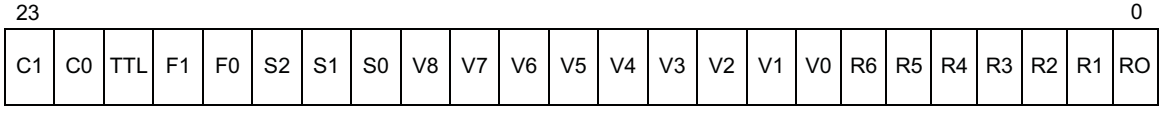


Figure 4-13 Clock control word

The control parameters are as follows:

C[1:0] Internal load capacitance for the crystal. Always set this to 00.

TTL Duty cycle setting. Always set this to 1.

F[1:0] Function setting for CLK2:

- 00 selects CLK2=Ref (this is the default)
- 01 selects CLK2=Ref/2
- 10 selects CLK2 = OFF (LOW).
- 11 selects CLK2=CLK1/2.

Note

The C, TTL, and F parameters are set to the default values at reset and are normally not modifiable. For the basic example image however, these values are programmable, see *Basic example oscillator divisor registers* on page 5-10.

S[2:0] OD selection:

- 000 = divide by 10
- 001 = divide by 2
- 010 = divide by 8
- 011 = divide by 4
- 100 = divide by 5
- 101 = divide by 7
- 110 = divide by 3
- 111 = divide by 6.

V[8:0] VDW.

R[6:0] RDW.

The configuration images provide registers that you can use to program some or all of these parameters, and each provides a functional block to manage the transfer of the parameters into the ICS307s.

- for the Basic example image, see *Basic example image clock control* on page 5-10
- for the CM image, see *CM clock control* on page 6-18.
- for the CP image, see *Integrator/CP922T system clocks* on page 7-28
- for the IM-PD1 image, see *IntegratorCM922T-XA10 and IM-PD1 clock control* on page 8-26.

4.7 Reset architecture

The core module provides several reset signal inputs to and outputs from the PLD. The PLD images supplied with the core module incorporate a reset control logic block that enables the core module to be reset as a standalone unit or as part of a larger development system. The core module is designed to be reset from the following sources:

- a *Power On Reset* (POR)
- push button resets
- resets received from a motherboard or other modules
- resets from a Multi-ICE unit or ByteBlaster equipment
- resets generated by software writing to a register within the PLD.

Figure 4-14 shows the architecture of the reset control subsystem.

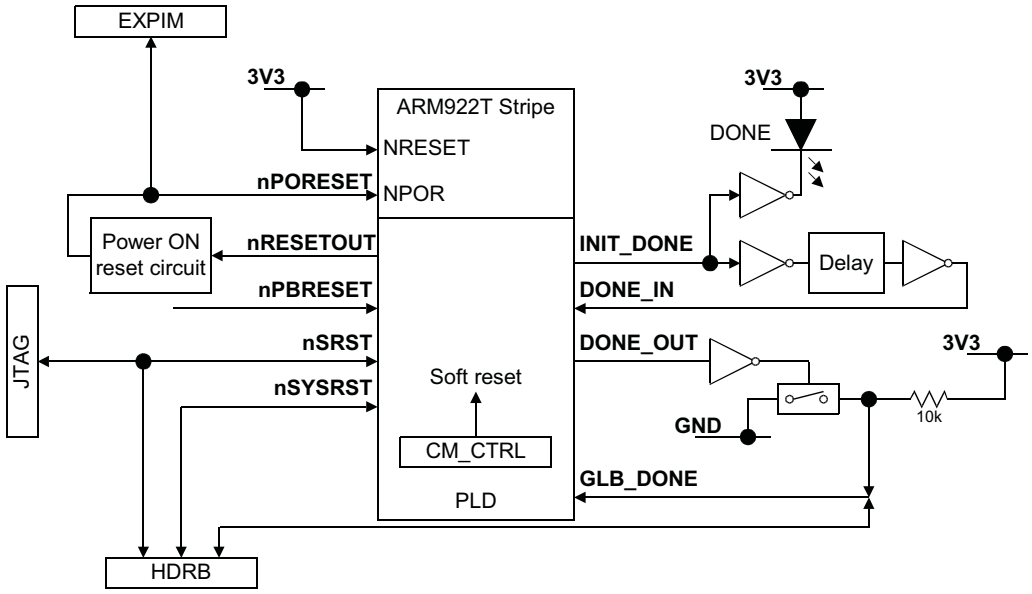


Figure 4-14 Reset architecture

4.7.1 Reset sequence

Resetting the core module causes any other Integrator modules to which it is attached to be reset. The reset control logic instantiated into the PLD and any other programmable devices on other Integrator modules coordinate the reset sequence to ensure that the system is initialized in a predictable and reliable way.

The reset sequence for the core module is as follows:

1. When the core module is powered on the signal **nPORESET** is asserted and held for 100ms by the POR circuit. The delay before releasing **nPORESET** ensures that power supplies are stable before PLD configuration begins.

   Assertion of the **nPORESET** signal resets the core and stripe components, causing the PLD configuration to be reloaded.

2. For subsequent resets, the PLD detects the assertion of one of the other reset signal sources and asserts **nRESETOUT** and **nSRST**. The **nSRST** is used to trigger a reset of other modules in the system.

   The **nRESETOUT** signal is used to assert **nPORESET.** The POR delay circuit is used to ensure that the previous PLD configuration is cleared and that the reset sequence completes reliably.

3. When PLD configuration is complete, the PLD asserts **INIT_DONE** and illuminates the DONE LED.

4. A delayed version of **INIT_DONE** signal, called **DONE_IN**, is fed back into the PLD to assert **DONE_OUT**. This signal is used to control the state of the **GLB_DONE** signal.

5. The **GLB_DONE** signal is an open drain signal shared by the Excalibur PLD and FPGAs on other Integrator modules. Each module monitors and controls **GLB_DONE**. It goes HIGH only when all the programmable devices have been configured. This ensures that the release of **nSRST** by all of the modules is properly coordinated.

   All Integrator modules with programmable logic provide an LED to give a visual indication that they have completed their configuration sequence. See the user guide for your Integrator module.

4.7.2 Reset signals

Table 4-2 describes the reset signals.

Table 4-2 Reset signal descriptions

Name	Description	Type	Function
nPORESET	Power-ON reset	Input	This reset is asserted when the system is powered ON or when the **nRESETOUT** signal is asserted by the reset controller in the PLD. It resets the ARM core and stripe logic and causes the PLD configuration to be reloaded.
nRESETOUT	System reset	Output	This signal triggers a full reset of the PLD and stripe by asserting the **nPORESEST** input to the stripe. It is generated when any of the reset sources (for example, **nPBRESET**) or software reset is asserted.
nPBRESET	Push-button reset	Input	The **PBRESET** signal is generated by pressing the reset button on the core module.
nSRST	System reset	Output	As an output the **nSRST** signal is driven LOW by the core module PLD when any reset source (for example, **nPBRESET**) or software reset is asserted.
		Input	As an input, **nSRST** can be driven LOW by Multi-ICE or by another module connected to the HDRB connector.
nSYSRST	System reset	Output	As an output, **nSYSRST** is generated by the PLD if the core module is used standalone or with an Integrator/CP baseboard.
		Input	As an input if the core module is used with an Integrator/AP motherboard, **nSYSRST** is generated by the system controller FPGA on the motherboard.
INIT_DONE	PLD configuration complete	Output	This signal is generated by the PLD to indicate that the configuration process is complete.
GLB_DONE	PLD/ FPGA configured	Open drain	This signal is routed round the system through the HDRB connectors to all FPGAs and large PLDs in the system. The system is held in reset until **GLB_DONE** is driven HIGH.

4.7.3 Software resets

The PLD images provide a software reset signal that can be triggered by writing to the reset bit in the CM_CTRL register. This generates SOFTRESET and asserts **nSRST**, resetting the whole system. For information about the CM-CTRL register see *Core module control register, CM_CTRL* on page 6-10.

4.8 Interrupt architecture

The stripe provides an interrupt controller that takes interrupt requests from devices within the stripe and six interrupt request lines assigned to the PLD. This is shown in Figure 4-15.

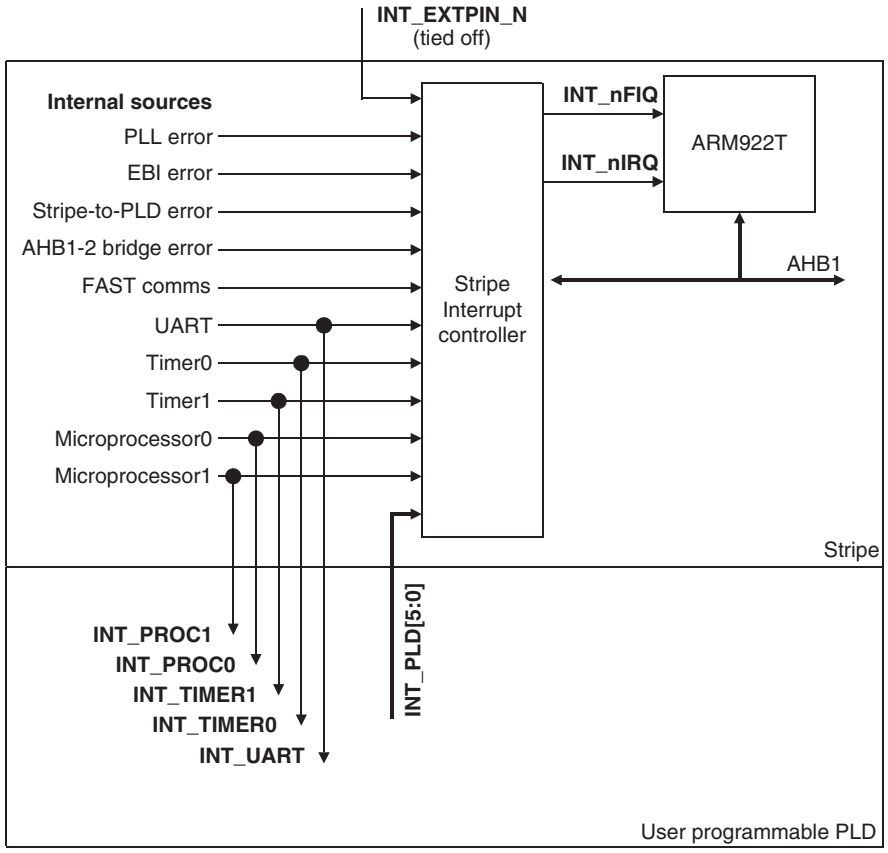


Figure 4-15 Interrupt control

Interrupt request signals from other Integrator modules are routed onto the PLD interrupt requests **INT_PLD[5:0]** by functional blocks implemented in the PLD.

For example, the Integrator/AP provides an interrupt controller within the system controller FPGA, and requests from this are routed to the PLD using the **nFIQ[3:0]** and **nIRQ[3:0]** signal pins on the HDRA connector. The CM image routes **nIRQ[0]** and **nFIQ[0]** onto the **INT_PLD[3]** and **INT_PLD[1]** signals as shown in Table 4-3.

Table 4-3 Assignment of interrupts for the PLD images

INT_PLD	PLD Image		
	CM	CP	IM-PD1
0	**CM_FIQ**	**CM_FIQ**	**CM_FIQ**
1	**nFIQ[0]**	**CP_FIQ**	**nFIQ[0]**
2	**CM_IRQ**	**CM_IRQ**	**CM_IRQ**
3	**nIRQ[0]**	**CP_IRQ**	**nIRQ[0]**
4	-	-	**VIC_FIQ**
5	-	-	**VIC_IRQ**

Table 4-3 shows how the CM, CP, and IM-PD1 images supplied with the core module assign interrupt requests to the **INT_PLD[5:0]** signals, and Table 4-4 describes the interrupt request signals.

Table 4-4 Interrupt sources

Signal	Source
nFIQ[0]	These interrupt requests are generated on the Integrator/AP and routed to the core module PLD using the HDRA or EXPA connectors. The interrupt controller for these is instantiated into the system controller FPGA on the Integrator/AP motherboard. See the *Integrator/AP User Guide*.
nIRQ[0]	
CM_FIQ	These interrupt requests are from the CM interrupt controller instantiated into the core module PLD. Requests are generated by the COMMRx and COMMTx interrupts. See *CM interrupt control* on page 6-21.
CM_IRQ	
CP_FIQ	These interrupt requests are from the *Primary Interrupt Controller* (PIC) instantiated into the core module PLD in the CP image. See *Integrator/CP922T interrupt control* on page 7-33.
CP_IRQ	
VIC_FIQ	These interrupt requests are the *Vectored Interrupt Controller* (VIC) instantiated into the core module PLD by the IM-PD1 image. See *Integrator/CM922T-XA10 and IM-PD1 interrupt control* on page 8-20.
VIC_IRQ	

Chapter 5
Basic Example Image for Simple Standalone Operation

This chapter describes the Basic Example PLD image supplied with the core module. It contains the following:

5.1 About the basic example image

This image is designed to get the core module started at a very basic level. It provides you with an interface to the AHB buses within the stripe and basic register support for the clock generators, resets, LEDs and switches, and a default AHB slave that provides responses to any accesses to the AHB bus. The designer must implement other functional blocks such as memory controllers and peripherals.

Note

The core module cannot be used with an Integrator/AP, CP, or IM-PD1 with the basic example image selected. All signals on the header connectors (HDRA and HDRB) are unused.

This section provides the following information:

- *Selecting the basic example image on your core module*
- *Basic example image functional block diagram* on page 5-3
- *Basic example image functional block HDL files* on page 5-4.

5.1.1 Selecting the basic example image on your core module

To select the basic example image on your core module, set the DIP switches as follows:

- S1[1] = ON
- S1[2] = ON
- S1[3] = ON
- S1[4] = OFF.

This image is signified by the code b01010101 being displayed by the user LEDs (0=OFF, 1=ON) when the system is powered ON.

5.1.2 Basic example image functional block diagram

Figure 5-1 shows the architecture of the Basic Example image.

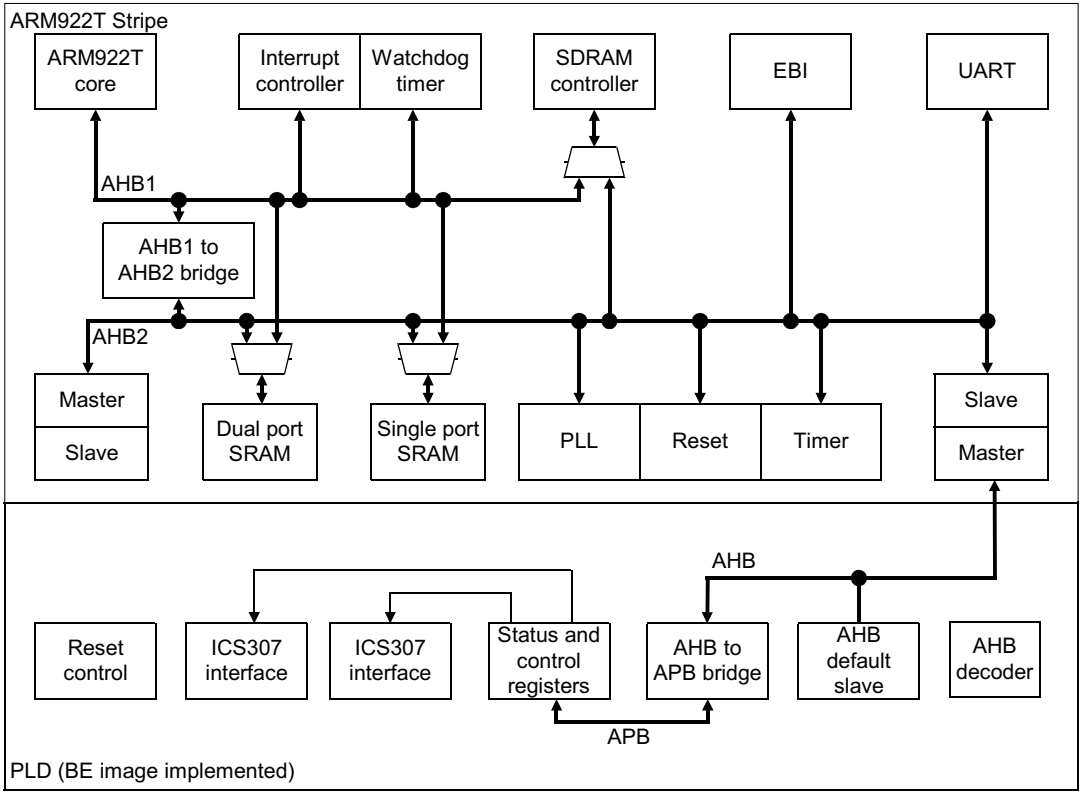


Figure 5-1 PLD Basic example image

5.1.3 Basic example image functional block HDL files

The basic example image provides the functional blocks as a set of HDL files. These are described in Table 5-1.

Table 5-1 Basic example image functional block HDL file descriptions

Block	HDL file	Description
Top level	saxa10fpga.vhd	This is the top level file for this image.
AHB decoder	AhbArmDecoder.vhd	The decoder block provides the high-speed peripherals with select lines. These are generated from the address lines.
AHB default slave	AhbDefaultSlave.vhd	This is a default slave that correctly generates Error responses to accesses to unused address space.
AHB multiplexor	AhbMuxStoM.vhd	This is the AHB multiplexor that connects the read data buses from all of the slaves to the AHB master(s).
AHB to APB bridge	AhbtoApb.vhd	This is the bridge block required to connect APB peripherals to the high-speed AMBA AHB bus. It produce the peripheral select signals for each of the APB peripherals.
Stripe to PLD	arm_excal.vhd	This is a wrapper for the Excalibur stripe.
Status and control registers	Saxa10Regs.vhd	The APB register peripheral provides memory-mapped registers that you can use to: • identify the PLD image • configure the two clock generators • write to the user LEDs • read the user switch inputs • reset the system.
ICS307 interface	ics307.vhd	These blocks are parallel to serial converters used to program the ICS307 devices from the register outputs.

5.2 Basic example image memory map

The memory map for the core module with the basic example PLD image implemented is shown in Figure 5-2. When the core module is powered up, the boot code (in the config flash in EBI0) is mapped to address `0x0`. After the core has booted, SDRAM0 is mapped to appear at this location.

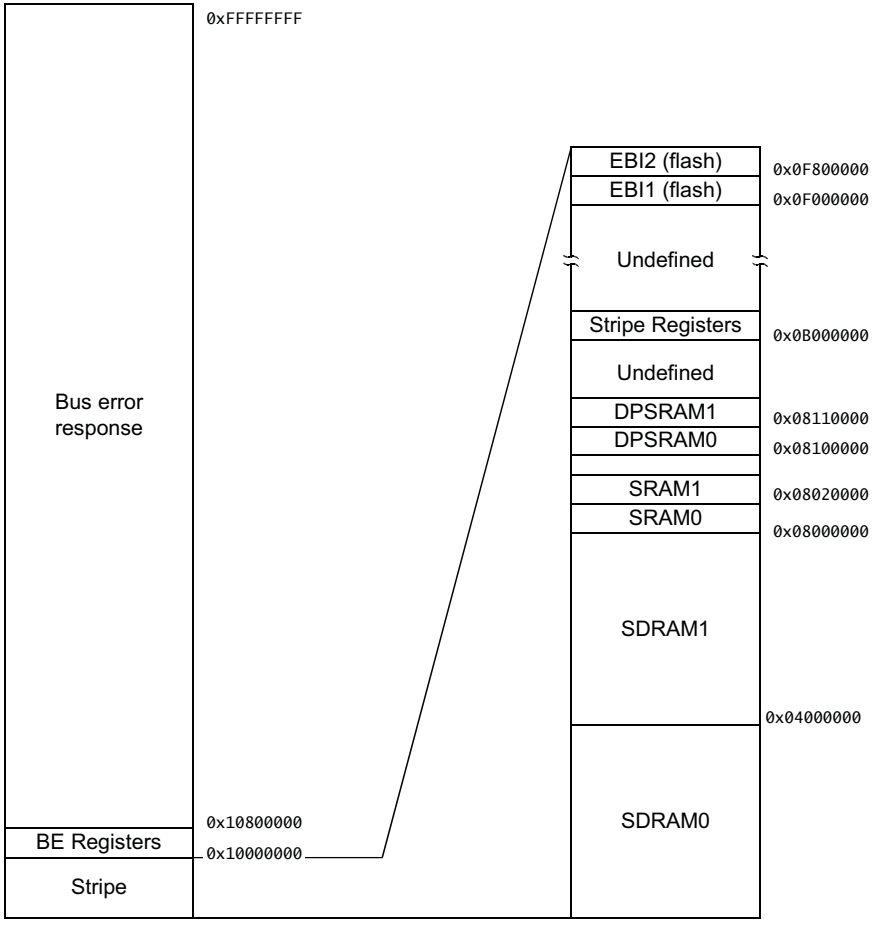


Figure 5-2 Basic example image memory map

The Integrator memory map for the basic example image is shown in Table 5-2.

Table 5-2 Basic example image memory map

Address	Function
0x00000000-0x001FFFFF	Boot flash. Mapped at this address only at power ON, and then disabled to allow access to SDRAM0.
0x00000000-0x03FFFFFF	On-board SDRAM0.
0x04000000-0x07FFFFFF	On-board SDRAM1.
0x08000000-0x0801FFFF	Embedded SRAM0.
0x08020000-0x0803FFFF	Embedded SRAM1.
0x08100000-0x0810FFFF	Embedded DPSRAM0.
0x08110000-0x0811FFFF	Embedded DPSRAM1.
0x0B000000-0x0B003FFF	Stripe registers.
0x0F000000-0x0F7FFFFF	User flash.
0x0F800000-0x0FFFFFFF	User flash.
0x1000000-0x1000001B	Basic example registers.
0x100001C-0x107FFFFF	Reserved.
0x1080000-0xFFFFFFFF	Bus error response.

Note

The basic example image does not provide controllers in the PLD for the on-board SSRAM or for an optional SDRAM DIMM.

5.3 Basic example image registers

Table 5-3 shows the mapping of the core module registers.

Table 5-3 Basic example image registers

Address	Name	Type	Size	Function
0x100000000	BE_ID	Read	32	Identification register
0x100000004	BE_PROC	Read	32	Processor register
0x100000008	BE_OSC1	Read/write	24	Oscillator register 1
0x10000000C	BE_OSC2	Read/write	24	Oscillator register 2
0x100000010	BE_LEDS	Read/write	9	User LEDs control register
0x100000014	BE_SW	Read	8	User switch register
0x100000018	BE_RST	Read/write	2	Reset register

5.3.1 Basic example ID register, BE_ID

The BE_ID register is a read-only register that identifies the board manufacturer, board type, PLD type, and revision.

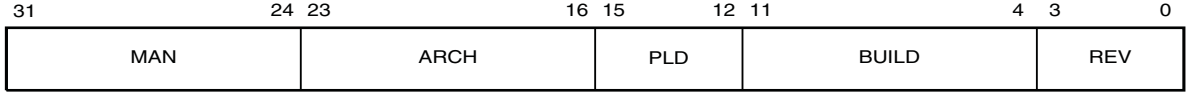


Figure 5-3 BE_ID register

Table 5-4 describes the core module ID register bits.

Table 5-4 BE_ID register bit descriptions

Bits	Name	Access	Function
[31:24]	MAN	Read	Manufacturer: 0x41 = ARM
[23:16]	ARCH	Read	Architecture: 0x0A = AHB interface, 32-bit SDRAM

Table 5-4 BE_ID register bit descriptions (continued)

Bits	Name	Access	Function
[15:12]	PLD	Read	FPGA type: 0x5 = XA10
[11:4]	BUILD	Read	Build value (ARM internal use)
[3:0]	REV	Read	Revision: 0x0 = Rev A 0x1 = Rev B.

5.3.2 Basic example processor register, BE_PROC

The BE_PROC register is a read-only register that contains the value 0x00000000. This is provided for compatibility with processors that do not have a system control coprocessor, CP15. For the ARM922T core, you can obtain information about the processor by reading coprocessor 15 register 0, CP15 c0.

5.3.3 Oscillator divisor registers, BE_OSCx

The oscillator registers control the frequency of the clocks generated by the two clock generators (see *Core module clocks* on page 4-23). See *Basic example oscillator divisor registers* on page 5-10.

5.3.4 User LEDs control register, BE_LEDs

The LEDs register is used to control the user LEDs (see *Core module LEDs, switches, and links* on page 1-6). Writing a 1 to a bit lights the associated LED.

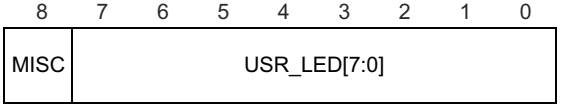


Figure 5-4 BE_LEDs register

5.3.5 User switch register, BE_SW

This register is used to read the setting of the 8-way user switch. A 0 indicates that the associated switch element is OFF.

5.3.6 Reset register, BE_RST

This register is used for flash memory protection and software reset.

Table 5-5 BE_RST register bit descriptions

Bits	Name	Access	Function
[31:2]	Reserved	-	-
[1]	Protect	Read/Write	Write a 1 to write-protect the flash memory. Write a 0 to enable writing.
[0]	Software reset	Write	Write a 1 to this bit to initiate a software reset.

5.4 Basic example image clock control

The two clock generators are independently programmable using their serial programming interfaces. The basic example image provides two registers that allow you to set the clock parameters. The serial transfer of the register data into the clock generators is handled by the register interface to the generators.

5.4.1 Calculating the output frequencies of the ICS307

The general formula used to calculate the output clock frequency of CLK1 from an ICS307 for this image is:

freq = 2*REFCLK*(VDW+ 8)/((RDW+2)*OD)

For a description of the parameters, see *Clock control parameters* on page 4-26.

5.4.2 Basic example oscillator divisor registers

There are two oscillator control register that control the frequency of the clocks generated by the two clock generators (see *Core module clocks* on page 4-23). BE_OSC0 controls U13 and BE_OSC1 controls U15. Table 5-6 describes the oscillator register bits.

Table 5-6 BE_OSCx registers

Bits	Name	Access	Function	Default
[23:22]	C	Read/write	Internal load capacitance for crystal	00
[21]	TTL	Read/write	Output duty cycle configuration	1
[20:19]	CLK2	Read/write	Function of CLK2 output: 00 = Reference 01 = Reference/2 10 = OFF 11 = CLK1/2	00

Table 5-6 BE_OSCx registers (continued)

Bits	Name	Access	Function	Default
[18:16]	OD	Read/write	Output divider select: 000 = divide by 10 001 = divide by 2 010 = divide by 8 011 = divide by 4 100 = divide by 5 101 = divide by 7 110 = divide by 3 111 = divide by 6.	011
[15:7]	VDW	Read/write	VCO divider word.	000001000
[6:0]	RDW	Read/write	Reference divider word.	0000110

Chapter 6
CM Image for Operation Standalone or with an Integrator/AP

This chapter describes the architecture of the core module when the CM image is implemented in the PLD. It contains the following sections:

6.1 About the Integrator CM image

The CM PLD image implements the functional blocks that are common to the Integrator family of core modules. The image enables you to use the core module on its own (standalone) or with an Integrator/AP motherboard. This section provides the following information:

- *Selecting the CM image on a standalone core module*
- *Selecting the CM image on a core module mounted on an Integrator/AP*
- *CM image functional block diagram* on page 6-3
- *The functional block HDL files* on page 6-4.

6.1.1 Selecting the CM image on a standalone core module

To select the CM image on a standalone core module, set the DIP switches as follows:

- S1[1] = x
- S1[2] = x
- S1[3] = OFF
- S1[4] = OFF.

The core module signals **CFGSEL[1:0]** default to `b10` to select the core module image. This image can be used either standalone or with an Integrator/AP. The default value is selected by pullup and pulldown resistors on the core module. If the core module is attached to a CP, a different image is automatically selected for loading, see *PLD image selection* on page 4-8.

6.1.2 Selecting the CM image on a core module mounted on an Integrator/AP

To select the CM image on a core module mounted on an Integrator/AP, set the DIP switches as follows:

- S1[1] = x
- S1[2] = x
- S1[3] = OFF
- S1[4] = OFF.

The motherboard must drive the signals **CFGSEL[1:0]** correctly to select the correct image, see *PLD image selection* on page 4-8.

This image is signified by the code b11100111 being displayed by the user LEDs (0=OFF, 1=ON) when the system is powered ON.

6.1.3 CM image functional block diagram

The CM image contains a number of functional blocks as shown in Figure 6-1.

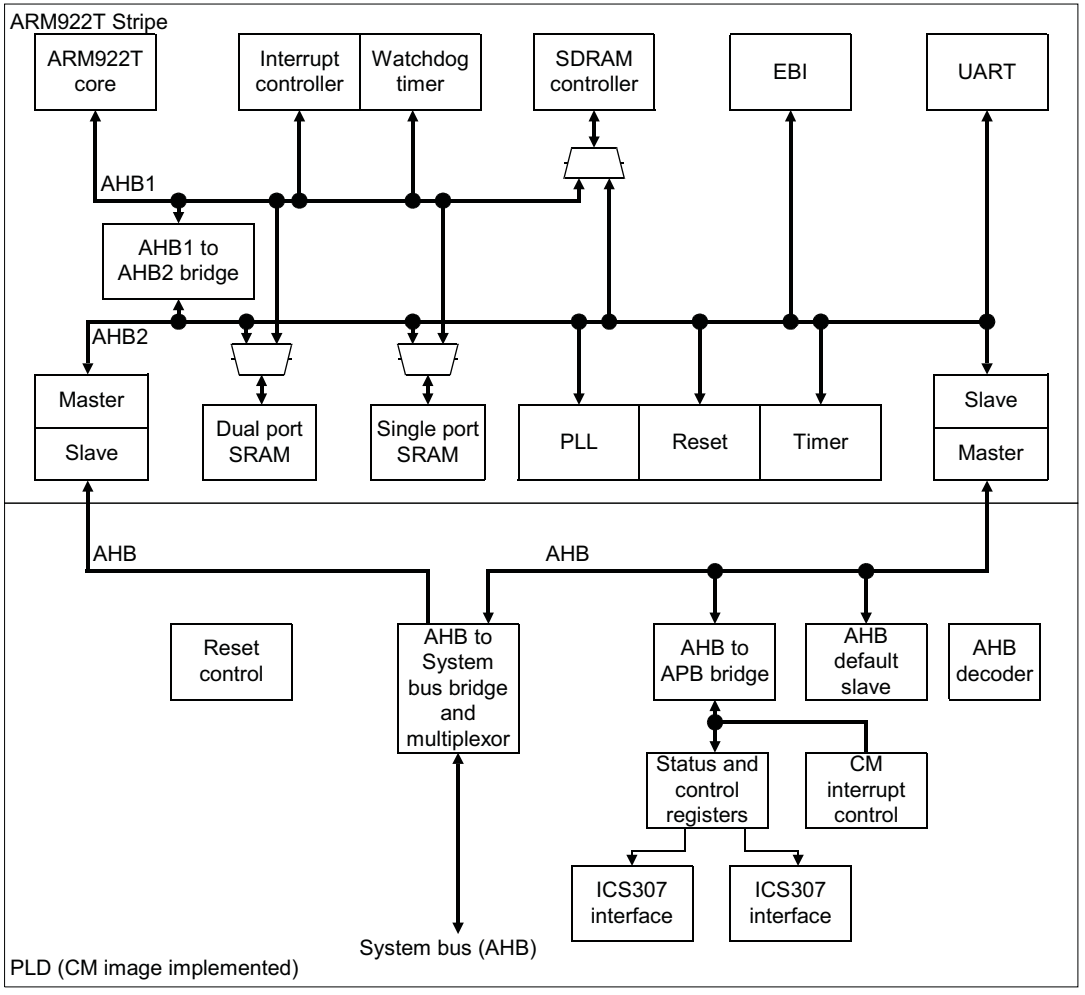


Figure 6-1 Core module image architecture

Note

The CM image is not compatible with the Integrator/CP baseboard. See Chapter 7 *CP Image for Operation with an Integrator/CP Baseboard* for more information.

6.1.4 The functional block HDL files

The CM image provides the functional blocks as a set of HDL files. These are described in Table 6-1.

Table 6-1 CM image functional block HDL file descriptions

Block	HDL file	Description
Top level	`cmxa10fpga.vhd`	This is the top level file for this image.
AHB decoder	AhbArmDecoder.vhd	The decoder block provides the high-speed peripherals with select lines. These are generated from the address lines and the module ID (position in stack) signals from the motherboard. The decoder blocks also contain the default slave peripheral. The Integrator family of boards uses a distributed address decoding system.
AHB multiplexor	`AhbMuxStoM.vhd`	This is the AHB multiplexor that connects the read data buses from all of the slaves to the AHB master(s).
AHB to APB bridge	`AhbtoApb.vhd`	This is the bridge block required to connect APB peripherals to the high-speed AMBA AHB bus. It produce the peripheral select signals for each of the APB peripherals.
AHB default slave	`AhbDefaultSlave.vhd`	This is a default slave that correctly generates Error responses to accesses to unused address space.
Stripe to PLD	`arm_excal.vhd`	This is a wrapper for the Excalibur stripe.
CM registers	`cmxa10Regs.vhd`	The APB register peripheral provides memory-mapped status and control registers that you can use to: • identify the PLD image • configure the two clock generators (protected by the CM_LOCK register) • write to the user LEDs • read the user switch inputs • reset the system.
CM interrupt controller	`CMIntcon.vhd`	The APB interrupt controller contains all of the standard interrupt controller registers.
ICS307 interface	`ics307.vhd`	These blocks are parallel to serial converters used to program the ICS307 devices from the register outputs.
Stripe to PLD	`arm_excal.vhd`	This block instantiates the connections between the stripe and the PLD.
APB multiplexor	`ApbMux2.vhd`	This block multiplexes the APB bus to the two APB slaves.

6.2 CM image memory map

The Integrator memory map for the CM image is shown in Figure 6-2. When the core module is powered up, the boot code (in the config flash in the EBI0 region) is mapped to `0x0`. After the core has booted, SDRAM0 is mapped to appear at this location, see Table 6-2 on page 6-6.

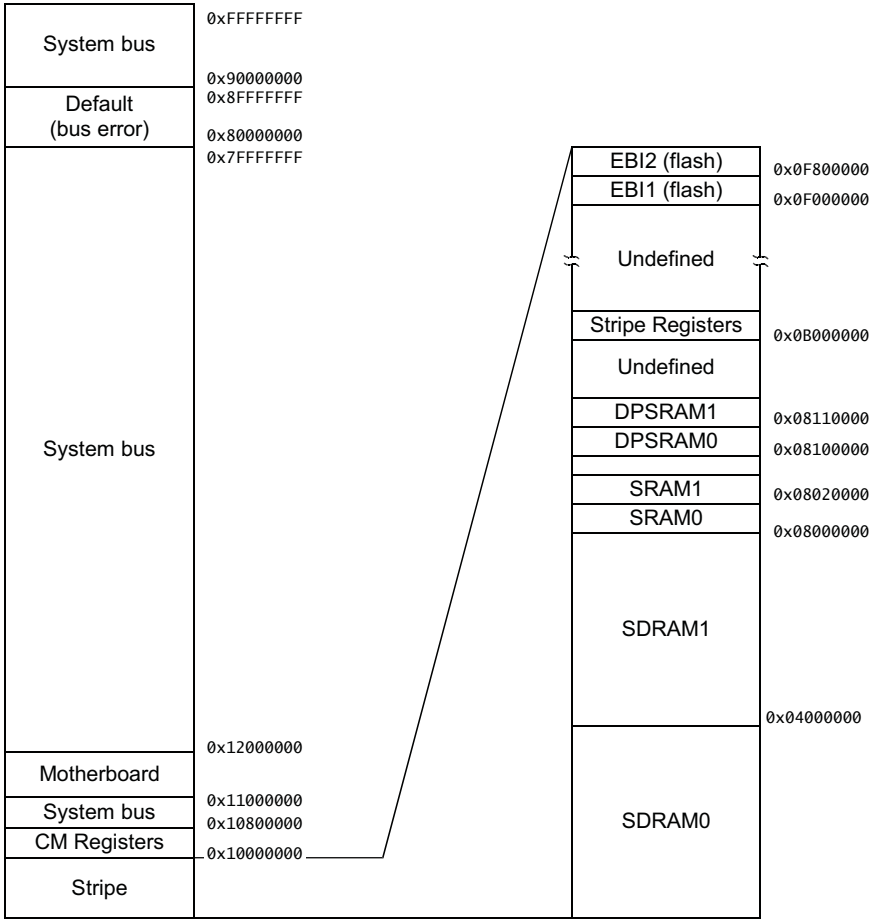


Figure 6-2 CM image memory map

The Integrator memory map for the CM image is shown in Table 6-2.

Table 6-2 CM image memory map

Address	Function
0x00000000-0x001FFFFF	Boot flash (EBI0). Mapped at this address only at power ON, and then disabled to allow access to SDRAM0.
0x00000000-0x03FFFFFF	On-board SDRAM0.
0x04000000-0x07FFFFFF	On-board SDRAM1.
0x08000000-0x0801FFFF	Embedded SRAM0.
0x08020000-0x0803FFFF	Embedded SRAM1.
0x08100000-0x0810FFFF	Embedded DPSRAM0.
0x08110000-0x0811FFFF	Embedded DPSRAM1.
0x0B000000-0x0B003FFF	Stripe registers.
0x0F000000-0x0F7FFFFF	User flash (EBI1).
0x0F800000-0x0FFFFFFF	User flash (EBI2).
0x1000000-0x1000001F	Core module registers.
0x1000100-0x107FFFFF	Reserved. Accesses result in undefined behavior.
0x1080000-0x10FFFFFF	System bus.
0x1100000-0x1100003F	Motherboard registers (if the core module is attached to a motherboard).
0x1100040-0x11FFFFFF	Motherboard.
0x1200000-0x7FFFFFFF	System bus.
0x8000000-0x8FFFFFFF	Default when core module is ID0, see *Module ID selection* on page 4-15.
0x9000000-0xFFFFFFFF	System bus.

System bus transactions all go onto the bus routed between Integrator modules. If there is more than one module, then the slave address location of each module is determined by their positions in the stack (see *Module ID selection* on page 4-15).

You can assign the optional SDRAM DIMM to any of the PLD regions. However, if you use the core module with other Integrator modules, you must ensure that it does not clash with other devices in the Integrator system memory map to maintain full system functionality. To understand how the other modules are memory mapped, refer to the user guide supplied with each module.

Note

The CM image does not provide an SSRAM controller. To ensure compatibility with other Integrator core modules, the SSRAM should be located at `0x1080000-108FFFFF`.

6.3 CM control and status registers

The CM PLD image provides a set of status and control registers, interrupt registers and flag registers. These are located within the PLD0 memory map region. Table 6-3 lists the CM registers.

Table 6-3 Core module status, control, and interrupt registers

Register Name	Address	Access	Description
CM_ID	0x10000000	Read	*Core module ID register, CM_ID* on page 6-9
CM_PROC	0x10000004	Read	*Core module processor register, CM_PROC* on page 6-10
CM_OSC	0x10000008	Read/write	*Core module oscillator register, CM_OSC* on page 6-10
CM_CTRL	0x1000000C	Read/write	*Core module control register, CM_CTRL* on page 6-10
CM_STAT	0x10000010	Read	*Core module status register, CM_STAT* on page 6-13
CM_LOCK	0x10000014	Read/write	*Core module lock register, CM_LOCK* on page 6-14
CM_COUNTER	0x10000018	Read	*Core module bus cycle counter, CM_COUNTER* on page 6-14
-	0x1000001C	-	Reserved
CM_SDRAM	0x10000020	Read/write	*SDRAM status and control register, CM_SDRAM* on page 6-15
-	0x10000024	-	Reserved
CM_REFCNT	0x10000028	Read	*Core module reference clock cycle counter, CM_REFCNT* on page 6-16
-	0x1000002C	-	Reserved
CM_FLAGS	0x10000030	Read	*CM flag registers* on page 6-16
CM_FLAGSS	0x10000030	Write	
CM_FLAGSC	0x10000034	Write	
CM_NVFLAGS	0x10000038	Read	
CM_NVFLAGSS	0x10000038	Write	
CM_NVFLAGSC	0x1000003C	Write	

Table 6-3 Core module status, control, and interrupt registers (continued)

Register Name	Address	Access	Description
CM_IRQ_STAT	0x10000040	Read	*CM interrupt controller* on page 6-23
CM_IRQ_RSTAT	0x10000044	Read	
CM_IRQ_ENSET	0x10000048	Read/write	
CM_IRQ_ENCLR	0x1000004C	Write	
CM_SOFT_INTSET	0x10000050	Read/write	
CM_SOFT_INTCLR	0x10000054	Write	
CM_FIQ_STAT	0x10000060	Read	
CM_FIQ_RSTAT	0x10000064	Read	
CM_FIQ_ENSET	0x10000068	Read/write	
CM_FIQ_ENCLR	0x1000006C	Write	
-	0x10000070	-	Reserved.
-	0x10000080	-	Reserved
-	0x10000090	-	Reserved
-	0x10000080 - 0x107FFFFF	-	Reserved

Note

All registers are 32-bits wide and only support word-wide writes. Do not implement registers at locations marked reserved. Preserve the register bits marked as *reserved* in the following sections by using read-modify-write operations.

6.3.1 Core module ID register, CM_ID

The core module ID register at 0x10000000 is a read-only register that identifies the board manufacturer, board type, and revision.

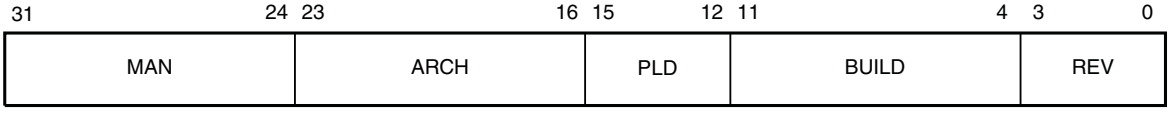


Figure 6-3 CM_ID register

Table 6-4 describes the core module ID register bits.

Table 6-4 CM_ID register bit descriptions

Bits	Name	Access	Function
[31:24]	MAN	Read	Manufacturer: 0x41 = ARM
[23:16]	ARCH	Read	Architecture: 0x0A = AHB interface, 32-bit SDRAM
[15:12]	PLD	Read	PLD type: 0x5 = XA10
[11:4]	BUILD	Read	Build value (ARM internal use)
[3:0]	REV	Read	Revision: 0x0 = Rev A 0x1 = Rev B

6.3.2 Core module processor register, CM_PROC

The core module processor register at 0x10000004 is a read-only register that contains the value 0x00000000. This is provided for compatibility with processors that do not have a system control coprocessor, CP15. For the ARM922T core, information about the processor can be obtained by reading coprocessor 15 register 0, CP15 c0.

6.3.3 Core module oscillator register, CM_OSC

This register is described in *Core Module oscillator register, CM_OSC* on page 6-19.

6.3.4 Core module control register, CM_CTRL

The core module control register at 0x1000000C is a read/write register that provides control of a number of user-configurable features of the core module.

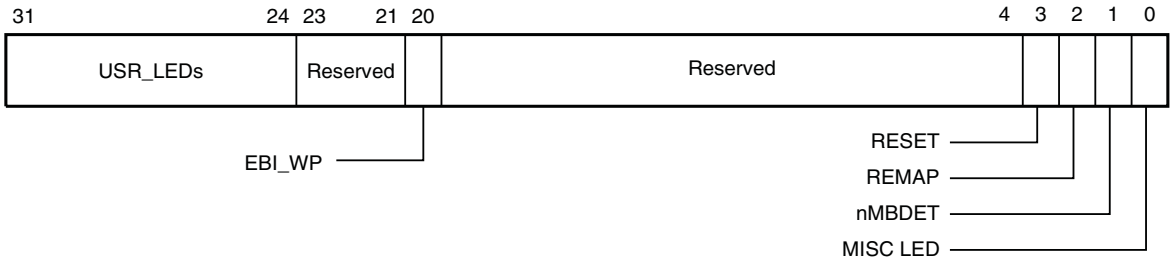


Figure 6-4 CM_CTRL register for the CM image

Table 6-5 describes the core module control register bits

Table 6-5 CM_CTRL register

Bits	Name	Access	Function
[31:24]	USR_LED	Write	These bits are used to control the user LEDs: 0 = LED OFF 1 = LED ON.
[23:21]	Reserved	Use read-modify-write to preserve value.	
[20]	EBI_WP	Write	This bit controls user flash write protect signal **EBI_WP** signal: 0 = Flash write protected (default) 1 = Flash write enabled.
[19:4]	Reserved	Use read-modify-write to preserve value.	
[3]	RESET	Write	This is used to reset the core module, the motherboard on which it is mounted, and any modules in a stack. A reset is triggered by writing a 1. Reading this bit always returns a 0 allowing you to use read-modify-write operations without masking the RESET bit.

Table 6-5 CM_CTRL register (continued)

Bits	Name	Access	Function
[2]	REMAP	Read/write	Remap is not supported by the Excalibur chip. This bit is hard coded to 1 for software compatibility.
[1]	nMBDET	Read	This bit indicates whether or not the core module is mounted on a motherboard: 0 = mounted on motherboard 1 = standalone.
[0]	LED	Read/write	This bit controls the green MISC LED on the core module: 0 = LED OFF 1 = LED ON.

6.3.5 Core module status register, CM_STAT

The core module status register at `0x10000010` is a read-only register that can be read to determine the size of the SSRAM present, the test chip type, and where in a stack this core module is positioned.

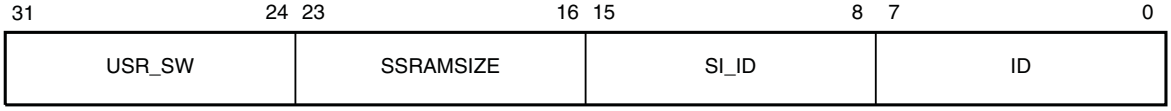


Figure 6-5 CM_STAT register

Table 6-6 describes the core module status register bits.

Table 6-6 CM_STAT register

Bit	Name	Access	Function	Default
[31:24]	USR_SW	Read	These are used to read the user switches. 1 = ON 0 = OFF.	-
[23:16]	SSRAMSIZE	Read	SSRAM size.	00000100
[15:8]	SI_ID	Read	Silicon manufacturer identification. This field is retained for compatibility with other core modules that use this to identify the manufacturer and type of core fitted to the module.	00000000
[7:0]	ID	Read	Card number in stack: `0x00` = core module 0 `0x01` = core module 1 `0x02` = core module 2 `0x03` = core module 3 `0xFF` = invalid or no motherboard attached.	-

6.3.6 Core module lock register, CM_LOCK

The core module lock register at `0x10000014` is a read/write register that is used to control access to the CM_OSC register, allowing it to be locked and unlocked. This mechanism prevents this register from being overwritten accidently.

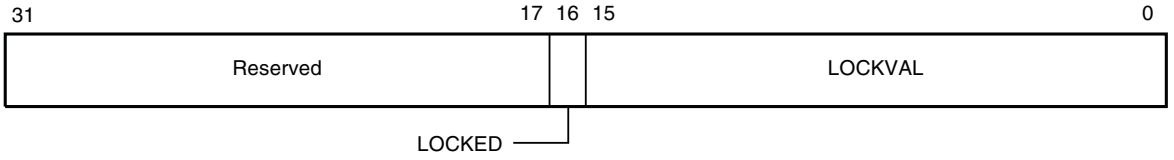


Figure 6-6 CM_LOCK register

Table 6-7 describes the core module lock register bits.

Table 6-7 CM_LOCK register

Bits	Name	Access	Function
[31:17]	Reserved	Use read-modify-write to preserve value.	
[16]	LOCKED	Read	This bit indicates if the CM_OSC register is locked or unlocked: 0 = unlocked 1 = locked.
[15:0]	LOCKVAL	Read/write	Write the value `0x0000A05F` to this register to enable write accesses to the CM_OSC register. Write any other value to this register to lock the CM_OSC register.

6.3.7 Core module bus cycle counter, CM_COUNTER

This register at `0x10000018` provides a 32-bit count value. The count increments at the **SYSCLK[0]** frequency and can be used as a cycle counter for performance measurement. The register is reset to zero by a reset.

6.3.8 SDRAM status and control register, CM_SDRAM

The SDRAM status and control register at `0x10000020` is a read-only register used to read the configuration parameters of an DDR SDRAM controlled by the stripe.

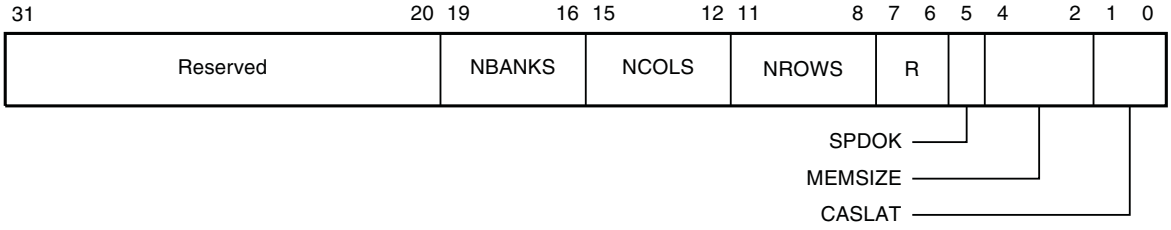


Figure 6-7 CM_SDRAM register

Table 6-8 describes the SDRAM status and control register bits.

Table 6-8 CM_SDRAM register

Bits	Name	Access	Function
[31:20]	Reserved	Use read-modify-write to preserve value.	
[19:16]	NBANKS	Read	Number of SDRAM banks.
[15:12]	NCOLS	Read	Number of SDRAM columns.
[11:8]	NROWS	Read	Number of SDRAM rows.
[7:6]	Reserved	Use read-modify-write to preserve value.	

Table 6-8 CM_SDRAM register (continued)

Bits	Name	Access	Function
[5]	SPDOK	Read	This bit is always 0.
[4:2]	MEMSIZE	Read	These bits specify the size of the SDRAM module fitted to the core module. The bits are encoded as follows: 000 = 16MB 001 = 32MB 010 = 64MB 011 = 128MB (default) 100 = 256MB 101 = Reserved 110 = Reserved 111 = Reserved.
[1:0]	CASLAT	Read	These bits specify the CAS latency for the core module SDRAM. The bits are encoded as follows: 00 = DDR SDRAM (default) 01 = Reserved 10 = 2 cycles 11 = 3 cycles.

6.3.9 Core module reference clock cycle counter, CM_REFCNT

This register at `0x10000028` provides a 32-bit count value. The count increments at the fixed reference clock frequency and can be used as a real-time counter. The register is reset to zero by a reset.

6.3.10 CM flag registers

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

The core module provides two distinct types of flag registers:

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

Flag/nonvolatile flag register, CM_FLAGS/CM_NVFLAGS

The status register contains the current state of the flags.

Flag/nonvolatile flag set register, CM_FLAGSS/CM_NVFLAGSS

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/nonvolatile flag clear register, CM_FLAGSC/CM_NVFLAGSC

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.

6.4 CM clock control

The programmable clocks are supplied by two Micrologic ICS307 clock generators, as described in:

- *Calculating the output frequencies of the ICS307*
- *Core Module oscillator register, CM_OSC* on page 6-19.

For more detail on how the clocks are used, see *Core module clocks* on page 4-23.

6.4.1 Calculating the output frequencies of the ICS307

The CM image provides registers to control the following parameters:

S[2:0] OD selection.

V[8:1] VDW. **V[0]** is always 0.

The function setting for CLK2 selects the reference input frequency as the output frequency for this clock (that is 24MHz).

The equation used to calculate the output clock frequency from an ICS307 for this image is:

freq = 48*(VDW+ 8)/(3*OD)

This equation can be used because:

- A reference clock is supplied at a known frequency of 24MHz to the first clock generator.
- RDW has a fixed value of 1 for both clock generators. This makes is possible to fit the control parameters for both clock generators into one register and maintains compatibility with earlier Integrator core modules.

6.4.2 Core Module oscillator register, CM_OSC

The core module oscillator register at `0x10000008` is a read/write register that controls the frequency of both clock generators.

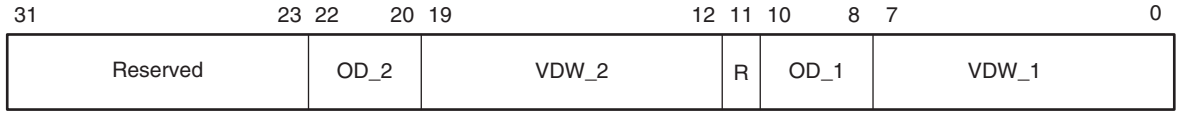


Figure 6-8 CM_OSC register

Note

Before writing to the CM_OSC register, you must unlock it by writing the value `0x0000A05F` to the CM_LOCK register. After writing the CM_OSC register, relock it by writing any value other than `0x0000A05F` to the CM_LOCK register.

Table 6-9 describes the core module oscillator register bits.

Table 6-9 CM_OSC register

Bits	Name	Access	Function	Default
[31:23]	Reserved	-	Use read-modify-write to preserve value.	-
[22:20]	OD_2	Read/write	Output divider select for U15: 000 = divide by 10 001 = divide by 2 010 = divide by 8 011 = divide by 4 100 = divide by 5 101 = divide by 7 110 = divide by 3 111 = divide by 6.	111
[19:12]	VDW_2	Read/write	These bits control bits [8:1] of the clock VCO divider word for U15. Bit [0] of the VCO divider word is always 0.	00000010

Table 6-9 CM_OSC register (continued)

Bits	Name	Access	Function	Default
[11]	Reserved	Use read-modify-write to preserve value.		-
[10:8]	OD_1	Read/write	Output divider select for U13: 000 = divide by 10 001 = divide by 2 010 = divide by 8 011 = divide by 4 100 = divide by 5 101 = divide by 7 110 = divide by 3 111 = divide by 6.	111
[7:0]	VDW_1	Read/write	These bits control bits [8:1] of the clock VCO divider word for U13. Bit [0] of the VCO divider word is always 0.	00000010

Table 6-10 shows the association between the clock generator chips and the signals that they generate.

Table 6-10 Clock signal association

Chip	Clock output	Signal name
U13	CLK1	**SYSCLK[3:0]** (when **BDET**=1)
		EX_CLK[1]
		LA_CLK[1]
	CLK2	**24MHz**
U15	CLK1	**IM_PLL1CLK[2:1]**
		EX_CLK[2]
		LA_CLK[2]
	CLK2	**EX_CLK[3]**

6.5 CM interrupt control

Figure 6-9 shows the architecture of the interrupt control system on the CM922T-XA10 core module fitted to an Integrator/AP motherboard. It shows the routing of the IRQ signals but omits the FIQ signals because they are routed in a similar way.

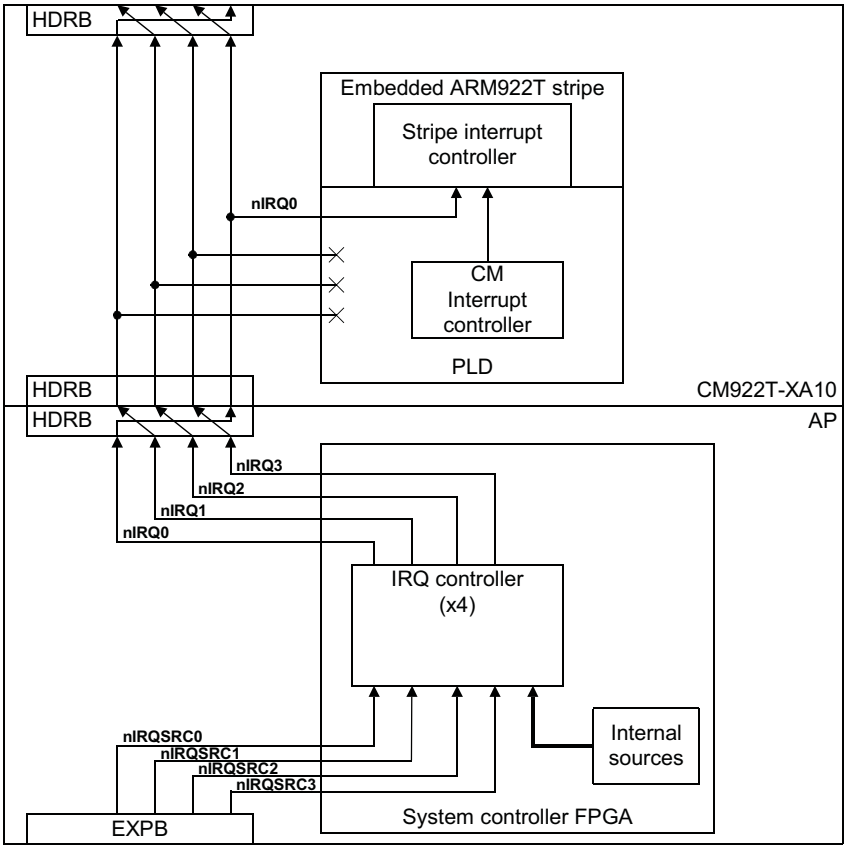


Figure 6-9 CM image interrupt control (showing IRQs only)

Note

The interrupt signals **IRQ[3:1]** and **FIQ[3:1]** on the core module are routed to the PLD but are unused by the CM image.

The Excalibur chip contains an interrupt controller within the stripe, which is present whether or not the PLD section is programmed. The CM image instantiates a second debug interrupt controller into the PLD that controls the COMMTx and COMMRx interrupts for the core.

The System controller FPGA on the AP also contains IRQ and FIQ interrupt controllers. These control interrupts from the logic module connector (EXPB), from internal peripherals, and from the PCI (not shown). Each interrupt source can be assigned to the IRQ or FIQ input to any of up to four processors. For more information about the interrupt controller on the AP, see the *Integrator/AP User Guide*.

The assignment of the interrupts from the AP and from the debug interrupt controller to the stripe controller inputs are shown in Table 6-11.

Table 6-11 CM image interrupt assignment

PLD_INT	CM
0	**CM_FIQ**
1	**nFIQ0**
2	**CM_IRQ**
3	**nIRQ0**
4	-
5	-

6.5.1 CM interrupt controller

The core module provides a 3-bit IRQ controller and 3-bit FIQ controller to support:

- *Comms interrupts*
- *Soft interrupts* on page 6-25.

Comms interrupts

The core module provides a 3-bit IRQ controller and 3-bit FIQ controller to support the debug communications channel used for passing information between applications software and the debugger. The interrupt control registers are listed in Table 6-12.

Table 6-12 Comms interrupt controller registers

Register Name	Address	Access	Description
CM_IRQ_STAT	0x10000040	Read	Core module IRQ status register
CM_IRQ_RSTAT	0x10000044	Read	Core module IRQ raw status register
CM_IRQ_ENSET	0x10000048	Read/write	Core module IRQ enable set register
CM_IRQ_ENCLR	0x1000004C	Write	Core module IRQ enable clear register
CM_SOFT_INTSET	0x10000050	Read/write	Core module software interrupt set
CM_SOFT_INTCLR	0x10000054	Write	Core module software interrupt clear
CM_FIQ_STAT	0x10000060	Read	Core module FIQ status register
CM_FIQ_RSTAT	0x10000064	Read	Core module FIQ raw status register
CM_FIQ_ENSET	0x10000068	Read/write	Core module FIQ enable set register
CM_FIQ_ENCLR	0x1000006C	Write	Core module FIQ enable clear register

Note

All registers are 32-bits wide and only support word-wide writes. Bits marked as *reserved* in the following sections should be preserved using read-modify-write operations.

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 bit assignments for the IRQ and FIQ status, raw status and enable register are shown in Table 6-13.

Table 6-13 IRQ and FIQ register bit assignment

Bit	Name	Function
31:3	Reserved	Write as 0. Reads undefined.
2	COMMTx	Debug communications transmit interrupt. This interrupt indicates that the communications channel is available for the processor to pass messages to the debugger.
1	COMMRx	Debug communications receive interrupt. This interrupt indicates to the processor that messages are available for the processor to read.
0	SOFT	Software interrupt

The way that the interrupt enable, clear, and status bits function for each interrupt is illustrated in Figure 6-10 and described in the following subsections. The illustration shows the control for one IRQ bit. The remaining IRQ bits and FIQ bits are controlled in a similar way.

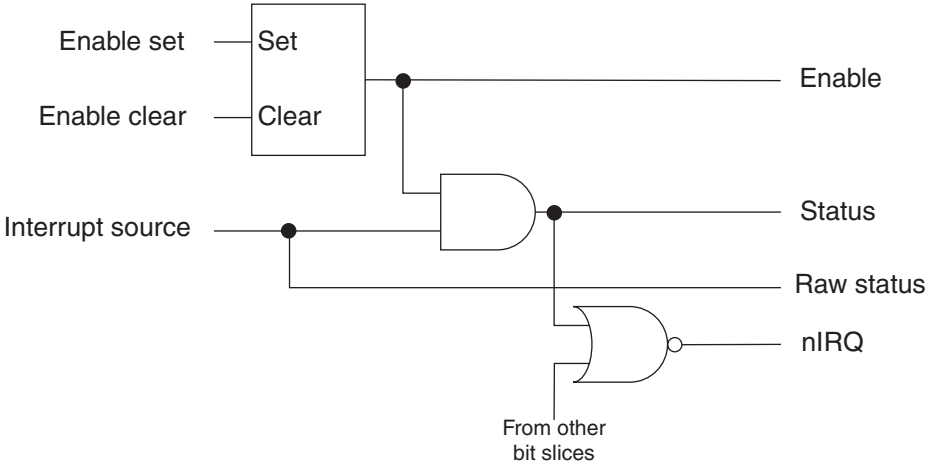


Figure 6-10 Interrupt control

IRQ/FIQ status register

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

IRQ/FIQ raw status register

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.

IRQ/FIQ enable set register

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

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

Read the current state of the enable bits from the ENSET location.

IRQ/FIQ enable clear register

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

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

Soft interrupts

The core module interrupt controller provides a register for controlling and clearing software interrupts. This register is accessed using the software interrupt set and software interrupt clear locations. The set and clear locations are used as follows:

- Set the software interrupt by writing to the CM_SOFT_INTSET location:

  write a 1 to SET the software interrupt

  write a 0 to leave the software interrupt unchanged.

- Read the current state of the of the software interrupt register from the CM_SOFT_INTSET location. A bit set to 1 indicates that the corresponding interrupt request is active.

- Clear the software interrupt by writing to the CM_SOFT_INTCLR location:

  write a 1 to CLEAR the software interrupt.

  write a 0 to leave the software interrupt unchanged.

The bit assignment for the software interrupt register is shown in Table 6-14.

Table 6-14 IRQ register bit assignment

Bit	Name	Function
31:1	Reserved	Write as 0. Reads undefined.
0	SOFT	Software interrupt.

Note

The *software interrupt* described in this section is used by software to generate IRQs or FIQs. It should not be confused with the ARM SWI software interrupt instruction. See the *ARM Architecture Reference Manual*.

6.6 CM system bus control

The system controller FPGA on the Integrator/AP normally provides control for the system bus. This includes bus arbitration and address decoding down to module level.

The core module and other modules added to the system must each provide address decoding for their assigned slave address in the memory map. The area of the memory map to which a module must respond is defined by its position in the stack, see *Module ID selection* on page 4-15.

When the core module is used without a motherboard, a default AHB peripheral in the PLD ensures that all accesses to nonexistant AHB locations generate an appropriate bus response.

If the core module is used with other modules but without a motherboard, then the core module or one of the other modules must provide the system control functions.

Chapter 7
CP Image for Operation with an Integrator/CP Baseboard

This chapter describes the system hardware. It contains the following sections:

7.1 About the CP PLD image

The CP PLD image implements the functional blocks that enables you to use the core with an Integrator/CP baseboard. This section provides the following information:

- *Selecting the CP image on a core module mounted on an Integrator/CP*
- *CP image functional block diagram* on page 7-3
- *The functional block HDL files* on page 7-4.

7.1.1 Selecting the CP image on a core module mounted on an Integrator/CP

To select the CP image on a core module mounted on an Integrator/CP, set the DIP switches as follows:

- S1[1] = x (x= *don't care*)
- S1[2] = x
- S1[3] = OFF
- S1[4] = OFF.

The baseboard drives the signals **CFGSEL[1:0]** correctly to select the correct image, see *PLD image selection* on page 4-8.

This image is signified by the code `0b00110011` being displayed by the user LEDs (0=OFF, 1=ON) when the system is powered ON.

Note

The bit file for the CP image is supplied on the CD for the CM922T-XA10 and is programmed into the image flash during board manufacture and test. However, the source files for the image are supplied with the CP baseboard.

Reprogramming of the core module must be performed standalone (that is, without a baseboard attached). Reprogramming requires code to be run on the processor. If a baseboard is attached, however, the processor is held in reset until configuration is complete.

7.1.2 CP image functional block diagram

The CP image contains the functional blocks shown in Figure 7-1.

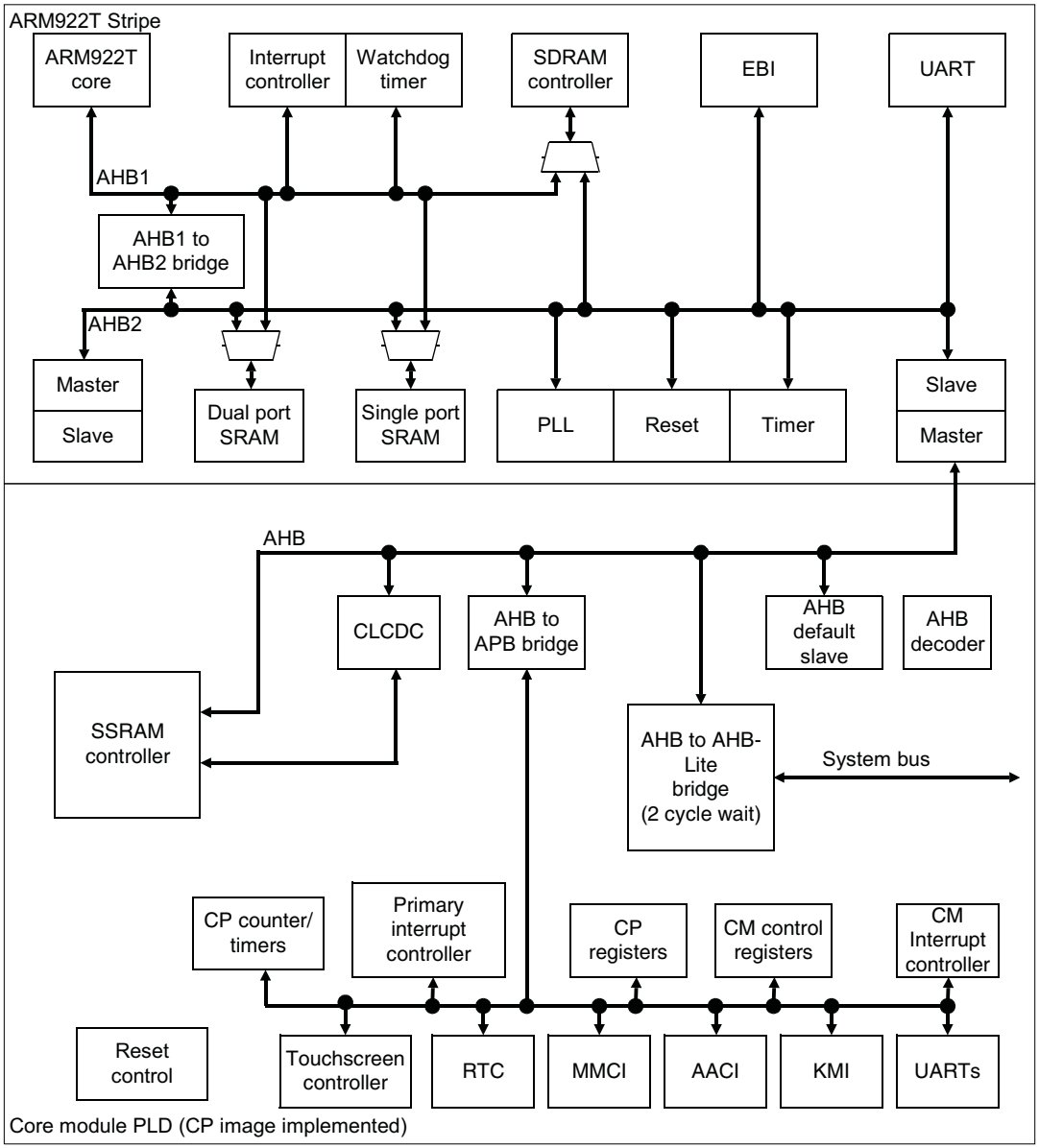


Figure 7-1 CP image architecture

7.1.3 The functional block HDL files

The CP image provides the functional blocks as a set of HDL files. These are described in Table 7-1.

Table 7-1 CM image functional block HDL file descriptions

Block	HDL file	Description
Top level	`Cp922fpga.vhd`	This is the top level file for this image.
AHB decoder	`AhbArmDecoder.vhd`	The decoder block provides the high-speed peripherals with select lines. These are generated from the address lines and the module ID (position in stack) signals from the motherboard. The decoder blocks also contain the default slave peripheral. The Integrator family of boards uses a distributed address decoding system.
AHB multiplexor	`CpAhbMux6StoM.vhd`	This is the AHB multiplexor that connects the read data buses from all of the slaves to the AHB master(s).
AHB to APB bridge	`AhbtoApb.vhd`	This is the bridge block required to connect APB peripherals to the high-speed AMBA AHB bus. It produce the peripheral select signals for each of the APB peripherals.
AHB default slave	`AhbDefaultSlave.vhd`	This is a default slave that correctly generates Error responses to accesses to unused address space.
Stripe to PLD	`arm_excal.vhd`	This is a wrapper for the Excalibur stripe.
APB multiplexor	`ApbMux16.vhd`	This block multiplexes the APB bus to the APB slaves.
CM registers	`cmxa10Regs.vhd`	The APB register peripheral provides memory-mapped status and control registers that you can use to: • identify the PLD image • configure the two clock generators (protected by the CM_LOCK register) • write to the user LEDs • read the user switch inputs • reset the system.
CP registers	`CpExcalregs.vhd`	This block contains additional registers used to control features of the CP baseboard.

Table 7-1 CM image functional block HDL file descriptions

Block	HDL file	Description
Reset controller	`CpExcalreset.vhd`	This block controls system reset. **Note** For the CP configuration, the reset button only resets the peripheral registers in the Excalibur PLD array. The processor is not reset. To reset the processor and reconfigure the PLD array, power cycle the board.
CM interrupt controller	`CMIntC.vhd`	The APB interrupt controller contains all of the standard interrupt controller registers.
Primary interrupt controller	`CpPrimaryIrqFiqCon.vhd`	This block is the primary interrupt controller for the CP image.
Timer	`Cnt1Hz.vhd`	This block provides a 1Hz clock signal used to generate the RTC enable signal.
ICS307 interface	`ics307.vhd`	These blocks are parallel to serial converters used to program the ICS307 devices from the register outputs.

7.2 Integrator/CP922T-XA10 system architecture

Figure 7-2 shows the architecture of the Integrator/CP922T-XA10.

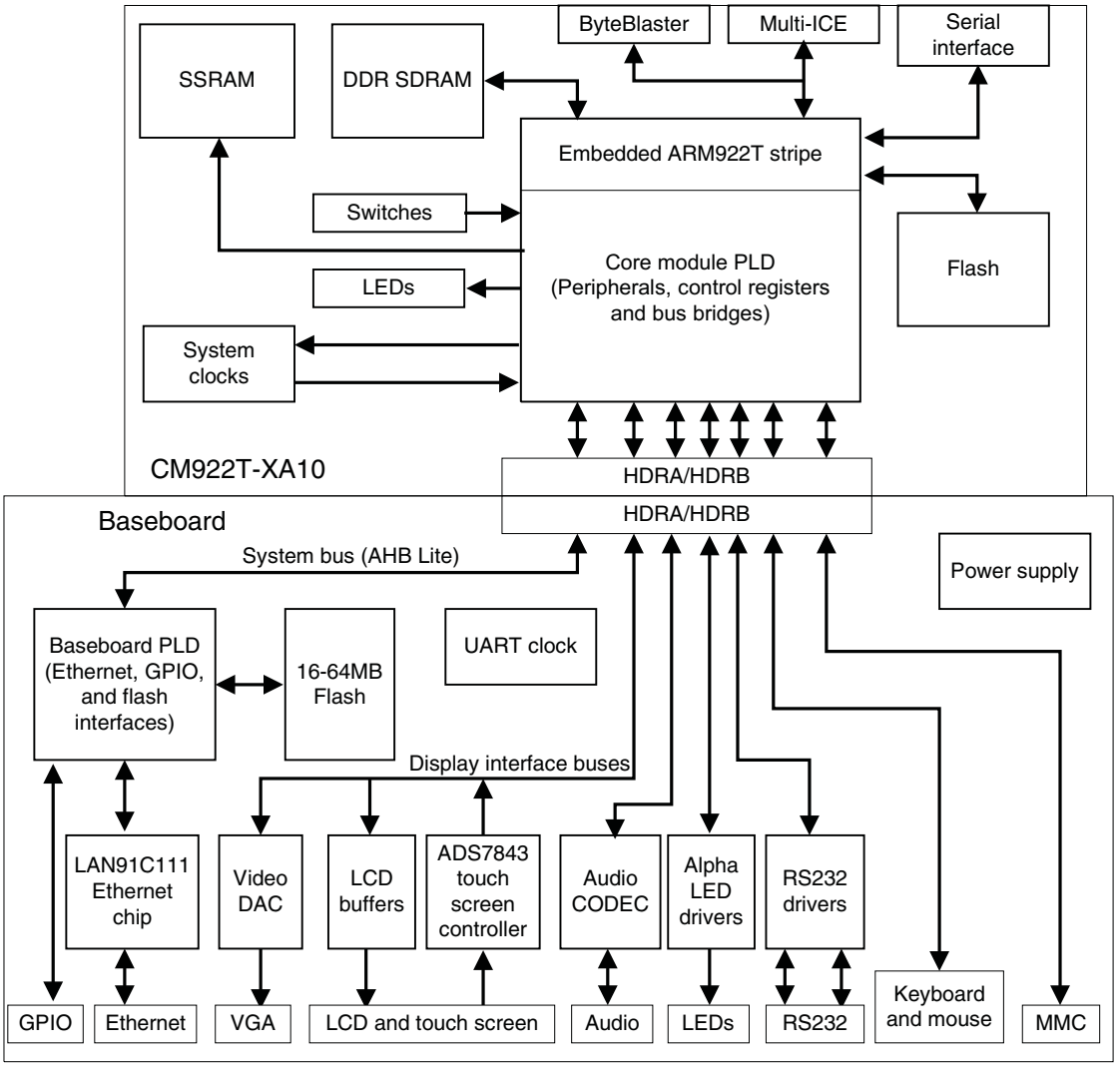


Figure 7-2 Integrator/CP922T-XA10 system architecture

Figure 7-2 shows a Integrator/CM922T-XA10 core module mounted onto an Integrator/CP baseboard. The combined system is given the name Integrator/CP922T-XA10 to signify that these two components are used together.

As well as the processor core in the PLD stripe, the core module provides the bus interfaces and peripheral interface controllers in the user programmable section of the PLD.

The baseboard provides an on-board power supply, interface connections, flash memory, an Ethernet controller, and a second PLD. This PLD provides:

- AHB slave interfaces for the Ethernet and flash
- the *Secondary Interrupt Controller* (SIC)
- an 8-bit GPIO interface.

The PLD is programmed during board manufacture and is not normally reprogrammed in the field. See the *Integrator/CP Baseboard User Guide* for more information.

7.3 Integrator/CP922T memory map

When the core module is powered up, the core module config flash in EBI0 is mapped to `0x0`. After the core has booted, SDRAM0 is mapped to appear at this location. The memory map for the CP image is shown in Table 7-2.

Table 7-2 Integrator/CP922T memory map

Address	Function
0x00000000-0x001FFFFF	Boot flash (EBI0). Mapped at this address only at power ON, and then disabled to allow access to SDRAM0.
0x00000000-0x03FFFFFF	On-board SDRAM0.
0x04000000-0x07FFFFFF	On-board SDRAM1.
0x08000000-0x0801FFFF	Embedded SRAM0.
0x08020000-0x0803FFFF	Embedded SRAM1.
0x08100000-0x0810FFFF	Embedded DPSRAM0.
0x08110000-0x0811FFFF	Embedded DPSRAM1.
0x0B000000-0x0B003FFF	Stripe registers.
0x0F000000-0x0F7FFFFF	User flash (EBI1).
0x0F800000-0x0FFFFFFF	User flash (EBI2).
0x10000000-0x100000FF	Core module registers.
0x10000100-0x107FFFFF	Reserved. Accesses result in undefined behavior.
0x10800000-0x10FFFFFF	SSRAM.
0x11000000-0x1FFFFFFF	CP APB peripherals.
0x20000000-0x27FFFFFF	System bus, baseboard flash memory.
0x28000000-0xBFFFFFFF	Default slave (bus error response).
0xC0000000-0xC0FFFFFF	Static
0xC1000000-0xC7FFFFFF	Static
0xC8000000-0xFFFFFFFF	POR

7.4 Integrator/CP922T peripherals and register memory map

Table 7-3 shows the register memory map for the Integrator/CP922T system and provides references to sections or other publications that contain more information about the registers.

Table 7-3 Integrator/CP922T system register map

Peripheral	Address range	Size	See
Core Module control registers	0x10000000 - 0x1000003F	64bytes	*CM control and status registers for the CP image* on page 7-11
CM interrupt controller	0x10000040 - 0x1000007F	64bytes	*CM interrupt controller* on page 7-35
Reserved	0x10000080 - 0x100001FF	128bytes	-
Counter/timers	0x13000000 - 0x13FFFFFF	16MB	The *Integrator/CP Baseboard User Guide.*
Primary interrupt controller registers	0x14000000 - 0x14FFFFFF	16MB	*Primary interrupt controller* on page 7-38
Real time clock	0x15000000 - 0x15FFFFFF	16MB	The *Integrator/CP Baseboard User Guide.*
UART0	0x16000000 - 0x16FFFFFF	16MB	The *Integrator/CP Baseboard User Guide.*
UART1	0x17000000 - 0x17FFFFFF	16MB	
Keyboard	0x18000000 - 0x18FFFFFF	16MB	The *Integrator/CP Baseboard User Guide.*
Mouse	0x19000000 - 0x19FFFFFF	16MB	
Debug LEDs and DIP switch	0x1A000000 - 0x1AFFFFFF	16MB	The *Integrator/CP Baseboard User Guide.*
Reserved	0x1B000000 - 0x1BFFFFFF	16MB	-
Multimedia Card Interface	0x1C000000 - 0x1CFFFFFF	16MB	The *Integrator/CP Baseboard User Guide.*
Advanced Audio CODEC Interface	0x1D000000 - 0x1DFFFFFF	16MB	The *Integrator/CP Baseboard User Guide.*
Touch Screen Controller Interface	0x1E000000 - 0x1EFFFFFF	16MB	The *Integrator/CP Baseboard User Guide.*
CLCD regs/palette	0xC0000000 - 0xC0FFFFFF	16MB	The *Integrator/CP Baseboard User Guide.*

Table 7-3 Integrator/CP922T system register map (continued)

Peripheral	Address range	Size	See
Ethernet	0xC8000000 - 0xC8FFFFFF	16MB	The *Integrator/CP Baseboard User Guide.*
GPIO	0xC9000000 - 0xC9FFFFFF	16MB	The *Integrator/CP Baseboard User Guide.*
Secondary interrupt controller	0xCA000000 - 0xCAFFFFFF	16MB	*Secondary interrupt controller* on page 7-40.
CP control registers	0xCB000000 - 0xCBFFFFFF	16MB	*CP baseboard registers* on page 7-21

Note

Device registers are usually mapped repeatedly to fill their assigned spaces. However, to ensure correct operation on future product versions, it is advisable to only access registers at their true addresses.

7.5 CM control and status registers for the CP image

The CP PLD image provides a set of status and control registers, interrupt registers and flag registers. These are located within the PLD0 memory map region. Table 7-4 lists the CM registers for the CP image.

Table 7-4 Core module status, control, and interrupt registers

Register Name	Address	Access	Reference or description
CM_ID	0x10000000	Read	*Core module ID register, CM_ID* on page 7-12
CM_PROC	0x10000004	Read	*Core module processor register, CM_PROC* on page 7-13
CM_OSC	0x10000008	Read/write	*Core module oscillator register for the CP image, CM_OSC* on page 7-31
CM_CTRL	0x1000000C	Read/write	*Core module control and status register for the CP image, CM_CTRL* on page 7-13
CM_STAT	0x10000010	Read	*Core module status register, CM_STAT* on page 7-16
CM_LOCK	0x10000014	Read/write	*Core module lock register, CM_LOCK* on page 7-17
CM_AUXOSC	0x1000001C	Read/write	*Core module auxiliary oscillator register for the CP image, CM_AUXOSC* on page 7-32
CM_SDRAM	0x10000020	Read/write	*SDRAM status and control register, CM_SDRAM* on page 7-18
-	0x10000024	-	Reserved.
CM_REFCNT	0x10000028	Read	*Core module reference clock cycle counter, CM_REFCNT* on page 7-19
-	0x1000002C	-	Reserved
CM_FLAGS	0x10000030	Read	*CM flag registers* on page 7-19
CM_FLAGSS	0x10000030	Write	
CM_FLAGSC	0x10000034	Write	
CM_NVFLAGS	0x10000038	Read	
CM_NVFLAGSS	0x10000038	Write	
CM_NVFLAGSC	0x1000003C	Write	

Table 7-4 Core module status, control, and interrupt registers (continued)

Register Name	Address	Access	Reference or description
CM_IRQ_STAT	0x10000040	Read	*CM interrupt controller* on page 7-35
CM_IRQ_RSTAT	0x10000044	Read	
CM_IRQ_ENSET	0x10000048	Read/write	
CM_IRQ_ENCLR	0x1000004C	Write	
CM_SOFT_INTSET	0x10000050	Read/write	
CM_SOFT_INTCLR	0x10000054	Write	
CM_FIQ_STAT	0x10000060	Read	
CM_FIQ_RSTAT	0x10000064	Read	
CM_FIQ_ENSET	0x10000068	Read/write	
CM_FIQ_ENCLR	0x1000006C	Write	
-	0x10000070	-	Reserved
-	0x10000080	-	Reserved
-	0x10000090	-	Reserved
-	0x10000080 - 0x107FFFFF	-	Reserved

Note

All registers are 32-bits wide and only support word-wide writes. Do not implement registers at locations marked reserved. Preserve the register bits marked as *reserved* in the following sections by using read-modify-write operations.

7.5.1 Core module ID register, CM_ID

The core module ID register at 0x10000000 is a read-only register that identifies the board manufacturer, board type, and revision.

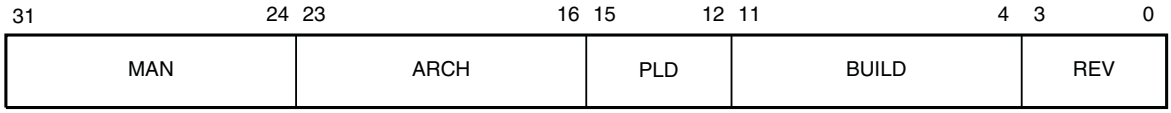


Figure 7-3 CM_ID register

Table 7-5 describes the core module ID register bits.

Table 7-5 CM_ID register bit descriptions

Bits	Name	Access	Function
[31:24]	MAN	Read	Manufacturer: 0x41 = ARM
[23:16]	ARCH	Read	Architecture: 0x0A = AHB interface, 32-bit SDRAM
[15:12]	PLD	Read	PLD type: 0x5 = XA10
[11:4]	BUILD	Read	Build value (ARM internal use)
[3:0]	REV	Read	Revision: 0x0 = Rev A 0x1 = Rev B

7.5.2 Core module processor register, CM_PROC

The core module processor register at 0x10000004 is a read-only register that contains the value 0x00000000. This is provided for compatibility with processors that do not have a system control coprocessor, CP15. For the ARM922T core, information about the processor can be obtained by reading coprocessor 15 register 0, CP15 c0.

7.5.3 Core module oscillator register, CM_OSC

This register is described in *Core module oscillator register for the CP image, CM_OSC* on page 7-31.

7.5.4 Core module control and status register for the CP image, CM_CTRL

The core module and LCD control register (CM_CTRL) at 0x1000000C is a read/write register that controls a number of user-configurable features of the core module and the display interface on the baseboard.

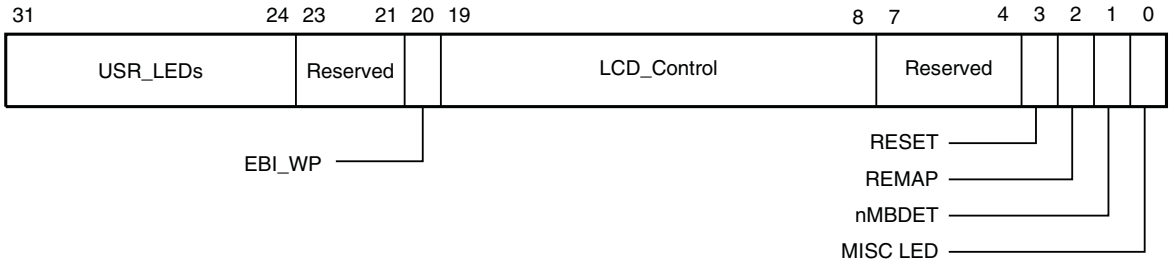


Figure 7-4 CM_CTRL register for the CP image

Table 7-6 describes the core module control register bits.

Table 7-6 CM_CTRL register

Bits	Name	Access	Function
[31:24]	USR_LEDs	Write	Writing a 1 to a bit in this register illuminates the associated LED.
[20]	EBI_WP	Write	This bit controls the write power signal to the core module user flash in EBI1 and EBI2. 0 = flash write protected 1= flash can be written to.
[19:8]	LCD_Control		
[19]	n24BITEN	Write	Select VGA depth: 0 = 24bit VGA 1 = 18bit VGA Must be 1 to enable BIAS control.
[18]	STATIC3	Write	No connection on Sharp panel.
[17]	STATIC2	Write	Up/down axis flip on Sharp panel.
[16]	STATIC1	Write	Right/left axis flip on Sharp panel.
[15]	Enable LCD1	Write	Enable, active high.
[14]	Enable LCD0	Write	Enable, active high.
[13:11]	LCDMUXSEL	Write	001 = Generic LCD connector, 24-bit mode 011 = Sharp LCD panel 100 = Sharp LCD panel 111 = 24-bit VGA.

Table 7-6 CM_CTRL register (continued)

Bits	Name	Access	Function
[10]	LCDBIASDN	Write	Low to high transition decreases LCD bias voltage (dimmer).
[9]	LCDBIASUP	Write	Low to high transition increases LCD bias voltage (brighter).
[8]	LCDBIASEN	Read/write	Enable LCD bias supply.
[7:4]	Reserved	Use read-modify-write to preserve value.	
[3]	RESET	Write	This is used to reset the core module, the baseboard on which it is mounted, and any modules in a stack. A reset is triggered by writing a 1. Reading this bit always returns a 0 allowing you to use read-modify-write operations without masking the RESET bit.
[2]	REMAP	Read/write	Remap is not supported by the Excalibur chip. This bit is hard coded to 1 for software compatibility.
[1]	nMBDET	Read	This bit indicates whether or not the core module is mounted on a baseboard: 0 = mounted on baseboard 1 = reserved.
[0]	LED	Read/write	This bit controls the green MISC LED on the core module: 0 = LED OFF 1 = LED ON.

7.5.5 Core module status register, CM_STAT

The core module status register at `0x10000010` is a read-only register that can be read to determine the size of the SSRAM present, the test chip type, and where in a stack this core module is positioned.

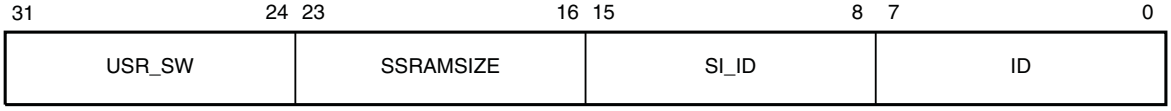


Figure 7-5 CM_STAT register

Table 7-7 describes the core module status register bits.

Table 7-7 CM_STAT register

Bit	Name	Access	Function	Default
[31:24]	USR_SW	Read	These are used to read the user switches. 1 = ON 0 = OFF.	-
[23:16]	SSRAMSIZE	Read	SSRAM size.	00000100
[15:8]	SI_ID	Read	Silicon manufacturer identification. Identifies the manufacturer and type of core fitted to the module.	00000000
[7:0]	ID	Read	Card number in stack: `0x00` = core module 0 `0x01` = core module 1 `0x02` = core module 2 `0x03` = core module 3 `0xFF` = invalid or no motherboard attached.	-

7.5.6 Core module lock register, CM_LOCK

The core module lock register at `0x10000014` is a read/write register that is used to control access to the CM_OSC register, allowing it to be locked and unlocked. This mechanism prevents this register from being overwritten accidently.

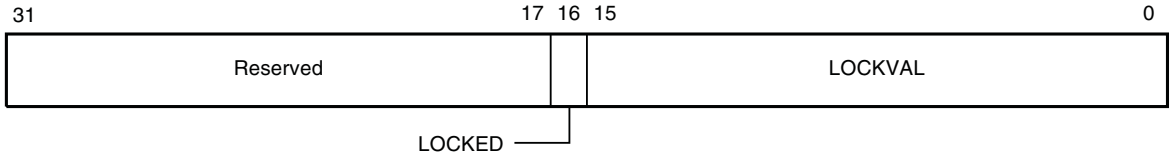


Figure 7-6 CM_LOCK register

Table 7-8 describes the core module lock register bits.

Table 7-8 CM_LOCK register

Bits	Name	Access	Function
[31:17]	Reserved	Use read-modify-write to preserve value.	
[16]	LOCKED	Read	This bit indicates if the CM_OSC register is locked or unlocked: 0 = unlocked 1 = locked.
[15:0]	LOCKVAL	Read/write	Write the value `0x0000A05F` to this register to enable write accesses to the CM_OSC register. Write any other value to this register to lock the CM_OSC register.

7.5.7 Core module auxiliary oscillator register, CM_AUXOSC

This register is described in *Core module auxiliary oscillator register for the CP image, CM_AUXOSC* on page 7-32.

7.5.8 SDRAM status and control register, CM_SDRAM

The SDRAM status and control register at `0x10000020` is a read-only register used to read the configuration parameters of an DDR SDRAM controlled by the stripe.

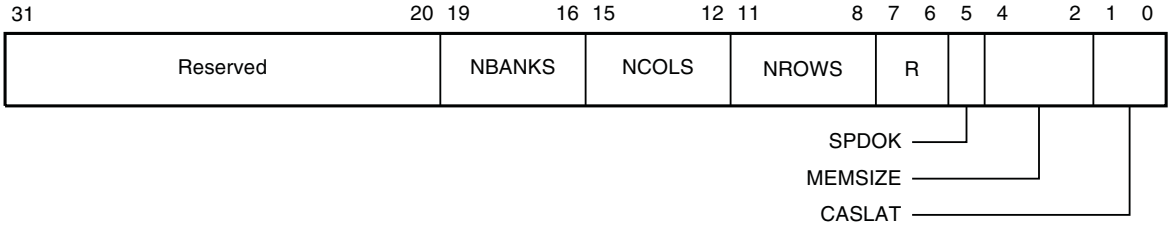


Figure 7-7 CM_SDRAM register

Table 7-9 describes the SDRAM status and control register bits.

Table 7-9 CM_SDRAM register

Bits	Name	Access	Function
[31:20]	Reserved	Use read-modify-write to preserve value.	
[19:16]	NBANKS	Read	Number of SDRAM banks.
[15:12]	NCOLS	Read	Number of SDRAM columns.
[11:8]	NROWS	Read	Number of SDRAM rows.
[7:6]	Reserved	Use read-modify-write to preserve value.	

Table 7-9 CM_SDRAM register (continued)

Bits	Name	Access	Function
[5]	SPDOK	Read	This bit is always 0.
[4:2]	MEMSIZE	Read	These bits specify the size of the SDRAM module fitted to the core module. The bits are encoded as follows: 000 = 16MB 001 = 32MB 010 = 64MB 011 = 128MB (default) 100 = 256MB 101 = Reserved 110 = Reserved 111 = Reserved.
[1:0]	CASLAT	Read	These bits specify the CAS latency for the core module SDRAM. The bits are encoded as follows: 00 = DDR SDRAM (default) 01 = Reserved 10 = 2 cycles 11 = 3 cycles.

7.5.9 Core module reference clock cycle counter, CM_REFCNT

This register at `0x10000028` provides a 32-bit count value. The count increments at the fixed reference clock frequency and can be used as a real-time counter. The register is reset to zero by a reset.

7.5.10 CM flag registers

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

The core module provides two distinct types of flag registers:

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

Flag/nonvolatile flag register, CM_FLAGS/CM_NVFLAGS

The status register contains the current state of the flags.

Flag/nonvolatile flag set register, CM_FLAGSS/CM_NVFLAGSS

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/nonvolatile flag clear register, CM_FLAGSC/CM_NVFLAGSC

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.

7.6 CP baseboard registers

The CP registers are instantiated into the PLD on the baseboard. These are listed in Table 7-10. These registers are described in the *Integrator/CP Baseboard User Guide*.

Table 7-10 CP registers

Register Name	Address	Access	Reset	Function
CP_IDFIELD	0xCB000000	Read	Static	CP build information register
CP_FLASHPROG	0xCB000004	Read	Static	Baseboard flash status and programming control register
CP_INTREG	0xCB000008	Read/write	POR	Secondary interrupt control register
CP_DECODE	0xCB00000C	Read/write	Reset	Fitted logic modules register

7.7 Integrator/CP922T system buses

Figure 7-8 shows how the external system bus connects the various boards together. This is described in the following subsections:

- *System bus routing and bus interfaces* on page 7-23.
- *APB peripheral bus* on page 7-23.

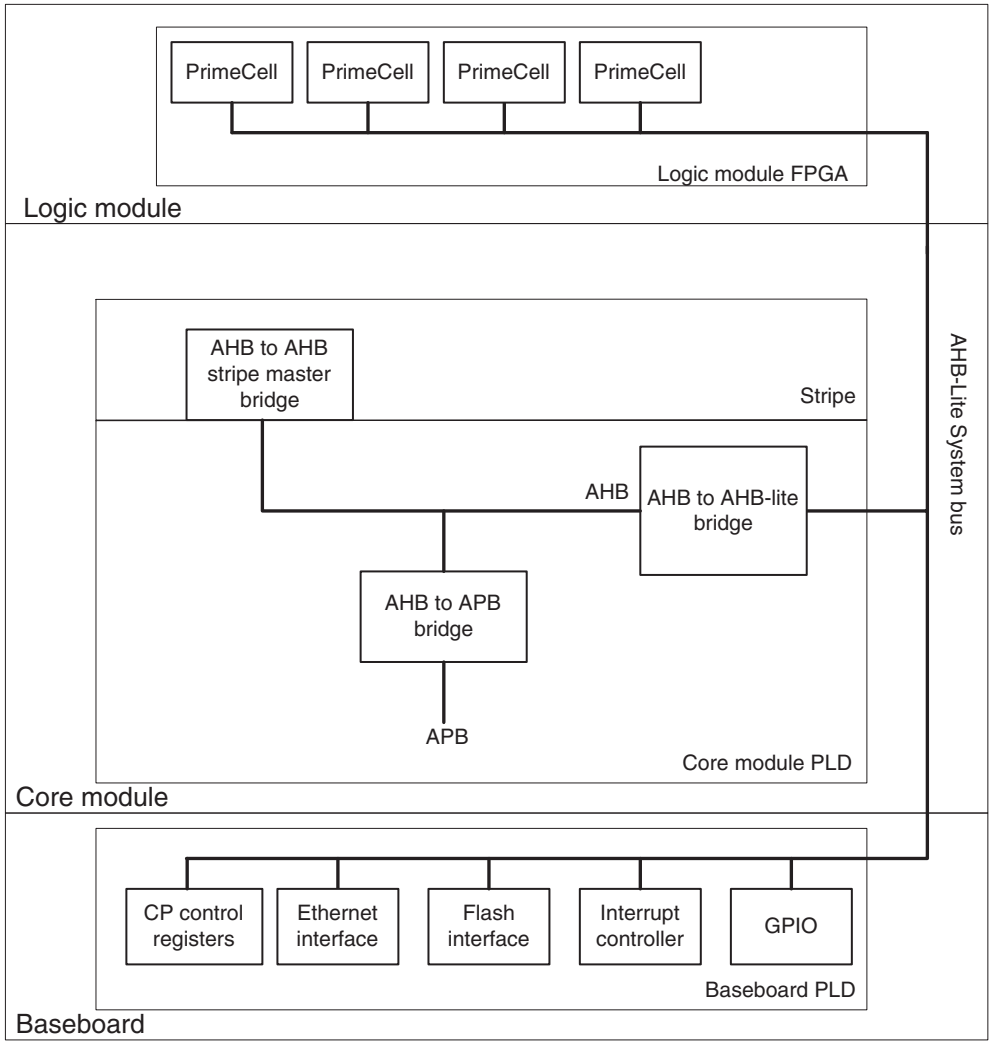


Figure 7-8 Bus routing

7.7.1 System bus routing and bus interfaces

The Integrator/CP uses an AMBA system bus. This comprises a number of segments:

AHB-Lite system bus

This is the AHB-Lite bus present on the HDRA and HDRB connectors and connects the baseboard, core module, and optional logic modules. The AHB-Lite system bus is a single-master bus with the ARM922T core as sole bus master.

AHB system bus

This is the AHB bus present within the PLD and connects the stripe AHB bus bridges bus some internal interface controllers, to the external system bus, and peripheral bus.

Stripe AHB system buses

The AHB bus within the stripe contains has two segments. They connect the between the core and other components within the stripe to the AHB bus bridges. See *About the Altera Excalibur EPXA10* on page 4-4.

Peripheral bus

This is the APB bus present within the PLD and connects the system bus to the APB peripherals and control registers.

There are several bus bridges that connect the segments of the system bus together:

Stripe master and slave AHB bridges

These bridges connect the AHB system bus within the stripe to the AHB system bus within the PLD.

AHB/APB Between the AHB and APB peripherals within the PLD.

AHB/system bus

Between the AHB within the PLD and the AHB-Lite system bus that connects between Integrator modules using the HDRA and HDRB connectors.

7.7.2 APB peripheral bus

The APB is an AMBA-compliant bus optimized for minimum power and reduced interface complexity. It is used to interface peripherals, such as the UARTs and the *Keyboard and Mouse Interface* (KMI), that do not require the high performance of the AHB.

The AHB-APB bridge is an AHB slave that provides an interface between the high-speed AHB domain and the low-power APB domain. The APB is not pipelined. Wait states are added during transfers between the APB and AHB when the AHB is required to wait for the APB protocol.

7.8 Configuring little or big-endian operation

The Integrator/CP922T can be configured to operate as a big-endian or little-endian system. To change to big-endian operation, write to the appropriate register in CP15 on the ARM core.

There is a delay between changing the endian configuration of the core and the system functioning in the new endian mode. Changing endianess should only be done at the start of a debugging session. The change must have taken effect before you can perform any subword accesses.

7.9 Integrator/CP922T system memory

This section describes the memory present on the Integrator/CP922T-XA10. The memory on the core module is described in *Core module memory* on page 4-17, and the additional memory element are described in:

- *Physical location of memory chips*
- *Baseboard flash memory* on page 7-27
- *Logic module SSRAM* on page 7-27.

7.9.1 Physical location of memory chips

Figure 7-9 shows where the different memory types are physically located in the Integrator/CP-based system.

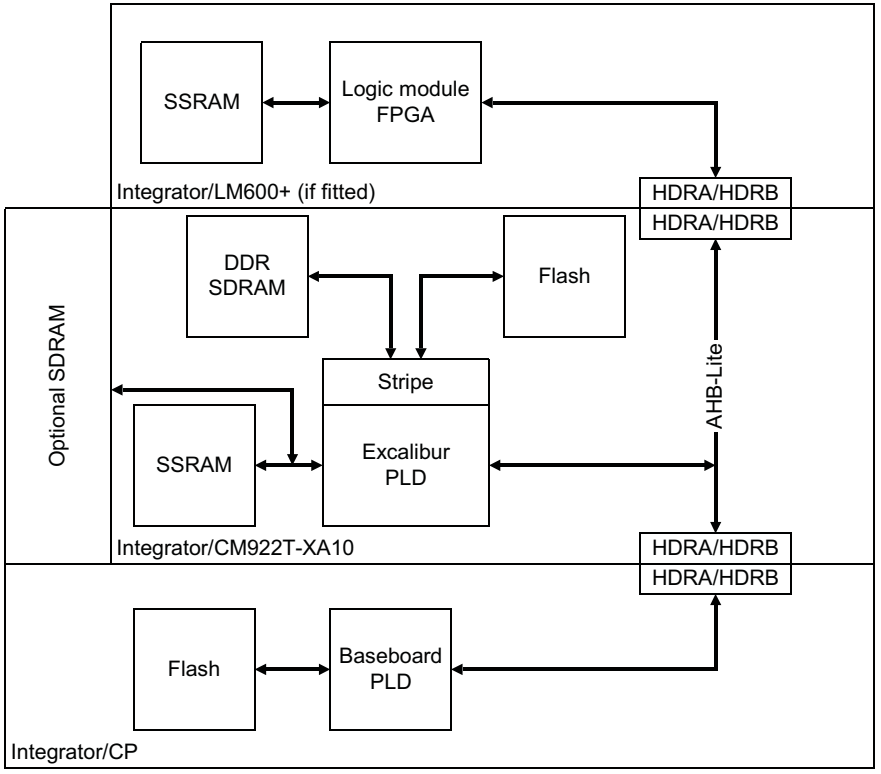


Figure 7-9 Physical memory locations

7.9.2 Baseboard flash memory

Note

The baseboard flash is factory programmed to contain boot code that maintains compatibility with a variety of Integrator systems but this is not used by the Integrator/CP922T. The CM922T-XA10 core module always boots from the flash on the core module itself (see *Flash memory* on page 4-17). You cannot use REMAP to boot from the baseboard flash.

If you want to use the baseboard flash during system power up, you can use a jump instruction in the core module boot flash in EB1 to jump to `0x24000000`.

The top 256KB of the baseboard flash memory is reserved for system boot code. If you want to retain this boot code for compatibility with other Integrator systems, use the remaining flash memory that is available for your own code requirements.

The address of the flash is `0x24000000` and the flash extends to the size of the device fitted. For details of the baseboard flash, see the *Integrator/CP Baseboard User Guide*.

7.9.3 Logic module SSRAM

The Integrator/LM600+ can be added to the Integrator/CP922T system if extra functionally is required. This logic module provides an area of SSRAM that requires a memory controller in the logic module FPGA. Also, because Integrator/CP uses a single-master AHB-Lite system bus, AHB devices instantiated into the logic module that connect to the system bus must be slaves.

7.10 Integrator/CP922T system clocks

Figure 7-10 shows the architecture of the Integrator/CP922T clock system.

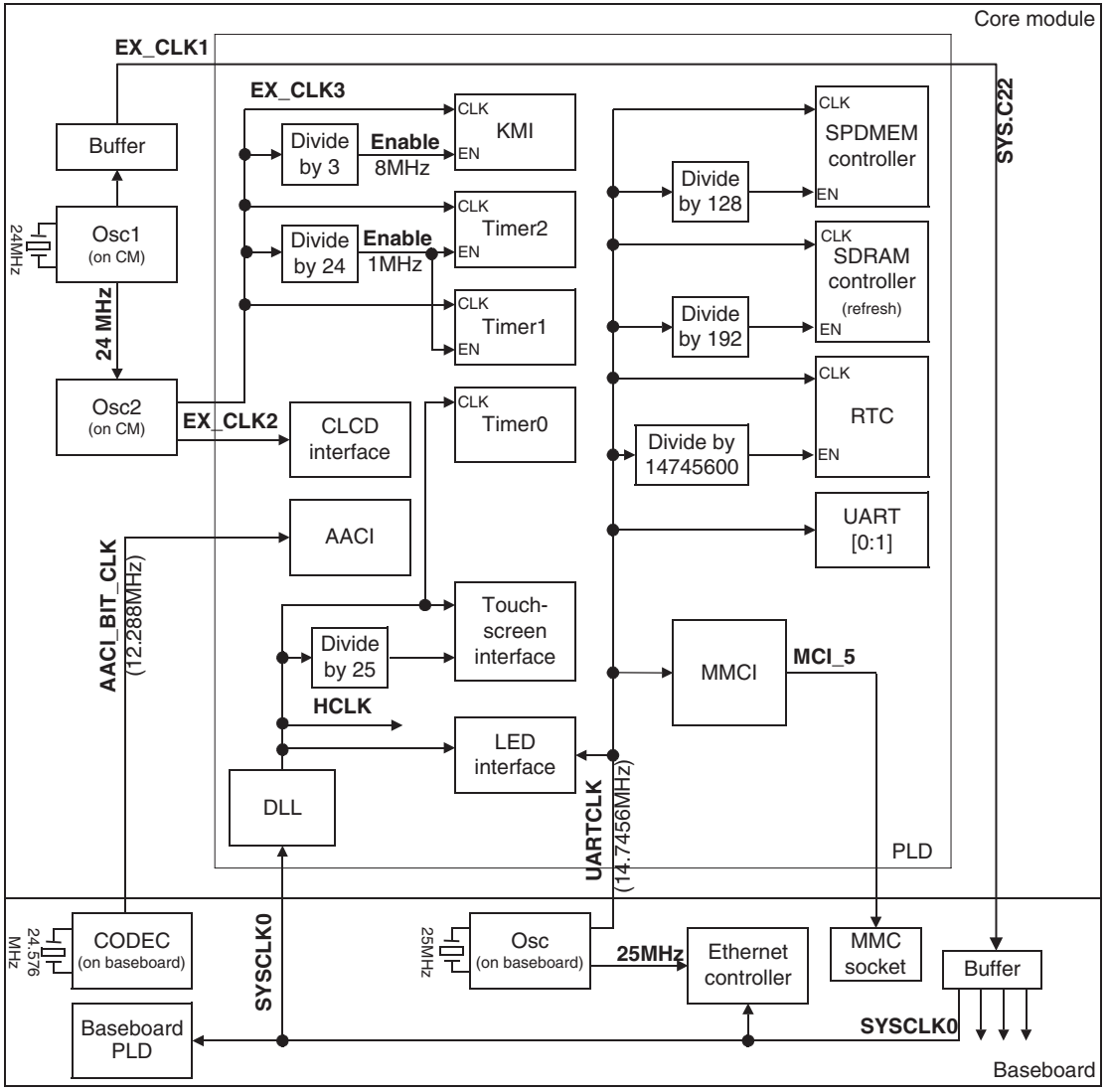


Figure 7-10 Integrator/CP922T clock architecture

The Integrator/CP922T uses the following clock sources:

SYSCLK[3:0]

These clocks are output from a buffer on the baseboard from a clock supplied by OSC1 (U13) on the core module. It is routed through the PLD down to the baseboard and then buffered to provide a high-speed clock signal **SYSCLK[0]** to the PLD. This is a programmable clock generated from **EX_CLK[1]**.

EX_CLK[3:2]

These clocks are output by OSC2 (U15) on the core module. **EX_CLK[2]** provides the clock source for the color LCD controller in the PLD. **EX_CLK[3]** provides the clock source for APB peripherals in the PLD. These are the two outputs from programmable clock generator U15.

UARTCLK

This clock is generated output by the oscillator on the baseboard. It provides the clock source for various controllers in the PLD. This clock has a fixed frequency of 14.7456MHz.

7.10.1 Calculating the output frequencies of the ICS307

The programmable clocks on the core module are supplied by two ICS307 clock generators. The CP image provides registers to control the following parameters of U15 (see *Clock control parameters* on page 4-26):

S[2:0] OD selection.

V[8:1] VDW. **V[0]** is always 0.

R[6:1] RDW. **R[0]** is always 0.

The general formula used to calculate the output clock frequency from an ICS307 is:

freq = 2*REFCLK*(VDW+ 8)/((RDW+2)*OD)

This formula is used for U15 but can be simplified for U13 because:

- REFCLK is supplied at a known frequency of 24MHz to the first clock generator.
- RDW has a fixed value of 6 for the first clock generator.
- OD has a fixed value of 6 for the first clock generator.

Table 7-11 shows the formulae you use to calculate the frequencies of the clocks, and their minimum and maximum values.

Table 7-11 Calculating the clock frequencies for the CP image

Chip	Clock output	Signal name	Formula	Max
U13	CLK1	**SYSCLK[3:0]**	freq = V+8	39MHz
		EX_CLK[1]		
		LA_CLK[1]		
	CLK2	-	24MHZ (fixed)	-
U15	CLK1	**IM_PLL1CLK[2:1]**	freq = 48*(VDW+ 8)/((RDW+2)*OD)	38MHz
		EX_CLK[3:2]		
		LA_CLK[2]		
	CLK2	**REF_CLK**	24MHz	-

Note

Table 7-11 shows the maximum operating frequencies for the core module. The clock generator outputs can typically be set to much higher values.

7.10.2 Core module oscillator register for the CP image, CM_OSC

The core module oscillator register at `0x10000008` is a read/write register that controls the output frequency of clock generator U13.

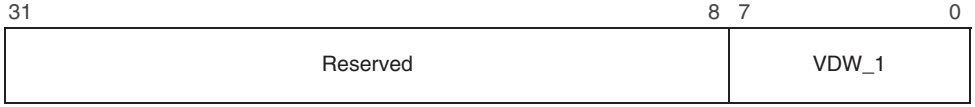


Figure 7-11 CM_OSC register for the CP image

Note

Before writing to the CM_OSC register, you must unlock it by writing the value `0x0000A05F` to the CM_LOCK register. After writing the CM_OSC register, relock it by writing any value other than `0x0000A05F` to the CM_LOCK register.

Table 7-12 describes the core module oscillator register bits.

Table 7-12 CM_OSC register for the CP image

Bits	Name	Access	Function
[31:8]	Reserved	Use read-modify-write to preserve value.	
[7:0]	VDW_1	Read/write	V[8:1] for U13. V[0] is always 0.

7.10.3 Core module auxiliary oscillator register for the CP image, CM_AUXOSC

The core module oscillator register at `0x1000001C` is a read/write register that controls the output frequency of clock generator U15.

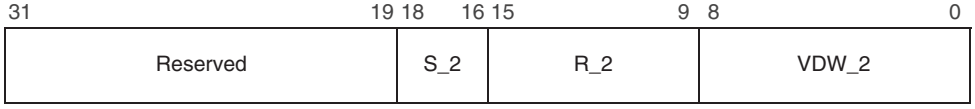


Figure 7-12 CM_AUXOSC register for the CP image

Note

Before writing to the CM_AUXOSC register, you must unlock it by writing the value `0x0000A05F` to the CM_LOCK register. After writing the CM_AUXOSC register, relock it by writing any value other than `0x0000A05F` to the CM_LOCK register.

Table 7-13 describes the auxiliary oscillator register bits.

Table 7-13 CM_AUXOSC register for the CP image

Bits	Name	Access	Function	Default
[31:19]	Reserved	Use read-modify-write to preserve value.		
[18:16]	OD_2	Read/write	Output divider select: 000 = divide by 10 001 = divide by 2 010 = divide by 8 011 = divide by 4 100 = divide by 5 101 = divide by 7 110 = divide by 3 111 = divide by 6.	011
[15:9]	RDW_2	Read/write	Reference divider word.	000001000
[8:0]	VDW_2	Read/write	VCO divider word.	0000110

7.11 Integrator/CP922T interrupt control

Figure 7-13 shows the interrupt control architecture for the Integrator/CP922T system. The CP image contains three interrupt controllers. Their use is described in:

- *CM interrupt controller* on page 7-35
- *Primary interrupt controller* on page 7-38
- *Secondary interrupt controller* on page 7-40
- *Interrupt routing between Integrator modules* on page 7-41
- *Handling interrupts* on page 7-42.

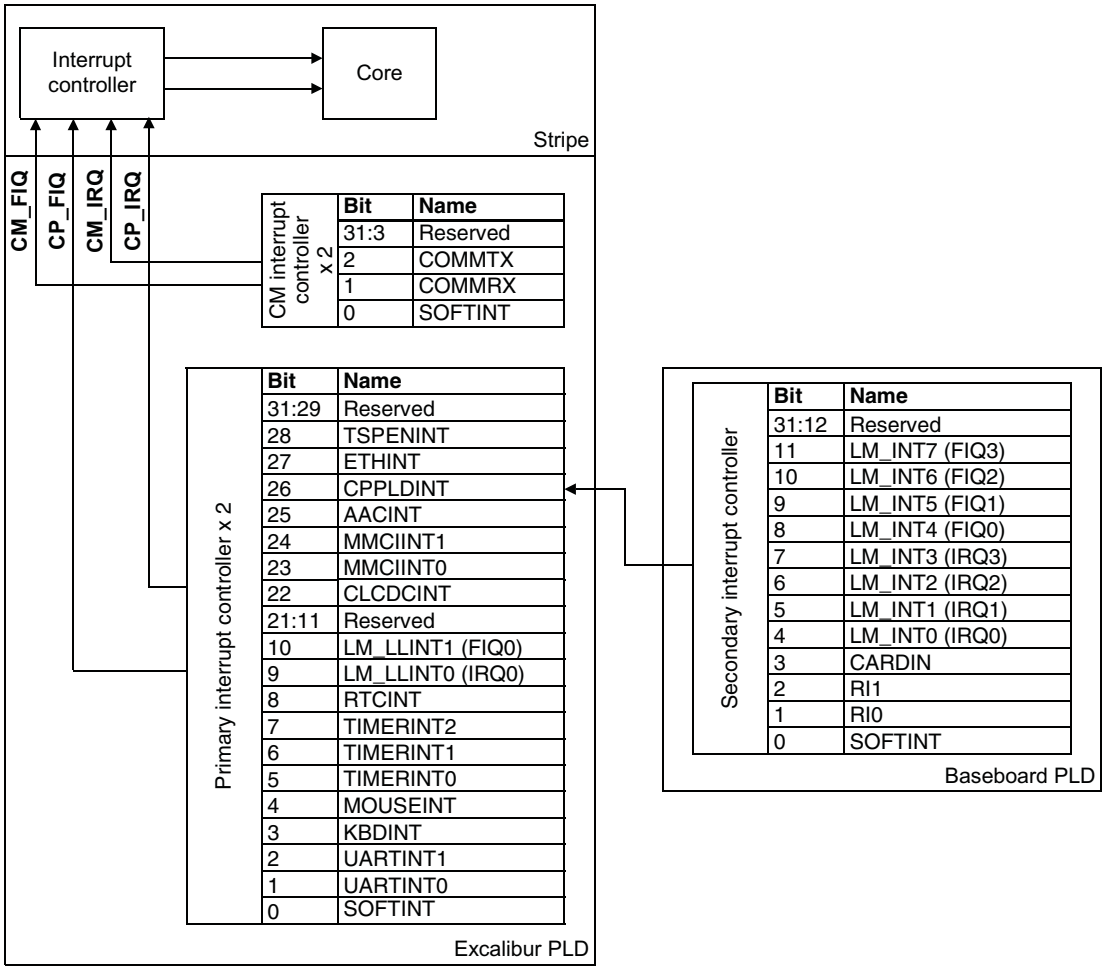


Figure 7-13 Interrupt architecture (CP image)

Integrator/CP system interrupts are generated by the *Primary Interrupt Controller* (PIC), the *Secondary Interrupt Controller* (SIC), and the *CM Interrupt Controller* (CIC). The PIC interrupts are connected to the interrupt controller within the stripe and are assigned to the **INT_PLD[3:0]** signals as shown in Table 7-14.

Table 7-14 CP image interrupt assignment

INT_PLD	CP
0	**CM_FIQ**
1	**CP_FIQ**
2	**CM_IRQ**
3	**CP_IRQ**
4	-
5	-

Detecting and clearing interrupts requires that each interrupt controller be correctly initialized as well as the interrupt control register in the individual peripheral.

7.11.1 CM interrupt controller

The core module provides a 3-bit IRQ controller and 3-bit FIQ controller to support:

- *Comms interrupts*
- *Soft interrupts* on page 7-37.

Comms interrupts

The core module provides a 3-bit IRQ controller and 3-bit FIQ controller to support the debug communications channel used for passing information between applications software and the debugger. The interrupt control registers are listed in Table 7-15.

Table 7-15 Comms interrupt controller registers

Register Name	Address	Access	Description
CM_IRQ_STAT	0x10000040	Read	Core module IRQ status register
CM_IRQ_RSTAT	0x10000044	Read	Core module IRQ raw status register
CM_IRQ_ENSET	0x10000048	Read/write	Core module IRQ enable set register
CM_IRQ_ENCLR	0x1000004C	Write	Core module IRQ enable clear register
CM_SOFT_INTSET	0x10000050	Read/write	Core module software interrupt set
CM_SOFT_INTCLR	0x10000054	Write	Core module software interrupt clear
CM_FIQ_STAT	0x10000060	Read	Core module FIQ status register
CM_FIQ_RSTAT	0x10000064	Read	Core module FIQ raw status register
CM_FIQ_ENSET	0x10000068	Read/write	Core module FIQ enable set register
CM_FIQ_ENCLR	0x1000006C	Write	Core module FIQ enable clear register

Note

All registers are 32-bits wide and only support word-wide writes. Bits marked as *reserved* in the following sections should be preserved using read-modify-write operations.

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 bit assignments for the IRQ and FIQ status, raw status and enable register are shown in Table 7-16.

Table 7-16 IRQ and FIQ register bit assignment

Bit	Name	Function
31:3	Reserved	Write as 0. Reads undefined.
2	COMMTx	Debug communications transmit interrupt. This interrupt indicates that the communications channel is available for the processor to pass messages to the debugger.
1	COMMRx	Debug communications receive interrupt. This interrupt indicates to the processor that messages are available for the processor to read.
0	SOFT	Software interrupt

The way that the interrupt enable, clear, and status bits function for each interrupt is illustrated in Figure 7-14 and described in the following subsections. The illustration shows the control for one IRQ bit. The remaining IRQ bits and FIQ bits are controlled in a similar way.

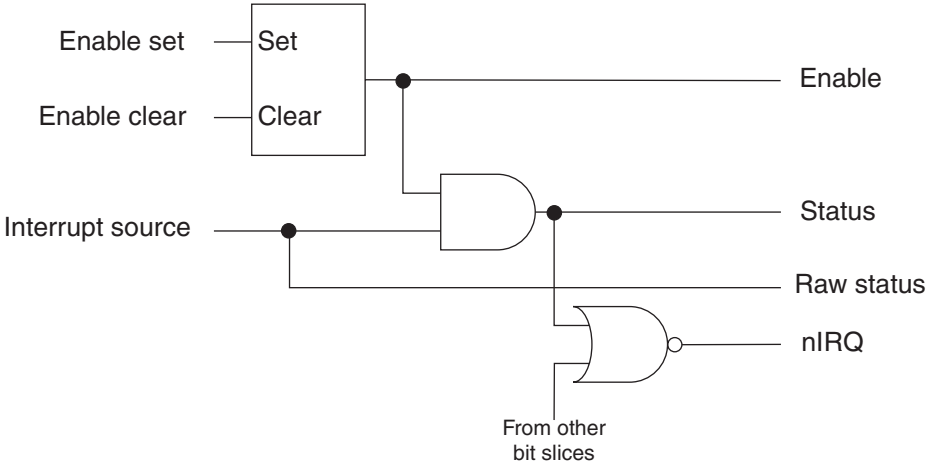


Figure 7-14 Interrupt control

IRQ/FIQ status register

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

IRQ/FIQ raw status register

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.

IRQ/FIQ enable set register

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

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

Read the current state of the enable bits from the ENSET location.

IRQ/FIQ enable clear register

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

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

Soft interrupts

The core module interrupt controller provides a register for controlling and clearing software interrupts. This register is accessed using the software interrupt set and software interrupt clear locations. The set and clear locations are used as follows:

- Set the software interrupt by writing to the CM_SOFT_INTSET location:

  write a 1 to SET the software interrupt

  write a 0 to leave the software interrupt unchanged.

- Read the current state of the of the software interrupt register from the CM_SOFT_INTSET location. A bit set to 1 indicates that the corresponding interrupt request is active.

- Clear the software interrupt by writing to the CM_SOFT_INTCLR location:

  write a 1 to CLEAR the software interrupt.

  write a 0 to leave the software interrupt unchanged.

The bit assignment for the software interrupt register is shown in Table 7-17.

Table 7-17 IRQ register bit assignment

Bit	Name	Function
31:1	Reserved	Write as 0. Reads undefined.
0	SOFT	Software interrupt.

Note

The *software interrupt* described in this section is used by software to generate IRQs or FIQs. It should not be confused with the ARM SWI software interrupt instruction. See the *ARM Architecture Reference Manual*.

7.11.2 Primary interrupt controller

The PIC is implemented within the core module PLD and handles the majority of interrupts from the system. A substantial number of interrupts are reserved to maintain compatibility with other modules within the Integrator family.

There are separate controllers for IRQ and FIQ, allowing any interrupt source to be assigned to either. The PIC provides a set of registers to control and handle interrupts. These are listed in Table 7-18.

Table 7-18 Primary interrupt registers

Address	Name	Type	Size	Function
0x14000000	PIC_IRQ_STATUS	Read	22	IRQ gated interrupt status
0x14000004	PIC_IRQ_RAWSTAT	Read	22	IRQ raw interrupt status
0x14000008	PIC_IRQ_ENABLESET	Read/write	22	IRQ enable set
0x1400000C	PIC_IRQ_ENABLECLR	Write	22	IRQ enable clear
0x14000010	PIC_INT_SOFTSET	Read/write	16	Software interrupt set
0x14000014	PIC_INT_SOFTCLR	Write	16	Software interrupt clear
0x14000020	PIC_FIQ_STATUS	Read	22	FIQ gated interrupt status

Table 7-18 Primary interrupt registers

Address	Name	Type	Size	Function
0x14000024	PIC_FIQ_RAWSTAT	Read	22	FIQ raw interrupt status
0x14000028	PIC_FIQ_ENABLESET	Read/write	22	FIQ enable set
0x1400002C	PIC_FIQ_ENABLECLR	Write	22	FIQ enable clear

The bit assignment for interrupts in the status, raw status, and enable registers for the IRQ and FIQ interrupt controllers is similar and is shown in Table 7-19.

Table 7-19 Primary interrupt register bit assignments

Bit	Name	Function
[31:29]	-	Reserved
[28]	TS_PENINT	Touchscreen pen-down event interrupt
[27]	ETH_INT	Ethernet interface interrupt
[26]	CPPLDINT	Interrupt from secondary interrupt controller, see *Secondary interrupt controller* on page 7-40
[25]	AACIINT	Audio interface interrupt
[24]	MMCIINT1	MultiMedia card interface
[23]	MMCIINT0	MultiMedia card interface
[22]	CLCDCINT	Display controller interrupt
[21:11]	-	Reserved
[10]	LM_LLINT1	Logic module low-latency interrupt 1
[9]	LM_LLINT0	Logic module low-latency interrupt 0
[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

Table 7-19 Primary interrupt register bit assignments (continued)

Bit	Name	Function
[2]	UARTINT1	UART 1 interrupt
[1]	UARTINT0	UART 0 interrupt
[0]	SOFTINT	Software interrupt

7.11.3 Secondary interrupt controller

The SIC is implemented in the baseboard PLD and combines interrupts from MMC socket, the UART ring indicator bits, installed logic modules, and a software generated interrupt to the CPPLDINT input of the PIC.

The MMC interrupt on the SIC is generated by the card insertion detect switch and is different from the MMCI interrupts in the PIC generated by the MMCI PrimeCell.

The secondary controller provides a set of registers to control and handle interrupts. These are described in the *Integrator/CP Baseboard User Guide*.

7.11.4 Interrupt routing between Integrator modules

Figure 7-15 shows how the IRQ signals are routed from logic modules down to the interrupt controllers.

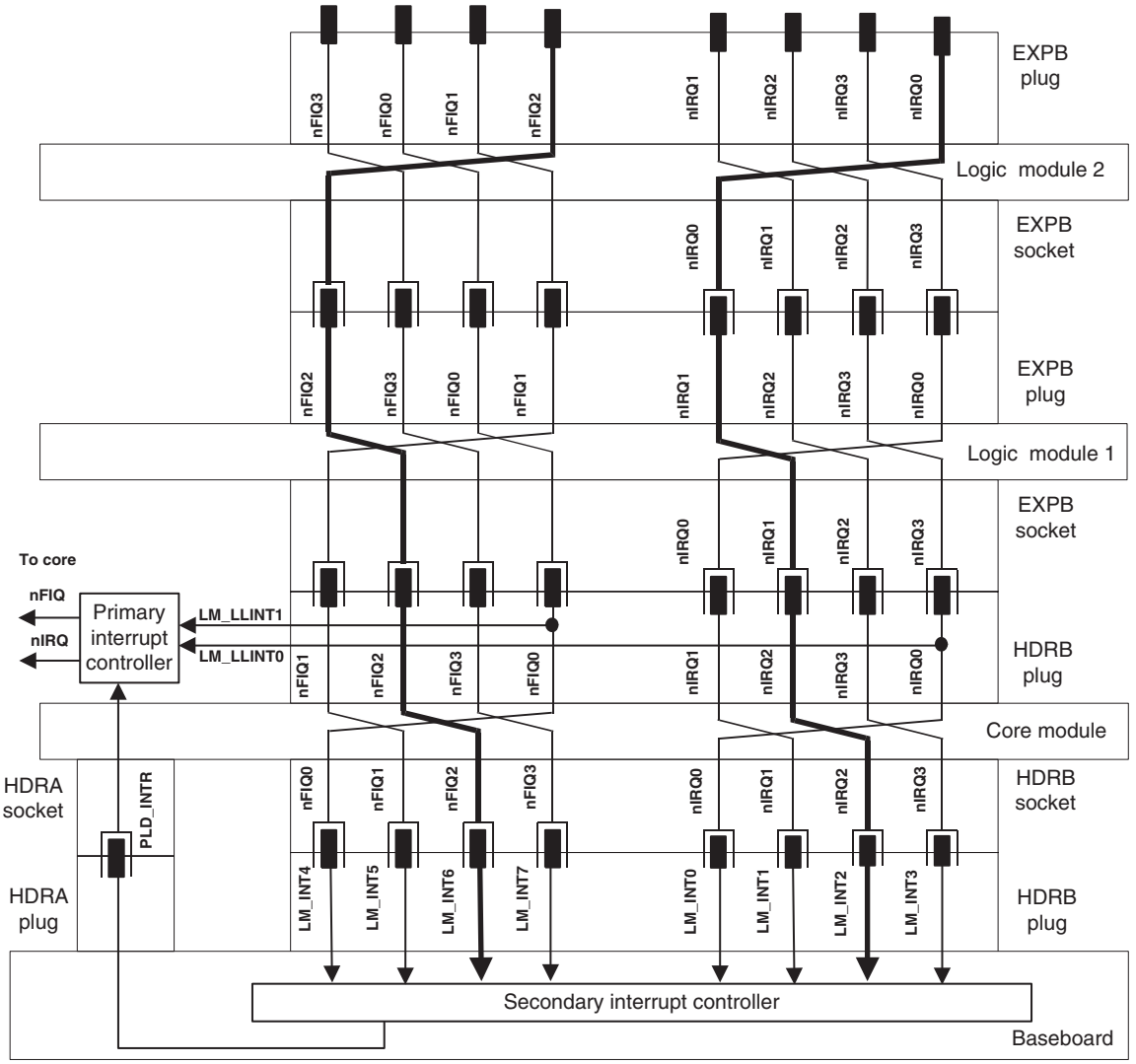


Figure 7-15 Interrupt signal routing

The diagram highlights how the signals are routed so that, for example, the interrupt request from logic module 2 connects to **LM_INT2** (**IRQ[2]**) on the baseboard. A similar routing applies for logic module 1 and 3. The operation of the interrupts relies on the core module being mounted on the baseboard first with any logic modules on top.

To simplify the description, this document refers to the logic modules source interrupts to the SIC as **LM_INT[7:0]**. The other signal names used in this diagram are those that appear on the schematic diagrams for the different modules. The signals names are not relevant to the final destination (IRQ or FIQ) of the signal in the ARM core.

For maximum flexibility, you can connect the eight logic module interrupt lines (**LM_INT[7:0]**) to the logic module interrupts as appropriate sources for your system. The SIC allows any of the interrupt sources **LM_INT[7:0]** to activate the **CPPLDINT** input on the PIC.

The signals route through the SIC. Determining the source of an interrupt requires interrogating first the primary and then the secondary controller. The time required for the extra instructions might cause a problem with interrupt latency in some situations. To improve latency, **FIQ[0]** and **IRQ[0]** (as **LM_LLINT[1:0]**) are also routed to the PIC as well.

Note

Ensure that the correct interrupt line is driven. See the routing pattern in Figure 7-15 on page 7-41.

Although the signal rotation scheme is identical, the logic module interrupt routing scheme used on the Integrator/CP is not the same as that used on the Integrator/AP. In an Integrator/AP system, the **nFIQ[3:0]** pins on a logic module are unused and each logic module typically outputs only one interrupt source on its **nIRQ[0]** line.

Modules produced for the Integrator/AP should work on the Integrator/CP without the need to re-assign the interrupt signal positions. Logic modules in positions 1, 2, and 3 appear on the Integrator/CP interrupt lines **LM_INT[1]** to **LM_INT[3]** and lines **LM_INT[7:4]** are not connected.

7.11.5 Handling interrupts

This section describes interrupt handling and clearing in general. For examples of interrupt detection and handling, see the *install_directory*\platform\software\selftest directory on the CD, the *ARM Firmware Suite User Guide*, the *ARM Developer Suite Developer Guide*, and the technical reference manual for your processor.

Enabling IRQ interrupts

The majority of peripheral interrupts are routed direct to the PIC. Each peripheral contains its own interrupt mask and clear registers. To enable interrupts, you must clear both the peripheral interrupt mask and the interrupt controller mask as well as clearing any previous interrupt flags:

1. Disable the primary interrupt by setting the appropriate bit in PIC_IRQ_ENCLR
2. Clear the peripheral interrupt by setting the appropriate bit in the peripheral interrupt clear register
3. Unmask the peripheral interrupt by clearing the appropriate bit in peripheral interrupt mask register
4. Enable the primary interrupt by setting the appropriate bit in PIC_IRQ_ENSET

The following C code stub demonstrates how the PIC UART0 CTS interrupt is cleared and re-enabled:

```
*PIC_IRQ_ENCLR    = PIC_UARTINT0;
*UART0_UARTICR    = UART_CTSINTR;
*UART0_UARTIMSC   &= ~UART_CTSINTR;
*PIC_IRQ_ENSET    |= PIC_UARTINT0;
```

The following C code stub demonstrates how the SIC MMCI CARDIN is cleared and re-enabled:

```
*PIC_IRQ_ENCLR   = PIC_CPPLDINT;
*CP_INTREG       = SIC_CARDIN;
*SIC_IRQ_ENSET   |= SIC_CARDIN;
*PIC_IRQ_ENSET   |= PIC_ CPPLDINT;
```

Note

The constants in these C code stubs must contain bit patterns necessary to set only the required interrupt mask bits. For example, `PIC_UARTINT0` must contain `0x02` to set only the UART0 bit in the PIC_IRQ_ENCLR register.

Determining and clearing IRQ interrupts

To determine an interrupt source, read the STATUS registers in the PIC and CIC to determine the interrupt controller that generated the interrupt. The sequence to determine and clear the interrupt is:

1. Determine the interrupt source by reading CIC_STATUS and PIC_STATUS.

The interrupt handler is directed by the status register information to the particular peripheral that generated the interrupt. In the case of SIC interrupts the interrupt handler must also read the SIC STATUS register to determine the interrupt source.

2. Determine the peripheral interrupt source by reading the peripheral masked interrupt status register.

3. Clear the peripheral interrupt by setting the appropriate bit in the peripheral interrupt clear register.

The following pseudo code example demonstrates how the UART0 CTS interrupt is detected:

```
If CIC_STATUS flags set,
      . . .  CIC interrupt handler

If PIC_STATUS flags set,
      . . . PIC interrupt handler
     If PIC_CPPLDINT set,
           . . .  SIC interrupt handler
     If PIC_UARTINT0 set,
           . . . UART0 interrupt handler
           If UART0_UARTMIS, UART_CTSINTR flag set,
                   . . . act on interrupt then clear flag with UARTICR
     . . . Test other PIC flags
```

Note

The PLD interrupts are routed through the stripe interrupt controller. This must also be set up correctly to ensure correct handling of interrupts. See the *Excalibur ARM-Based Embedded Processor PLD Hardware Reference Manual*.

Chapter 8
IM-PD1 Image for Operation with an Integrator/IM-PD1

This chapter describes the IM-PD1 in the baseboard PLD. It contains the following sections:

8.1 About the IM-PD1 image

The IM-PD1 PLD image implements the functional blocks that enables you to use the core with an Integrator/IM-PD1. This section provides the following information:

- *Selecting the IM-PD1 image*
- *IM-PD1 image functional block diagram* on page 8-3
- *The functional block HDL files* on page 8-4.

8.1.1 Selecting the IM-PD1 image

To select the IM-PD1 image on a core module with an Integrator/IM-PD1 fitted, set the DIP switches as follows:

- S1[1] = OFF
- S1[2] = ON
- S1[3] = ON
- S1[4] = OFF.

This image is signified by the code `0b11110000` being displayed by the user LEDs (0=OFF, 1=ON) when the system is powered ON.

Note

The bit file for the IM-PD1 image is supplied on the CD for the CM922T-XA10 and is programmed into the image flash during board manufacture and test. However, the source files for the image are supplied with the IM-PD1.

8.1.2 IM-PD1 image functional block diagram

The IM-PD1 image contains the functional blocks shown in Figure 8-1.

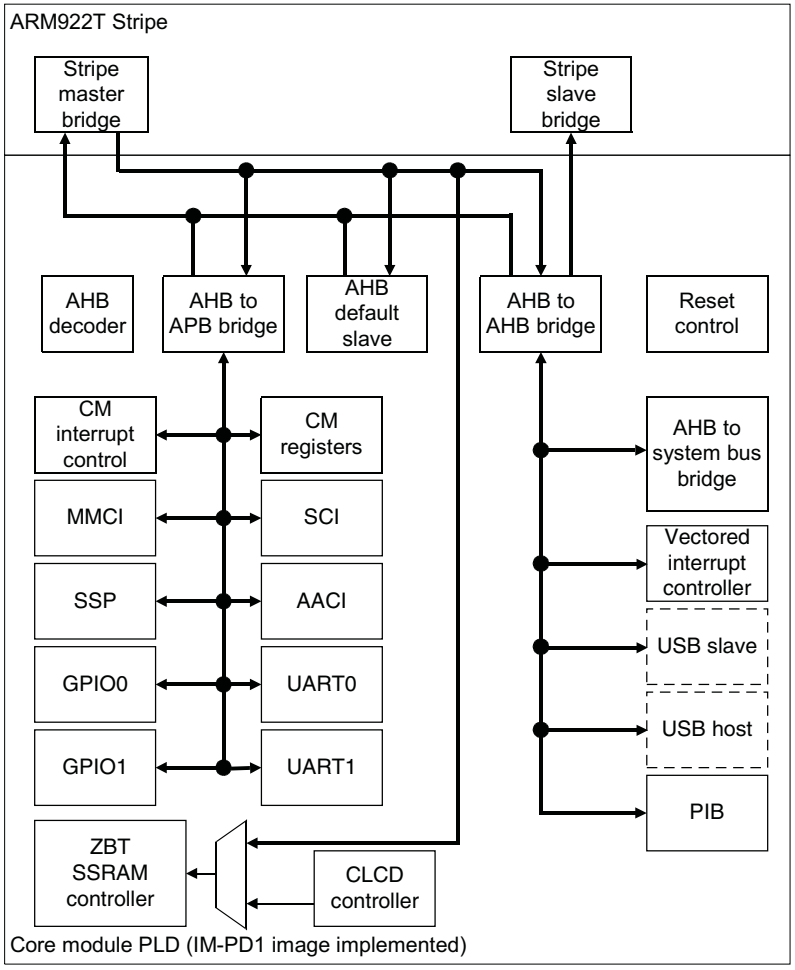


Figure 8-1 IM-PD1 image architecture

Universal serial bus

Connectors and circuitry are provided by the IM-PD1 for the USB logic blocks shown in Figure 8-1. However, there are no USB PrimeCell peripherals currently available from ARM Limited. These blocks can be licensed from other IP providers.

8.1.3 The functional block HDL files

The IM-PD1 image provides the functional blocks as a set of HDL files. These are described in Table 8-1.

Table 8-1 VHDL file descriptions

Block	File	Description
Top level	`imxa10fpga.vhd`	This file is the top-level VHDL that instantiates all of the PrimeCell peripherals for the image. The VHDL for the PrimeCell peripherals themselves are not supplied but are available from ARM as separate products.
AHB decoder	`AhbArmDecoder.vhd`	The decoder block provides the high-speed peripherals with select lines. These are generated from the address lines and the module ID (position in stack) signals from the motherboard. The Integrator family of boards uses a distributed address decoding system (see *Module ID selection* on page 4-15).
AHB default slave	`AHBDefaultSlave.vhd`	This block provides a default slave response when the address space is addressed but the address does not correspond to any of the instantiated slaves.
AHB multiplexor	`AhbMuxStoM.vhd`	This is the AHB multiplexor that connects the read data buses and the **HRESP** and **HREADY** signals from all of the slaves to the AHB master.
SSRAM controller	`AhbZbtRam.vhd`	This is the SSRAM controller block. It supports word, halfword and byte operations to the SSRAM on the core module.
AHB to APB bus bridge	`AhbtoApb.vhd`	This is the bridge block required to connect APB peripherals to the AMBA AHB bus. It produces the peripheral select signals for each of the APB peripherals.
Interrupt controller	`CMIntcon.vhd`	The APB interrupt controller contains all of the standard interrupt controller registers.

Table 8-1 VHDL file descriptions (continued)

Block	File	Description
CM registers	`cmxa10Regs`	The APB register peripheral provides memory-mapped status and control registers that you can use to: • identify the PLD image • configure the two clock generators (protected by the CM_LOCK register) • write to the user LEDs • read the user switch inputs • reset the system.
LM registers	`LMRegs.vhd`	The APB register peripheral provides memory-mapped registers. It also latches the pressing of the push button to generate an expansion interrupt.
-	`BuildOptions.vhd`	This file defines generation of the PrimeCell peripherals in the example and allows control over the synthesis so that PrimeCell peripherals can be included or excluded. It also specifies the base address of all the peripherals.

8.2 Integrator/CM922T-XA10 and IM-PD1 system architecture

Figure 8-2 shows the architecture of the Integrator/CM922T-XA10 and IM-PD1 system.

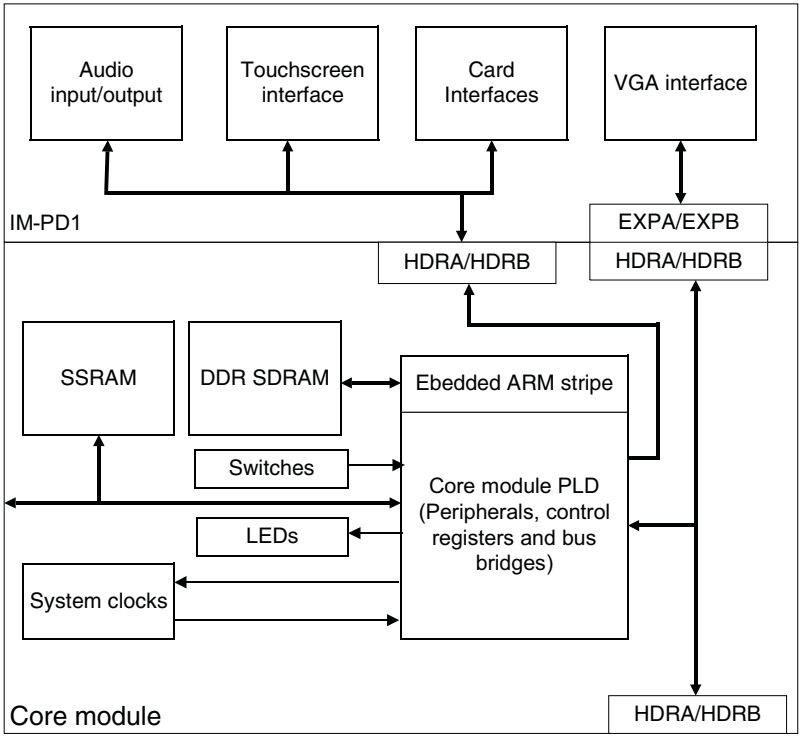


Figure 8-2 Integrator/CP922T-XA10 system architecture

Figure 8-2 shows an Integrator/CM922T-XA10 core module with an Integrator/IM-PD1 interface module mounted onto it.

As well as the processor core in the stripe, the core module provides the bus interfaces and peripheral interface controllers in the user-programmable section of the PLD.

8.3 Integrator/CM922T-XA10 and IM-PD1 memory map

When the core module is powered up, the core module flash is mapped to 0x0. After the core has booted, SDRAM0 is mapped to appear at this location. The memory map for the IM-PD1 image is shown in Table 8-2.

Table 8-2 Integrator/CM922T-XA10 and IM-PD1 memory map

Address	Function
0x00000000-0x001FFFFF	Boot flash (EBI0). Mapped at this address only at power ON, and then disabled to allow access to SDRAM0.
0x00000000-0x03FFFFFF	On-board SDRAM0.
0x04000000-0x07FFFFFF	On-board SDRAM1.
0x08000000-0x0801FFFF	Embedded SRAM0.
0x08020000-0x0803FFFF	Embedded SRAM1.
0x08100000-0x0810FFFF	Embedded DPSRAM0.
0x08110000-0x0811FFFF	Embedded DPSRAM1.
0x0B000000-0x0B003FFF	Stripe registers.
0x0F000000-0x0F7FFFFF	User flash (EBI1).
0x0F800000-0x0FFFFFFF	User flash (EBI2).
0x10000000-0x100000FF	Core module registers.
0x10000100-0x107FFFFF	Reserved. Accesses result in undefined behavior.
0x10800000-0x10FFFFFF	SSRAM
0x11000000-0x7FFFFFFF	System bus
0x80000000-0x8FFFFFFF	Default slave (bus error response).
0x90000000-0xBFFFFFF	System bus
0xC0000000-0xC0FFFFFF	IM-PD1 registers and peripherals
0xC1000000-0xC1FFFFFF	CLDC
0xC2000000-0xC2FFFFFF	SSRAM

Table 8-2 Integrator/CM922T-XA10 and IM-PD1 memory map (continued)

Address	Function
0xC3000000-0xC37FFFFF	VIC
0xCFC00000-0xCFFFFFFF	PIB
0xD0000000-0xFFFFFFFF	System bus

The PLD regions that are mapped to the system bus, can contain the resources within the PLD or on other Integrator modules. The default slave ensures that any accesses receive an appropriate response for any unoccupied locations.

8.4 Integrator/CM922T-XA10 and IM-PD1 register memory map

Table 8-3 shows the register memory map for the Integrator/CM922T-XA10 and IM-PD1 system and provides references to sections that contain more information about the registers.

Table 8-3 Integrator/CM922T-XA10 and IM-PD1 system register map

Peripheral	Address	See
Core Module control registers	0x10000000	*CM control and status registers for the IM-PD1 image* on page 8-10
Core Module interrupt controller	0x10000040	*CM interrupt controller* on page 8-22
IM-PD1 control registers	0xC0000000	*IM-PD1 registers* on page 8-29
UART0	0xC01000000	*UART interface* on page 8-32
UART1	0xC0200000	*IrDA interface* on page 8-32
SSP	0xC0300000	*Touchscreen controller* on page 8-33
GPIO0	0xC0400000	See *the Integrator/CP Baseboard User Guide*
GPIO1	0xC0500000	
SCI	0xC0600000	*Smart card interface* on page 8-32
MultiMedia Card Interface	0xC0700000	*MMC interface* on page 8-33
Advanced Audio CODEC Interface	0xC0800000	*Audio CODEC* on page 8-33
CLCD regs/palette	0xC1000000	*Display interface* on page 8-33
Vectored interrupt controller	0xC3000000	*Vectored interrupt controller* on page 8-25

Note

Device registers are usually mapped repeatedly to fill their assigned spaces. However, to ensure correct operation on future product versions, it is advisable to only access registers at their true addresses.

8.5 CM control and status registers for the IM-PD1 image

The CM PLD image provides a set of status and control registers, interrupt registers and flag registers. These are located within the PLD0 memory map region. Table 8-4 lists the CM registers.

Table 8-4 Core module status, control, and interrupt registers

Register Name	Address	Access	Description
CM_ID	0x10000000	Read	*Core module ID register, CM_ID* on page 8-11
CM_PROC	0x10000004	Read	*Core module processor register, CM_PROC* on page 8-12
CM_OSC	0x10000008	Read/write	*Core module oscillator register, CM_OSC* on page 8-12
CM_CTRL	0x1000000C	Read/write	*Core module control register, CM_CTRL* on page 8-12
CM_STAT	0x10000010	Read	*Core module status register, CM_STAT* on page 8-15
CM_LOCK	0x10000014	Read/write	*Core module lock register, CM_LOCK* on page 8-16
CM_COUNTER	0x10000018	Read	*Core module bus cycle counter, CM_COUNTER* on page 8-16
-	0x1000001C	-	Reserved
CM_SDRAM	0x10000020	Read/write	*SDRAM status and control register, CM_SDRAM* on page 8-17
-	0x10000024	-	Reserved.
CM_REFCNT	0x10000028	Read	*Core module reference clock cycle counter, CM_REFCNT* on page 8-18
-	0x1000002C	-	Reserved
CM_FLAGS	0x10000030	Read	*CM flag registers* on page 8-18
CM_FLAGSS	0x10000030	Write	
CM_FLAGSC	0x10000034	Write	
CM_NVFLAGS	0x10000038	Read	
CM_NVFLAGSS	0x10000038	Write	
CM_NVFLAGSC	0x1000003C	Write	

Table 8-4 Core module status, control, and interrupt registers (continued)

Register Name	Address	Access	Description
CM_IRQ_STAT	0x10000040	Read	*CM interrupt controller* on page 8-22
CM_IRQ_RSTAT	0x10000044	Read	
CM_IRQ_ENSET	0x10000048	Read/write	
CM_IRQ_ENCLR	0x1000004C	Write	
CM_SOFT_INTSET	0x10000050	Read/write	
CM_SOFT_INTCLR	0x10000054	Write	
CM_FIQ_STAT	0x10000060	Read	
CM_FIQ_RSTAT	0x10000064	Read	
CM_FIQ_ENSET	0x10000068	Read/write	
CM_FIQ_ENCLR	0x1000006C	Write	
-	0x10000070	-	Reserved
-	0x10000080	-	Reserved
-	0x10000090	-	Reserved
-	0x10000080 - 0x107FFFFF	-	Reserved

Note

All registers are 32-bits wide and only support word-wide writes. Do not implement registers at locations marked reserved. Preserve the register bits marked as *reserved* in the following sections by using read-modify-write operations.

8.5.1 Core module ID register, CM_ID

The core module ID register at 0x10000000 is a read-only register that identifies the board manufacturer, board type, and revision.

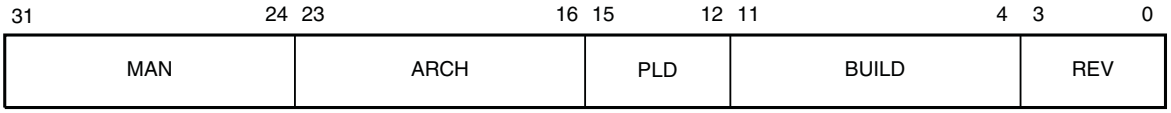


Figure 8-3 CM_ID register

Table 8-5 describes the core module ID register bits.

Table 8-5 CM_ID register bit descriptions

Bits	Name	Access	Function
[31:24]	MAN	Read	Manufacturer: `0x41` = ARM
[23:16]	ARCH	Read	Architecture: `0x0A` = AHB interface, 32-bit SDRAM
[15:12]	PLD	Read	PLD type: `0x5` = XA10
[11:4]	BUILD	Read	Build value (ARM internal use)
[3:0]	REV	Read	Revision: `0x0` = Rev A `0x1` = Rev B

8.5.2 Core module processor register, CM_PROC

The core module processor register at `0x10000004` is a read-only register that contains the value `0x00000000`. This is provided for compatibility with processors that do not have a system control coprocessor, CP15. For the ARM922T core, information about the processor can be obtained by reading coprocessor 15 register 0, CP15 c0.

8.5.3 Core module oscillator register, CM_OSC

This register is described in *Core Module oscillator register for the IM-PD1 image, CM_OSC* on page 8-27.

8.5.4 Core module control register, CM_CTRL

The core module control register at `0x1000000C` is a read/write register that provides control of a number of user-configurable features of the core module.

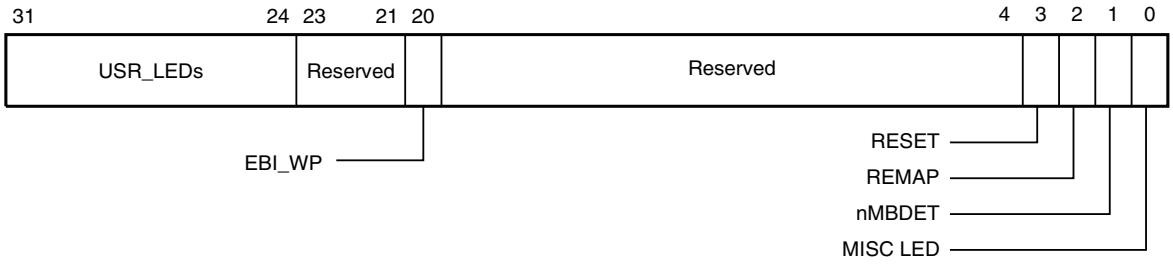


Figure 8-4 CM_CTRL register for the IM-PD1 image

Table 8-6 describes the core module control register bits

Table 8-6 CM_CTRL register

Bits	Name	Access	Function
[31:24]	USR_LED	Write	These bits are used to control the user LEDs: 0 = LED OFF 1 = LED ON.
[23:21]	Reserved	Use read-modify-write to preserve value.	
[20]	EBI_WP	Write	This bit controls user flash write protect signal **EBI_WP**: 0 = Flash write protected (default) 1 = Flash write enabled.
[19:4]	Reserved	Use read-modify-write to preserve value.	
[3]	RESET	Write	This is used to reset the core module, the motherboard on which it is mounted, and any modules in a stack. A reset is triggered by writing a 1. Reading this bit always returns a 0 allowing you to use read-modify-write operations without masking the RESET bit.

Table 8-6 CM_CTRL register (continued)

Bits	Name	Access	Function
[2]	REMAP	Read/write	Remap is not supported by the Excalibur. This bit contains 1 for software compatibility.
[1]	nMBDET	Read	This bit indicates whether or not the core module is mounted on a motherboard: 0 = mounted on motherboard 1 = standalone.
[0]	LED	Read/write	This bit controls the green MISC LED on the core module: 0 = LED OFF 1 = LED ON.

8.5.5 Core module status register, CM_STAT

The core module status register at `0x10000010` is a read-only register that can be read to determine the size of the SSRAM present, the test chip type, and where in a stack this core module is positioned.

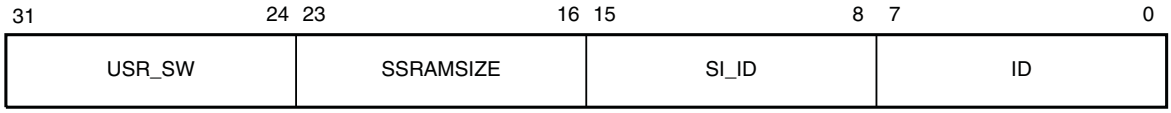


Figure 8-5 CM_STAT register

Table 8-7 describes the core module status register bits.

Table 8-7 CM_STAT register

Bit	Name	Access	Function	Default
[31:24]	USR_SW	Read	These are used to read the user switches. 1 = ON 0 = OFF.	-
[23:16]	SSRAMSIZE	Read	SSRAM size.	00000100
[15:8]	SI_ID	Read	Silicon manufacturer identification. Identifies the manufacturer and type of core fitted to the module.	00000000
[7:0]	ID	Read	Card number in stack: `0x00` = core module 0 `0x01` = core module 1 `0x02` = core module 2 `0x03` = core module 3 `0xFF` = invalid or no motherboard attached.	-

8.5.6 Core module lock register, CM_LOCK

The core module lock register at `0x10000014` is a read/write register that is used to control access to the CM_OSC register, allowing it to be locked and unlocked. This mechanism prevents this register from being overwritten accidently.

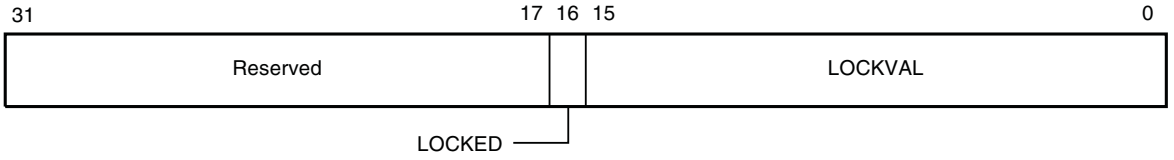


Figure 8-6 CM_LOCK register

Table 8-8 describes the core module lock register bits.

Table 8-8 CM_LOCK register

Bits	Name	Access	Function
[31:17]	Reserved	Use read-modify-write to preserve value.	
[16]	LOCKED	Read	This bit indicates if the CM_OSC register is locked or unlocked: 0 = unlocked 1 = locked.
[15:0]	LOCKVAL	Read/write	Write the value `0x0000A05F` to this register to enable write accesses to the CM_OSC register. Write any other value to this register to lock the CM_OSC register.

8.5.7 Core module bus cycle counter, CM_COUNTER

This register at `0x10000018` provides a 32-bit count value. The count increments at the **SYSCLK[0]** frequency and can be used as a cycle counter for performance measurement. The register is reset to zero by a reset.

8.5.8 SDRAM status and control register, CM_SDRAM

The SDRAM status and control register at `0x10000020` is a read-only register used to read the configuration parameters of an DDR SDRAM controlled by the stripe.

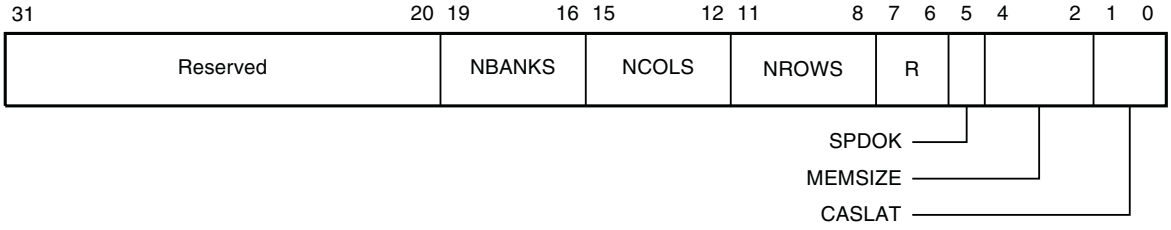


Figure 8-7 CM_SDRAM register

Table 8-9 describes the SDRAM status and control register bits.

Table 8-9 CM_SDRAM register

Bits	Name	Access	Function
[31:20]	Reserved	Use read-modify-write to preserve value.	
[19:16]	NBANKS	Read	Number of SDRAM banks.
[15:12]	NCOLS	Read	Number of SDRAM columns.
[11:8]	NROWS	Read	Number of SDRAM rows.
[7:6]	Reserved	Use read-modify-write to preserve value.	

Table 8-9 CM_SDRAM register (continued)

Bits	Name	Access	Function
[5]	SPDOK	Read	This bit is always 0.
[4:2]	MEMSIZE	Read	These bits specify the size of the SDRAM module fitted to the core module. The bits are encoded as follows: 000 = 16MB 001 = 32MB 010 = 64MB 011 = 128MB (default) 100 = 256MB 101 = Reserved 110 = Reserved 111 = Reserved.
[1:0]	CASLAT	Read	These bits specify the CAS latency for the core module SDRAM. The bits are encoded as follows: 00 = DDR SDRAM (default) 01 = Reserved 10 = 2 cycles 11 = 3 cycles.

8.5.9 Core module reference clock cycle counter, CM_REFCNT

This register at `0x10000028` provides a 32-bit count value. The count increments at the fixed reference clock frequency and can be used as a real-time counter. The register is reset to zero by a reset.

8.5.10 CM flag registers

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

The core module provides two distinct types of flag registers:

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

Flag/nonvolatile flag register, CM_FLAGS/CM_NVFLAGS

The status register contains the current state of the flags.

Flag/nonvolatile flag set register, CM_FLAGSS/CM_NVFLAGSS

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/nonvolatile flag clear register, CM_FLAGSC/CM_NVFLAGSC

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.

8.6 Integrator/CM922T-XA10 and IM-PD1 interrupt control

This section describes the interrupt control for the IM-PD1 image in:

- *Interrupt architecture*
- *CM interrupt controller* on page 8-22
- *Vectored interrupt controller* on page 8-25.

8.6.1 Interrupt architecture

Figure 8-8 shows the architecture of the interrupt control subsystem for the IM-PD1 image.

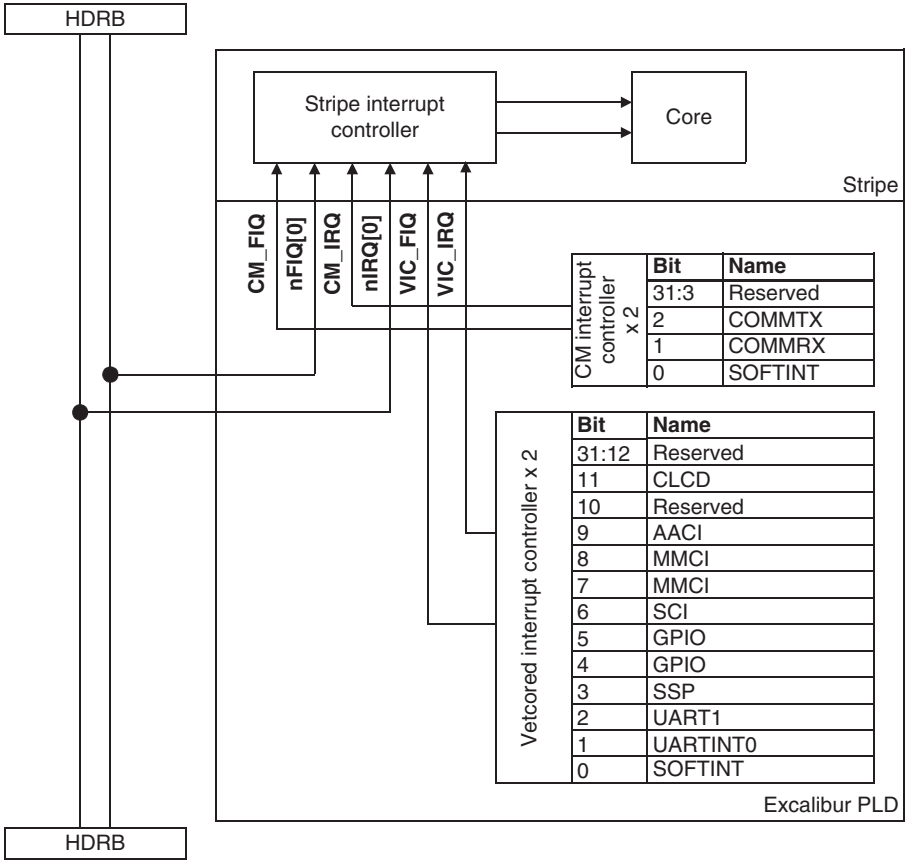


Figure 8-8 Interrupt architecture (IM-PD1 image)

The interrupt controllers provide separate register sets for IRQs and FIQs, as shown in Figure 8-8 on page 8-20. Interrupt requests from the controllers are assigned to the to the **PLD_INT[5:0]** signal inputs of the stripe interrupt controller as shown in Table 8-10.

Table 8-10 IM-PD1 image interrupt assignment

PLD_INT	IM-PD1
0	**CM_FIQ**
1	**nFIQ0**
2	**CM_IRQ**
3	**nIRQ0**
4	**VIC_FIQ**
5	**VIC_IRQ**

The architecture also allows for interrupt requests from another module in the system (if present) such as an Integrator/LM-EP20K600+ logic module.

Detecting and clearing interrupts requires that each interrupt controller be correctly initialized as well as the interrupt control register in the individual peripheral.

8.6.2 CM interrupt controller

The core module provides a 3-bit IRQ controller and 3-bit FIQ controller to support:

- *Comms interrupts*
- *Soft interrupts* on page 8-24.

Comms interrupts

The core module provides a 3-bit IRQ controller and 3-bit FIQ controller to support the debug communications channel used for passing information between applications software and the debugger. The interrupt control registers are listed in Table 8-11.

Table 8-11 Comms interrupt controller registers

Register Name	Address	Access	Description
CM_IRQ_STAT	0x10000040	Read	Core module IRQ status register
CM_IRQ_RSTAT	0x10000044	Read	Core module IRQ raw status register
CM_IRQ_ENSET	0x10000048	Read/write	Core module IRQ enable set register
CM_IRQ_ENCLR	0x1000004C	Write	Core module IRQ enable clear register
CM_SOFT_INTSET	0x10000050	Read/write	Core module software interrupt set
CM_SOFT_INTCLR	0x10000054	Write	Core module software interrupt clear
CM_FIQ_STAT	0x10000060	Read	Core module FIQ status register
CM_FIQ_RSTAT	0x10000064	Read	Core module FIQ raw status register
CM_FIQ_ENSET	0x10000068	Read/write	Core module FIQ enable set register
CM_FIQ_ENCLR	0x1000006C	Write	Core module FIQ enable clear register

Note

All registers are 32-bits wide and only support word-wide writes. Bits marked as *reserved* in the following sections should be preserved using read-modify-write operations.

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 bit assignments for the IRQ and FIQ status, raw status and enable register are shown in Table 8-12.

Table 8-12 IRQ and FIQ register bit assignment

Bit	Name	Function
31:3	Reserved	Write as 0. Reads undefined.
2	COMMTx	Debug communications transmit interrupt. This interrupt indicates that the communications channel is available for the processor to pass messages to the debugger.
1	COMMRx	Debug communications receive interrupt. This interrupt indicates to the processor that messages are available for the processor to read.
0	SOFT	Software interrupt

The way that the interrupt enable, clear, and status bits function for each interrupt is illustrated in Figure 8-9 and described in the following subsections. The illustration shows the control for one IRQ bit. The remaining IRQ bits and FIQ bits are controlled in a similar way.

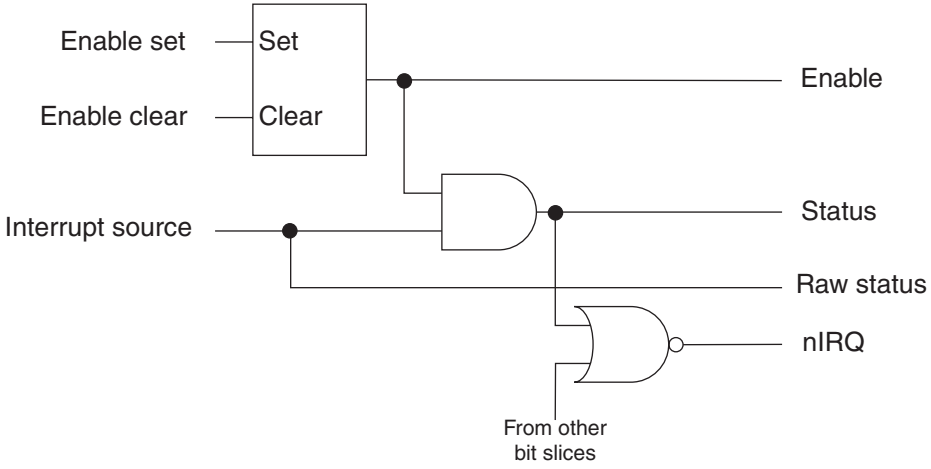


Figure 8-9 Interrupt control

IRQ/FIQ status register

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

IRQ/FIQ raw status register

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.

IRQ/FIQ enable set register

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

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

Read the current state of the enable bits from the ENSET location.

IRQ/FIQ enable clear register

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

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

Soft interrupts

The core module interrupt controller provides a register for controlling and clearing software interrupts. This register is accessed using the software interrupt set and software interrupt clear locations. The set and clear locations are used as follows:

- Set the software interrupt by writing to the CM_SOFT_INTSET location:

  write a 1 to SET the software interrupt

  write a 0 to leave the software interrupt unchanged.

- Read the current state of the of the software interrupt register from the CM_SOFT_INTSET location. A bit set to 1 indicates that the corresponding interrupt request is active.

- Clear the software interrupt by writing to the CM_SOFT_INTCLR location:

  write a 1 to CLEAR the software interrupt.

  write a 0 to leave the software interrupt unchanged.

The bit assignment for the software interrupt register is shown in Table 8-13.

Table 8-13 IRQ register bit assignment

Bit	Name	Function
31:1	Reserved	Write as 0. Reads undefined.
0	SOFT	Software interrupt.

Note

The *software interrupt* described in this section is used by software to generate IRQs or FIQs. It should not be confused with the ARM SWI software interrupt instruction. See the *ARM Architecture Reference Manual*.

8.6.3 Vectored interrupt controller

The interrupt controller implemented in the IM-PD1 image, is the PrimeCell PL190 *Vectored Interrupt Controller* (VIC). It provides a software interface to the interrupt system and supports two levels of interrupt:

- *Fast Interrupt Request* (FIQ) for fast, low latency interrupt handling
- *Interrupt ReQuest* (IRQ) for more general interrupts.

Only a single FIQ source at a time is generally defined in a system, to provide a true low-latency interrupt. This has the following benefits:

- You can execute the interrupt service routine directly without determining the source of the interrupt.
- Interrupt latency is reduced. You can use the banked registers available for FIQ interrupts more efficiently, because a context save is not required.

There are 32 interrupt lines. The VIC uses a bit position for each different interrupt source. The software can control each request line to generate software interrupts.

8.7 IntegratorCM922T-XA10 and IM-PD1 clock control

The programmable clocks are supplied by two Micrologic ICS307 clock generators, as described in:

- *Calculating the output frequencies of the ICS307*
- *Core Module oscillator register for the IM-PD1 image, CM_OSC* on page 8-27.

8.7.1 Calculating the output frequencies of the ICS307

The CM image provides registers to control the following parameters (see *Clock control parameters* on page 4-26):

S[2:0] OD selection.

V[8:1] VDW. **V[0]** is always 0.

The function setting for CLK2 selects the reference input frequency as the output frequency for this clock (that is 24MHz).

The equation used to calculate the output clock frequency from an ICS307 for this image is:

freq = 48*(VDW+ 8)/(3*OD)

This equation can be used because:

- A reference clock is supplied at a known frequency of 24MHz to the first clock generator.
- RDW has a fixed value of 1 for both clock generators. This makes is possible to fit the control parameters for both clock generators into one register and maintains compatibility with earlier Integrator core modules.

8.7.2 Core Module oscillator register for the IM-PD1 image, CM_OSC

The core module oscillator register at 0x10000008is a read/write register that controls the frequency of both clock generators.

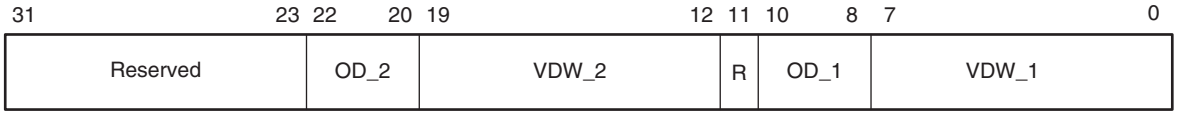


Figure 8-10 CM_OSC register

Note

Before writing to the CM_OSC register, you must unlock it by writing the value 0x0000A05F to the CM_LOCK register. After writing the CM_OSC register, relock it by writing any value other than 0x0000A05F to the CM_LOCK register.

Table 8-14 describes the core module oscillator register bits.

Table 8-14 CM_OSC register

Bits	Name	Access	Function	Default
[31:23]	Reserved	Use read-modify-write to preserve value.		-
[22:20]	OD_2	Read/write	Output divider select for U15: 000 = divide by 10 001 = divide by 2 010 = divide by 8 011 = divide by 4 100 = divide by 5 101 = divide by 7 110 = divide by 3 111 = divide by 6.	111
[19:12]	VDW_2	Read/write	These bits control bits [8:1] of the clock VCO divider word for U15. Bit [0] of the VCO divider word is always 0.	00000010

Table 8-14 CM_OSC register (continued)

Bits	Name	Access	Function	Default
[11]	Reserved	Use read-modify-write to preserve value.		-
[10:8]	OD_1	Read/write	Output divider select for U13: 000 = divide by 10 001 = divide by 2 010 = divide by 8 011 = divide by 4 100 = divide by 5 101 = divide by 7 110 = divide by 3 111 = divide by 6.	111
[7:0]	VDW_1	Read/write	These bits control bits [8:1] of the clock VCO divider word for U13. Bit [0] of the VCO divider word is always 0.	00000010

Table 8-15 shows the association between the clock generator chips and the signals that they generate.

Table 8-15 Clock signal association

Chip	Clock output	Signal name
U13	CLK1	**SYSCLK[3:0]** (when **MBDET**=1)
		EX_CLK[1]
		LA_CLK[1]
	CLK2	**24MHz**
U15	CLK1	**IM_PLL1CLK[2:1]**
		EX_CLK[2]
		LA_CLK[2]
	CLK2	**EX_CLK[3]**

8.8 IM-PD1 registers

The IM-PD1 image provides a set of registers specifically for the IM-PD1 card. The registers are called LM registers to preserve code compatibility with previous Integrator implementations that used a logic module to support the IM-PD1. Only the display interface controls in the LM_CONTROL register are functional. The LED, switches, and interrupt control registers are non-functional.

Table 8-16 shows the mapping of the LM registers in the Integrator/CM922T-XA10 and IM-PD1 implementation.

Table 8-16 LM registers for the IM-PD1 image

Address	Name	Type	Function
0xC0000000	Reserved	-	These register locations are non-functional in this implementation. They are provided for software compatibility with Integrator/LM and IM-PD1 implementations.
0xC0000004	Reserved	-	
0xC0000008	Reserved	-	
0xC000000C	LM_LEDS	Read/write	
0xC0000010	LM_INT	Read/write	
0xC0000014	LM_SW	Read	
0xC0000018	LM_CONTROL	Read/write	Display controls

8.8.1 Control register, LM_CONTROL

This register controls the multiplexors that are used to select the display type. Table 8-17 describes the operation of this register.

Table 8-17 LM_CONTROL register

Bits	Name	Access	Function
[7:3]	RESERVED	-	-
[2]	DISPLAY ENABLE	Read/write	This bit enables and disables the selected display: 0 = DISABLED 1 = ENABLED (default).
[1:0]	DISPLAY SELECT	Read/write	These bits control the display outputs: 00 = Sharp 8.4 inch display (default) 01= VGA/SVGA monitor 16bpp 10 = LCD1 connector 11 = VGA/SVGA monitor 24bpp.

8.9 IM-PD1 Interfaces

This section provides an overview the hardware on the interface module. The descriptions assume that PrimeCell peripherals in the core module PLD are being used to control these interfaces:

- *UART interface* on page 8-32
- *IrDA interface* on page 8-32
- *Smart card interface* on page 8-32
- *USB interface* on page 8-32
- *Audio CODEC* on page 8-33
- *MMC interface* on page 8-33
- *Display interface* on page 8-33
- *Touchscreen controller* on page 8-33.

8.9.1 Peripheral information block

The PLD image contains a *Peripheral Information Block* (PIB). This is a data structure that provides information about the peripherals present in the design. The format of entries in the PIB is:

`0xaabbbbcc`

where:

`0xaa` Indicates bits [27:20] of the base address of the peripheral. Bits [31:28] are determined by the location of the core module in the stack and bits [19:0] are always 0.

`0xbbbb` Indicates the PrimeCell type number. For example 0041 = AACI (PL041).

If the peripheral is not a PrimeCell, this field contains `0xFFbb`, where `bb` is a lookup value.

`0xcc` Is the Primecell revision number. For example, 11 = rev 1.1.

For example, and entry of `0x08004111` would indicate an AACI rev 1.1 with base address `0xC0800000`.

A PIB entry of `0x00000000` indicates that there is no peripheral at this location and that you should check the next address.

Note

You can use the utility `read_pib.axf` supplied on the distribution CD to read the PIB and display a list the peripherals present. It also displays the PLD build number.

8.9.2 UART interface

The interface module provides two serial transceivers and connectors. These are controlled by a PrimeCell UART (PL011) instantiated into the core module PLD. The interface signals are routed between the core module and IM-PD1 using the EXPIM connectors.

The PrimeCell UART is the same as that used in the CP image and is described in the *ARM PrimeCell UART (PL011) Technical Reference Manual*.

8.9.3 IrDA interface

The IM-PD1 provides an IRMS6400 IrDA interface transmitter and receiver. This is controlled by a PrimeCell UART (PL011) instantiated into the core module PLD. Interface connections are routed between the core module and IM-PD1 through the EXPIM connectors.

The PrimeCell UART is the same as that used in the CP image and is described in the *ARM PrimeCell UART (PL011) Technical Reference Manual*.

8.9.4 Smart card interface

The IM-PD1 provides a card socket. This interface is controlled by a PrimeCell the *Smart Card Interface* (SCI) instantiated into the core module PLD. The signals between the core module and IM-PD1 are routed through the EXPIM connectors. For more information about this PrimeCell, refer to the *ARM PrimeCell Smart Card Interface (PL130) Technical Reference Manual*.

8.9.5 USB interface

The IM-PD1 provides two USB interface connectors and two PDIUSB11AD USB transceivers. The interface signals from the USB transceiver are routed to the core module PLD using the EXPIM connectors.

To use the USB interface, you must instantiate a USB controller into the core module PLD.

8.9.6 Audio CODEC

The interface module provides a National Semiconductors LM4549 audio CODEC. The audio CODEC is compatible with AC'97 Rev 2.1, is PC98 compliant, and features sample rate conversion and analog 3D sound. The CODEC is driven with a PrimeCell AACI (PL041) instantiated into the core module PLD. The interface signals between the core module and interface module are routed through the EXPIM connector.

The PrimeCell AACI is the same as that used in the CP image and is described in the *ARM PrimeCell Advanced Audio CODEC Interface (PL041) Technical Reference Manual*.

8.9.7 MMC interface

The interface module provide an MMC socket. This is controlled by the MMC interface is the PrimeCell MMCI (PL181) instantiated into the core module PLD.The interface signals between the core module and interface module are routed through the EXPIM connectors.

The PrimeCell MMCI is the same as that used in the CP image and is described in the *ARM PrimeCell Multimedia Card Interface (PL181) Technical Reference Manual*.

8.9.8 Display interface

The interface module provides a flexible display interface that provides support for two types of LCD display or a VGA display. These are supported by the PrimeCell CLCD controller (PL110) instantiated into the core module PLD. The display interface signals are routed between the core module and interface module using the B bus signals on the HDRA connector.

The PrimeCell CLDC is the same as that used in the CP image and uses the registers described in the *ARM PrimeCell Color LCD Controller (PL110) Technical Reference Manual*.

8.9.9 Touchscreen controller

The touchscreen interface is designed to connect to a 4-wire resistive touchscreen. It is driven by the PrimeCell SSP (PL022) instantiated into the core module PLD. The signals to the touchscreen are routed through the EXPIM connectors. For more information about this PrimeCell, refer to the *ARM PrimeCell Synchronous Serial Port Master and Slave (PL022) Technical Reference Manual*.

Appendix A
Signal Descriptions

This index provides a summary of signals present on the core module main connectors. It contains the following sections:

Note

For the Multi-ICE connector pinout and signal descriptions see *JTAG signal descriptions* on page 3-8.

A.1 HDRA

Table A-1 on page A-3 shows the pin numbers of the HDRA plug.

Note

All pins on the HDRA socket are connected to the corresponding pins on the HDRA plug. The socket layout, therefore, is a mirror image of the plug.

The signals present on the pins labeled A[31:0], B[31:0], and C[31:0] are described in in Table A-1 for an AHB system bus.

Table A-1 Bus bit assignment (for an AMBA AHB bus)

Pin label	Signal	Description
A[31:0]	**HADDR[3:0]**	System address bus
B[31:0]	Not used	-
C[31:0]	**CONT[31:0]**	See below
C[31:16]	Not used	-
C15	**HLOCKALL**	Locked transaction
C[14:13]	**HRESP[1:0]**	Slave response
C12	**HREADY**	Slave ready
C11	**HWRITE**	Write transaction
C[10:8]	**HPROT[2:0]**	Transaction protection type
C[7:5]	**HBURST[2:0]**	Transaction burst size
C4	**HPROT[3]**	Transaction protection type
C[3:2]	**HSIZE[1:0]**	Transaction width
C[1:0]	**HTRAN[1:0]**	Transaction type
D[31:0]	**HDATA[31:0]**	System data bus

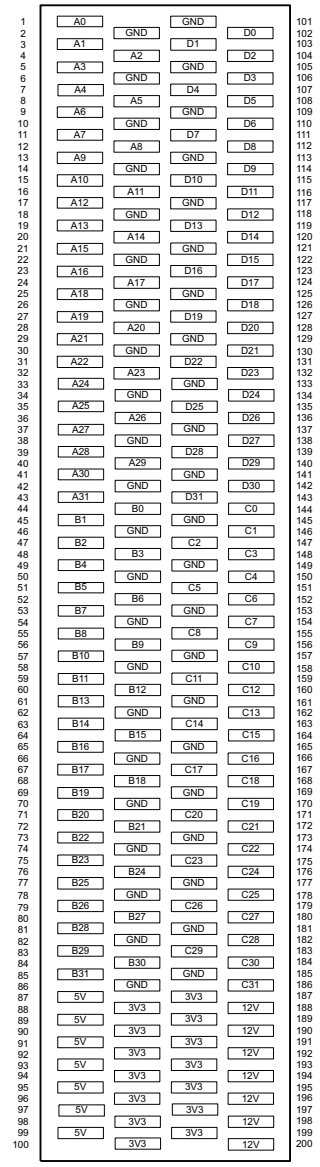


Figure A-1 HDRA plug pin numbering

A.2 HDRB

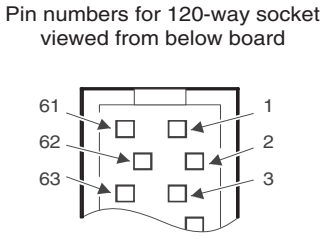
The HDRB plug and socket have slightly different pinouts, as described below.

A.2.1 HDRB socket pinout

Figure A-2 shows the pin numbers of the socket HDRB on the underside of the core module.

Note

The HDRB socket layout is, in general, a mirror image of the plug. Some pins are placed differently, however, as part of the signal rotation mechanism.



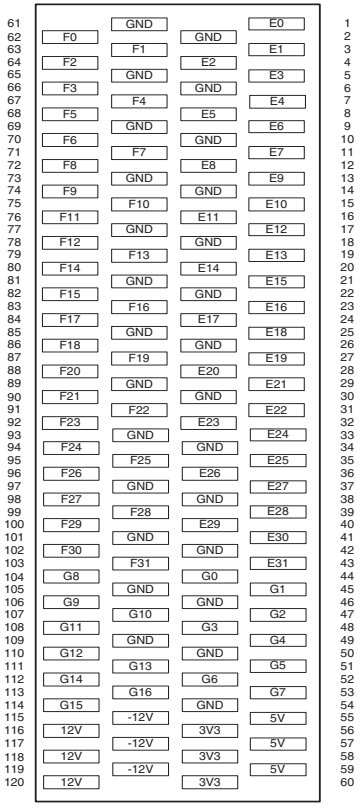


Figure A-2 HDRB socket pin numbering

A.2.2 HDRB plug pinout

Figure A-3 shows the pin numbers of the HDRB plug on the top of the core module.

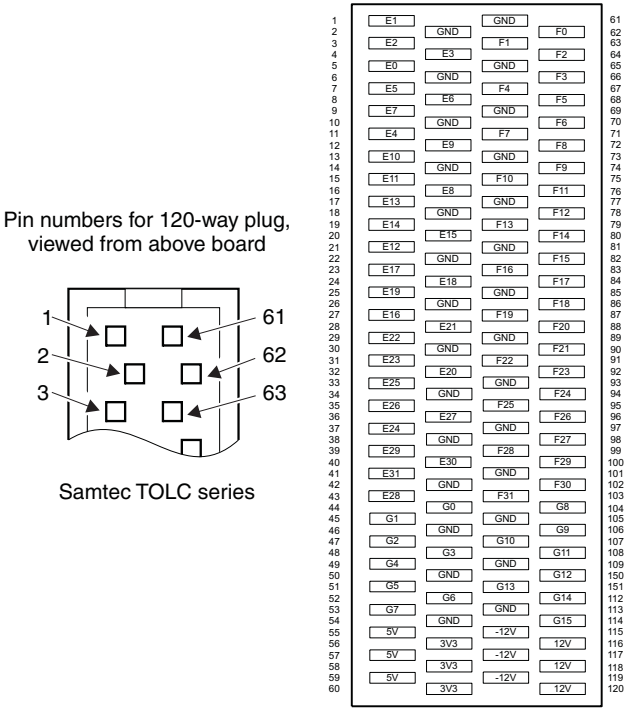


Figure A-3 HDRB plug pin numbering

A.2.3 Through-board signal connections

The signals on the pins labeled E[31:0] are cross-connected between the plug and socket so that the signals are rotated through the stack in groups of four. For example, the first block of four are connected as shown in Table A-2.

Table A-2 Signal cross-connections (example)

Plug		Socket
E0	connects to	E1
E1	connects to	E2
E2	connects to	E3
E3	connects to	E0

For details about the signal rotation scheme, see *System bus signal routing* on page 4-11.

A.2.4 HDRB signal descriptions

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-3 HDRB signal description (AHB)

Pin label	Name	Description
E[31:28]	**SYSCLK[3:0]**	System clock to each core module/expansion card
E[27:24]	**nPPRES[3:0]**	Processor present
E[23:20]	**nIRQ[3:0]**	Interrupt request to processors 3, 2, 1, and 0 respectively
E[19:16]	**nFIQ[3:0]**	Fast interrupt requests to processors 3, 2, 1, and 0 respectively
E[15:12]	**ID[3:0]**	Core module stack position indicator
E[11:8]	**HLOCK[3:0]**	System bus lock from processor 3, 2, 1, and 0 respectively
E[7:4]	**HGRANT[3:0]**	System bus grant to processor 3, 2, 1, and 0 respectively
E[3:0]	**HBUSREQ[3:0]**	System bus request from processors 3, 2, 1, and 0 respectively
F[31:0]	-	Not connected
G16	**nRTCKEN**	RTCK AND gate enable
G[15:14]	**CFGSEL[1:0]**	FPGA configuration select
G13	**nCFGEN**	Sets motherboard into configuration mode
G12	**nSRST**	Multi-ICE reset (open collector)
G11	**FPGADONE**	Indicates when FPGA configuration is complete (open collector)
G10	**RTCK**	Returned JTAG test clock
G9	**nSYSRST**	Buffered system reset
G8	**nTRST**	JTAG reset
G7	**TDO**	JTAG test data out

Table A-3 HDRB signal description (AHB) (continued)

Pin label	Name	Description
G6	**TDI**	JTAG test data in
G5	**TMS**	JTAG test mode select
G4	**TCK**	JTAG test clock
G[3:1]	**MASTER[2:0]**	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**	Motherboard detect pin

A.3 EXPIM plug and socket

Figure A-4 shows the pin numbers for the EXPIM plug on the interface module. NC indicates no connection. The layout for the EXPIM socket is a mirror image of the EXPIM plug.

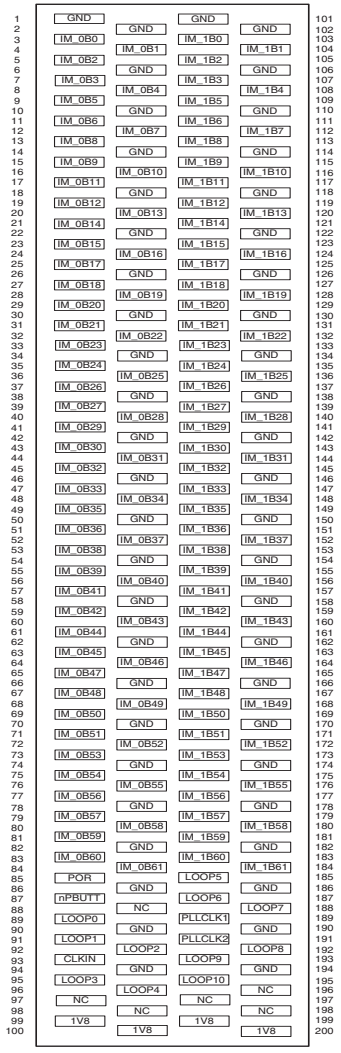


Figure A-4 EXPIM plug pin numbering

A.3.1 EXPIM signals

Table A-4 shows the signals that differ between the EXPIM socket and plug.

The EXPIM socket is not normally fitted to the core module. Contact your supplier for details on ordering a version of the core module with the socket already fitted.

Table A-4 Signal differences between EXPIM socket and plug

Pin	Socket	Plug	Description
85	NC	**nPORESET**	Power On Reset. This is the master reset for this board and the Integrator stack. It is supplied to the expansion board so that it can also be held in reset.
87	NC	**nPBUTT**	Pushbutton signal. This is a general purpose IO/O signal, but for legacy reasons, it is labeled **pBUTT** for pushbutton.
89	**LOOP0**	**LOOP0**	The loop signals are not used by the core module. They route through the connector to enable custom use of the signals.
91	**LOOP1**	**LOOP1**	
92	**LOOP2**	**LOOP2**	
93	**CLKIN**	**CLKIN**	**CLKIN** is for clocks generated on expansion board. Goes into fast input (FAST4) of the FPGA.
95	**LOOP3**	**LOOP3**	
96	**LOOP4**	**LOOP4**	
99	NC	**1V8**	
100	NC	**1V8**	
185	**LOOP5**	**LOOP5**	
187	**LOOP6**	**LOOP6**	
188	**LOOP7**	**LOOP7**	
189	NC	**IM_PLL1CLK1**	**IM_PLL1CLK1** and **IM_PLL1CLK2** are buffer signals from ICS307M B (U15) and provide a clock to the expansion board that is synchronous to CLK2 in the FPGA and LA_CLK2 on the mictor connectors.
191	NC	**IM_PLL1CLK2**	

Table A-4 Signal differences between EXPIM socket and plug (continued)

Pin	Socket	Plug	Description
192	**LOOP8**	**LOOP8**	
193	**LOOP9**	**LOOP9**	
195	**LOOP10**	**LOOP10**	
199	NC	**1V8**	
200	NC	**1V8**	

A.4 Serial interface connector

This section provides the pinout of the serial interface connector on the core module. The connector is shown in Figure A-5.

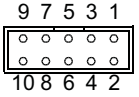


Figure A-5 Serial interface connector (J1)

Table A-5 lists the signals on the serial interface connector.

Table A-5 Serial interface signal assignment

Pin	Signal
1	**SER0_DCD**
2	**SER0_DSR**
3	**SER0_RX**
4	**SER0_RTS**
5	**SER0_TX**
6	**SER0_CTS**
7	**SER0_DTR**
8	**SER0_RI**
9	**GND**
10	Not connected

A.5 Multi-ICE connector

Figure A-6 shows the pinout of the Multi-ICE connector. For a description of the signals see, *JTAG signal descriptions* on page 3-8.

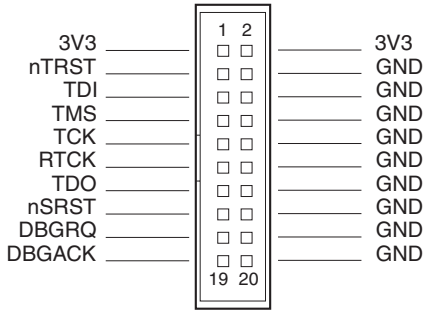


Figure A-6 Multi-ICE connector pinout

A.6 Trace connector pinout

Table A-6 shows the pinout of the Trace connector.

Table A-6 Trace connector pinout

Channel	Pin	Pin	Channel
no connect	1	2	no connect
no connect	3	4	no connect
GND	5	6	**TRACECLK**
DBGRQ	7	8	**DBGACK**
nSRST	9	10	**EXTTRIG**
TDO	11	12	**VDD** (3.3V)
RTCK	13	14	**VDD** (3.3V)
TCK	15	16	**TRACEPKT7**
TMS	17	18	**TRACEPKT6**
TDI	19	20	**TRACEPKT5**
nTRST	21	22	**TRACEPKT4**
TRACEPKT15	23	24	**TRACEPKT3**
TRACEPKT14	25	26	**TRACEPKT2**
TRACEPKT13	27	28	**TRACEPKT1**
TRACEPKT12	29	30	**TRACEPKT0**
TRACEPKT11	31	32	**TRACESYNC**
TRACEPKT10	33	34	**PIPESTAT2**
TRACEPKT9	35	36	**PIPESTAT1**
TRACEPKT8	37	38	**PIPESTAT0**

Appendix B
Excalibur PLD Pinout

This appendix provides a listing of the signal connections to the Excalibur XA10. It contains the following section:

B.1 Excalibur XA10 pinlist

Table B-1 shows the signal to pin assignment for the Excalibur XA10.

Table B-1 Signal assignments to Excalibur XA10 pins

Signal	Direction	Bus	Pin
EX_SD_DQM[3:0]	O	3	H9
		2	K9
		1	L14
		0	H14
EX_SD_DQS[3:0]	IO	3	H10
		2	K10
		1	K14
		0	J14
SD_CLKE	O		F14
EX_SD_A[14_0]	O	14	G6
		13	G7
		12	F6
		11	F7
		10	G8
		9	F8
		8	G9
		7	F9
		6	F10
		5	G10
		4	F11
		3	F12
		2	G11
		1	F13
		0	G12
EX_SD_nCAS	O		F18
EX_SD_nRAS	O		F17
EX_SD_nCS[1:0]	O	1	F16
		0	G14
EX_SD_nWE	O		G18

Table B-1 Signal assignments to Excalibur XA10 pins

Signal	Direction	Bus	Pin
EX_SD_DQ[31:0]	IO	31	H11
		30	J9
		29	J10
		28	J11
		27	J12
		26	J13
		25	H12
		24	H13
		23	L9
		22	L10
		21	K11
		20	L11
		19	K12
		18	L12
		17	K13
		16	L13
		15	L15
		14	K15
		13	L16
		12	L17
		11	K16
		10	L18
		9	K17
		8	K18
		7	J15
		6	J16
		5	H15
		4	J17
		3	J18
		2	H16
		1	H17
		0	H18
SD_nCLK	O		G13
SD_CLK	O		F15
UART_TXD	O		D28

Table B-1 Signal assignments to Excalibur XA10 pins

Signal	Direction	Bus	Pin
UART_RTS	O		C28
UART_DTR	O		G26
UART_RXD	I		D29
UART_DCD	IO		G27
UART_DSR	I		F28
UART_CTS	I		G28
UART_RI	IO		E28
EBI_A[24:0]	O	24	K21
		23	L21
		22	G21
		21	F21
		20	G22
		19	H22
		18	J22
		17	K22
		16	L22
		15	E21
		14	F22
		13	E22
		12	H23
		11	J23
		10	K23
		9	G23
		8	F23
		7	E23
		6	H24
		5	J24
		4	G24
		3	F24
		2	E24
		1	D24
		0	H25

Table B-1 Signal assignments to Excalibur XA10 pins

Signal	Direction	Bus	Pin
EBI_DQ[15:0]	IO	15	H19
		14	J19
		13	K19
		12	L19
		11	G19
		10	F19
		9	G20
		8	H20
		7	J20
		6	K20
		5	L20
		4	E19
		3	F20
		2	E20
		1	H21
		0	J21
EBI_nCS[3:0]	O	3	D25
		2	C25
		1	B25
		0	A25
EBI_nWE	O		E26
EBI_nOE	O		F26
EBI_WP	O		P7
EBI_ACK	I		F25
DIMM_CLK[3:0]	O	3	AE17
		2	AD17
		1	AC17
		0	AB17

Table B-1 Signal assignments to Excalibur XA10 pins

Signal	Direction	Bus	Pin
DIMM_A[13:0]	O	13	AC16
		12	AB16
		11	AE15
		10	AD15
		9	AC15
		8	AB15
		7	AE14
		6	AD14
		5	AC14
		4	AB14
		3	AE13
		2	AD13
		1	AC13
		0	AB13
DIMM_BANK[1:0]	O	1	AD16
		0	AE16
DIMM_CKE[1:0]	O	1	AC18
		0	AB18
DIMM_DQM[7:0]	O	7	AC20
		6	AB20
		5	AE19
		4	AD19
		3	AC19
		2	AB19
		1	AE18
		0	AD18
DIMM_nCS[3:0]	O	3	AE21
		2	AD21
		1	AE20
		0	AD20
DIMM_nCAS	O		M19
DIMM_nRAS	O		N19
DIMM_SCL			

Table B-1 Signal assignments to Excalibur XA10 pins

Signal	Direction	Bus	Pin
DIMM_nWE	O		N20
DIMM_DQ[63:32]	IO	63	T13
		62	R13
		61	P13
		60	N13
		59	AE12
		58	AD12
		57	AC12
		56	AB12
		55	AA12
		54	Y12
		53	W12
		52	V12
		51	U12
		50	T12
		49	R12
		48	P12
		47	N12
		46	AF11
		45	AE11
		44	AD11
		43	AC11
		42	AB11
		41	AA11
		40	Y11
		39	W11
		38	V11
		37	U11
		36	T11
		35	R11
		34	P11
		33	N11
		32	AF10

Table B-1 Signal assignments to Excalibur XA10 pins

Signal	Direction	Bus	Pin
DIMM_DQ[31:0]	IO	31	AE10
		30	AD10
		29	AC10
		28	AB10
		27	AA10
		26	Y10
		25	W10
		24	V10
		23	U10
		22	T10
		21	R10
		20	P10
		19	AF9
		18	AE9
		17	AD9
		16	AC9
		15	AB9
		14	U9
		13	R9
		12	AF8
		11	AE8
		10	AD8
		9	AC8
		8	AB8
		7	AA8
		6	Y8
		5	W8
		4	V8
		3	AB7
		2	AA7
		1	Y7
		0	W7
DIMM_SPDATA	IO		P20
SRAM_nCE	O		A15
SRAM_CLK	O		C14

Table B-1 Signal assignments to Excalibur XA10 pins

Signal	Direction	Bus	Pin
SRAM_ZZ	O		B14
DIL_SW[7:0]	I	7	L5
		6	K5
		5	K4
		4	J5
		3	J4
		2	J3
		1	H3
		0	E3
USR_LED[7:0]	O	7	T6
		6	T5
		5	T4
		4	R7
		3	R6
		2	R5
		1	R4
		0	R3
CLKDATA	O		T7
SCLK1	O		U6
SCLK2	O		U4
CLK1STRB	O		T8
CLK2STRB	O		U5
CLK_REF	I		A28
HCLK	I		N30
EX_CLK1	I		W30
EX_CLK2	I		P3
EX_CLK3	I		Y3

Table B-1 Signal assignments to Excalibur XA10 pins

Signal	Direction	Bus	Pin
LA_SPARE[95:64]	O	95	A20
		94	A19
		93	B20
		92	B19
		91	B18
		90	C20
		89	C19
		88	D19
		87	D18
		86	D17
		85	D16
		84	D15
		83	E18
		82	E17
		81	E16
		80	E15
		79	A24
		78	A23
		77	B24
		76	B23
		75	C23
		74	D23
		73	D20
		72	J26
		71	K26
		70	K24
		69	L26
		68	L24
		67	L23
		66	M26
		65	M23
		64	M22

Table B-1 Signal assignments to Excalibur XA10 pins

Signal	Direction	Bus	Pin
LA_SPARE[63:32]	O	63	M22
		62	M21
		61	N25
		60	N23
		59	N22
		58	N21
		57	P25
		56	P22
		55	P21
		54	R25
		53	R24
		52	R23
		51	R22
		50	R21
		49	R20
		48	T23
		47	T22
		46	T21
		45	U23
		44	U22
		43	U21
		42	V23
		41	V22
		40	V21
		39	W23
		38	W22
		37	W21
		36	Y23
		35	Y22
		34	Y21
		33	AA22
		32	AA21

Table B-1 Signal assignments to Excalibur XA10 pins

Signal	Direction	Bus	Pin
LA_SPARE[31:0]	O	31	AB24
		30	AB23
		29	AB22
		28	AB21
		27	AC25
		26	AC24
		25	AC23
		24	AC22
		23	AC21
		22	AD25
		21	AD24
		20	AD23
		19	AD22
		18	AE24
		17	AE23
		16	AE22
		15	A18
		14	U8
		13	V4
		12	V5
		11	V6
		10	V7
		9	W4
		8	W5
		7	W6
		6	Y4
		5	Y5
		4	Y6
		3	AA5
		2	AA6
		1	AB5
		0	AB6
PIPESTAT[2:0]	O	2	M9
		1	N9
		0	N10

Table B-1 Signal assignments to Excalibur XA10 pins

Signal	Direction	Bus	Pin
TRACEPKT[15:0]	O	15	H6
		14	J6
		13	K6
		12	L6
		11	M6
		10	N6
		9	H7
		8	J7
		7	K7
		6	L7
		5	M7
		4	N7
		3	H8
		2	J8
		1	K8
		0	L8
TRACECLK	O		N8
TRACESYNC	O		M8
DBGRQ	I		N5
DBGACK	O		M5
EXTTRIG	I		G25
EX_TDI	I		AD3
EX_TDO	O		E11
EX_TCK	I		AM19
EX_TMS	IO		AM20
PROC_TDI	I		H27
PROC_TDO	O		H26
PROC_TCK	I		D30
PROC_TMS	I		E29
PROC_TRST	I		E30

Table B-1 Signal assignments to Excalibur XA10 pins

Signal	Direction	Bus	Pin
PLL1OUT	O		AM29
PLL2OUT	O		AH3
nCFGEN	I		H4
nCONFIG	I		R30
nRESET	IO		B13
LA_CLK3	O		A14
LA_CLK4	O		B15
nPPRES[3:0]	I	3	A4
		2	B4
		1	A5
		0	B5
SLOCK[3:0]	O	3	D8
		2	E8
		1	D9
		0	E9
SREQ[3:0]	O	3	H29
		2	J29
		1	N26
		0	P26

Table B-1 Signal assignments to Excalibur XA10 pins

Signal	Direction	Bus	Pin
A[31:0]	IO	31	AJ14
		30	AJ15
		29	AK15
		28	AL15
		27	AJ16
		26	AK16
		25	AJ17
		24	AK17
		23	AJ18
		22	AK18
		21	AL18
		20	AJ19
		19	AK19
		18	AL19
		17	AJ20
		16	AK20
		15	AL20
		14	AJ23
		13	AK23
		12	AL23
		11	AM23
		10	AJ24
		9	AK24
		8	AL24
		7	AM24
		6	AH25
		5	AJ25
		4	AK25
		3	AL25
		2	AM25
		1	AH26
		0	AH27

Table B-1 Signal assignments to Excalibur XA10 pins

Signal	Direction	Bus	Pin
B[31:0]	IO	31	AJ3
		30	AH4
		29	AJ4
		28	AE30
		27	AH5
		26	AJ5
		25	AG28
		24	AH6
		23	AH7
		22	AH8
		21	AJ8
		20	AK8
		19	AL8
		18	AM8
		17	AH9
		16	AJ9
		15	AK9
		14	AL9
		13	AM9
		12	AH10
		11	AJ10
		10	AK10
		9	AL10
		8	AM10
		7	AH11
		6	AH12
		5	AH13
		4	AJ13
		3	AK13
		2	AL13
		1	AK14
		0	AL14

Table B-1 Signal assignments to Excalibur XA10 pins

Signal	Direction	Bus	Pin
C[31:0]	IO	31	AC4
		30	AD4
		29	AE4
		28	AC5
		27	AD5
		26	AE5
		25	AF5
		24	AG5
		23	AC6
		22	AD6
		21	AE6
		20	AF6
		19	AG6
		18	AC7
		17	AD7
		16	AE7
		15	AF7
		14	AG7
		13	AG8
		12	AG9
		11	AG10
		10	AG11
		9	AF12
		8	AG12
		7	AF13
		6	AG13
		5	AF14
		4	AG14
		3	AH14
		2	AF15
		1	AG15
		0	AH15

Table B-1 Signal assignments to Excalibur XA10 pins

Signal	Direction	Bus	Pin
D[31:0]	IO	31	AF16
		30	AG16
		29	AH16
		28	AF17
		27	AG17
		26	AH17
		25	AF18
		24	AG18
		23	AH18
		22	AF19
		21	AG19
		20	AH19
		19	AF20
		18	AG20
		17	AH20
		16	AF21
		15	AG21
		14	AH21
		13	AF22
		12	AG22
		11	AH22
		10	AF23
		9	AG23
		8	AH23
		7	AF24
		6	AG24
		5	AH24
		4	AE25
		3	AF25
		2	AG25
		1	AF26
		0	AG26

Table B-1 Signal assignments to Excalibur XA10 pins

Signal	Direction	Bus	Pin
F[31:0]	IO	31	R26
		30	T25
		29	T26
		28	U24
		27	U25
		26	U26
		25	V25
		24	V26
		23	V27
		22	V28
		21	W25
		20	W26
		19	W27
		18	Y24
		17	Y25
		16	Y26
		15	Y27
		14	AA23
		13	AA24
		12	AA25
		11	AA26
		10	AA27
		9	AB26
		8	AB27
		7	AC26
		6	E13
		5	AD26
		4	AD27
		3	AE26
		2	AE27
		1	AF27
		0	AG27
CONF	IO		AM13
nSRST	IO		P5
nTRST	IO		C13

Table B-1 Signal assignments to Excalibur XA10 pins

Signal	Direction	Bus	Pin
nIRQ[3:0]	I	3	D5
		2	E5
		1	F5
		0	E6
nFIQ[3:0]	I	3	A8
		2	B8
		1	C8
		0	E7
nMBDET	I		E4
nSYSRST	I		P4
SGNT[3:0]	I	3	D10
		2	E10
		1	E14
		0	J27
nPBUTT	IO		N4
nPOR	I		B28

Table B-1 Signal assignments to Excalibur XA10 pins

Signal	Direction	Bus	Pin
IM_0BANK[63:32]	IO	61	H30
		60	K28
		59	K29
		58	L28
		57	M28
		56	N29
		55	P28
		54	P29
		53	R29
		52	T28
		51	AC30
		50	U29
		49	V29
		48	E2
		47	W28
		46	Y28
		45	AA28
		44	AB28
		43	AC28
		42	AD28
		41	AD29
		40	AE28
		39	AK5
		38	AH28
		37	AJ29
		36	AJ28
		35	AK28
		34	V30
		33	AL4
		32	AM5

Table B-1 Signal assignments to Excalibur XA10 pins

Signal	Direction	Bus	Pin
IM_0BANK[31:0]	IO	31	AJ32
		30	B29
		29	E32
		28	J32
		27	N32
		26	R32
		25	W32
		24	AC32
		23	AE32
		22	AJ2
		21	AE2
		20	AC2
		19	W2
		18	R2
		17	N2
		16	J2
		15	D31
		14	H31
		13	K31
		12	P31
		11	V31
		10	Y31
		9	AD31
		8	AH31
		7	AH1
		6	AD1
		5	Y1
		4	V1
		3	P1
		2	K1
		1	H1
		0	D1

Table B-1 Signal assignments to Excalibur XA10 pins

Signal	Direction	Bus	Pin
IM_1BANK[63:32]	IO	61	J28
		60	K27
		59	L27
		58	M27
		57	N27
		56	N28
		55	P27
		54	R27
		53	R28
		52	T27
		51	U27
		50	U28
		49	U7
		48	E1
		47	W29
		46	Y29
		45	Y30
		44	AC29
		43	T29
		42	AD30
		41	AE29
		40	AK4
		39	AF28
		38	AH29
		37	AH30
		36	AJ30
		35	AK29
		34	AE3
		33	AM4
		32	AL5

Table B-1 Signal assignments to Excalibur XA10 pins

Signal	Direction	Bus	Pin
IM_1BANK[31:0]	IO	31	AJ31
		30	A29
		29	E31
		28	J31
		27	N31
		26	R31
		25	W31
		24	AC31
		23	AE31
		22	AJ1
		21	AE1
		20	AC1
		19	W1
		18	R1
		17	N1
		16	J1
		15	D32
		14	H32
		13	K32
		12	P32
		11	V32
		10	Y32
		9	AD32
		8	AH32
		7	AH2
		6	AD2
		5	Y2
		4	V2
		3	P2
		2	K2
		1	H2
		0	D2
HMASTER[2:0]	I	2	C4
		1	D3
		0	D4

Table B-1 Signal assignments to Excalibur XA10 pins

Signal	Direction	Bus	Pin
ID[3:0]	I	3	A9
		2	B9
		1	C9
		0	A13
IM_CLKIN	I		Y3
GLB_DONE	I		P8
nRESETOUT	O		AC27
DONE_IN	I		AC6
DONE_OUT	O		G5
CLKLK_ENA	I		P30
IM_CLKIN	I		AM15
nPBRESET	I		AM18
MISC_LED	O		H5
ALWAYSONE	I		P6
SPAREA	O		R8

Appendix C
Mechanical Specification

This appendix provides a listing of the signal connections to the Excalibur XA10. It contains the following section:

- *Mechanical information* on page C-2.

C.1 Mechanical information

The Integrator/CM922T-XA10 is designed to be used in stackable system. Figure C-1 shows the mechanical outline of a board that you can mount onto an Integrator/CM922T-XA10 board. It shows the location of pin 1 of the Samtec connectors. (Dimensions are in millimeters.)

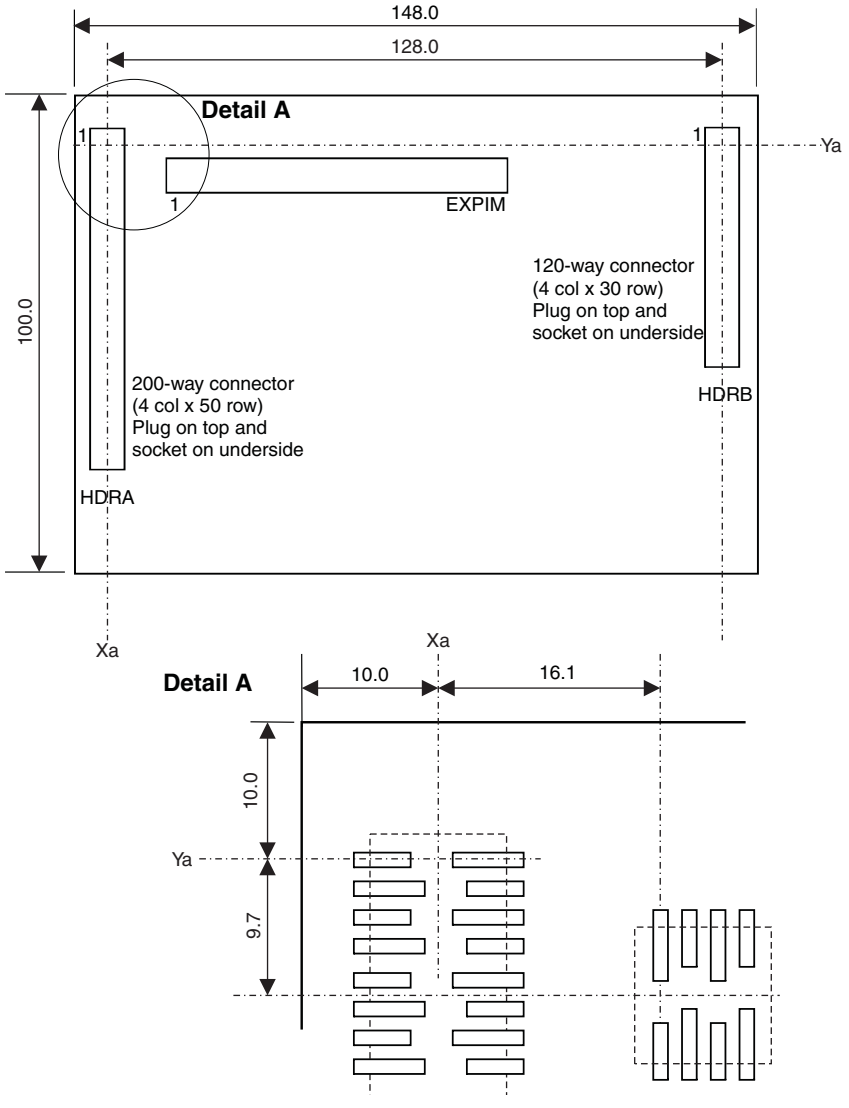


Figure C-1 Board dimensions

Note

In Figure C-1 on page C-2, the 148.0 and 100.0 dimensions show the size of a typical Integrator module produced by ARM Limited.

C.1.1 Connector part numbers

The Samtec connector part numbers are listed in Table C-1.

Table C-1 Samtec connector part numbers

Type	Part number
200 way connector	TOLC-150-32-F-Q-P-A
120 way connector	TOLC-130-32-F-Q-P-A

Glossary

AFS See *ARM Firmware Suite*.

ARM Boot Flash Utility
The *ARM Boot Flash Utility* (BootFU) allows modification of the specific boot flash sector on the system.

ARM Firmware Suite A collection of utilities to assist in developing applications and operating systems on ARM-based systems.

ARM Flash Utility The *ARM Flash Utility* (AFU) is an application for manipulating and storing data within a system that uses the flash library.

Big-Endian Memory organization where the least significant byte of a word is at a higher address than the most significant byte. See also *Little-Endian*.

BootFU See *ARM Boot Flash Utility*.

Boot monitor A ROM-based monitor that communicates with a host computer using simple commands over a serial port. Typically this application is used to display the contents of memory and provide system debug and self-test functions.

Boot switcher The boot switcher selects and runs an image in application flash. You can store one or more code images in flash memory and use the boot switcher to start the image at reset.

Debugger An application that monitors and controls the execution of a second application. Usually used to find errors in the application program flow.

Double word A 64-bit unit of information. Contents are taken as being an unsigned integer unless otherwise stated.

Flash memory Nonvolatile memory that is used to hold boot code and application code.

Halfword A 16-bit unit of information. Contents are taken as being an unsigned integer unless otherwise stated.

Host A computer which provides data and other services to another computer.

Image A hardware description file that is used to configure the PLD during board initialization.

Linker Software which produces a single image from one or more source assembler or compiler output objects.

Little-endian Memory organization where the least significant byte of a word is at a lower address than the most significant byte. See also *Big-endian*.

Memory management unit
Hardware that controls caches and access permissions to blocks of memory, and translates virtual to physical addresses.

Memory protection unit
Hardware that controls permissions to blocks of memory. Unlike an MMU, a MPU does not translate virtual addresses to physical addresses.

MMU See *Memory Management Unit*.

MPU See *Memory Protection Unit*.

Multi-ICE Multi-processor JTAG emulator. ARM registered trademark.

Peripheral information block
A data structure used to identify peripherals included in the design instantiated into a logic module FPGA.

Remapping Changing the address of physical memory or devices after the application has started executing. This is typically done to allow RAM to replace ROM once the initialization has been done.

SIB See *System Information Block*.

SWI Software Interrupt. An instruction that causes the processor to call a programer-specified subroutine. Used by ARM to handle semihosting.

System Information Block
A block of user-defined nonvolatile storage.

Target The actual target processor, real or simulated, on which the application is running.

Watchpoint A location within the image which will be monitored and which will cause execution to break when it changes.

Word A 32-bit unit of information. Contents are taken as being an unsigned integer unless otherwise stated.

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

B

C

D

E

F

G

H

I

J

L

M

N

O

P

R

S

T

U