PrimeCell® Vectored Interrupt Controller (PL190)
Revision: r1p2
PrimeCell Vectored Interrupt Controller (PL190)

Copyright © 2000, 2003-2004 ARM Limited. All rights reserved.

Release Information

The following changes have been made to this document.

<table>
<thead>
<tr>
<th>Date</th>
<th>Issue</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>30 June 2000</td>
<td>A</td>
<td>First release</td>
</tr>
<tr>
<td>August 2000</td>
<td>B</td>
<td>Small corrections to code examples</td>
</tr>
<tr>
<td>September 2000</td>
<td>C</td>
<td>VICITIP1 &amp; 2 changed to read-only. Changes to Figs 2-5 &amp; 2-6</td>
</tr>
<tr>
<td>02 July 2003</td>
<td>D</td>
<td>Incorporate errata, revision r1p1</td>
</tr>
<tr>
<td>30 November 2004</td>
<td>E</td>
<td>Incorporate erratum, revision r1p2</td>
</tr>
</tbody>
</table>

Proprietary Notice

Words and logos marked with ® or ™ are registered trademarks or trademarks owned by ARM Limited, except as otherwise stated below in this proprietary notice. Other brands and names mentioned herein may be the trademarks of their respective owners.

Neither the whole nor any part of the information contained in, or the product described in, this document may be adapted or reproduced in any material form except with the prior written permission of the copyright holder.

The product described in this document is subject to continuous developments and improvements. All particulars of the product and its use contained in this document are given by ARM in good faith. However, all warranties implied or expressed, including but not limited to implied warranties of merchantability, or fitness for purpose, are excluded.

This document is intended only to assist the reader in the use of the product. ARM Limited shall not be liable for any loss or damage arising from the use of any information in this document, or any error or omission in such information, or any incorrect use of the product.

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
Contents
PrimeCell Vectored Interrupt Controller (PL190)

Preface
About this manual ................................................................................................. x
Feedback ........................................................................................................... xiv

Chapter 1  Introduction
1.1  About the VIC ............................................................................................ 1-2
1.2  Product revisions ....................................................................................... 1-3

Chapter 2  Functional Overview
2.1  About the VIC ............................................................................................ 2-2
2.2  Operation .................................................................................................... 2-9
2.3  Connectivity .............................................................................................. 2-11

Chapter 3  Programmer’s Model
3.1  About the programmer’s model ................................................................. 3-2
3.2  Summary of VIC registers ......................................................................... 3-3
3.3  Register descriptions ................................................................................ 3-6
3.4  Interrupt latency ....................................................................................... 3-18
3.5  Interrupt priority ....................................................................................... 3-21
Chapter 4  Programmer's Model for Test
4.1 VIC test harness overview ............................................................... 4-2
4.2 Scan testing .................................................................................. 4-3
4.3 Summary of test registers ............................................................... 4-4

Appendix A  Signal Descriptions
A.1 AMBA AHB signals ....................................................................... A-2
A.2 Interrupt controller signals .......................................................... A-3
A.3 Daisy-chain signals ....................................................................... A-4
A.4 Scan test control signals ............................................................... A-5

Appendix B  Example Code
B.1 About the example code ............................................................... B-2

Glossary
List of Tables
PrimeCell Vectored Interrupt Controller (PL190)

Change history ........................................................................................................... ii
Table 2-1 Recommended interrupt standard configuration ........................................ 2-3
Table 3-1 VIC register summary .................................................................................. 3-3
Table 3-2 VICIRQSTATUS Register bit assignments .................................................. 3-6
Table 3-3 VICFIQSTATUS Register bit assignments ................................................... 3-6
Table 3-4 VICRAWINTR Register bit assignments ..................................................... 3-6
Table 3-5 VICINTSELECT Register bit assignments .................................................. 3-7
Table 3-6 VICINENABLE Register bit assignments .................................................... 3-7
Table 3-7 VICINTENCLEAR Register bit assignments ............................................... 3-7
Table 3-8 VICSOFTINT Register bit assignments ...................................................... 3-8
Table 3-9 VICSOFTINTCLEAR Register bit assignments .......................................... 3-8
Table 3-10 VICPROTECTION Register bit assignments ............................................. 3-9
Table 3-11 VICVECTADDR Register bit assignments ............................................... 3-9
Table 3-12 VICDEFVECTADDR Register bit assignments ......................................... 3-10
Table 3-13 VICVECTADDR Registers bit assignments .............................................. 3-10
Table 3-14 VICVECTCNTL Registers bit assignments .............................................. 3-11
Table 3-15 VICPERIPHID0 Register bit assignments ............................................... 3-12
Table 3-16 VICPERIPHID1 Register bit assignments .............................................. 3-13
Table 3-17 VICPERIPHID2 Register bit assignments .............................................. 3-14
Table 3-18 VICPERIPHID3 Register bit assignments .............................................. 3-14
Table 3-19 VICPCELLID0 Register bit assignments .................................................. 3-15
Table 3-20  VICPCELLID1 Register bit assignments ...............................................................  3-16
Table 3-21  VICPCELLID2 Register bit assignments ...............................................................  3-16
Table 3-22  VICPCELLID3 Register bit assignments ...............................................................  3-17
Table 3-23  FIQ interrupt latency ...............................................................................................  3-18
Table 3-24  IRQ interrupt latency ...............................................................................................  3-19
Table 3-25  Fast IRQ interrupt latency ........................................................................................  3-19
Table 4-1   Test registers memory map ......................................................................................  4-4
Table 4-2   VICITCR Register bit assignments ...........................................................................  4-4
Table 4-3   VICITIP1 Register bit assignments .........................................................................  4-5
Table 4-4   VICITIP2 Register bit assignments .........................................................................  4-5
Table 4-5   VICITOP1 Register bit assignments .........................................................................  4-6
Table 4-6   VICITOP2 Register bit assignments .........................................................................  4-6
Table A-1   AMBA AHB signal descriptions ................................................................................. A-2
Table A-2   Interrupt controller signals ..................................................................................... A-3
Table A-3   Daisy-chain signals ................................................................................................ A-4
Table A-4   Scan test control signals ........................................................................................ A-5
# List of Figures

**PrimeCell Vectored Interrupt Controller (PL190) Technical Reference Manual**

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-1</td>
<td>VIC block diagram</td>
<td>2-4</td>
</tr>
<tr>
<td>2-2</td>
<td>Interrupt request logic</td>
<td>2-5</td>
</tr>
<tr>
<td>2-3</td>
<td>Non-vectored FIQ interrupt logic</td>
<td>2-6</td>
</tr>
<tr>
<td>2-4</td>
<td>Non-vectored IRQ interrupt logic</td>
<td>2-6</td>
</tr>
<tr>
<td>2-5</td>
<td>Vectored interrupt block</td>
<td>2-7</td>
</tr>
<tr>
<td>2-6</td>
<td>Interrupt priority logic</td>
<td>2-8</td>
</tr>
<tr>
<td>2-7</td>
<td>Standalone interrupt controller connectivity</td>
<td>2-11</td>
</tr>
<tr>
<td>2-8</td>
<td>Daisy-chained interrupt controller connectivity</td>
<td>2-12</td>
</tr>
<tr>
<td>3-1</td>
<td>VICPROTECTION Register bit assignments</td>
<td>3-8</td>
</tr>
<tr>
<td>3-2</td>
<td>VICVECTCNTL Register bit assignments</td>
<td>3-10</td>
</tr>
<tr>
<td>3-3</td>
<td>Peripheral Identification Register bit allocation</td>
<td>3-12</td>
</tr>
<tr>
<td>3-4</td>
<td>VICPERIPHID0 Register bit assignments</td>
<td>3-12</td>
</tr>
<tr>
<td>3-5</td>
<td>VICPERIPHID1 Register bit assignments</td>
<td>3-13</td>
</tr>
<tr>
<td>3-6</td>
<td>VICPERIPHID2 Register bit assignments</td>
<td>3-13</td>
</tr>
<tr>
<td>3-7</td>
<td>VICPERIPHID3 Register bit assignments</td>
<td>3-14</td>
</tr>
<tr>
<td>3-8</td>
<td>PrimeCell Identification Register bit assignment</td>
<td>3-15</td>
</tr>
<tr>
<td>3-9</td>
<td>VICPCELLID0 Register bit assignments</td>
<td>3-15</td>
</tr>
<tr>
<td>3-10</td>
<td>VICPCELLID1 Register bit assignments</td>
<td>3-16</td>
</tr>
<tr>
<td>3-11</td>
<td>VICPCELLID2 Register bit assignments</td>
<td>3-16</td>
</tr>
<tr>
<td>3-12</td>
<td>VICPCELLID3 Register bit assignments</td>
<td>3-17</td>
</tr>
</tbody>
</table>

Key to timing diagram conventions | xii
| Figure 4-1 | VICITCR Register bit assignments | 4-4 |
| Figure 4-2 | VICITIP1 Register bit assignments | 4-5 |
| Figure 4-3 | VICITOP1 Register bit assignments | 4-6 |
Preface

This preface introduces the PrimeCell Vectored Interrupt Controller (PL190) r1p2 Technical Reference Manual. It contains the following sections:

- About this manual on page x
- Feedback on page xiv.
About this manual

This is the Technical Reference Manual for the ARM PrimeCell Vectored Interrupt Controller (VIC).

Product revision status

The rnpn identifier indicates the revision status of the product described in this manual, where:

- \( r_n \): Identifies the major revision of the product.
- \( p_n \): Identifies the minor revision or modification status of the product.

Intended audience

This manual is written for hardware and software engineers implementing System-on-Chip (SoC) designs. It provides the necessary information to enable designers to integrate the peripheral into a target system as quickly as possible. The manual describes the VIC.

Using this manual

This manual is organized into the following chapters:

Chapter 1 Introduction
Read this chapter for an introduction to the VIC and its features.

Chapter 2 Functional Overview
Read this chapter for a description of the major functional blocks of the VIC.

Chapter 3 Programmer’s Model
Read this chapter for a description of the registers and programming details of the VIC.

Chapter 4 Programmer’s Model for Test
Read this chapter for a description of the test registers and signals of the VIC.

Appendix A Signal Descriptions
Read this appendix for a description of the VIC signals.

Appendix B Example Code
Read this appendix for a description of the VIC example code.
Glossary  Read the Glossary for definitions of terms used in this manual.

Conventions

Conventions that this manual can use are described in:

- Typographical
- Timing diagrams on page xii
- Signals on page xii
- Numbering on page xiii.

Typographical

The typographical conventions are:

*italic*  Highlights important notes, introduces special terminology, denotes internal cross-references, and citations.

*bold*  Highlights interface elements, such as menu names. Denotes signal names. Also used for terms in descriptive lists, where appropriate.

*monospace*  Denotes text that you can enter at the keyboard, such as commands, file and program names, and source code.

*monospace*  Denotes a permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name.

*monospace italic*  Denotes arguments to monospace text where the argument is to be replaced by a specific value.

*monospace bold*  Denotes language keywords when used outside example code.

*< and >*  Angle brackets enclose replaceable terms for assembler syntax where they appear in code or code fragments. They appear in normal font in running text. For example:

- `MRC p15, 0 <Rd>, <CRn>, <CRm>, <Opcode_2>`
- The Opcode_2 value selects the register that is accessed.
Timing diagrams

The figure named *Key to timing diagram conventions* explains the components that timing diagrams use. Variations, when they occur, have clear labels. You must not assume any timing information that is not explicit in the diagrams.

Shaded bus and signal areas are undefined, so the bus or signal can assume any value within the shaded area at that time. The actual level is unimportant and does not affect normal operation.

![Key to timing diagram conventions](image)

**Signals**

The signal conventions are:

- **Signal level**
  - The level of an asserted signal depends on whether the signal is active-HIGH or active-LOW. Asserted means HIGH for active-HIGH signals and LOW for active-LOW signals.

- **Prefix A**
  - Denotes *Advanced eXtensible Interface* (AXI) global and address channel signals.

- **Prefix B**
  - Denotes AXI write response channel signals.

- **Prefix C**
  - Denotes AXI low-power interface signals.

- **Prefix H**
  - Denotes *Advanced High-performance Bus* (AHB) signals.

- **Prefix n**
  - Denotes active-LOW signals except in the case of AXI, AHB, or *Advanced Peripheral Bus* (APB) reset signals.

- **Prefix P**
  - Denotes APB signals.
Prefix R  Denotes AXI read channel signals.
Prefix W  Denotes AXI write channel signals.
Suffix n  Denotes AXI, AHB, and APB reset signals.

Numbering

The numbering convention is:

<size in bits>'<base><number>

This is a Verilog method of abbreviating constant numbers. For example:

- 'h7B4 is an unsized hexadecimal value.
- 'o7654 is an unsized octal value.
- 8'd9 is an eight-bit wide decimal value of 9.
- 8'h3F is an eight-bit wide hexadecimal value of 0x3F. This is equivalent to b00111111.
- 8'b1111 is an eight-bit wide binary value of b00001111.

Further reading

This section lists publications by ARM Limited, and by third parties.

ARM periodically provides updates and corrections to its documentation. See http://www.arm.com for current errata sheets, addenda, and the Frequently Asked Questions list.

ARM publications

This manual contains information that is specific to the ARM PrimeCell Vectored Interrupt Controller (PL190). See the following documents for other relevant information:

- AMBA Specification (Rev 2.0) (ARM IHI 00011).
Feedback

ARM Limited welcomes feedback, both on the ARM PrimeCell Vectored Interrupt Controller (PL190) and its documentation.

Feedback on the product

If you have any comments or suggestions about the VIC, contact your supplier giving:
- the product name
- a concise explanation of your comments.

Feedback on this manual

If you have any comments on this manual, send email to errata@arm.com giving:
- the title
- the number
- the relevant page number(s) to which your comments apply
- a concise explanation of your comments.

ARM Limited also welcomes general suggestions for additions and improvements.
Chapter 1
Introduction

This chapter introduces the ARM PrimeCell Vectored Interrupt Controller (VIC) (PL190). It contains the following section:

- About the VIC on page 1-2
- Product revisions on page 1-3.
1.1 About the VIC

The VIC is an Advanced Microcontroller Bus Architecture (AMBA) compliant System-on-Chip (SoC) peripheral that is developed, tested, and licensed by ARM.

The VIC provides an interface to the interrupt system, and improves interrupt latency in two ways:

- moves the interrupt controller to the AMBA AHB bus
- provides vectored interrupt support for high-priority interrupt sources.

1.1.1 Features of the VIC

The VIC has the following features:

- compliance to the AMBA Specification (Rev 2.0) onwards for easy integration into SoC implementation
- support for 32 standard interrupts
- support for 16 vectored IRQ interrupts
- hardware interrupt priority
- IRQ and FIQ generation
- AHB mapped for faster interrupt response
- software interrupt generation
- test registers
- raw interrupt status
- interrupt request status
- interrupt masking
- privileged mode support
- vector interrupt controller daisy-chaining support.
1.2 Product revisions

This section describes differences in functionality between product revisions of the VIC:

**r1p-r1p2** Contains no change to functionality. See the Errata document for details of erratum that have been fixed in this release.

These changes have no effect on the information provided in this manual.
Chapter 2
Functional Overview

This chapter describes the major functional blocks of the VIC (PL190). It contains the following sections:
- *About the VIC* on page 2-2
- *Operation* on page 2-9
- *Connectivity* on page 2-11.
2.1 About the VIC

The VIC provides a software interface to the interrupt system. In a system with an interrupt controller, software must determine the source that is requesting service and where its service routine is loaded. A VIC does both of these in hardware. It supplies the starting address, or vector address, of the service routine corresponding to the highest priority requesting interrupt source.

In an ARM system, two levels of interrupt are available:

**Fast Interrupt reQuest (FIQ)**

For fast, low latency interrupt handling.

**Interrupt ReQuest (IRQ)**

For more general interrupts.

Generally, you only use a single FIQ source at a time 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.
- It reduces interrupt latency. You can use the banked registers available for FIQ interrupts more efficiently, because you do not require a context save.

The interrupt inputs must be level sensitive, active HIGH, and held asserted until the interrupt service routine clears the interrupt. Edge-triggered interrupts are not compatible.

The interrupt inputs do not have to be synchronous to **HCLK**.

--- **Note** ---

The VIC does not handle interrupt sources with transient behavior. For example, an interrupt is asserted and then deasserted before software can clear the interrupt source. In this case, the CPU acknowledges the interrupt and obtains the vectored address for the interrupt from the VIC, assuming that no other interrupt has occurred to overwrite the vectored address. However, when a transient interrupt occurs, the priority logic of the VIC is not set, and lower priority interrupts can interrupt the transient interrupt service routine, assuming interrupt nesting is permitted.

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.
There are 16 vectored interrupts. These interrupts can only generate an IRQ interrupt. The vectored and non-vectored IRQ interrupts provide an address for an Interrupt Service Routine (ISR). Reading from the Vector Interrupt Address Register, VICVECTADDR, provides the address of the ISR, and updates the interrupt priority hardware that masks out the current, and any lower priority interrupt requests. Writing to the VICVECTADDR Register indicates to the interrupt priority hardware that the current interrupt is serviced, enabling lower priority or the same priority interrupts to be removed, and for the interrupts to become active to go active.

The FIQ interrupt has the highest priority, followed by interrupt vector 0 to interrupt vector 15. Non-vectored IRQ interrupts have the lowest priority. A programmed interrupt request enables you to generate an interrupt under software control. This register typically downgrades an FIQ interrupt to an IRQ interrupt.

— Note —
- The ARM core sets the priority of the FIQ over IRQ.
- The VIC can raise both an FIQ and an IRQ at the same time.

——

The IRQ and FIQ request logic has an asynchronous path to the nVICIRQ and nVICFIQ outputs respectively. This enables you to assert interrupts when the VIC AHB clock, HCLK, is disabled in a low-power mode. It is expected that the power control logic enables the processor and VIC AHB clocks when an interrupt is received, so that the interrupt service routine can be performed.

By convention, for the IRQ interrupt, you are recommended to use bits 1 to 5 that Table 2-1 defines. Bit 0 and bit 6 upwards are available for you to use. For the FIQ interrupt, you can use all the bits.

<table>
<thead>
<tr>
<th>Bit</th>
<th>Interrupt source</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Software interrupt</td>
</tr>
<tr>
<td>2</td>
<td>Comms Rx</td>
</tr>
<tr>
<td>3</td>
<td>Comms Tx</td>
</tr>
<tr>
<td>4</td>
<td>Timer 1</td>
</tr>
<tr>
<td>5</td>
<td>Timer 2</td>
</tr>
</tbody>
</table>
A space is reserved for the software interrupt so that you can use it without masking out a valid hardware interrupt. You can program any of the interrupt bits through software using the VICSOFTINT Register but, by reserving a specific software interrupt bit, it is easier to differentiate between hardware and software interrupts.

The Comms RX and TX lines are debug channel interrupts that the system processor uses, and they are required in any system that uses these debug features.

Spaces are reserved for two timers because a typical system has at least two timers.

Figure 2-1 shows a block diagram of the VIC.
The following sections describe the main components of the VIC:

- Interrupt request logic
- Non-vectored FIQ interrupt logic
- Non-vectored IRQ interrupt logic on page 2-6
- Vectored interrupt block on page 2-6
- Interrupt priority logic on page 2-7.

### 2.1.1 Interrupt request logic

The interrupt request logic receives the interrupt requests from the peripheral and combines them with the software interrupt requests. It then masks out the interrupt requests that are not enabled, and routes the enabled interrupt requests to either `IRQSTATUS` or `FIQSTATUS`. Figure 2-2 shows a block diagram of the interrupt request logic.

![Figure 2-2 Interrupt request logic](image)

### 2.1.2 Non-vectored FIQ interrupt logic

The non-vectored FIQ interrupt logic generates the FIQ interrupt signal by combining the FIQ interrupt requests in the interrupt controller and any requests from daisy-chained interrupt controllers. Figure 2-3 on page 2-6 shows a block diagram of the non-vectored FIQ interrupt logic.
2.1.3 Non-vectored IRQ interrupt logic

The non-vectored IRQ interrupt logic combines the non-vectored interrupt requests to generate the non-vectored IRQ interrupt signal. This signal is then sent to the IRQ vector address and priority logic. Figure 2-4 shows a block diagram of the non-vectored IRQ interrupt logic.

2.1.4 Vectored interrupt block

There are 16 vectored interrupt blocks. The vectored interrupt blocks receive the IRQ interrupt requests and set $\text{VECTIRQX}$ if the following are true:
- the selected interrupt is active
- the selected interrupt is currently the highest requesting interrupt.

Each vectored interrupt block also provides a $\text{VECTADDRX}[31:0]$ output that you can use in the interrupt priority block. Figure 2-5 on page 2-7 shows a block diagram of two of the vectored interrupt blocks.
2.1.5 Interrupt priority logic

The interrupt priority block prioritizes the following requests:

- non-vectored interrupt requests
- vectored interrupt requests
- external interrupt requests.

The highest priority request generates an IRQ interrupt if the interrupt is not currently being serviced. Figure 2-6 on page 2-8 shows a block diagram of the interrupt priority logic.

Note

nVICIRQIN is the daisy-chained IRQ request input.
2.1.6 Vectored interrupts

A vectored interrupt is only generated if the following are true:

- you enable it in the interrupt enable register, VICIntEnable
- you set it to generate an IRQ interrupt in the interrupt select register, VICIntSelect
- you enable it in the relevant vector control register, VICVectCntl[0-15].

This prevents multiple interrupts being generated from a single interrupt request if the controller is incorrectly programmed.

2.1.7 Software interrupts

The software can control the source interrupt lines to generate software interrupts. These interrupts are generated before interrupt masking, in the same way as external source interrupts. You clear software interrupts by writing to the Software Interrupt Clear Register, VICSoftIntClear. See Software Interrupt Clear Register on page 3-8. This is normally done at the end of the interrupt service routine.
2.2 Operation

The following sections describe the operation of the VIC:

- Vectored interrupt flow sequence
- Simple interrupt flow.

2.2.1 Vectored interrupt flow sequence

The following procedure shows the sequence for the vectored interrupt flow:

1. An interrupt occurs.
2. The ARM processor branches to either the IRQ or FIQ interrupt vector.
3. If the interrupt is an IRQ, read the VICVectAddr Register and branch to the interrupt service routine. You can do this using an LDR PC instruction. Reading the VICVectorAddr Register updates the interrupt controllers hardware priority register.
4. Stack the workspace so that you can re-enable IRQ interrupts.
5. Enable the IRQ interrupts so that a higher priority can be serviced.
7. Clear the requesting interrupt in the peripheral, or write to the VICSoftIntClear Register if the request was generated by a software interrupt.
8. Disable the interrupts and restore the workspace.
9. Write to the VICVectAddr Register. This clears the respective interrupt in the internal interrupt priority hardware.
10. Return from the interrupt. This re-enables the interrupts.

2.2.2 Simple interrupt flow

The following procedure describes how to use the interrupt controller without using vectored interrupts or the interrupt priority hardware. For example, you can use it for debugging:

1. An interrupt occurs.
2. Branch to the IRQ or FIQ interrupt vector.
3. Branch to the interrupt handler.
4. Interrogate the VICIRQStatus Register to determine the source that generated the interrupt, and prioritize the interrupts if there are multiple active interrupt sources. This takes a number of instructions to compute.

5. Branch to the correct ISR.

6. Execute the ISR.

7. Clear the interrupt. If a software interrupt generated the request, you must write to the VICSoftIntClear Register.

8. Check the VICIRQStatus Register to ensure that no other interrupt is active. If there is an active request, go to Step 4.

9. Return from the interrupt.

——— Note ————
If you use this flow, do not read or write to the VICVectorAddr Register.

———
2.3 Connectivity

You normally use the VIC as a standalone interrupt controller. Where required, you can daisy-chain a second VIC.

Note

The interrupt latency increases if you use daisy-chaining. See Daisy-chained interrupts on page 3-20.

The VIC connects to the processor as a standard AHB slave, with the FIQ and IRQ signals connected to the FIQ and IRQ inputs on the processor. The interrupt request lines from the peripheral connect to the VICINTSOURCE inputs of the VIC. To ensure that you can read the vector address register in a single instruction, you must put the VIC in the upper 4K of memory, at 0xFFFFF000. See Vector Address Register on page 3-9.

Note

If the VIC is located at a different address, interrupt latency increases.

The following sections describe the connectivity for the two options:

- Standalone interrupt controller
- Daisy-chained interrupt controller on page 2-12.

2.3.1 Standalone interrupt controller

If you use the VIC as a standalone interrupt controller, connect the signals as follows:

- tie nVICIRQIN and nVICFIQIN HIGH
- tie VICVECTADDRIN[31:0] LOW.

Figure 2-7 shows the connections between the VIC and the processor when you use it as a standalone interrupt controller.

![Figure 2-7 Standalone interrupt controller connectivity](image-url)
2.3.2 Daisy-chained interrupt controller

If you use the VIC in a daisy-chain, connect the signals between the VICs as follows:

- \text{nVICIRQIN} on the first VIC to the \text{nVICIRQ} output of the second VIC
- \text{nVICFIQIN} on the first VIC to the \text{nVICFIQ} output of the second VIC
- \text{VICVECTADDRIN}[31:0] on the first VIC to the \text{VICVECTADDROUT}[31:0] of the second VIC.

\textit{Standalone interrupt controller} on page 2-11 describes how to connect the final VIC in the chain, the VIC furthest from the processor.

Figure 2-8 shows the connections between the VICs and the processor when you use them in a daisy-chain.
Daisy-chain interrupt priority

The interrupt priority in Figure 2-8 on page 2-12 is as follows:

1. FIQ
2. VIC0: VIRQ0, VIRQ1-VIRQ15, NonVectIRQ
3. VIC1: VIRQ0, VIRQ1-VIRQ15, NonVectIRQ
4. VICn: ...
Chapter 3
Programmer’s Model

This chapter describes the VIC (PL190) registers. It provides information that you require to program the microcontroller. It contains the following sections:

- About the programmer’s model on page 3-2
- Summary of VIC registers on page 3-3
- Register descriptions on page 3-6
- Interrupt latency on page 3-18
- Interrupt priority on page 3-21.
3.1 About the programmer’s model

To ensure that you can read the vector address register in a single instruction, the VIC base address must be 0xFFFFF000, that is the upper 4K of memory. See Vector Address Register on page 3-9. Placing the VIC anywhere else in memory increases interrupt latency because the ARM processor cannot access the VICVectorAddr Register using a single instruction.

The read, LDR, instruction has a maximum address offset of 12 bits, equivalent to 4K, meaning that it can read from an address up to 4K away from the current address with a single read instruction. If the address to be read from is more than 4K away, you require a second instruction to read in the full address value. This takes longer to perform. When an interrupt occurs, the current address is either the IRQ or FIQ exception vector location, 0x00000018 or 0x0000001C for normal low exception vectors. A 4K offset from the exception address is the upper 4K of memory, so placing the VIC in this area of memory enables the read of the VICADDRESS Register, at 0xFFFFF000, to be performed using an address offset with a single instruction. For example, at location 0x18 LDR pc, [pc, #–0x120] to access VICADDRESS at location 0xFFFFFF00.

If you use a processor that supports high exception vectors, and you tie the HIVECS configuration pin HIGH, you must put the VIC at 0xFFFEF000 to enable the exception vectors that are located at 0xFFFFF000. The VIC is not located at 0x00000000, because this is the standard location for the system memory. The offset of any particular register from the base address is fixed.
### 3.2 Summary of VIC registers

Table 3-1 lists the VIC registers.

#### Table 3-1 VIC register summary

<table>
<thead>
<tr>
<th>Register</th>
<th>Address offset</th>
<th>Type</th>
<th>Reset value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VICIRQSTATUS</td>
<td>0x000</td>
<td>RO</td>
<td>0x00000000</td>
<td>See IRQ Status Register on page 3-6</td>
</tr>
<tr>
<td>VICFIQSTATUS</td>
<td>0x004</td>
<td>RO</td>
<td>0x00000000</td>
<td>See FIQ Status Register on page 3-6</td>
</tr>
<tr>
<td>VICRAWINTR</td>
<td>0x008</td>
<td>RO</td>
<td>-</td>
<td>See Raw Interrupt Status Register on page 3-6</td>
</tr>
<tr>
<td>VICINTSELECT</td>
<td>0x00C</td>
<td>R/W</td>
<td>0x00000000</td>
<td>See Interrupt Select Register on page 3-7</td>
</tr>
<tr>
<td>VICINTENABLE</td>
<td>0x010</td>
<td>R/W</td>
<td>0x00000000</td>
<td>See Interrupt Enable Register on page 3-7</td>
</tr>
<tr>
<td>VICINTENCLEAR</td>
<td>0x014</td>
<td>Write</td>
<td>-</td>
<td>See Interrupt Enable Clear Register on page 3-7</td>
</tr>
<tr>
<td>VICSOFTINT</td>
<td>0x018</td>
<td>R/W</td>
<td>0x00000000</td>
<td>See Software Interrupt Register on page 3-8</td>
</tr>
<tr>
<td>VICSOFTINTCLEAR</td>
<td>0x01C</td>
<td>WO</td>
<td>-</td>
<td>See Software Interrupt Clear Register on page 3-8</td>
</tr>
<tr>
<td>VICPROTECTION</td>
<td>0x020</td>
<td>R/W</td>
<td>0x0</td>
<td>See Protection Enable Register on page 3-8</td>
</tr>
<tr>
<td>VICVECTADDR</td>
<td>0x030</td>
<td>R/W</td>
<td>0x00000000</td>
<td>See Vector Address Register on page 3-9</td>
</tr>
<tr>
<td>VICDEFVECTADDR</td>
<td>0x034</td>
<td>R/W</td>
<td>0x00000000</td>
<td>See Default Vector Address Register on page 3-10</td>
</tr>
<tr>
<td>VICVECTADDR0</td>
<td>0x100</td>
<td>R/W</td>
<td>0x00000000</td>
<td>See Vector Address Registers on page 3-10</td>
</tr>
<tr>
<td>VICVECTADDR1</td>
<td>0x104</td>
<td>R/W</td>
<td>0x00000000</td>
<td></td>
</tr>
<tr>
<td>VICVECTADDR2</td>
<td>0x108</td>
<td>R/W</td>
<td>0x00000000</td>
<td></td>
</tr>
<tr>
<td>VICVECTADDR3</td>
<td>0x10C</td>
<td>R/W</td>
<td>0x00000000</td>
<td></td>
</tr>
<tr>
<td>VICVECTADDR4</td>
<td>0x110</td>
<td>R/W</td>
<td>0x00000000</td>
<td></td>
</tr>
<tr>
<td>VICVECTADDR5</td>
<td>0x114</td>
<td>R/W</td>
<td>0x00000000</td>
<td></td>
</tr>
<tr>
<td>VICVECTADDR6</td>
<td>0x118</td>
<td>R/W</td>
<td>0x00000000</td>
<td></td>
</tr>
<tr>
<td>VICVECTADDR7</td>
<td>0x11C</td>
<td>R/W</td>
<td>0x00000000</td>
<td></td>
</tr>
<tr>
<td>VICVECTADDR8</td>
<td>0x120</td>
<td>R/W</td>
<td>0x00000000</td>
<td></td>
</tr>
<tr>
<td>VICVECTADDR9</td>
<td>0x124</td>
<td>R/W</td>
<td>0x00000000</td>
<td></td>
</tr>
</tbody>
</table>
Table 3-1 VIC register summary (continued)

<table>
<thead>
<tr>
<th>Register</th>
<th>Address offset</th>
<th>Type</th>
<th>Reset value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VICVECTADDR10</td>
<td>0x128</td>
<td>R/W</td>
<td>0x00000000</td>
<td>See Vector Address Registers on page 3-10</td>
</tr>
<tr>
<td>VICVECTADDR11</td>
<td>0x12C</td>
<td>R/W</td>
<td>0x00000000</td>
<td></td>
</tr>
<tr>
<td>VICVECTADDR12</td>
<td>0x130</td>
<td>R/W</td>
<td>0x00000000</td>
<td></td>
</tr>
<tr>
<td>VICVECTADDR13</td>
<td>0x134</td>
<td>R/W</td>
<td>0x00000000</td>
<td></td>
</tr>
<tr>
<td>VICVECTADDR14</td>
<td>0x138</td>
<td>R/W</td>
<td>0x00000000</td>
<td></td>
</tr>
<tr>
<td>VICVECTADDR15</td>
<td>0x13C</td>
<td>R/W</td>
<td>0x00000000</td>
<td></td>
</tr>
<tr>
<td>VICVECTCNTL0</td>
<td>0x200</td>
<td>R/W</td>
<td>0x00</td>
<td>See Vector Control Registers on page 3-10</td>
</tr>
<tr>
<td>VICVECTCNTL1</td>
<td>0x204</td>
<td>R/W</td>
<td>0x00</td>
<td></td>
</tr>
<tr>
<td>VICVECTCNTL2</td>
<td>0x208</td>
<td>R/W</td>
<td>0x00</td>
<td></td>
</tr>
<tr>
<td>VICVECTCNTL3</td>
<td>0x20C</td>
<td>R/W</td>
<td>0x00</td>
<td></td>
</tr>
<tr>
<td>VICVECTCNTL4</td>
<td>0x210</td>
<td>R/W</td>
<td>0x00</td>
<td></td>
</tr>
<tr>
<td>VICVECTCNTL5</td>
<td>0x214</td>
<td>R/W</td>
<td>0x00</td>
<td></td>
</tr>
<tr>
<td>VICVECTCNTL6</td>
<td>0x218</td>
<td>R/W</td>
<td>0x00</td>
<td></td>
</tr>
<tr>
<td>VICVECTCNTL7</td>
<td>0x21C</td>
<td>R/W</td>
<td>0x00</td>
<td></td>
</tr>
<tr>
<td>VICVECTCNTL8</td>
<td>0x220</td>
<td>R/W</td>
<td>0x00</td>
<td></td>
</tr>
<tr>
<td>VICVECTCNTL9</td>
<td>0x224</td>
<td>R/W</td>
<td>0x00</td>
<td></td>
</tr>
<tr>
<td>VICVECTCNTL10</td>
<td>0x228</td>
<td>R/W</td>
<td>0x00</td>
<td></td>
</tr>
<tr>
<td>VICVECTCNTL11</td>
<td>0x22C</td>
<td>R/W</td>
<td>0x00</td>
<td></td>
</tr>
<tr>
<td>VICVECTCNTL12</td>
<td>0x230</td>
<td>R/W</td>
<td>0x00</td>
<td></td>
</tr>
<tr>
<td>VICVECTCNTL13</td>
<td>0x234</td>
<td>R/W</td>
<td>0x00</td>
<td></td>
</tr>
<tr>
<td>VICVECTCNTL14</td>
<td>0x238</td>
<td>R/W</td>
<td>0x00</td>
<td></td>
</tr>
<tr>
<td>VICVECTCNTL15</td>
<td>0x23C</td>
<td>R/W</td>
<td>0x00</td>
<td></td>
</tr>
<tr>
<td>VICPERIPHID0</td>
<td>0xFE0</td>
<td>RO</td>
<td>0x90</td>
<td>See Peripheral Identification Registers on page 3-11</td>
</tr>
</tbody>
</table>

See Vector Address Registers on page 3-10

See Vector Control Registers on page 3-10

See Peripheral Identification Registers on page 3-11
### Table 3-1 VIC register summary (continued)

<table>
<thead>
<tr>
<th>Register</th>
<th>Address offset</th>
<th>Type</th>
<th>Reset value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VICPERIPHID1</td>
<td>0xFE4</td>
<td>RO</td>
<td>0x11</td>
<td>See Peripheral Identification Registers on page 3-11</td>
</tr>
<tr>
<td>VICPERIPHID2</td>
<td>0xFE8</td>
<td>RO</td>
<td>0x04</td>
<td></td>
</tr>
<tr>
<td>VICPERIPHID3</td>
<td>0xFEC</td>
<td>RO</td>
<td>0x00</td>
<td></td>
</tr>
<tr>
<td>VICCELLID0</td>
<td>0xFF0</td>
<td>RO</td>
<td>0x0D</td>
<td>See PrimeCell Identification Registers on page 3-14</td>
</tr>
<tr>
<td>VICCELLID1</td>
<td>0xFF4</td>
<td>RO</td>
<td>0xF0</td>
<td></td>
</tr>
<tr>
<td>VICCELLID2</td>
<td>0xFF8</td>
<td>RO</td>
<td>0x05</td>
<td></td>
</tr>
<tr>
<td>VICCELLID3</td>
<td>0xFFC</td>
<td>RO</td>
<td>0xB1</td>
<td></td>
</tr>
</tbody>
</table>
3.3 Register descriptions

This section describes the VIC registers.

3.3.1 IRQ Status Register

The read-only VICIRQSTATUS Register, with address offset of 0x000, provides the status of interrupts [31:0] after IRQ masking. Table 3-2 lists the bit assignments for this register.

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>[31:0]</td>
<td>IRQStatus</td>
<td>Shows the status of the interrupts after masking by the VICINTENABLE and VICINTSELECT Registers. A HIGH bit indicates that the interrupt is active, and generates an interrupt to the processor.</td>
</tr>
</tbody>
</table>

3.3.2 FIQ Status Register

The read-only VICFIQSTATUS Register, with address offset of 0x004, provides the status of the interrupts after FIQ masking. Table 3-3 lists the bit assignments for this register.

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>[31:0]</td>
<td>FIQStatus</td>
<td>Shows the status of the interrupts after masking by the VICINTENABLE and VICINTSELECT Registers. A HIGH bit indicates that the interrupt is active, and generates an interrupt to the processor.</td>
</tr>
</tbody>
</table>

3.3.3 Raw Interrupt Status Register

The read-only VICRAWINTR Register, with address offset of 0x008, provides the status of the source interrupts, and software interrupts, to the interrupt controller. Table 3-4 lists the bit assignments for this register.

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>[31:0]</td>
<td>RawInterrupt</td>
<td>Shows the status of the interrupts before masking by the enable registers. A HIGH bit indicates that the appropriate interrupt request is active before masking.</td>
</tr>
</tbody>
</table>
3.3.4 Interrupt Select Register

The read/write VICINTSELECT Register, with address offset of 0x00C, selects whether the corresponding interrupt source generates an FIQ or an IRQ interrupt. Table 3-5 lists the bit assignments for this register.

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>[31:0]</td>
<td>IntSelect</td>
<td>Selects the type of interrupt for interrupt requests:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = FIQ interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = IRQ interrupt</td>
</tr>
</tbody>
</table>

3.3.5 Interrupt Enable Register

The read/write VICINTENABLE Register, with address offset of 0x010, enables the interrupt request lines, by masking the interrupt sources for the IRQ interrupt. Table 3-6 lists the bit assignments for this register.

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>[31:0]</td>
<td>IntEnable</td>
<td>Enables the interrupt request lines:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = Interrupt enabled. Enables interrupt request to processor.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = Interrupt disabled.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>On reset, all interrupts are disabled. A HIGH bit sets the corresponding bit in the VICINTENABLE Register. A LOW bit has no effect.</td>
</tr>
</tbody>
</table>

3.3.6 Interrupt Enable Clear Register

The write-only VICINTENCLEAR Register, with address offset of 0x014, clears bits in the VICIntEnable Register. Table 3-7 lists the bit assignments for this register.

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>[31:0]</td>
<td>IntEnable Clear</td>
<td>Clears bits in the VICINTENABLE Register. A HIGH bit clears the corresponding bit in the VICINTENABLE Register. A LOW bit has no effect.</td>
</tr>
</tbody>
</table>
3.3.7 Software Interrupt Register

The read/write VICSOFTINT Register, with address offset of 0x018, generates software interrupts. Table 3-8 lists the bit assignments for this register.

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>[31:0]</td>
<td>SoftInt</td>
<td>Setting a bit generates a software interrupt for the specific source interrupt before interrupt masking. A HIGH bit sets the corresponding bit in the VICSOFTINT Register. A LOW bit has no effect.</td>
</tr>
</tbody>
</table>

3.3.8 Software Interrupt Clear Register

The write-only VICSOFTINTCLEAR Register, with address offset of 0x01C, clears bits in the VICSoftInt Register. Table 3-9 lists the bit assignments for this register.

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>[31:0]</td>
<td>SoftIntClear</td>
<td>Clears bits in the VICSOFTINT Register. A HIGH bit clears the corresponding bit in the VICSOFTINT Register. A LOW bit has no effect.</td>
</tr>
</tbody>
</table>

3.3.9 Protection Enable Register

The read/write VICPROTECTION Register, with address offset of 0x020, enables or disables protected register access. Figure 3-1 shows the bit assignments for this register.

![Figure 3-1 VICPROTECTION Register bit assignments](image-url)
Table 3-10 lists the bit assignments for this register.

### Table 3-10 VICPROTECTION Register bit assignments

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>[31:1]</td>
<td>-</td>
<td>Read undefined. Write as zero.</td>
</tr>
<tr>
<td>[0]</td>
<td>Protection</td>
<td>Enables or disables protected register access. When enabled, only privileged mode accesses, reads and writes, can access the interrupt controller registers. When disabled, both User mode and Privileged mode can access the registers. This register is cleared on reset, and can only be accessed in privileged mode.</td>
</tr>
</tbody>
</table>

---

**Note**

If the bus master cannot generate accurate protection information, leave this register in its reset state to enable User mode access.

### 3.3.10 Vector Address Register

The read/write VICVECTADDR Register, with address offset of 0x030, contains the *Interrupt Service Routine* (ISR) address of the currently active interrupt. Table 3-11 lists the bit assignments for this register.

### Table 3-11 VICVECTADDR Register bit assignments

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>[31:0]</td>
<td>VectorAddr</td>
<td>Contains the address of the currently active ISR. Any writes to this register clear the interrupt.</td>
</tr>
</tbody>
</table>

---

**Note**

Reading from this register provides the address of the ISR, and indicates to the priority hardware that the interrupt is being serviced. Writing to this register indicates to the priority hardware that the interrupt has been serviced. You must use this register as follows:

- the ISR reads the VICVectAddr Register when an IRQ interrupt is generated
- at the end of the ISR, the VICVectAddr Register is written to, to update the priority hardware.

Reading from or writing to the register at other times can cause incorrect operation.
3.3.11 Default Vector Address Register

The read/write VICDEFVECTADDR Register, with address offset of 0x034, contains the default ISR address. Table 3-12 lists the bit assignments for this register.

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>[31:0]</td>
<td>Default VectorAddr</td>
<td>Contains the address of the default ISR handler</td>
</tr>
</tbody>
</table>

3.3.12 Vector Address Registers

The read/write VICVECTADDR[0-15] Registers span address locations 0x100-0x13C and contain the ISR vector addresses. Table 3-13 lists the bit assignments for these registers.

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>[31:0]</td>
<td>VectorAddr 0-15</td>
<td>Contains ISR vector addresses</td>
</tr>
</tbody>
</table>

3.3.13 Vector Control Registers

The read/write VICVECTCNTL[0-15] Registers span address locations 0x200-0x23C and select the interrupt source for the vectored interrupt. Figure 3-2 shows the bit assignments for these registers.

![Figure 3-2 VICVECTCNTL Register bit assignments](image)
Table 3-14 lists the bit assignments for these registers.

**Table 3-14 VICVECTCNTL Registers bit assignments**

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>[5]</td>
<td>E</td>
<td>Enables vector interrupt. This bit is cleared on reset.</td>
</tr>
<tr>
<td>[4:0]</td>
<td>IntSource</td>
<td>Selects interrupt source. You can select any of the 32 interrupt sources.</td>
</tr>
</tbody>
</table>

**Note**

Vectored interrupts are only generated if the interrupt is enabled. The specific interrupt is enabled in the VICIntEnable Register, and the interrupt is set to generate an IRQ interrupt in the VICIntSelect Register. This prevents multiple interrupts being generated from a single request if the controller is incorrectly programmed.

### 3.3.14 Peripheral Identification Registers

The read-only VICPeriphID0-3 Registers are four 8-bit registers, that span address locations 0xFE0-0xFEC. You can treat the registers conceptually as a single 32-bit register. The read-only registers provide the following options for the peripheral:

**Part number [11:0]**

This identifies the peripheral. The VIC uses the three-digit product code 0x90.

**Designer [19:12]**

This is the identification of the designer. ARM Limited is 0x41, ASCII A.

**Revision number [23:20]**

This is the revision number of the peripheral. The revision number starts from 0.

**Configuration [31:24]**

This is the configuration option of the peripheral. The configuration value is 0.

Figure 3-3 on page 3-12 shows the bit assignments for these registers.
The following sections describe the four 8-bit Peripheral Identification Registers:

- **VICPERIPHID0 Register**
- **VICPERIPHID1 Register** on page 3-13
- **VICPERIPHID2 Register** on page 3-13
- **VICPERIPHID3 Register** on page 3-14.

**VICPERIPHID0 Register**

The read-only VICPERIPHID0 Register, with address offset of 0xFE0, is hard-coded, and the fields within the register determine the reset value. Figure 3-4 shows the bit assignments for this register.

Table 3-15 lists the bit assignments for this register.

![Figure 3-4 VICPERIPHID0 Register bit assignments](image)

Table 3-15 VICPERIPHID0 Register bit assignments

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>[31:8]</td>
<td>-</td>
<td>Read undefined</td>
</tr>
<tr>
<td>[7:0]</td>
<td>Partnumber0</td>
<td>These bits read back as 0x90</td>
</tr>
</tbody>
</table>
VICPERIPHID1 Register

The read-only VICPERIPHID1 Register, with address offset of 0xFE4, is hard-coded, and the fields within the register determine the reset value. Figure 3-5 shows the bit assignments for this register.

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>[31:8]</td>
<td>-</td>
<td>Read undefined</td>
</tr>
<tr>
<td>[7:4]</td>
<td>Designer0</td>
<td>These bits read back as 0x1</td>
</tr>
<tr>
<td>[3:0]</td>
<td>Partnumber1</td>
<td>These bits read back as 0x1</td>
</tr>
</tbody>
</table>

Figure 3-5 VICPERIPHID1 Register bit assignments

Table 3-16 lists the bit assignments for this register.

VICPERIPHID2 Register

The read-only VICPERIPHID2 Register, with address offset of 0xFE8, is hard-coded and the fields within the register determine the reset value. Figure 3-6 shows the bit assignments for this register.

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>[31]</td>
<td>-</td>
<td>Read undefined</td>
</tr>
<tr>
<td>[8:3]</td>
<td>Revision</td>
<td></td>
</tr>
<tr>
<td>[2:0]</td>
<td>Designer1</td>
<td></td>
</tr>
</tbody>
</table>

Figure 3-6 VICPERIPHID2 Register bit assignments
Table 3-17 lists the bit assignments for this register.

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>[31:8]</td>
<td></td>
<td>Read undefined</td>
</tr>
<tr>
<td>[7:4]</td>
<td>Revision</td>
<td>These bits read back as 0x1</td>
</tr>
<tr>
<td>[3:0]</td>
<td>Designer1</td>
<td>These bits read back as 0x0</td>
</tr>
</tbody>
</table>

### VICPERIPHID3 Register

The read-only VICPERIPHID3 Register, with address offset of 0xFEC, is hard-coded and the fields within the register determine the reset value. Figure 3-7 shows the bit assignments for this register.

![Figure 3-7 VICPERIPHID3 Register bit assignments](image)

Table 3-18 lists the bit assignments for this register.

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>[31:8]</td>
<td></td>
<td>Read undefined</td>
</tr>
<tr>
<td>[7:0]</td>
<td>Configuration</td>
<td>These bits read back as 0x0</td>
</tr>
</tbody>
</table>

### PrimeCell Identification Registers

The read-only VICPCELLID0-3 Registers are four 8-bit registers that span address locations 0xFF0-0xFFC. You can treat them conceptually as a single 32-bit register. Use the register as a standard cross-peripheral identification system. on page 3-15 shows the bit assignment for these registers.
VICPCELLID0 Register

The read-only VICPCELLID0 Register, with address offset of 0xFF0, is hard-coded and the fields within the register determine the reset value. Figure 3-9 shows the bit assignments for this register.

Table 3-19 VICPCELLID0 Register bit assignments

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>[31:8]</td>
<td>-</td>
<td>Read undefined</td>
</tr>
<tr>
<td>[7:0]</td>
<td>VICPCellID0</td>
<td>These bits read back as 0x0D</td>
</tr>
</tbody>
</table>

VICPCELLID1 Register

The read-only VICPCELLID1 Register, with address offset of 0xFF4, is hard-coded and the fields within the register determine the reset value. Figure 3-10 on page 3-16 shows the bit assignments for this register.
Figure 3-10 VICPCELLID1 Register bit assignments

Table 3-20 lists the bit assignments for this register.

Table 3-20 VICPCELLID1 Register bit assignments

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>[31:8]</td>
<td>-</td>
<td>Read undefined</td>
</tr>
<tr>
<td>[7:0]</td>
<td>VICPCellID1</td>
<td>These bits read back as 0xF0</td>
</tr>
</tbody>
</table>

VICPCELLID2 Register

The read-only VICPCELLID2 Register, with address offset of 0xFF8, is hard-coded and the fields within the register determine the reset value. Figure 3-11 shows the bit assignments for this register.

Figure 3-11 VICPCELLID2 Register bit assignments

Table 3-21 lists the bit assignments for this register.

Table 3-21 VICPCELLID2 Register bit assignments

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>[31:8]</td>
<td>-</td>
<td>Read undefined</td>
</tr>
<tr>
<td>[7:0]</td>
<td>VICPCellID2</td>
<td>These bits read back as 0x05</td>
</tr>
</tbody>
</table>

VICPCELLID3 Register

The read-only VICPCELLID3 Register, with address offset of 0xFFC, is hard-coded and the fields within the register determine the reset value. Figure 3-12 on page 3-17 shows the bit assignments for this register.
Figure 3-12 VICPCELLID3 Register bit assignments

Table 3-22 lists the bit assignments for this register.

Table 3-22 VICPCELLID3 Register bit assignments

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Type</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>[31:8]</td>
<td>-</td>
<td>-</td>
<td>Read undefined</td>
</tr>
<tr>
<td>[7:0]</td>
<td>VICPCellID3</td>
<td>RO</td>
<td>These bits read back as 0x81</td>
</tr>
</tbody>
</table>
3.4 Interrupt latency

The calculations in this section show the number of cycles required to service interrupts, using the following types of interrupt:

- **FIQ interrupts**
- **IRQ interrupts** on page 3-19
- **Fast IRQ interrupts** on page 3-19
- **Daisy-chained interrupts** on page 3-20.

--- Note

The calculations are based on the assumption that the ISRs are in zero wait state memory.

---

3.4.1 FIQ interrupts

FIQ interrupts have the highest priority in the VIC, and are not nested. In FIQ mode, seven 32-bit registers are banked into the system. This enables the VIC to process the interrupt as quickly as possible. Table 3-23 lists the worst case cycles for FIQ interrupts.

<table>
<thead>
<tr>
<th>Event</th>
<th>Worst case (cycles)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt synchronization.</td>
<td>3</td>
</tr>
<tr>
<td>Worst case instruction execution. This assumes that a standard switch reduces STM and LDM. You can reduce this to 7 cycles to avoid data aborts.</td>
<td>7</td>
</tr>
<tr>
<td>Entry to first instruction.</td>
<td>2</td>
</tr>
<tr>
<td>Total.</td>
<td>12</td>
</tr>
</tbody>
</table>

--- Note

For the best results, start the FIQ handler at the FIQ vector address, 0x1c.
3.4.2 IRQ interrupts

In IRQ mode, you can nest interrupt levels lower than the highest priority FIQ interrupt level. To provide this nesting, the return address, stored in the Link Register (LR), and the status register, stored in the Saved Processor Status Register (SPSR) must be available before more IRQ interrupts can be accepted. This increases the interrupt latency, but provides a scalable nested interrupt system. Table 3-24 lists the worst case cycles for IRQ interrupts.

<table>
<thead>
<tr>
<th>Event</th>
<th>Worst case (cycles)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt synchronization</td>
<td>3</td>
</tr>
<tr>
<td>Worst case interrupt disable period</td>
<td>10</td>
</tr>
<tr>
<td>Entry to first instruction</td>
<td>2</td>
</tr>
<tr>
<td>Nesting, assuming single-state AHB</td>
<td>10</td>
</tr>
<tr>
<td>Total</td>
<td>25</td>
</tr>
</tbody>
</table>

3.4.3 Fast IRQ interrupts

Fast IRQ mode is similar to IRQ mode, except that the highest-level IRQ interrupt handler assumes that no other IRQ interrupt occurs during its operation, and therefore, you do not require LR and SPSR. Table 3-25 lists the worst case cycles for fast IRQ interrupts.

<table>
<thead>
<tr>
<th>Event</th>
<th>Worst case (cycles)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt synchronization</td>
<td>3</td>
</tr>
<tr>
<td>Worst case interrupt disable period</td>
<td>10</td>
</tr>
<tr>
<td>Entry to first instruction</td>
<td>2</td>
</tr>
<tr>
<td>Load IRQ vector into PC</td>
<td>5</td>
</tr>
<tr>
<td>Total</td>
<td>20</td>
</tr>
</tbody>
</table>
3.4.4 Daisy-chained interrupts

Because of the additional read required to read both the primary VICVectAddr Register and the daisy-chained VICVectAddr Register, the worst-case latency of the primary VIC increases by one cycle, to 26 cycles. The worst-case latency for the secondary, daisy-chained VIC increases by two cycles, to 27 cycles. This latency applies to any number of secondary VICS. See Daisy-chained vectored interrupt service routine on page B-6 for more information.
3.5 Interrupt priority

The hardware regulates the interrupt priority. FIQ interrupts have the highest priority, followed by vectored interrupt 0 to vectored interrupt 15. Non-vectored interrupts have the lowest priority.

To reduce interrupt latency, you can re-enable the IRQ interrupts in the processor after the Interrupt Service Routine (ISR) is entered. See Interrupt latency on page 3-18. In this case, the current ISR is interrupted, and the higher-priority ISR is executed. The VIC then only enables a higher priority interrupt than the interrupt currently being serviced. If a higher priority interrupt goes active, the current ISR is interrupted and the higher-priority ISR is executed.

Before the interrupt enable bits in the processor can be re-enabled, the LR and SPSR must be saved, preferably on a software stack. When the ISR is exited, you must disable the interrupts, reload the LR and SPSR, and write to the Vector Address Register, VICVectAddr. See Vectored interrupt service routine on page B-6.

When you daisy-chain VICs, the interrupt priority is as follows:

- FIQ interrupts
- primary VIC vectored interrupts
- primary VIC non-vectored interrupts
- daisy-chained VIC vectored interrupts
- daisy-chained VIC non-vectored interrupts.
Chapter 4
Programmer’s Model for Test

This chapter describes the additional logic for functional verification and provisions made for production testing. It contains the following sections:

- VIC test harness overview on page 4-2
- Scan testing on page 4-3
- Summary of test registers on page 4-4.
4.1 VIC test harness overview

The additional logic for functional verification and production testing enables:

- capture of input signals to the block
- stimulation of the output signals.

The integration vectors provide a way of verifying that the VIC is correctly wired into a system. Do this by separately testing two groups of signals:

AMBA signals

Test these by checking the connections of all the address and data bits.

Intra-chip signals

The tests for these signals are system-specific, and enable you to write the necessary tests. Additional logic is implemented enabling you to read/write to each intra-chip input/output signal.

Test registers control these test features, and enable you to test the VIC in isolation from the rest of the system using only transfers from the AMBA AHB.

Off-chip test vectors are supplied using a 32-bit parallel *External Bus Interface* (EBI) and converted to internal AMBA bus transfers. The *Test Interface Controller* (TIC) AMBA bus master module controls the application of test vectors.
4.2 Scan testing

The VIC simplifies:

- insertion of scan test cells
- use of *Automatic Test Pattern Generation* (ATPG).

This provides an alternative method of manufacturing test.
4.3 Summary of test registers

Table 4-1 lists how the VIC test registers are memory-mapped.

<table>
<thead>
<tr>
<th>Register</th>
<th>Address offset</th>
<th>Type</th>
<th>Reset value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VICITCR</td>
<td>0x300</td>
<td>R/W</td>
<td>-</td>
<td>See Test Control Register</td>
</tr>
<tr>
<td>VICITIP1</td>
<td>0x304</td>
<td>RO</td>
<td>0x0</td>
<td>See Integration Test Input Registers on page 4-5</td>
</tr>
<tr>
<td>VICITIP2</td>
<td>0x308</td>
<td>RO</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>VICITOP1</td>
<td>0x30C</td>
<td>RO</td>
<td>0x0</td>
<td>See Integration Test Output Registers on page 4-6</td>
</tr>
<tr>
<td>VICITOP2</td>
<td>0x310</td>
<td>RO</td>
<td>0x00000000</td>
<td></td>
</tr>
</tbody>
</table>

4.3.1 Test Control Register

The read/write VICITCR Register, with address offset of 0x300, is a single-bit test control register. The ITEN bit in this register controls the input test multiplexors. Figure 4-1 shows the bit assignments for this register.

![Figure 4-1 VICITCR Register bit assignments](image)

Table 4-2 lists the bit assignments for this register.

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[31:1]</td>
<td>-</td>
<td>Read undefined. Write as zero.</td>
</tr>
<tr>
<td>[0]</td>
<td>ITEN</td>
<td>Integration test enable:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = normal mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = test mode.</td>
</tr>
</tbody>
</table>
### 4.3.2 Integration Test Input Registers

The read-only VICITIP1 Register, with address offset of 0x304, is a 2-bit register that returns the values of the `nVICIRQIN` and `nVICFIQIN` inputs. Figure 4-2 shows the bit assignments for this register.

![VICITIP1 Register bit assignments](image)

Table 4-3 lists the bit assignments for this register.

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[31:8]</td>
<td>-</td>
<td>Read undefined</td>
</tr>
<tr>
<td>[7]</td>
<td>nVICIRQIN</td>
<td>Reads return the value on <code>nVICIRQIN</code> when the VICITCR Register is LOW</td>
</tr>
<tr>
<td>[6]</td>
<td>nVICFIQIN</td>
<td>Reads return the value on <code>nVICFIQIN</code> when the VICITCR Register is LOW</td>
</tr>
<tr>
<td>[5:0]</td>
<td>-</td>
<td>Read undefined</td>
</tr>
</tbody>
</table>

The VICITIP2 Register, with address offset of 0x308, is a read-only register. It is a 32-bit register that returns the value of the `VICVECTADDRIN` input. Table 4-4 lists the bit assignments for this register.

Table 4-4 VICITIP2 Register bit assignments

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[31:0]</td>
<td>VICVECTADDRIN</td>
<td>Reads return the value on <code>VICVECTADDRIN</code> when the VICITCR Register is LOW.</td>
</tr>
</tbody>
</table>
4.3.3 Integration Test Output Registers

The read-only VICITOP1 Register, with address offset of 0x30C, is a 32-bit register that controls the nVICIRQ and nVICFIQ outputs. Figure 4-3 shows the bit assignments for this register.

Table 4-5 VICITOP1 Register bit assignments

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[31:8]</td>
<td>-</td>
<td>Read undefined.</td>
</tr>
<tr>
<td>[7]</td>
<td>VICIRQ</td>
<td>Reads return the value on the internal VICIRQ line. This is the pre-inverted version of the final output, and is inverted to create the final nVICIRQ output.</td>
</tr>
<tr>
<td>[6]</td>
<td>VICFIQ</td>
<td>Reads return the value on the internal VICFIQ line. This is the pre-inverted version of the final output, and is inverted to create the final nVICFIQ output.</td>
</tr>
<tr>
<td>[5:0]</td>
<td>-</td>
<td>Read undefined.</td>
</tr>
</tbody>
</table>

The read-only VICITOP2 Register, with address offset of 0x310, is a 32-bit register that controls the VICVECTADDROUT output. Table 4-6 lists the bit assignments for this register.

Table 4-6 VICITOP2 Register bit assignments

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[31:0]</td>
<td>VICVECTADDROUT</td>
<td>Reads return the value on the VICVECTADDROUT lines.</td>
</tr>
</tbody>
</table>
Appendix A
Signal Descriptions

This appendix describes the signals that interface with the ARM PrimeCell Vectored Interrupt Controller (PL190). It contains the following sections:

- AMBA AHB signals on page A-2
- Interrupt controller signals on page A-3
- Daisy-chain signals on page A-4
- Scan test control signals on page A-5.
A.1 AMBA AHB signals

The VIC module is connected to the AMBA AHB as a bus slave. Table A-1 lists the AHB signals that are used and produced.

Table A-1 AMBA AHB signal descriptions

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Source/destination</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HCLK</td>
<td>Input</td>
<td>Clock source</td>
<td>AMBA AHB bus clock. Times all bus transfers. All signal timings are related to the rising edge of HCLK.</td>
</tr>
<tr>
<td>HRESETn</td>
<td>Input</td>
<td>Reset controller</td>
<td>AHB bus reset, active LOW.</td>
</tr>
<tr>
<td>HTRANS</td>
<td>Input</td>
<td>Master</td>
<td>Transfer type. This can be NONSEQUENTIAL, SEQUENTIAL, IDLE, or BUSY. You must connect this signal to HTRANS[1] on the AHB interface. HTRANS[0] is not used.</td>
</tr>
<tr>
<td>HWRITE</td>
<td>Input</td>
<td>Master</td>
<td>Transfer direction. Indicates a write transfer when HIGH, and a read transfer when LOW.</td>
</tr>
<tr>
<td>HSIZE[2:0]</td>
<td>Input</td>
<td>Master</td>
<td>Size of the transfer. This must be a word, 32-bit, for the VIC, HSIZE[2:0] = 0b010.</td>
</tr>
<tr>
<td>HPROT</td>
<td>Input</td>
<td>Master</td>
<td>Memory access protection type. This can be User mode (0) or privileged mode (1). You must connect this signal to HPROT[1] on the AHB interface. HPROT[3], HPROT[2] and HPROT[0] are not used.</td>
</tr>
<tr>
<td>HWDATA[31:0]</td>
<td>Input</td>
<td>Master</td>
<td>Write data bus. Transfers data from bus master to bus slaves during write operations.</td>
</tr>
<tr>
<td>HSELVIC</td>
<td>Input</td>
<td>Decoder</td>
<td>Slave select signal. This is a combinatorial decode of the address bus. It indicates that the current transfer is intended for the selected slave.</td>
</tr>
<tr>
<td>HRDATA[31:0]</td>
<td>Output</td>
<td>Slave</td>
<td>Read data bus. Transfers data from bus slaves to bus master during read operations.</td>
</tr>
<tr>
<td>HREADYIN</td>
<td>Input</td>
<td>External slave</td>
<td>Transfer done signal, generated by an alternate slave. When HIGH, indicates that a transfer is complete. You can drive it LOW to extend a transfer.</td>
</tr>
<tr>
<td>HREADYOUT</td>
<td>Output</td>
<td>Slave</td>
<td>Transfer done signal, generated by the VIC. When HIGH, indicates that a transfer is complete. You can drive it LOW to extend a transfer.</td>
</tr>
<tr>
<td>HRESP[1:0]</td>
<td>Output</td>
<td>Slave</td>
<td>Transfer response. This provides additional transfer status information. The response can be OKAY, ERROR, RETRY, or SPLIT. The VIC responds with either OKAY or ERROR.</td>
</tr>
</tbody>
</table>
## A.2 Interrupt controller signals

Table A-2 lists the signals for the VIC that interface to the processor interrupt sources.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Source/destination</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VICINTSOURCE [31:0]</td>
<td>Input</td>
<td>Peripheral interrupt request</td>
<td>Interrupt source input</td>
</tr>
<tr>
<td>nVICIRQ</td>
<td>Output</td>
<td>Interrupt controller</td>
<td>Interrupt request to processor</td>
</tr>
<tr>
<td>nVICFIQ</td>
<td>Output</td>
<td>Interrupt controller</td>
<td>Fast interrupt request to processor</td>
</tr>
</tbody>
</table>
A.3 Daisy-chain signals

You use daisy-chain signals when two or more VICs are daisy-chained. See Daisy-chained interrupt controller on page 2-12. Table A-3 lists the daisy-chain signals.

Table A-3 Daisy-chain signals

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Source/destination</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VICVECTADDRIN[31:0]</td>
<td>Input</td>
<td>External interrupt controller</td>
<td>Connects to the VICVECTADDROUT[31:0] signal of the previous VIC if you use daisy-chaining. Connects to logic 0 if the VIC is not daisy-chained.</td>
</tr>
<tr>
<td>VICVECTADDROUT[31:0]</td>
<td>Output</td>
<td>Interrupt controller</td>
<td>Connects to the VICVECTADDRIN[31:0] signal of the next VIC if you use daisy-chaining. Left unconnected if the VIC is not daisy-chained.</td>
</tr>
<tr>
<td>nVICIRQIN</td>
<td>Input</td>
<td>External interrupt controller</td>
<td>Connects to the nVICIRQ signal of the previous VIC if you use daisy-chaining. Connects to logic 1 if the VIC is the last in the daisy-chain, or if VIC is not daisy-chained.</td>
</tr>
<tr>
<td>nVICFIQIN</td>
<td>Input</td>
<td>External interrupt controller</td>
<td>Connects to the nVICFIQ signal of the previous VIC if you use daisy-chaining. Connects to logic 1 if the VIC is the last in the daisy-chain, or if VIC is not daisy-chained.</td>
</tr>
</tbody>
</table>
A.4 Scan test control signals

Table A-4 lists the internal scan test control signals.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Source/destination</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCANENABLE</td>
<td>Input</td>
<td>Scan controller</td>
<td>Scan enable</td>
</tr>
<tr>
<td>SCANINHCLK</td>
<td>Input</td>
<td>Scan controller</td>
<td>Scan data input for HCLK domain</td>
</tr>
<tr>
<td>SCANOUTHCLK</td>
<td>Output</td>
<td>Scan controller</td>
<td>Scan data output for HCLK domain</td>
</tr>
</tbody>
</table>
Appendix B
Example Code

This appendix provides examples of the code required when setting up the ARM PrimeCell Vectored Interrupt Controller (PL190). It contains the following section:

- *About the example code* on page B-2.
B.1 About the example code

This section provides the following examples of code:

- **Enable interrupts**
- **Disable interrupts**
- **Interrupt polling** on page B-3
- **Generate software interrupt** on page B-3
- **Clear software interrupt** on page B-3
- **FIQ interrupt initialization** on page B-4
- **FIQ interrupt handler** on page B-4
- **Simple interrupt initialization** on page B-4
- **Simple interrupt service routine** on page B-5
- **Vectored interrupt initialization** on page B-5
- **Vectored interrupt service routine** on page B-6
- **Daisy-chained vectored interrupt service routine** on page B-6
- **Highest level vectored IRQ interrupt service routine** on page B-7.

### B.1.1 Enable interrupts

Example B-1 gives an example of the enable interrupt code.

**Example B-1 Enable interrupts**

```
LDR r0, =IntCntlBase                      ; where IntCntlBase is a predefined constant
MOV r1, #<interrupt to enable>           ; for example, IntCntlBase EQU 0xFFFFF000
STR r1, [r0, #IntEnableOffset]
```

### B.1.2 Disable interrupts

Example B-2 gives an example of the disable interrupt code.

**Example B-2 Disable interrupts**

```
LDR r0, =IntCntlBase
MOV r1, #<interrupt to disable>
STR r1, [r0, #IntEnableClearOffset]
```
B.1.3  Interrupt polling

Example B-3 gives an example of the interrupt polling code.

Example B-3 Interrupt polling

LDR   r0, =IntCntlBase
Loop  LDR   r1, [r0, #RawInterruptOffset]
       CMP  r1, #0
       BEQ    loop
       ; Scan r1 for source of interrupt & branch to relevant routine

B.1.4  Generate software interrupt

Example B-4 gives an example of the generate software interrupt code.

Example B-4 Generate software interrupt

; Generate software interrupt on interrupt request line 1
LDR    r0, =IntCntlBase
MOV    r1, #2 ; Interrupt source/request 1
STR    r1, [r0, #SoftIntOffset]

B.1.5  Clear software interrupt

Example B-5 gives an example of the clear software interrupt code.

Example B-5 Clear software interrupt

; Clear software interrupt on interrupt request line 1.
LDR    r0, =IntCntlBase ; where IntCntlBase is a predefined constant,  
MOV    r1, #2 ; for example, IntCntlBase EQU 0xFFFFF000
STR    r1, [r0, #SoftIntClearOffset]
B.1.6  FIQ interrupt initialization

Example B-6 gives an example of the FIQ interrupt initialization code.

Example B-6 FIQ interrupt initialization

LDR    r0, =IntCnt1Base
MOV    r1, #<interrupt_to_enable>
STR    r1, [r0, #IntSelectOffset] ; Select FIQ interrupt and clear other FIQs
STR    r1, [r0, #IntEnableOffset]     ; Enable interrupt
MRS    CPSR_c, #(DISABLE_IRQ + MODE_SYS_32) ; Enable FIQ interrupts

B.1.7  FIQ interrupt handler

Example B-7 gives an example of the FIQ interrupt handler code.

Example B-7 FIQ interrupt handler

; IRQ and FIQ interrupts are automatically masked until return from interrupt performed

0x1c ; Interrupt service routine
; Clear interrupt request
SUBS    pc, r14, #4

B.1.8  Simple interrupt initialization

Example B-8 shows how you can use the interrupt controller without using vectored interrupts, or the interrupt priority hardware. For example, you can use it for debugging.

Example B-8 Simple interrupt initialization

LDR    r0, =IntCnt1Base
MOV    r1, #<interrupt_to_enable>
LDR    r2, [r0, #IntSelectOffset] ; Select IRQ interrupt
BIC    r2, r2, r1
STR    r2, [r0, #IntSelectOffset]
STR    r1, [r0, #IntEnableOffset] ; Enable interrupt
MRS    CPSR_c, #(DISABLE_IRQ + MODE_SYS_32) ; Enable FIQ interrupts
**B.1.9 Simple interrupt service routine**

Example B-9 shows how you can use the interrupt controller without using vectored interrupts, or the interrupt priority hardware. For example, you can use it for debugging.

**Example B-9 Simple interrupt service routine**

```assembly
; This interrupt service routine assumes that there are no vectored interrupts. It also
; assumes that interrupts are disabled until the interrupt service routine has been exited.

; IRQ interrupts are masked until a return from interrupt is performed
; The FIQ interrupt is enabled

0x18   B      IRQ_ISR                        ; Branch to interrupt service routine

IRQ_ISR

STMFD  sp!, {r0, r1}                  ; Store r0 and r1
LDR    r0, [IntCntlBase]              ; Discover source of interrupt
LDR    r1, [r0, #IRQStatusOffset]    ; Scan r1 for source of interrupt & branch to relevant routine ISR

ISR

Interrupt service routine

Clear interrupt request

LDMFD  sp!, {r0, r1}                  ; Restore r0 and r1
SUBS   pc, r14, #4                   ; Exit from IRQ
```

**B.1.10 Vectored interrupt initialization**

Example B-10 gives an example of the vectored interrupt initialization code.

**Example B-10 Vectored interrupt initialization**

```assembly
LDR    r0, =IntCntlBase
MOV    r1, #<interrupt_to_enable>
STR    r1, [r0, #IntEnableClearOffset] ; Disable interrupt

LDR    r2, =default_vector_address   ; Set default vector address
STR    r2, [r0, #DefaultVectorAddressOffset] ; Setup and enable vectored interrupt 15
MOV    r2, #vector_address          ; Set vector address
STR    r2, [r0, #VectorAddr15Offset]
MOV    r2, #interrupt_source       ; Set interrupt source
ORR    r2, r2, #0x20                ; and enabled vector interrupt
STR    r2, [r0, #VectorCntl15Offset]
LDR    r2, [r0, #IntSelectOffset]   ; Select IRQ interrupt
```
B.1.11 Vectored interrupt service routine

Example B-11 gives an example of the vectored interrupt service routine code.

Example B-11 Vectored interrupt service routine

Example Code

```
BIC    r2, r2, r1
STR    r2, [r0, #IntSelectOffset]
STR    r1, [r0, #IntEnableOffset] ; Enable interrupt
MRS    CPSR_c, #(DISABLE_IRQ + MODE_SYS_32) ; Enable FIQ interrupts
```

B.1.12 Daisy-chained vectored interrupt service routine

Example B-12 on page B-7 gives an example of the daisy-chained vectored interrupt service routine code.

Example B-12 Daisy-chained vectored interrupt service routine

Example Code

```
0x18    LDR pc,    [pc, #-0xff0]   ; Load Vector into PC
; .....................................................
vector_handler
    ; Code to enable interrupt nesting
    STMFD r13!, {r12, r14} ; stack lr_irq and r12 [plus other regs used below, if appropriate]
    MRS r12, spsr ; Copy spsr into r12...
    STMFD r13!, {r12} ; and save to stack

    ; Read from VICIRQStatus to determine the source of the interrupt
    MSR cpsr_c, #0x1f ; Switch to SYS mode, re-enable IRQ
    STMFD r13!, {r0-r3, r14} ; stack lr_sys and r0-r3

    ; Interrupt service routine...
    ; NOTE: ADS 1.2 requires preservation of 8-byte stack alignment with respect to all external
    ; interfaces. See ADS 1.2 Developer Guide - Section 2.3.3
    ; ...
    BL 2nd_level_handler ; this corrupts lr_sys and r0-r3
    ; ...

    ; Add code to clear the interrupt source; Code to exit handler
    LDMFD r13!, {r0-r3, r14} ; unstack lr_sys and r0-r3
    MSR cpsr_c, #0x92 ; Disable IRQ, and return to IRQ mode
    LDMFD r13!, {r12} ; unstack r12...
    MSR spsr_cxsf, r12 ; and restore spsr...
    LDMFD r13!, {r12, r14} ; unstack registers
    LDR r1, =VectorAddr
    STR r0, [r1] ; Acknowledge VIRQ serviced
    SUBS pc, lr, #4 ; Return from ISR
```

Example Code

```
BIC    r2, r2, r1
STR    r2, [r0, #IntSelectOffset]
STR    r1, [r0, #IntEnableOffset] ; Enable interrupt
MRS    CPSR_c, #(DISABLE_IRQ + MODE_SYS_32) ; Enable FIQ interrupts
```
Example B-12 Daisy-chained vectored interrupt service routine

```
0x18  LDR   pc, [pc, #-0xff0]     ; Load vector into PC

vector_handler ; Code to enable interrupt nesting. First, stack off registers you know will be
corrupted STMFD r13!, {r0-r3, r12, r14} ; Use r12 to stack off the spsr MRS   r12, spsr
; Copy spsr to r12 STMFD r13!, {r12}; Stack spsr in r12 ; Change from IRQ mode to System
mode, and re-enable interrupts MSR   cpsr_c, #0x1F ; Branch to the function that:
; 1. Clears the peripheral interrupt. Do this first ; 2. Performs the interrupt function
; Stack the link register of System mode
STMFD SP!, {lr} BL some_interrupt_code LDMFD SP!, {lr} ; When the interrupt has
finished, disable interrupts so that you can update the VIC and your
; mode without worrying about being interrupted MSR   cpsr_c, #0x92 ; Disable
interrupts and return to IRQ mode ; Acknowledge that the IRQ has finished being serviced. You can
do this because the interrupts
; are now disabled, so the ARM core runs this section of code up until the end,
uninterrupted LDR   r12, =VectorAddr ; VectorAddr should be = 0xFFFFF030 STR   r0,
[r12] ; Not important what r0 contains ; Stacking operations - first, restore the spsr
using r12 as a temporary register LDMFD r13!, {r12} ; Pop the spsr off the stack MSR
spsr_cxsf, r12 ; and restore it ; Pop remaining registers off the stack. This corresponds
to the first STMFD of this function LDMFD r13!, {r0-r3, r12, r14} ; Return from the
interrupt handler SUBS   pc, lr, #4
```

B.1.13 Highest level vectored IRQ interrupt service routine

Example B-13 gives an example of the highest level vectored IRQ interrupt service
routine code.

Example B-13 Highest level vectored IRQ interrupt service routine

```
0x18  LDR   pc, [pc, #-0xff0]     ; Load vector into PC

highest_priority_vector_handler

Interrupt_service_routine
;
; Code to exit handler
STR    r0, VectorAddr             ; Acknowledge Vectored IRQ has
; finished
SUBS   pc, r14, #4                ; Return from IRQ
```
Glossary

This glossary describes some of the terms used in ARM manuals. Where terms can have several meanings, the meaning presented here is intended.

Abort

A mechanism that indicates to a core that the value associated with a memory access is invalid. An abort can be caused by the external or internal memory system as a result of attempting to access invalid instruction or data memory. An abort is classified as either a Prefetch or Data Abort, and an internal or External Abort.

See also Data Abort, External Abort and Prefetch Abort.

Advanced eXtensible Interface (AXI)

This is a bus protocol that supports separate address/control and data phases, unaligned data transfers using byte strobes, burst-based transactions with only start address issued, separate read and write data channels to enable low-cost DMA, ability to issue multiple outstanding addresses, out-of-order transaction completion, and easy addition of register stages to provide timing closure. The AXI protocol also includes optional extensions to cover signaling for low-power operation.

AXI is targeted at high performance, high clock frequency system designs and includes a number of features that make it very suitable for high speed sub-micron interconnect.
Advanced High-performance Bus (AHB)
The AMBA Advanced High-performance Bus system connects embedded processors such as an ARM core to high-performance peripherals, DMA controllers, on-chip memory, and interfaces. It is a high-speed, high-bandwidth bus that supports multi-master bus management to maximize system performance.

See also Advanced Microcontroller Bus Architecture and AHB-Lite.

Advanced Microcontroller Bus Architecture (AMBA)
AMBA is the ARM open standard for multi-master on-chip buses, capable of running with multiple masters and slaves. It is an on-chip bus specification that details a strategy for the interconnection and management of functional blocks that make up a System-on-Chip (SoC). It aids in the development of embedded processors with one or more CPUs or signal processors and multiple peripherals. AMBA complements a reusable design methodology by defining a common backbone for SoC modules. AHB, APB, and AXI conform to this standard.

Advanced Peripheral Bus (APB)
The AMBA Advanced Peripheral Bus is a simpler bus protocol than AHB. It is designed for use with ancillary or general-purpose peripherals such as timers, interrupt controllers, UARTs, and I/O ports. Connection to the main system bus is through a system-to-peripheral bus bridge that helps to reduce system power consumption.

See also Advanced High-performance Bus.

AHB
See Advanced High-performance Bus.

AHB-Lite
AHB-Lite is a subset of the full AHB specification. It is intended for use in designs where only a single AHB master is used. This can be a simple single AHB master system or a multi-layer AHB system where there is only one AHB master on a layer.

AMBA
See Advanced Microcontroller Bus Architecture.

APB
See Advanced Peripheral Bus.

Architecture
The organization of hardware and/or software that characterizes a processor and its attached components, and enables devices with similar characteristics to be grouped together when describing their behavior, for example, Harvard architecture, instruction set architecture, ARMv6 architecture.

ARM instruction
A word that specifies an operation for an ARM processor to perform. ARM instructions must be word-aligned.

ASIC
See Application Specific Integrated Circuit.

ATPG
See Automatic Test Pattern Generation.
Automatic Test Pattern Generation (ATPG)  
The process of automatically generating manufacturing test vectors for an ASIC design, using a specialized software tool.

AXI  
See Advanced eXtensible Interface.

Beat  
Alternative word for an individual transfer within a burst. For example, an INCR4 burst comprises four beats.

See also Burst.

Burst  
A group of transfers to consecutive addresses. Because the addresses are consecutive, there is no requirement to supply an address for any of the transfers after the first one. This increases the speed at which the group of transfers can occur. Bursts over AHB buses are controlled using the HBURST signals to specify if transfers are single, four-beat, eight-beat, or 16-beat bursts, and to specify how the addresses are incremented.

See also Beat.

Byte  
An 8-bit data item.

Core  
A core is that part of a processor that contains the ALU, the datapath, the general-purpose registers, the Program Counter, and the instruction decode and control circuitry.

Data Abort  
An indication from a memory system to the core of an attempt to access an illegal data memory location. An exception must be taken if the processor attempts to use the data that caused the abort.

See also Abort, External Abort, and Prefetch Abort.

Direct Memory Access (DMA)  
An operation that accesses main memory directly, without the processor performing any accesses to the data concerned.

DMA  
See Direct Memory Access.

Event  
1 (Simple) An observable condition that can be used by an ETM to control aspects of a trace.

2 (Complex) A boolean combination of simple events that is used by an ETM to control aspects of a trace.

Exception  
A fault or error event that is considered serious enough to require that program execution is interrupted. Examples include attempting to perform an invalid memory access, external interrupts, and undefined instructions. When an exception occurs,
normal program flow is interrupted and execution is resumed at the corresponding exception vector. This contains the first instruction of the interrupt handler to deal with the exception.

**Exception vector**  
*See* Interrupt vector.

**External Abort**  
An indication from an external memory system to a core that the value associated with a memory access is invalid. An external abort is caused by the external memory system as a result of attempting to access invalid memory.

*See also* Abort, Data Abort and Prefetch Abort.

**High vectors**  
Alternative locations for exception vectors. The high vector address range is near the top of the address space, rather than at the bottom.

**Interrupt handler**  
A program that control of the processor is passed to when an interrupt occurs.

**Interrupt vector**  
One of a number of fixed addresses in low memory, or in high memory if high vectors are configured, that contains the first instruction of the corresponding interrupt handler.

**Microprocessor**  
*See* Processor.

**Multi-master**  
An AMBA bus sharing scheme (not in AMBA Lite) where different masters can gain a bus lock (Grant) to access the bus in an interleaved fashion.

**Prefetch Abort**  
An indication from a memory system to the core that an instruction has been fetched from an illegal memory location. An exception must be taken if the processor attempts to execute the instruction. A Prefetch Abort can be caused by the external or internal memory system as a result of attempting to access invalid instruction memory.

*See also* Data Abort, External Abort and Abort.

**Processor**  
A processor is the circuitry in a computer system required to process data using the computer instructions. It is an abbreviation of microprocessor. A clock source, power supplies, and main memory are also required to create a minimum complete working computer system.

**Read**  
Reads are defined as memory operations that have the semantics of a load. That is, the ARM instructions LDM, LDRD, LDC, LDR, LDRT, LDRSH, LDRH, LDRSB, LDRB, LDRBT, LDREX, RFE, STREX, SWP, and SWPB, and the Thumb instructions LDM, LDR, LDRSH, LDRH, LDRSB, LDRB, and POP.

Java instructions that are accelerated by hardware can cause a number of reads to occur, according to the state of the Java stack and the implementation of the Java hardware acceleration.
**Reserved**
A field in a control register or instruction format is reserved if the field is to be defined by the implementation, or produces Unpredictable results if the contents of the field are not zero. These fields are reserved for use in future extensions of the architecture or are implementation-specific. All reserved bits not used by the implementation must be written as 0 and read as 0.

**Unaligned**
A data item stored at an address that is not divisible by the number of bytes that defines the data size is said to be unaligned. For example, a word stored at an address that is not divisible by four.

**Unpredictable**
Means that you cannot rely on the behavior of the ETM. Such conditions have not been validated. When applied to the programming of an event resource, only the output of that event resource is Unpredictable. Unpredictable behavior can affect the behavior of the entire system, because the ETM is capable of causing the core to enter debug state, and external outputs can be used for other purposes.

**Unpredictable**
For reads, the data returned when reading from this location is unpredictable. It can have any value. For writes, writing to this location causes unpredictable behavior, or an unpredictable change in device configuration. Unpredictable instructions must not halt or hang the processor, or any part of the system.

**Word**
A 32-bit data item.

**Write**
Writes are defined as operations that have the semantics of a store. That is, the ARM instructions SRS, STM, STRD, STC, STRT, STRH, STRB, STRBT, STREX, SWP, and SWPB, and the Thumb instructions STM, STR, STRH, STRB, and PUSH.

Java instructions that are accelerated by hardware can cause a number of writes to occur, according to the state of the Java stack and the implementation of the Java hardware acceleration.
Index

A
AMBA AHB signals  A-2

B
Block diagram  2-4

C
Clear software interrupt code  B-3
Code
clear software interrupt  B-3
daisy-chained vectored interrupt service routine  B-6
disable interrupts  B-2
enable interrupts
Enable interrupts code  B-2
FIQ interrupt handler  B-4
FIQ interrupt initialization  B-4
generate software interrupt  B-3

D
Daisy-chain  2-5, 2-11, 3-20
signals  A-4
Daisy-chained interrupt controller  2-12

E
highest level vectored IRQ interrupt service routine  B-7
interrupt polling  B-3
simple interrupt initialization  B-4
simple interrupt service routine  B-5
vectored interrupt initialization  B-5
vectored interrupt service routine  B-6
Code examples  B-2
Conventions
numerical  xiii
signal naming  xii
timing diagram  xii
typographical  xi

F
Fast interrupt request  2-2
Fast IRQ  3-19
Features  1-2
FIQ  2-2, 3-18
FIQ interrupt handler code  B-4
FIQ interrupt initialization code  B-4
FIQ Status Register  3-6
Further reading  xiii

G
Generate software interrupt code  B-3

vectored interrupt service routine code  B-6
Default Vector Address Register  3-10
Disable interrupts code  B-2

ARM DDI 0181E  Copyright © 2000, 2003-2004 ARM Limited. All rights reserved.
Index

Index-2

Copyright © 2000, 2003-2004 ARM Limited. All rights reserved.

H
HADDR A-2
HCLK A-2
Highest level vectored IRQ interrupt service routine code B-7
HPROT A-2
HRDATA A-2
HREADYIN A-2
HREADYOUT A-2
HRESETn A-2
HRESP A-2
HSELVIC A-2
HSIZE A-2
HTRANS A-2
HWDATA A-2
HWRITE A-2

N
Non-vectored FIQ interrupt 2-5
Non-vectored interrupt 2-3
Non-vectored IRQ interrupt 2-6
Numerical conventions xiii

P
Peripheral Identification Registers 3-11
PrimeCell Identification Registers 3-14
Product revision status x
Protection Enable Register 3-8

R
Raw Interrupt Status Register 3-6
Registers
VICDEFVECTADDR 3-10
VICFIQSTATUS 3-6
VICINTENABLE 3-7
VICINTEXELECT 3-7
VICIRQSTATUS 3-6
VICITCR 4-4
VICITOP 4-6
VICPCELLID 3-14
VICPERIPHID 3-11
VICPROTECTION 3-8
VICRAWINTR 3-6
VICSOFTING 3-8
VICSOFTINTCLEAR 3-8
VICVECTADDR 3-9, 3-10
VICVECTCNTL 3-10
VIDINTENCLEAR 3-7
Revision status x

S
Scan test control signals A-5
Scan testing 4-3
SCANENABLE A-5
SCANINHCLK A-5
SCANOUTHCLK A-5
Signal naming conventions xii
Signals

V
Vector Address Register 3-9
Vector Address Registers 3-10
Vector Control Registers 3-10
Vectored interrupt 2-3, 2-8
Vectored interrupt blocks 2-6
Vectored interrupt initialization code B-5
Vectored interrupt service routine code B-6
VIC registers 3-3
VICDEFVECTADDR Register 3-10
VICFIQSTATUS Register 3-6
VICINTENABLE Register 3-7
VICINTEXELECT Register 3-7
VICINTSOURCE A-3
VICIRQSTATUS Register 3-6
VICITCR Register 4-4
VICITIP 4-5
VICITOP Register 4-6
VICPCELLID Register 3-14
VICPERIPHID Register 3-11
VICPROTECTION Register 3-8
VICRAWINTR Register 3-6
VICSOFTING Register 3-8
VICSOFTINTCLEAR Register 3-8

AMBA AHB A-2
daisy-chain A-4
interrupt controller A-3
test control signals A-5
Simple interrupt initialization code B-4
Simple interrupt service routine code B-5
Software interrupt 2-2, 2-8, 2-9
Software Interrupt Clear Register 3-8
Software Interrupt Register 3-8
Standalone interrupt controller 2-11

T
Test Control Register 4-4
Test harness 4-2
Test registers 4-4
Timing diagram conventions xii
Typographical conventions xi
VICVECTADDR Register  3-9, 3-10
VICVECTADDRIN  A-4
VICVECTADDROUT  A-4
VICVECTCNTL Register  3-10
VIDINTENCLEAR Register  3-7