ARM PrimeCell™ Vectored Interrupt Controller (PL192)

Technical Reference Manual

ARM®

ARM DDI 0273A

ARM PrimeCell Vectored Interrupt Controller (PL192)
Technical Reference Manual

Release Information

The following changes have been made to this document.

Change history

Date	Issue	Change
December 2002	A	First issue.

Proprietary Notice

Confidentiality Status

Product Status

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

Web Address

`http://www.arm.com`

Contents
ARM PrimeCell Vectored Interrupt Controller (PL192) Technical Reference Manual

List of Tables
ARM PrimeCell Vectored Interrupt Controller (PL192) Technical Reference Manual

List of Figures
ARM PrimeCell Vectored Interrupt Controller (PL192) Technical Reference Manual

Preface

This preface introduces the ARM PrimeCell Vectored Interrupt Controller (PL192) and its reference documentation. It contains the following sections:

- *About this document* on page x
- *Further reading* on page xii
- *Feedback* on page xiii.

About this document

This document is the technical reference manual for the ARM PrimeCell *Vectored Interrupt Controller* (VIC).

Intended audience

This document has been written for hardware and software engineers implementing System-on-Chip designs. It provides the necessary information to enable designers to integrate the peripheral into a target system as quickly as possible.

Organization

This document is organized as follows:

Chapter 1 ***Introduction***

Read this chapter for an introduction to the PrimeCell VIC and its features.

Chapter 2 ***Functional Overview***

Read this chapter for a description of the major functional blocks of the PrimeCell VIC.

Chapter 3 ***Programmer's Model***

Read this chapter for a description of the registers and programming details of the PrimeCell VIC.

Chapter 4 ***Programmer's Model for Test***

Read this chapter for a description of the test registers and signals of the PrimeCell VIC.

Appendix A ***Signal Descriptions***

Read this appendix for a description of the PrimeCell VIC signals.

Appendix B ***Example Code***

Read this appendix for a description of the PrimeCell VIC example code.

Appendix C ***Troubleshooting***

Read this appendix for a description of troubleshooting the PrimeCell VIC.

Typographical conventions

The following typographical conventions are used in this book:

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

Further reading

This section lists publications by ARM Limited.

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

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

ARM publications

This document contains information that is specific to the ARM PrimeCell Vectored Interrupt Controller (PL192). Refer to 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 (PL192), and on the documentation.

Feedback on the ARM PrimeCell Vectored Interrupt Controller (PL192)

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

- the product name
- a concise explanation of your comments.

Feedback on this document

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

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

General suggestions for additions and improvements are also welcome.

Chapter 1
Introduction

This chapter introduces the ARM PrimeCell Vectored Interrupt Controller (PL192). It contains the following sections:

- *About the ARM PrimeCell Vectored Interrupt Controller (PL192)* on page 1-2
- *Release information* on page 1-3.

1.1 About the ARM PrimeCell Vectored Interrupt Controller (PL192)

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

The PrimeCell 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 all interrupt sources
- provides support for the ARM v6 processor VIC port, compatible with ARM11 and ARM1026EJ processors.

1.1.1 Features of the PrimeCell VIC

The PrimeCell VIC has the following features:

- compliance to the *AMBA Specification (Rev 2.0)* for easy integration into *System-on-Chip* (SoC) implementation
- support for 32 vectored IRQ interrupts
- fixed hardware interrupt priority levels
- programmable interrupt priority levels
- hardware interrupt priority level masking
- programmable interrupt priority level masking
- IRQ and FIQ generation
- AHB mapped for faster interrupt response
- software interrupt generation
- test registers
- raw interrupt status
- interrupt request status
- privileged mode support for restricted access
- interrupt controller daisy chaining (supports PL190, PL192, and *AMBA Development Kit* (ADK)) interrupt controller
- support for the ARM v6 processor VIC port in synchronous and asynchronous mode, enabling faster interrupt servicing when connected to an ARM11 or ARM1026EJ processor.

1.2 Release information

The following list shows the functional differences between the PL192 and PL190 PrimeCell VICs:

- VIC port fully supported, both in synchronous and asynchronous modes.
- The programmers model is different between PL192 and PL190, so code is not backwards-compatible.
- 16 standard interrupts replaced with vectored interrupts to give 32 vectored interrupts.
- 16 software-programmable interrupt priority levels added, in addition to the hardware priority levels.
- Software-programmable interrupt priority level masking added, in addition to the hardware priority level masking.
- Register map changed
  - VICDefVectAddr Register removed
  - 16 x VICVectCntl Register removed
  - 16 x VICVECTADDR Registers added
  - 32 x VICVECTPRIORITY Registers added
  - VICINTSSTATUS Register added
  - VICINTSSTATUSCLEAR Register added.
- VICVECTPRIORITY Register added. This enables the priority level of each of the 33 interrupt sources to be programmed by software.
- VICINTSSTATUS and VICINTSSTATUSCLEAR Registers added. These enable a sampled version of the status of the interrupt sources to be read, and are used to check for interrupts that are cleared prematurely.
- Ports **VICIRQINREG** and **VICFIQINREG** added to enable the daisy-chained interrupts to be registered. If the interrupts are not registered, the latency for daisy-chained interrupts is decreased.
- Port **VICIRQACKOUT** added to enable daisy-chained VICs to receive the processor response to an interrupt. This saves the software overhead of writing to the VICADDRESS Register in the daisy-chained VIC that has generated the interrupt.

Chapter 2
Functional Overview

This chapter describes the major functional blocks of the ARM PrimeCell Vectored Interrupt Controller (PL192). It contains the following sections:

2.1 PrimeCell VIC overview

The PrimeCell 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 processor based system, two levels of interrupt are available:

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

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

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

The FIQ interrupt has the highest priority, followed by interrupt vector 0-31, and the daisy-chained interrupt. The priority of each of the vectored interrupts is programmable, enabling the order the interrupts are serviced in to be dynamically changed. This is done by programming the values in the vector priority registers. If multiple interrupts are set to the same programmed priority level, the fixed hardware priority levels are used to determine the order the interrupts on that level are serviced. This is also applicable when the priority registers are not programmed. Interrupt 0 has the highest hardware priority level, and the daisy-chained interrupt has the lowest. The software can control each request line to generate software interrupts.

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

——— **Note** ———

The PrimeCell 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 are able to interrupt the transient interrupt service routine, assuming interrupt nesting is allowed.

There are 32 vectored interrupts available, which can also be used as nonvectored interrupts if the system does not support interrupt vector addresses, but requires the interrupt priority and nesting features of the VIC. These interrupts can only generate an IRQ interrupt. The IRQ interrupts provide an address for an *Interrupt Service Routine* (ISR). Reading from the Vector Interrupt Address Register, VICADDRESS, 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 VICADDRESS Register indicates to the interrupt priority hardware that the current interrupt is serviced, enabling the masking of lower priority or the same priority interrupts to be removed and for the interrupts to become active.

Note

After the VICADDRESS Register is read, the current and lower priority interrupts are masked from the **nVICIRQ** output. However, the values in the VICIRQSTATUS and VICRAWINTR Registers are not affected.

If the system does not support interrupt vector addresses, the VICVECTADDR Registers can be programmed with the number of the interrupt source ports they relate to, so that the source of the active interrupt can be easily determined.

A programmed interrupt request (for each of the 32 IRQ lines) enables you to generate an interrupt under software control. This register is typically used to downgrade an FIQ interrupt to an IRQ interrupt. This is done by clearing the FIQ and setting up a software IRQ instead.

Note

The priority of the FIQ over IRQ is set by the ARM processor. 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 allows interrupts to be asserted 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 interrupts, bits 1-5 must be used as defined in Table 2-1. Bit 0 and bit 6 upwards are available for use as required. For the FIQ interrupt, the bits can be used as required.

Table 2-1 Interrupt standard configuration

Bit	Interrupt source
1	Software interrupt
2	Comms Rx
3	Comms Tx
4	Timer 1
5	Timer 2

Note

This is only a recommended interrupt configuration, and there is no requirement that it is followed.

A space is reserved for the software interrupt so that it can be used without masking out a valid hardware interrupt. Any of the interrupt bits can be programmed 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 used by the system processor, and are required in any system using these debug features.

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

Figure 2-1 on page 2-5 shows a block diagram of the PrimeCell VIC.

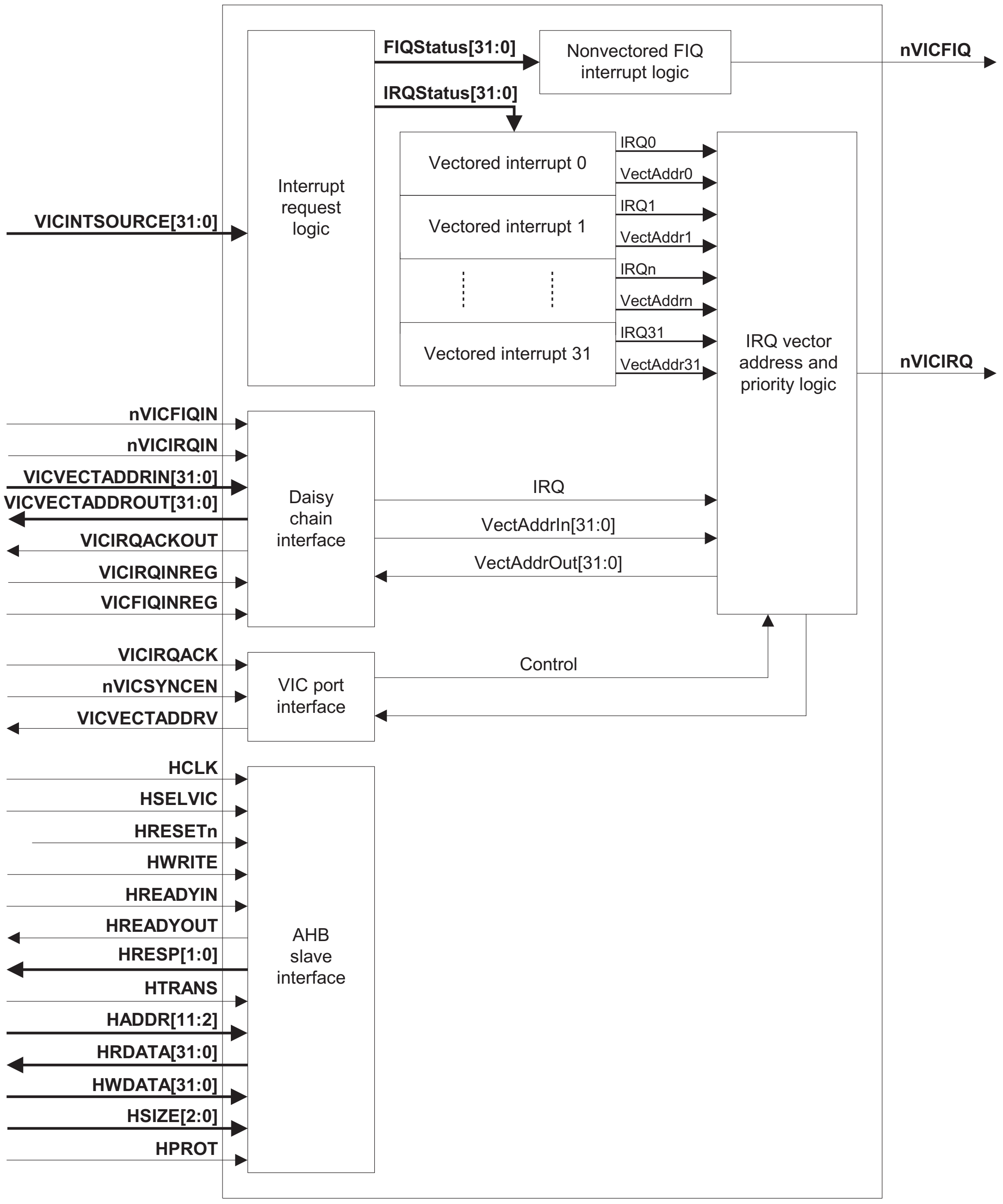

Figure 2-1 VIC block diagram

The main components of the PrimeCell VIC are described in the following sections:

- *Interrupt request logic*
- *Nonvectored FIQ interrupt logic* on page 2-7
- *Vectored IRQ interrupt logic* on page 2-7
- *Interrupt priority logic* on page 2-8
- *Interrupt priority masking* on page 2-10
- *Vectored interrupts* on page 2-11
- *Software interrupts* on page 2-11
- *Interrupt service routine addresses* on page 2-11.

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 which 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.

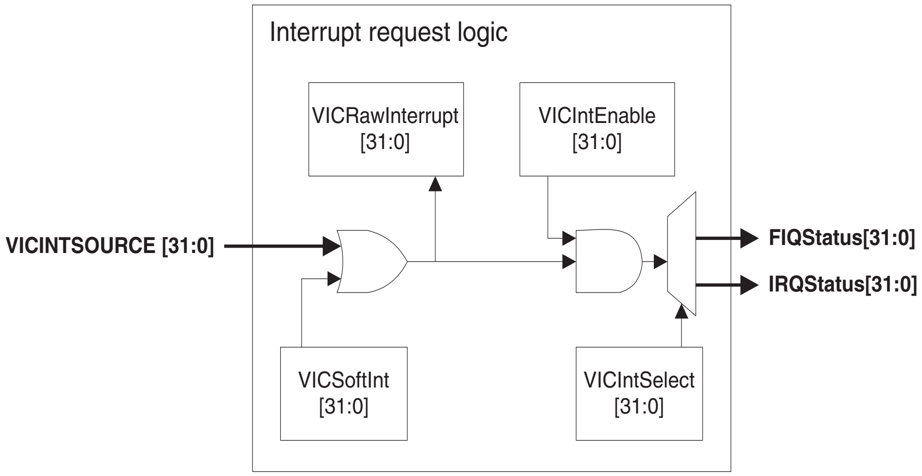

Figure 2-2 Interrupt request logic

2.1.2 Nonvectored FIQ interrupt logic

The nonvectored 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 shows a block diagram of the nonvectored FIQ interrupt logic.

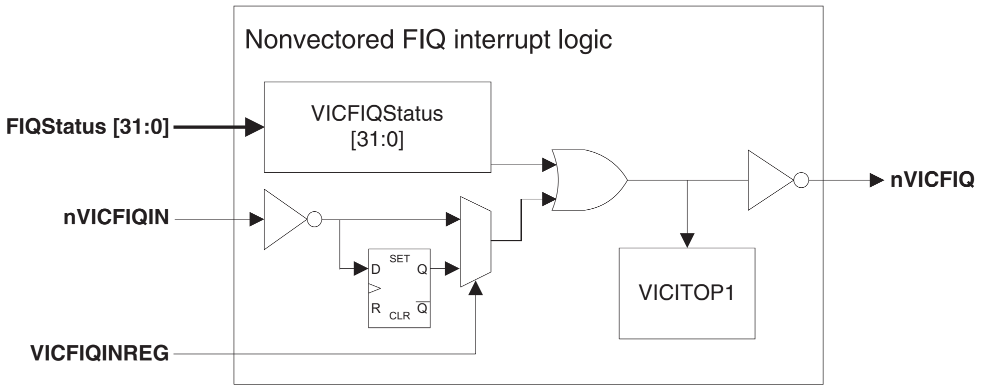

Figure 2-3 Nonvectored FIQ interrupt logic

2.1.3 Vectored IRQ interrupt logic

There are 32 vectored interrupt blocks. The vectored interrupt blocks receive the IRQ interrupt requests and set **VectIRQx** if the following are true:

- the interrupt is active
- the interrupt's programmed priority level is not masked
- the interrupt is currently the highest priority requesting interrupt.

Each vectored interrupt block also provides a **VectAddrx[31:0]** output for use in the interrupt priority block. Figure 2-4 on page 2-8 shows a block diagram of the vectored IRQ interrupt logic.

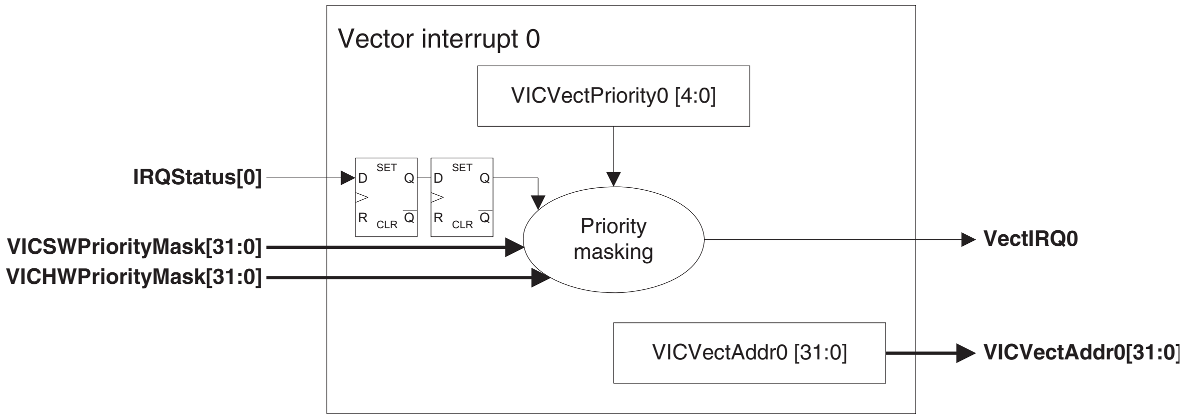

Figure 2-4 Vectored IRQ interrupt logic

2.1.4 Interrupt priority logic

The interrupt priority block prioritizes the interrupt requests using the programmed and hardware priority levels in the following order:

- vectored interrupt requests
- external interrupt requests (from daisy chain).

The highest-priority request generates an IRQ interrupt if the interrupt is not currently being serviced. Figure 2-5 on page 2-9 shows a block diagram of the interrupt priority logic.

Note

nVICIRQIN is the daisy-chained IRQ request input.

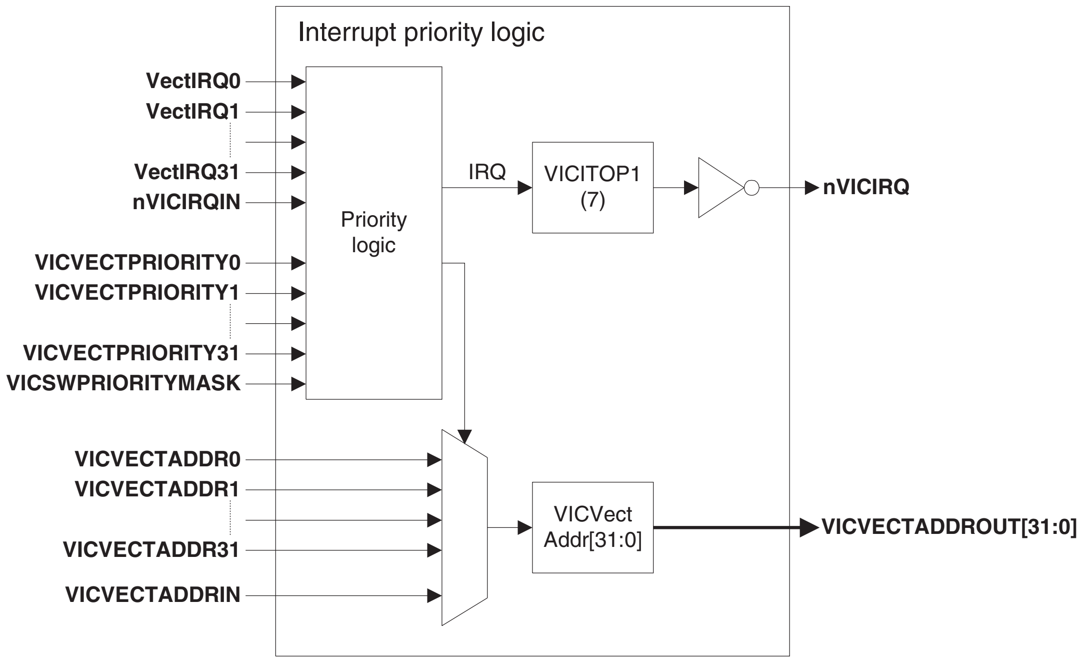

Figure 2-5 Interrupt priority logic

The priority of each of the vectored interrupts is programmable, allowing the order in which interrupts are serviced to be dynamically changed. This is done by programming the values in the Vector Priority Registers, VICVECTPRIORITY[0-31] (see *Vector Priority Registers, VICVECTPRIORITY[0-31] and VICVECTPRIORITYDAISY* on page 3-16), to reflect the priority level required for that vectored interrupt. There are 16 programmable priority levels available, 0-15. By default, all interrupts are set to the lowest priority level (15), but each interrupt can be set to any priority level required.

If multiple interrupts are set to the same programmed priority level, the fixed hardware priority levels are used to determine the order the interrupts on that level are serviced. This is also applicable when the priority registers are not programmed. Interrupt 0 has the highest hardware priority level, and the daisy-chained interrupt has the lowest. This ensures that for any programmed priority level setting, there is a defined order in which the interrupts are serviced.

2.1.5 Interrupt priority masking

The PrimeCell VIC implements two forms of interrupt priority masking:

Hardware masking

The hardware masking is applied whenever an interrupt is being serviced, either with a read from the VICADDRESS Register, or by asserting the **VICIRQACK** input when the VIC port is used. This prevents other active interrupts of an equal or lower priority generating a new IRQ while the interrupt service routine is being executed. When the interrupt routine has completed and the VICADDRESS Register has been written to, the interrupt mask is cleared to allow all enabled interrupt sources through.

Note

The VICIRQSTATUS and VICRAWINTR Registers are not affected by this masking.

Software masking

The software masking is applied using the value programmed into the VICSWPRIORITYMASK Register. This mask is applied continuously, and at the same time as the hardware mask when an interrupt is being serviced.

Note

The values in the VICIRQSTATUS and VICRAWINTR Registers do not reflect the masking by the VICSWPRIORITYMASK Register.

If two interrupts with the same programmed priority level are asserted simultaneously, the interrupt connected to the lowest **VICINTSOURCE** is serviced first. If an interrupt occurs while another interrupt of the same priority level is being serviced, it is masked until the first interrupt service routine has completed, regardless of whether the second interrupt is connected to a lower **VICINTSOURCE**.

Note

Hardware priority levels only take effect when multiple interrupts are programmed to have the same priority level, and occur at the same time. In this case, vectored interrupt 0 has the highest priority, and interrupt 31 has the lowest priority.

2.1.6 Vectored interrupts

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

- it is enabled in the Interrupt Enable Register, VICINTENABLE
- it is set to generate an IRQ interrupt in the Interrupt Select Register, VICINTSELECT
- the priority level of the interrupt is not masked out by the Software Priority Mask Register, VICSWPRIORITYMASK.

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. Software interrupts are cleared by writing to the Software Interrupt Clear Register, VICSOFTINTCLEAR (see *Software Interrupt Clear Register, VICSOFTINTCLEAR* on page 3-13). This is normally done at the end of the interrupt service routine.

2.1.8 Interrupt service routine addresses

The VICADDRESS Register provides the location of the interrupt service routine for the currently active interrupt, which is also made available on the **VICVECTADDROUT** output ports. This value reflects the relevant programmed value from one of the 32 VICVECTADDR Registers, or the address input from the daisy-chain interface. If no interrupt is currently active, the VICADDRESS Register holds the previous active interrupt address. This means that the address of the now non-active interrupt is the one presented to the CPU when:

- an interrupt has occurred
- the CPU has acknowledged the interrupt
- the interrupt source has gone inactive
- no other interrupt is active.

The vectored interrupt address registers must be programmed with the locations of the specific routines for each of the vectored interrupt sources. If multiple vectored interrupt sources use the same interrupt service routine, the same address must be programmed into their respective address registers.

For systems that cannot use the vector address information, the registers can be left with their default values, or programmed with the interrupt source number.

2.2 Operation

The operation of the PrimeCell VIC is described in the following sections:

- *Vectored interrupt flow sequence using AHB*
- *Nonvectored interrupt flow sequence using AHB*
- *FIQ interrupt flow sequence* on page 2-13
- *Vectored IRQ interrupt flow sequence using VIC port* on page 2-14.

2.2.1 Vectored interrupt flow sequence using AHB

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

1. An interrupt occurs.
2. The ARM processor branches to the IRQ interrupt vector.
3. Read the VICADDRESS Register and branch to the interrupt service routine. This can be done using an `LDR` PC instruction. Reading the VICADDRESS Register updates the hardware priority register of the interrupt controller.
4. Stack the workspace so that IRQ interrupts can be re-enabled.
5. Enable the IRQ interrupts on the processor so that a higher priority can be serviced.
6. Execute the *Interrupt Service Routine* (ISR).
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 on the processor and restore the workspace.
9. Write to the VICADDRESS 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 Nonvectored interrupt flow sequence using AHB

The following procedure show the sequence for the nonvectored IRQ interrupt flow:

1. An IRQ interrupt occurs.
2. The ARM processor branches to the IRQ interrupt vector
3. Stack the workspace so that IRQ interrupts can be re-enabled later.

4. Perform a dummy read to the VICADDRESS Register to set up priority status control in the VIC.
5. Read the VICIRQSTATUS Register and determine which interrupt sources have to be service.
6. Execute the ISR. At the beginning of the ISR, the interrupt of the processor can be re-enabled so that a higher priority interrupt 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 interrupt on the processor and restore the workspace.
9. Write to the VICADDRESS Register. This clears the respective interrupt in the internal interrupt priority hardware.
10. Return from the interrupt. This re-enables the interrupts.

2.2.3 FIQ interrupt flow sequence

The following procedure shows the sequence for the nonvectored FIQ interrupt flow.

1. An FIQ interrupt occurs.
2. The ARM processor branches to the FIQ interrupt vector.
3. Branch to the ISR.
4. Execute the ISR.
5. Clear the requesting interrupt in the peripheral, or write to the VICSOFTINTCLEAR Register if the request was generated by a software interrupt.
6. Disable the interrupts and restore the workspace.
7. Return from the interrupt. This re-enables the interrupts

Note

If the above flow is used, you must not read or write to the VICADDRESS Register.

2.2.4 Vectored IRQ interrupt flow sequence using VIC port

The PrimeCell VIC provides direct support for the VIC port on the ARM11 and ARM1026EJ processors. This interface is used to access the vectored interrupt address and acknowledge servicing of the interrupt without using the AHB, reducing interrupt latency. Using the VIC port to obtain the vectored interrupt address decreases the latency of the time it takes to service an interrupt, because the processor does not have to make an access out onto the AHB to read the VICADDRESS Register. The time of this access can vary depending on what access is currently happening on the AHB.

When a vectored IRQ is triggered, the CPU reads the address using the **VICVECTADDROUT** daisy-chain bus. This contains identical address information to the VICADDRESS Register.

To update the hardware priority register of the interrupt controller, and to clear a nested interrupt:

- the processor sets the **VICIRQACK** signal HIGH to indicate that the interrupt has been detected
- the VIC sets the **VICVECTADDRV** signal HIGH to indicate that the vector address value is valid and stable
- the processor samples the vector address and deasserts **VICIRQACK**
- the VIC deasserts both the **nIRQ** and **VICVECTADDRV** signals.

This sequence is equivalent to reading the VICADDRESS Register.

Note

The AHB must be used at the end of the interrupt service routine to perform a write to the VICADDRESS Register to indicate that the interrupt has been serviced.

The following procedure shows the sequence for the vectored interrupt flow using the VIC port:

1. Ensure CP15 Register 1 VE (VIC Enable) bit is set in the ARM11/ARM1026EJ processor. If this bit is clear, the processor jumps to the legacy vector address (`0x00000018` or `0xFFFF0018`) and does not start the VICIRQACK to VICVECTADDRV handshake. If it is set, the processor drives the VIC port to obtain the IRQ vector address. The VE bit resets to 0.
2. An IRQ interrupt occurs.

3. Perform handshaking with the VIC using the **VICIRQACK** and **VICVECTADDRV** signals to update the hardware priority register of the interrupt controller, which includes reading the vector address through the **VICVECTADDROUT** port. Branch to the interrupt service routine when the handshaking has finished.

   Note

   This operation is performed automatically by the processor when an IRQ is received.

4. Stack the workspace so that IRQ interrupts can be re-enabled.
5. Enable the IRQ interrupts so that a higher priority can be serviced.
6. Execute the ISR.
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 VICADDRESS Register. This clears the respective interrupt in the internal interrupt priority hardware.
10. Return from the interrupt. This re-enables the interrupts.

FIQ support for the ARM11 and ARM1026EJ processors is the same as described in *FIQ interrupt flow sequence* on page 2-13, because the VIC port is not used.

2.3 Connectivity

The PrimeCell VIC is normally used as a standalone interrupt controller. Where required, you can daisy-chain with another interrupt controller, such as a second PL192, a PL190, or an ADK interrupt controller.

——— **Note** ———

The interrupt latency increases if daisy chaining is used.

————————

The PrimeCell VIC is connected 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 are connected to the **VICINTSOURCE** inputs of the PrimeCell VIC, with any unused interrupt inputs tied LOW. To ensure that the Vector Address Register (see *Vector Address Register, VICADDRESS* on page 3-15) can be read in a single instruction, the PrimeCell VIC must be located in the upper 4K of memory, at `0xFFFFF000` for normal exception vectors, or `0xFFFEF000` for high exception vectors. For more details information on the reason for this, see *About the programmer's model* on page 3-2.

——— **Note** ———

If the PrimeCell VIC is located at a different address (outside the 4K range of the exception vectors), interrupt latency is increased.

Because the ARM11 and ARM1026EJ processors access the interrupt address directly rather than using the VICADDRESS Register, the interrupt latency is not affected by the location of the VIC in the system memory map. This enables the VIC to be located at any free position in the memory map.

————————

The connectivity for the various options is described in the following sections:

- *Single interrupt controller connectivity to a processor without VIC port* on page 2-17
- *Daisy-chained interrupt controller connectivity to processor without VIC port* on page 2-17
- *Daisy-chained VIC mode* on page 2-22
- *VIC port connections* on page 2-23
- *Synchronous mode VIC port timing* on page 2-25
- *Asynchronous mode VIC port timing* on page 2-26.

2.3.1 Single interrupt controller connectivity to a processor without VIC port

If the PrimeCell VIC is used as a standalone interrupt controller connected to a processor that does not support the VIC port, connect the signals as follows:

- **nVICIRQIN**, **nVICFIQIN**, and **nVICSYNCEN** must be tied HIGH
- all bits of **VICVECTADDRIN[31:0]**, **VICIRQACK**, **VICIRQINREG**, and **VICFIQINREG** must be tied LOW.

Figure 2-6 shows the connections between the PrimeCell VIC and the processor when used as a standalone interrupt controller.

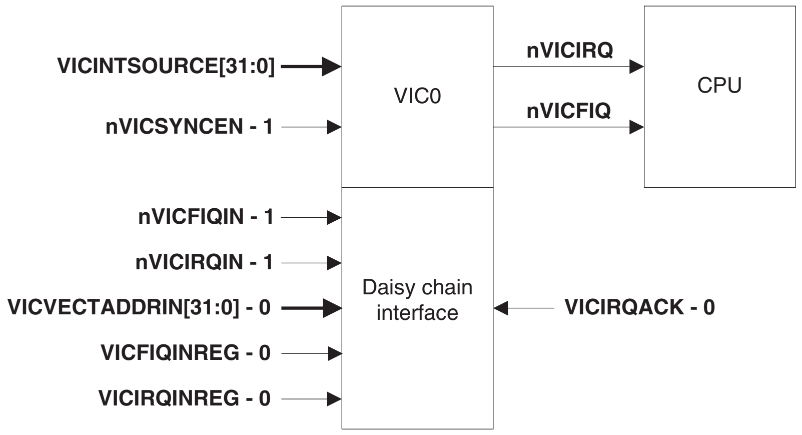

Figure 2-6 Single interrupt controller connectivity

2.3.2 Daisy-chained interrupt controller connectivity to processor without VIC port

If the PrimeCell VIC is used in a daisy chain connected to a processor without a VIC port, connect the signals between the VICs as follows:

- **nVICIRQIN** on VIC0 connects to the **nVICIRQ** output of VIC1
- **nVICFIQIN** on VIC0 connects to the **nVICFIQ** output of VIC1
- **VICVECTADDRIN[31:0]** on VIC0 connects to the **VICVECTADDROUT[31:0]** output of the VIC1
- **nVICSYNCEN** on all VICs tied must be tied HIGH
- **VICVECTADDRV** on all VICs must be left open

- **VICIRQACK** on VIC0 must be tied LOW
- **VICIRQACKOUT** on VIC0 must be left open
- **VICIRQACK** input on VIC1 must be tied LOW
- **VICIRQINREG/VICFIQINREG** on all VICs must be tied LOW, assuming the timing for interrupt source to CPU is less than one clock cycle.

Connect the final VIC in the chain (the VIC furthest from the processor) as described in *Single interrupt controller connectivity to a processor without VIC port* on page 2-17.

Note

It is recommended that no more than eight interrupt controllers are daisy-chained.

Figure 2-7 shows the connections between two PrimeCell VICs and the processor when used in a daisy chain. This configuration is referred to as VIC0 blocking mode.

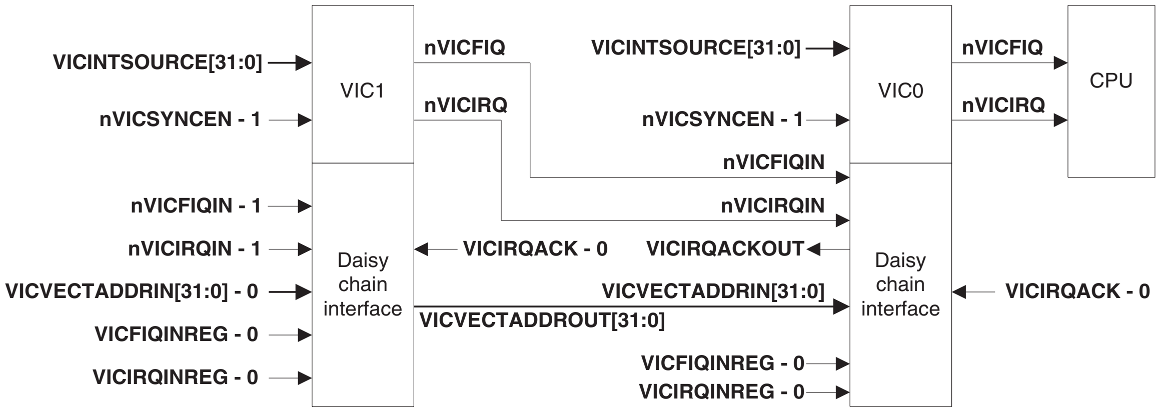

Figure 2-7 Daisy-chained VIC0 blocking mode

In this mode, VIC0 is totally responsible for blocking lower level interrupts from the daisy chained VICs. Because the **VICIRQACKOUT** signal is not used in this configuration, ensure that the VICVECTPRIORITYDAISY Register of VIC0 is not changed to a higher priority while servicing an interrupt. This can allow lower-level VIC1 interrupts to break into the interrupt currently being serviced.

——— **Note** ———

VIC0 (closest to the processor) must be located 4K from the processor exception vectors to reduce interrupt latency. Because only one VIC can be located in that region of memory, all other daisy-chained VICs (VIC1 onwards) must be located outside this region. This does not apply to ARM11 or ARM1026EJ processors, which can read the interrupt address directly.

If the PrimeCell VIC is used in a daisy-chain configuration in an implementation within a chip, you must determine if the delay from the interrupt source to **nVICIRQ** and **nVICFIQ** into the CPU is greater than one clock cycle:

- if the delay is less than one clock cycle, you can tie the ports **VICIRQINREG** and **VICFIQINREG** on all the VICs LOW
- if the delay is greater than one clock cycle, you must tie some or all of the **VICIRQINREG** and **VICFIQINREG** ports HIGH.

Which **VICIRQINREG** and **VICFIQINREG** ports are tied HIGH depends on the delay across each VIC. The delays to be taken into account are as follows:

- the delay from **VICINTSOURCE** to **nVICIRQ** and **nVICFIQ** (for example, VIC7) for the VIC furthest away from the processor
- the delay from **nVICIRQIN** and **nVICFIQIN** to **nVICIRQ** and **nVICFIQ** for subsequent VICs in the chain.

The **VICIRQINREG** and **VICFIQINREG** ports enable registers on the **nVICIRQIN** and **nVICFIQIN** input ports. The **nVICIRQIN** and **nVICFIQIN** input port connections originate from the previous daisy-chained VIC **nVICIRQ** and **nVICFIQ**.

Figure 2-8 shows eight VICs in a daisy-chained configuration. Not all connections are shown, only those relevant for this example.

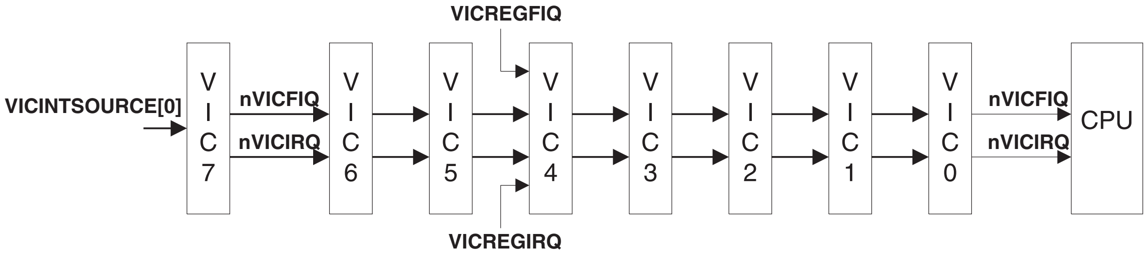

Figure 2-8 Daisy-chained interrupt controller connectivity

Use Figure 2-8 on page 2-19 as an example, with the following assumptions:

- bus clock period is 6ns (approximately 166MHz)
- an IRQ interrupt occurs on **VICINTSOURCE[0]**
- the delay for VIC7 from **VICINTSOURCE[0]** to **nVICIRQ** of VIC7 is 2ns
- the delay from the **nVICIRQ** output of the previous VIC to the **nVICIRQ** output of the next VIC is 1ns, for example, the delay from **nVICIRQ** output of VIC7 to the **nVICIRQ** output of VIC6 is 1ns.

The delay from **VICINTSOURCE[0]** of VIC7 to the **nIRQ** into the CPU is 2ns + (1ns * 7) = 9ns, which is greater than the clock period of 6ns. You must pipeline the **nVICIRQ** path. Pipelining can be done by having all the VIC **VICIRQINREG** signals tied HIGH, however this adds a latency of seven clock cycles to the **nVICIRQ**. To achieve the optimum performance for the circuit in Figure 2-8 on page 2-19, you can tie the **VICIRQINREG** port HIGH on VIC4, which only adds a latency of one clock cycle. Therefore the delay from **VICINTSOURCE[0]** of VIC7 to the **nVICIRQIN** input of VIC4 is 2ns (VIC7) + 1ns (VIC6) + 1ns (VIC5) = 4ns and the delay though the other VICs to the CPU is (1ns * 5) = 5ns.

To have interrupts active while the bus clock is turned off, ensure that there is no pipelining between the interrupt source and the CPU. If the delay means that every VIC must have **VICIRQINREG** and **VICFIQINREG** ports tied HIGH, route their interrupt though **VICINTSOURCE** of VIC0, effectively bypassing the registers on the daisy-chained interrupts. Which VICs you choose to enable the pipelining depends on the manufacturing process and the way the VICs are placed in layout. The timing for the circuit described above is shown in Figure 2-9 on page 2-21.

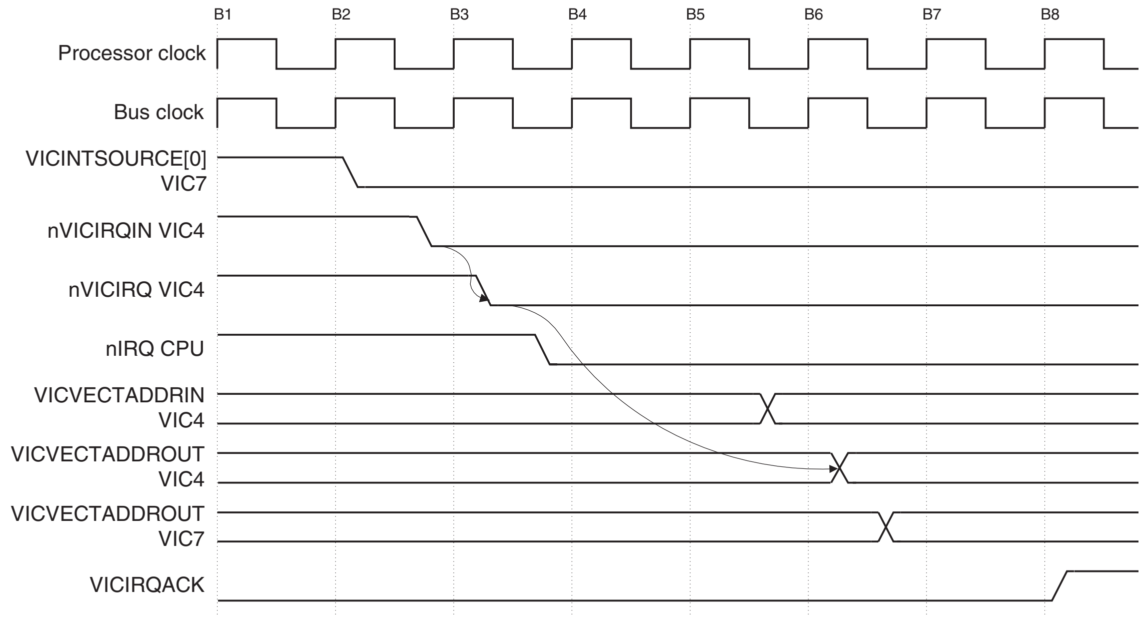

Figure 2-9 Daisy-chained timing example

Note

In Figure 2-9, the processor and bus clocks are synchronous and at the same frequency.

Figure 2-9 shows the timing for the daisy-chained example in Figure 2-8 on page 2-19. The timing of the **VICINTSOURCE[0]** for VIC7 to the **nVICIRQIN** of the VIC4 is shown occurring at clock time B2. VIC4 has the **VICIRQINREG** set HIGH so the **nVICIRQ** of the VIC4 does not go LOW until clock time B3. **VICVECTADDROUT** for each of the VICs is generated three clock cycles after the interrupt is received (two clock cycles for synchronization and one for priority decode). Because the **nVICIRQIN** for VIC4 is registered, the **VICVECTADDROUT** for VIC4 is not generated until clock time B6.

The IRQ path through the VIC has a longer delay than the FIQ path so, depending on the delays, it is possible that the IRQ path requires pipelining and the FIQ path does not.

Caution

If the bus clock is turned off, any VIC that has the **VICIRQINREG** and **VICFIQINREG** ports tied HIGH does not propagate the interrupt from the previous daisy-chained VIC until the bus clock is turned on by a system controller. It is therefore mandatory that any interrupt that operates while the bus clock is turned off is, connected to a part of the chain that does not have any subsequent **VICIRQINREG** and **VICFIQINREG** ports tied HIGH. The interrupt (wake-up interrupt) can then activate the system controller to turn the bus clock on. In the example shown in Figure 2-8 on page 2-19, the wake-up interrupt can be connected to the **VICINTSOURCE** of either VIC0, VIC1, VIC2, VIC3, or VIC4 (the VIC4 daisy-chained **nVICIRQIN** is registered, not its **VICINTSOURCE**).

2.3.3 Daisy-chained VIC mode

In this mode, the VIC0 and the daisy-chained VIC are responsible for blocking lower-level or equal-level interrupts. This enables you to modify the VICVECTPRIORITYDAISY Register of VIC0 during the interrupt service routine. To change the VICVECTPRIORITYDAISY Register while servicing an interrupt, ensure that the **VICIRQACKOUT** between the VICs is connected. To enable higher priority interrupts from the daisy-chained VIC1 to be acknowledged while servicing a lower level interrupt, set the VICVECTPRIORITYDAISY Register of VIC0 to a higher level while servicing the lower-level interrupt. To clear the hardware priority register, the VICADDRESS Registers for all of the active VICs must be written to at the end of the interrupt service routine.

The generation of the **VICIRQACKOUT** signal depends on whether the VIC port connections are used. If the VIC port is used, the CPU **IRQACK** generates the signal. If not, a write to the VICADDRESS Register generates the **VICIRQACKOUT** signal. **VICIRQACKOUT** is only generated if a daisy-chained VIC has generated the IRQ being acknowledge.

Note

This mode must be used with care. Ensure that the timing of the **VICIRQACKOUT** path is met across all the VICs in the daisy chain. If the timing cannot be met for the process being used, use the VIC0 blocking mode as shown in Figure 2-7 on page 2-18.

Figure 2-10 on page 2-23 shows the connections between two PrimeCell VICs and the processor when used in daisy-chained VIC blocking mode.

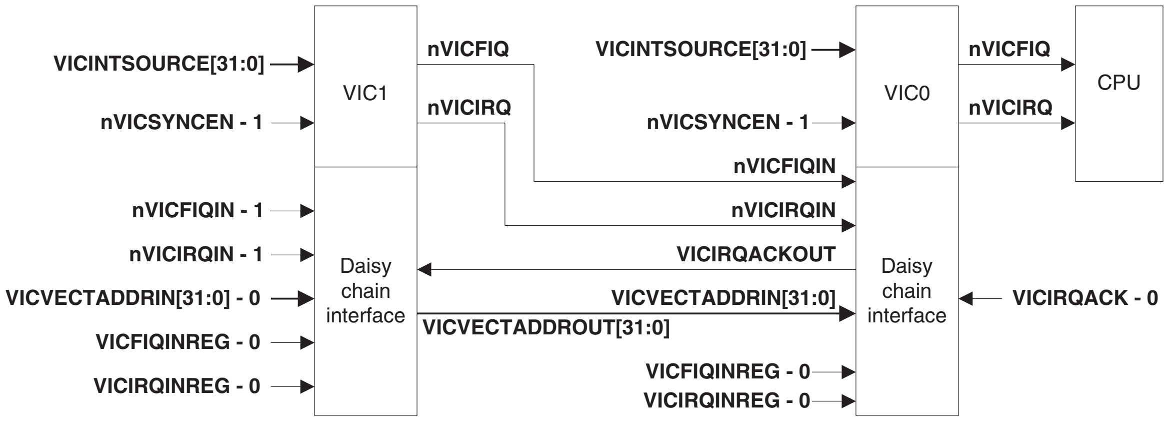

Figure 2-10 Daisy-chained VIC blocking mode

Note

When implementing the daisy chain, ensure the total propagation delay for **VICIRQACK** across the all the VICs is within one clock cycle. The time taken to read the VICADDRESS Register must be included in the total propagation delay.

2.3.4 VIC port connections

The VIC provides direct support for the ARM11 and ARM1026EJ processor VIC ports. This consists of an address input connected to the daisy chain **VICVECTADDROUT** output (which contains the address for the currently active vectored interrupt), a synchronous/asynchronous tie-off (**nVICSYNCEN**), and two handshaking signals (**VICIRQACK** and **VICVECTADDRV**). The connections between the VIC and a VIC port compatible processor are shown in Figure 2-11 on page 2-24.

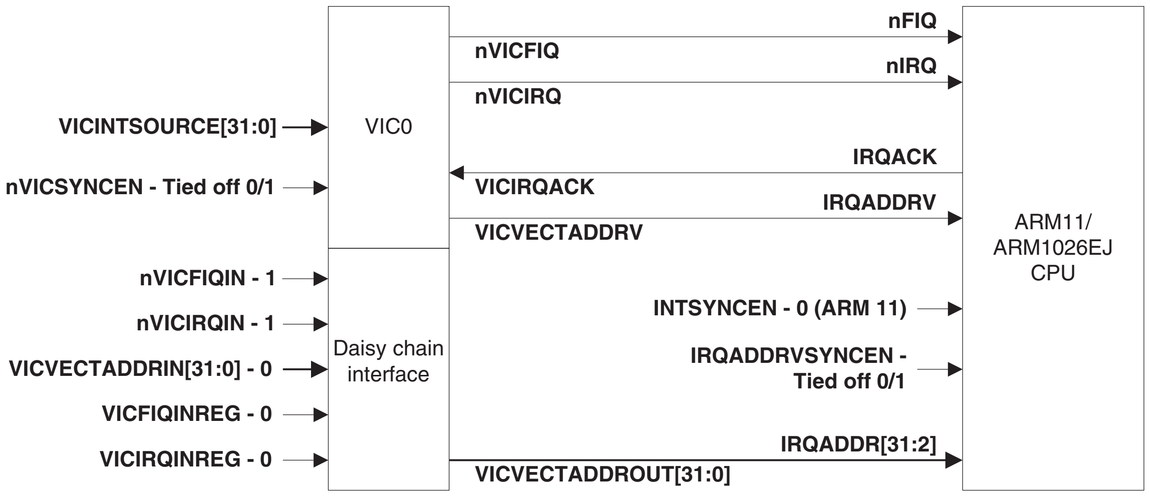

Figure 2-11 VIC port connectivity for synchronous mode

For synchronous mode, the **nVICSYNCEN** input is tied HIGH to indicate that the interface operates without synchronization of the handshaking signals. For asynchronous mode (ARM11 processor only), the **nVICSYNCEN** input is tied LOW to indicate that the interface handshaking signals are synchronized internally in the VIC before use. The ARM1026EJ processor runs synchronously to the system clock.

Note

The ARM11 processor synchronous/asynchronous interface control port **IRQADDRVSYNCEN** must be tied off to the same value as **nVICSYNCEN**, to ensure that both the VIC and the processor are operating in the same mode. **INTSYNCEN** is always set to 0 because the PL192 VIC **nVICFIQ** and **nVICIRQ** outputs are asynchronous, that is, there are combinational paths from the **VICINTSOURCE** inputs to these outputs.

These connections enable the ARM11 and ARM1026EJ processors to read the vectored interrupt address more quickly than through the VICADDRESS Register, which would require an AHB read transfer to be performed.

If the PrimeCell VIC is used in a daisy chain, all other signal connections are as shown in Figure 2-7 on page 2-18. The **VICVECTADDROUT** port is not used as part of the daisy chain for VIC 0 (closest to the processor) when multiple VICs are used.

2.3.5 Synchronous mode VIC port timing

In synchronous mode, the ARM11 and ARM1026EJ processors can run at any multiple of the bus clock frequency. The interface between the ARM11 and ARM1026EJ processor and VIC handles this by using handshaking.

Figure 2-12 shows the minimum timing for an active IRQ when the processor and bus clocks are the same.

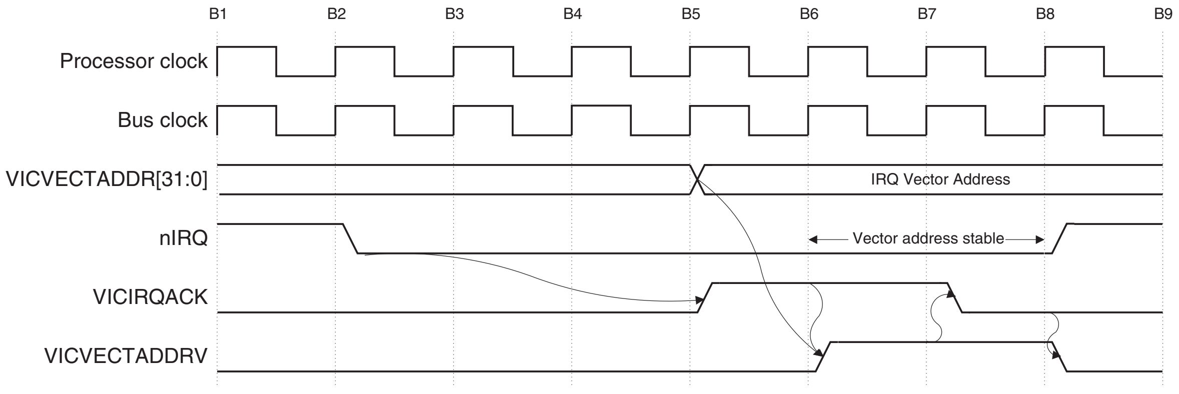

Figure 2-12 VIC port timing example, processor and bus clocks synchronous and same frequency

When the CPU detects the **nIRQ** signal is active, it asserts the **VICIRQACK** input at bus clock time B5 to indicate that it is ready to service the interrupt. The time taken for the CPU to respond to the interrupt depends on the current state of the processor, but the interrupt is always synchronized so the timing shown is the minimum possible. The VIC then asserts the output **VICVECTADDRV**, indicating that the value on the address bus is stable and does not change until after the processor acknowledges sampling the address value (to avoid a higher priority interrupt changing the address value). The vector address is sampled by the processor at B7, when it has detected that **VICVECTADDRV** is asserted. **VICIRQACK** is then deasserted, and one cycle later both **VICVECTADDRV** and **nIRQ** are deasserted to prevent the processor sampling the IRQ a second time before the VIC has cleared the interrupt. The processor only samples **nIRQ** while **VICVECTADDRV** is deasserted.

Figure 2-13 on page 2-26 shows the basic timing for an active IRQ when the processor clock is twice the frequency of the bus clock, and the interrupt acknowledge is asserted on the falling edge of the bus clock.

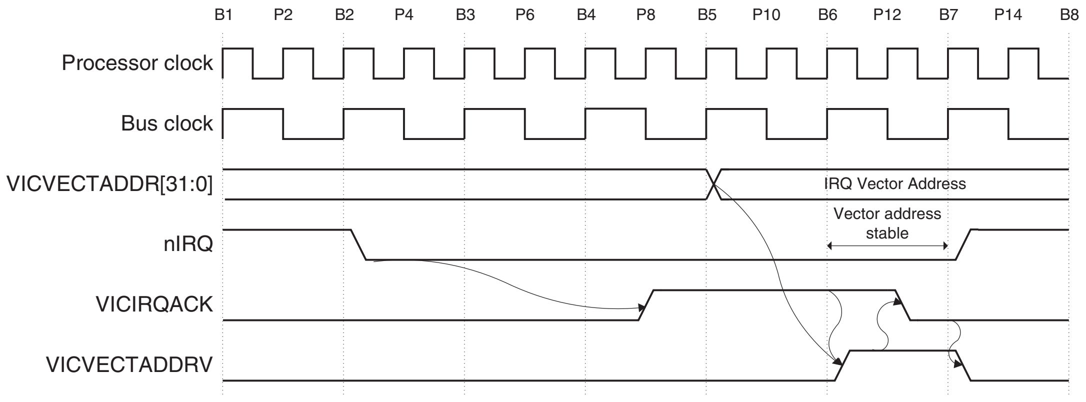

Figure 2-13 VIC port timing example, processor clock synchronous and twice bus clock frequency

Because the processor clock is running at twice the speed of the bus clock, the **VICIRQACK** response from the processor is valid earlier than when the processor and bus clocks are the same, at time P8. The **VICVECTADDRV** output is asserted at time B6 after the address has been generated and the processor acknowledge has been detected. This is the same as when the clocks are at the same frequency, because of the synchronization logic required on the **nIRQ** path. The vector address value is then sampled on processor clock edge P12, and the acknowledge is deasserted. The **nIRQ** and address valid outputs are deasserted on bus clock edge B7, after the acknowledge signal has been sampled.

VICVECTADDR[31:0] is generated from **nIRQ** which is then synchronized to the bus clock (two clock cycles) and then though the priority logic (one clock cycle) into the address generation logic which produces the address.

2.3.6 Asynchronous mode VIC port timing

The ARM11 processor can operate using a clock that is asynchronous to the system bus clock. In this mode, the operation of the interrupt interface is slightly different, to account for the synchronization that must occur between the control signals of the two devices.

Figure 2-14 on page 2-27 shows the basic timing for an active IRQ when operating in asynchronous mode.

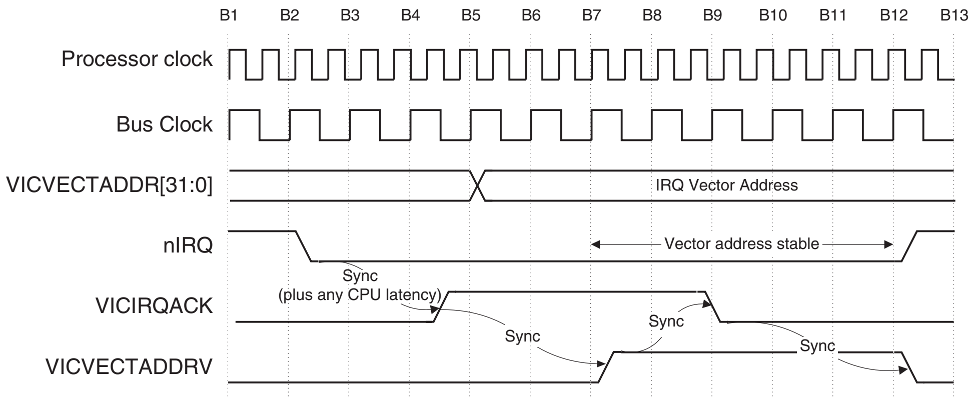

Figure 2-14 VIC port timing example, processor and bus clocks asynchronous

The **nIRQ** output is asserted on bus clock edge B2, causing the **VICIRQACK** processor output to be asserted after a synchronization delay. This response is then synchronized back to the bus clock, and the **VICVECTADDRV** output is asserted at B7. When the processor has detected the response from the VIC, it samples the vector address and deasserts the **VICIRQACK** output to indicate that the address has been sampled. The **VICVECTADDRV** output is deasserted after a bus clock synchronization delay, along with the **nIRQ** output. From this point, the address might change to indicate the vector for a higher priority active interrupt.

Note

Because of the synchronization delays required by both the processor and the VIC, the handshaking process takes longer when running in asynchronous mode.

Chapter 3
Programmer's Model

This chapter describes the ARM PrimeCell VIC (PL192) registers, and provides details needed when programming the microcontroller. It contains the following sections:

3.1 About the programmer's model

To ensure that the Vector Address Register (see *Vector Address Register, VICADDRESS* on page 3-15) can be read in a single instruction, the PrimeCell VIC base address must be `0xFFFFF000`, the upper 4K of memory. Placing the PrimeCell VIC anywhere else in memory increases interrupt latency because the ARM processor is unable to access the VICADDRESS 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, a second instruction is required to read in the full address value, and takes longer to be performed.

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 allows the read of the VICADDRESS Register (at `0xFFFFFF00`) 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 a processor supporting high exception vectors is used and the **HIVECS** configuration pin is tied HIGH, the VIC must be located at `0xFFFEF000` to allow for the exception vectors which 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.

Note

Because the ARM11 and ARM1026EJ processors can access the interrupt address directly rather than using the VICADDRESS Register, the interrupt latency is not affected by the location of the VIC in the system memory map.

3.2 Summary of PrimeCell VIC registers

The PrimeCell VIC registers are shown in Table 3-1.

Table 3-1 PrimeCell VIC register summary

Address	Type	Width	Reset value	Name	Description
VIC base +0x000	Read	32	0x00000000	VICIRQSTATUS	IRQ Status Register
VIC base +0x004	Read	32	0x00000000	VICFIQSTATUS	FIQ Status Register
VIC base +0x008	Read	32	-	VICRAWINTR	Raw Interrupt Status Register
VIC base +0x00C	Read/write	32	0x00000000	VICINTSELECT	Interrupt Select Register
VIC base +0x0010	Read/write	32	0x00000000	VICINTENABLE	Interrupt Enable Register
VIC base +0x014	Write	32	-	VICINTENCLEAR	Interrupt Enable Clear Register
VIC base +0x018	Read/write	32	0x00000000	VICSOFTINT	Software Interrupt Register
VIC base +0x01C	Write	32	-	VICSOFTINTCLEAR	Software Interrupt Clear Register
VIC base +0x020	Read/write	1	0x0	VICPROTECTION	Protection Enable Register
VIC base +0x024	Read/write	16	0xFFFF	VICSWPRIORITY MASK	Software Priority Mask Register
VIC base +0x028	Read/write	4	0xF	VICPRIORITYDAISY	Vector Priority Register for Daisy Chain
VIC base +0x100	Read/write	32	0x00000000	VICVECTADDR0	Vector Address 0 Register
VIC base +0x104	Read/write	32	0x00000000	VICVECTADDR1	Vector Address 1 Register
VIC base +0x108	Read/write	32	0x00000000	VICVECTADDR2	Vector Address 2 Register

Table 3-1 PrimeCell VIC register summary (continued)

Address	Type	Width	Reset value	Name	Description
VIC base +0x10C	Read/write	32	0x00000000	VICVECTADDR3	Vector Address 3 Register
VIC base +0x110	Read/write	32	0x00000000	VICVECTADDR4	Vector Address 4 Register
VIC base +0x114	Read/write	32	0x00000000	VICVECTADDR5	Vector Address 5 Register
VIC base +0x118	Read/write	32	0x00000000	VICVECTADDR6	Vector Address 6 Register
VIC base +0x11C	Read/write	32	0x00000000	VICVECTADDR7	Vector Address 7 Register
VIC base +0x120	Read/write	32	0x00000000	VICVECTADDR8	Vector Address 8 Register
VIC base +0x124	Read/write	32	0x00000000	VICVECTADDR9	Vector Address 9 Register
VIC base +0x128	Read/write	32	0x00000000	VICVECTADDR10	Vector Address 10 Register
VIC base +0x12C	Read/write	32	0x00000000	VICVECTADDR11	Vector Address 11 Register
VIC base +0x130	Read/write	32	0x00000000	VICVECTADDR12	Vector Address 12 Register
VIC base +0x134	Read/write	32	0x00000000	VICVECTADDR13	Vector Address 13 Register
VIC base +0x138	Read/write	32	0x00000000	VICVECTADDR14	Vector Address 14 Register
VIC base +0x13C	Read/write	32	0x00000000	VICVECTADDR15	Vector Address 15 Register
VIC base +0x140	Read/write	32	0x00000000	VICVECTADDR16	Vector Address 16 Register
VIC base +0x144	Read/write	32	0x00000000	VICVECTADDR17	Vector Address 17 Register
VIC base +0x148	Read/write	32	0x00000000	VICVECTADDR18	Vector Address 18 Register

Table 3-1 PrimeCell VIC register summary (continued)

Address	Type	Width	Reset value	Name	Description
VIC base +0x14C	Read/write	32	0x00000000	VICVECTADDR19	Vector Address 19 Register
VIC base +0x150	Read/write	32	0x00000000	VICVECTADDR20	Vector Address 20 Register
VIC base +0x154	Read/write	32	0x00000000	VICVECTADDR21	Vector Address 21 Register
VIC base +0x158	Read/write	32	0x00000000	VICVECTADDR22	Vector Address 22 Register
VIC base +0x15C	Read/write	32	0x00000000	VICVECTADDR23	Vector Address 23 Register
VIC base +0x160	Read/write	32	0x00000000	VICVECTADDR24	Vector Address 24 Register
VIC base +0x164	Read/write	32	0x00000000	VICVECTADDR25	Vector Address 25 Register
VIC base +0x168	Read/write	32	0x00000000	VICVECTADDR26	Vector Address 26 Register
VIC base +0x16C	Read/write	32	0x00000000	VICVECTADDR27	Vector Address 27 Register
VIC base +0x170	Read/write	32	0x00000000	VICVECTADDR28	Vector Address 28 Register
VIC base +0x174	Read/write	32	0x00000000	VICVECTADDR29	Vector Address 29 Register
VIC base +0x178	Read/write	32	0x00000000	VICVECTADDR30	Vector Address 30 Register
VIC base +0x17C	Read/write	32	0x00000000	VICVECTADDR31	Vector Address 31 Register
VIC base +0x200	Read/write	4	0xF	VICVECTPRIORITY0	Vector Priority 0 Register
VIC base +0x204	Read/write	4	0xF	VICVECTPRIORITY1	Vector Priority 1 Register
VIC base +0x208	Read/write	4	0xF	VICVECTPRIORITY2	Vector Priority 2 Register

Table 3-1 PrimeCell VIC register summary (continued)

Address	Type	Width	Reset value	Name	Description
VIC base +0x20C	Read/write	4	0xF	VICVECTPRIORITY3	Vector Priority 3 Register
VIC base +0x210	Read/write	4	0xF	VICVECTPRIORITY4	Vector Priority 4 Register
VIC base +0x214	Read/write	4	0xF	VICVECTPRIORITY5	Vector Priority 5 Register
VIC base +0x218	Read/write	4	0xF	VICVECTPRIORITY6	Vector Priority 6 Register
VIC base +0x21C	Read/write	4	0xF	VICVECTPRIORITY7	Vector Priority 7 Register
VIC base +0x220	Read/write	4	0xF	VICVECTPRIORITY8	Vector Priority 8 Register
VIC base +0x224	Read/write	4	0xF	VICVECTPRIORITY9	Vector Priority 9 Register
VIC base +0x228	Read/write	4	0xF	VICVECTPRIORITY10	Vector Priority 10 Register
VIC base +0x22C	Read/write	4	0xF	VICVECTPRIORITY11	Vector Priority 11 Register
VIC base +0x230	Read/write	4	0xF	VICVECTPRIORITY12	Vector Priority 12 Register
VIC base +0x234	Read/write	4	0xF	VICVECTPRIORITY13	Vector Priority 13 Register
VIC base +0x238	Read/write	4	0xF	VICVECTPRIORITY14	Vector Priority 14 Register
VIC base +0x23C	Read/write	4	0xF	VICVECTPRIORITY15	Vector Priority 15 Register
VIC base +0x240	Read/write	4	0xF	VICVECTPRIORITY16	Vector Priority 16 Register
VIC base +0x244	Read/write	4	0xF	VICVECTPRIORITY17	Vector Priority 17 Register
VIC base +0x248	Read/write	4	0xF	VICVECTPRIORITY18	Vector Priority 18 Register

Table 3-1 PrimeCell VIC register summary (continued)

Address	Type	Width	Reset value	Name	Description
VIC base +0x24C	Read/write	4	0xF	VICVECTPRIORITY19	Vector Priority 19 Register
VIC base +0x250	Read/write	4	0xF	VICVECTPRIORITY20	Vector Priority 20 Register
VIC base +0x254	Read/write	4	0xF	VICVECTPRIORITY21	Vector Priority 21 Register
VIC base +0x258	Read/write	4	0xF	VICVECTPRIORITY22	Vector Priority 22 Register
VIC base +0x25C	Read/write	4	0xF	VICVECTPRIORITY23	Vector Priority 23 Register
VIC base +0x260	Read/write	4	0xF	VICVECTPRIORITY24	Vector Priority 24 Register
VIC base +0x264	Read/write	4	0xF	VICVECTPRIORITY25	Vector Priority 25 Register
VIC base +0x268	Read/write	4	0xF	VICVECTPRIORITY26	Vector Priority 26 Register
VIC base +0x26C	Read/write	4	0xF	VICVECTPRIORITY27	Vector Priority 27 Register
VIC base +0x270	Read/write	4	0xF	VICVECTPRIORITY28	Vector Priority 28 Register
VIC base +0x274	Read/write	4	0xF	VICVECTPRIORITY29	Vector Priority 29 Register
VIC base +0x278	Read/write	4	0xF	VICVECTPRIORITY30	Vector Priority 30 Register
VIC base +0x27C	Read/write	4	0xF	VICVECTPRIORITY31	Vector Priority 31 Register
VIC base +0xF00	Read/write	32	0x00000000	VICADDRESS	Vector Address Register
VIC base +0xFE0	Read	8	0x92	VICPERIPHID0	Peripheral Identification Register bits 7:0
VIC base +0xFE4	Read	8	0x11	VICPERIPHID1	Peripheral Identification Register bits 15:8

Table 3-1 PrimeCell VIC register summary (continued)

Address	Type	Width	Reset value	Name	Description
VIC base +0xFE8	Read	8	0x04	VICPERIPHID2	Peripheral Identification Register bits 23:16
VIC base +0xFEC	Read	8	0x00	VICPERIPHID3	Peripheral Identification Register bits 31:24
VIC base +0xFF0	Read	8	0x0D	VICPCELLID0	PrimeCell Identification Register bits 7:0
VIC base +0xFF4	Read	8	0xF0	VICPCELLID1	PrimeCell Identification Register bits 15:8
VIC base +0xFF8	Read	8	0x05	VICPCELLID2	PrimeCell Identification Register bits 23:16
VIC base +0xFFC	Read	8	0xB1	VICPCELLID3	PrimeCell Identification Register bits 31:24

3.3 Register descriptions

The following registers are described in this section:

- *IRQ Status Register, VICIRQSTATUS*
- *FIQ Status Register, VICFIQSTATUS* on page 3-10
- *Raw Interrupt Status Register, VICRAWINTR* on page 3-11
- *Interrupt Select Register, VICINTSELECT* on page 3-11
- *Interrupt Enable Register, VICINTENABLE* on page 3-12
- *Interrupt Enable Clear Register, VICINTENCLEAR* on page 3-13
- *Software Interrupt Register, VICSOFTINT* on page 3-13
- *Software Interrupt Clear Register, VICSOFTINTCLEAR* on page 3-13
- *Protection Enable Register, VICPROTECTION* on page 3-14
- *Vector Address Register, VICADDRESS* on page 3-15
- *Software Priority Mask Register, VICSWPRIORITYMASK* on page 3-15
- *Vector Address Registers, VICVECTADDR[0-31]* on page 3-16
- *Vector Priority Registers, VICVECTPRIORITY[0-31] and VICVECTPRIORITYDAISY* on page 3-16
- *Peripheral Identification Registers, VICPERIPHID0-3* on page 3-17
- *PrimeCell Identification Registers, VICPCELLID0-3* on page 3-20.

3.3.1 IRQ Status Register, VICIRQSTATUS

The VICIRQSTATUS Register provides the status of interrupts [31:0] after IRQ masking. Because of the use of dual-stage synchronization logic, the VICIRQSTATUS Register takes two clock cycles to update. This register can be accessed with zero wait states.

Note

To access the sampled status of the interrupts, use the ISS bit in the VICITCR Register (see *Test Control Register, VICITCR* on page 4-4).

Table 3-2 shows the bit assignment of the VICIRQSTATUS Register.

Table 3-2 VICIRQSTATUS Register bit assignments

Bits	Name	Type	Function
[31:0]	IRQStatus	Read	Shows the status of the interrupts after masking by the VICINTENABLE and VICINTSELECT Registers: 0 = interrupt is inactive (reset) 1 = interrupt is active. There is one bit of the register for each interrupt source.

Note

When a system has multiple VICs, and the VICIRQSTATUS Register is used to determine which interrupt source must be served (instead of using the VIC port or reading the VICADDRESS Register), the interrupt handler might have to read the VICIRQSTATUS Register of all the VICs in the system. This is because the interrupt status from daisy-chained VICs cannot be observed from the VICIRQSTATUS Register for the first VIC connected to the processor.

3.3.2 FIQ Status Register, VICFIQSTATUS

The VICFIQSTATUS Register provides the status of the interrupts after FIQ masking. The VICFIQSTATUS Register is 32 bits wide. There is normally only one FIQ in the system. You can allow more than one interrupt source to generate a FIQ. The FIQ handler can then read this register to determine which FIQ interrupt source is active. Because of the use of dual-stage synchronization logic, the VICFIQSTATUS Register takes two clock cycles to update. This register can be accessed with zero wait states.

Table 3-3 shows the bit assignment of the VICFIQSTATUS Register.

Table 3-3 VICFIQSTATUS Register bit assignments

Bits	Name	Type	Function
[31:0]	FIQStatus	Read	Shows the status of the FIQ interrupts after masking by the VICINTENABLE and VICINTSELECT Registers: 0 = interrupt is inactive (reset) 1 = interrupt is active. There is one bit of the register for each interrupt source.

Note

This register is 32 bits wide to allow the FIQ to be placed on any of the input interrupt lines, but a typical system only contains one FIQ interrupt source.

When a system has multiple VICs, and the FIQ source can be located on a daisy-chained VIC, the interrupt handler might have to read the VICFIQSTATUS Register of all the VICs in the system. This is because the interrupt status from daisy-chained VICs cannot be observed from the VICFIQSTATUS Register for the first VIC connected to the processor.

3.3.3 Raw Interrupt Status Register, VICRAWINTR

The VICRAWINTR Register provides the unmasked status of the interrupt sources (either hardware or software). Because of the use of dual-stage synchronization logic, the VICRAWINTR Register takes two clock cycles to update. This register can be accessed with zero wait states.

Table 3-4 shows the bit assignment of the VICRAWINTR Register.

Table 3-4 VICRAWINTR Register bit assignments

Bits	Name	Type	Function
[31:0]	RawInterrupt	Read	Shows the status of the interrupts before masking by the Enable Registers: 0 = interrupt is inactive before masking 1 = interrupt is active before masking. Because this register provides a direct view of the raw interrupt inputs, the reset value is unknown. There is one bit of the register for each interrupt source.

3.3.4 Interrupt Select Register, VICINTSELECT

The VICINTSELECT Register selects whether the corresponding interrupt source generates an FIQ or IRQ interrupt. This register can be accessed with zero wait states.

Table 3-5 shows the bit assignment of the VICINTSELECT Register.

Table 3-5 VICINTSELECT Register bit assignments

Bits	Name	Type	Function
[31:0]	IntSelect	Read/write	Selects type of interrupt for interrupt request: 0 = IRQ interrupt (reset) 1 = FIQ interrupt. There is one bit of the register for each interrupt source.

Note

A standard system only has one FIQ source, so only one bit of this register must be set HIGH. This register must only be modified when the relevant interrupts are disabled. Changing the type of an interrupt when it is currently active and enabled can result in unpredictable behavior.

3.3.5 Interrupt Enable Register, VICINTENABLE

The VICINTENABLE Register enables the interrupt request lines, by unmasking the interrupt sources for the IRQ interrupt. This register can be accessed with zero wait states.

Table 3-6 shows the bit assignment of the VICINTENABLE Register.

Table 3-6 VICINTENABLE Register bit assignments

Bits	Name	Type	Function
[31:0]	IntEnable	Read/write	Enables the interrupt request lines, which allow the interrupts to reach the processor. Read: 0 = interrupt disabled (reset) 1 = interrupt enabled. The interrupt enable can only be set using this register. The VICINTENCLEAR Register must be used to disable the interrupt enable. Write: 0 = no effect 1 = interrupt enabled. On reset, all interrupts are disabled. There is one bit of the register for each interrupt source.

3.3.6 Interrupt Enable Clear Register, VICINTENCLEAR

The VICINTENCLEAR Register clears bits in the VICINTENABLE Register, and masks out the interrupt sources for the IRQ interrupt. This register can be accessed with zero wait states.

Table 3-7 shows the bit assignment of the VICINTENCLEAR Register.

Table 3-7 VICINTENCLEAR Register bit assignments

Bits	Name	Type	Function
[31:0]	IntEnable Clear	Write	Clears corresponding bits in the VICINTENABLE Register: 0 = no effect 1 = interrupt disabled in VICINTENABLE Register. There is one bit of the register for each interrupt source.

3.3.7 Software Interrupt Register, VICSOFTINT

The VICSOFTINT Register is used to generate software interrupts. This register can be accessed with zero wait states.

Table 3-8 shows the bit assignment of the VICSOFTINT Register.

Table 3-8 VICSOFTINT Register bit assignments

Bits	Name	Type	Function
[31:0]	SoftInt	Read/write	Setting a bit HIGH generates a software interrupt for the selected source before interrupt masking. Read: 0 = software interrupt inactive (reset) 1 = software interrupt active. Write: 0 = no effect 1 = software interrupt enabled. There is one bit of the register for each interrupt source.

3.3.8 Software Interrupt Clear Register, VICSOFTINTCLEAR

The VICSOFTINTCLEAR Register clears bits in the VICSOFTINT Register.This register can be accessed with zero wait states.

Table 3-9 shows the bit assignment of the VICSOFTINTCLEAR Register.

Table 3-9 VICSOFTINTCLEAR Register bit assignments

Bits	Name	Type	Function
[31:0]	SoftIntClear	Write	Clears corresponding bits in the VICSOFTINT Register: 0 = no effect 1 = software interrupt disabled in the VICSOFTINT Register. There is one bit of the register for each interrupt source.

3.3.9 Protection Enable Register, VICPROTECTION

The VICPROTECTION Register enables or disables protected register access, stopping register accesses when the processor is in User mode. This register can be accessed with zero wait states.

Table 3-10 shows the bit assignment of the VICPROTECTION Register.

Table 3-10 VICPROTECTION Register bit assignments

Bits	Name	Type	Function
[31:1]	Reserved	-	Reserved, read as zero, do not modify.
[0]	Protection	Read/write	Enables or disables protected register access: 0 = protection mode disabled (reset) 1 = protection mode enabled. When enabled, only privileged mode accesses (reads and writes) can access the interrupt controller registers, that is, when **HPROT[1]** is set HIGH for the current transfer. When disabled, both user mode and privileged mode can access the registers. This register can only be accessed in privileged mode, even when protection mode is disabled.

——— **Note** ———

If a system bus master accessing the VIC cannot generate accurate protection information, leave this register in its reset state to allow User mode access.

3.3.10 Vector Address Register, VICADDRESS

The VICADDRESS Register contains the *Interrupt Service Routine* (ISR) address of the currently active interrupt. If no interrupt is currently active, the register holds the ISR address of the last active interrupt. This register can be accessed with zero wait states.

Table 3-11 shows the bit assignment of the VICADDRESS Register.

Table 3-11 VICADRESS Register bit assignments

Bits	Name	Type	Function
[31:0]	VectAddr	Read/write	Contains the address of the currently active ISR, with reset value `0x00000000`. A read of this register returns the address of the ISR and sets the current interrupt as being serviced. A read must only be performed while there is an active interrupt. A write of any value to this register clears the current interrupt. A write must only be performed at the end of an interrupt service routine.

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. The register must be used as follows:

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

Reading or writing to this register at other times can cause incorrect operation.

3.3.11 Software Priority Mask Register, VICSWPRIORITYMASK

The VICSWPRIORITYMASK Register contains the mask value for the interrupt priority levels. This register can be accessed with zero wait states.

Table 3-12 shows the bit assignment of the VICSWPRIORITYMASK Register.

Table 3-12 VICSWPRIORITYMASK Register bit assignments

Bits	Name	Type	Function
[31:16]	Reserved	-	Reserved, read as zero, do not modify.
[15:0]	SWPriorityMask	Read/write	Controls software masking of the 16 interrupt priority levels: 0 = interrupt priority level is masked 1 = interrupt priority level is not masked (reset). Each bit of the register is applied to each of the 16 interrupt priority levels.

3.3.12 Vector Address Registers, VICVECTADDR[0-31]

The VICVECTADDR[0-31] Registers contain the ISR vector addresses. These registers can be accessed with one wait state.

Table 3-13 shows the bit assignment of the VICVECTADDR[0-31] Registers.

Table 3-13 VICVECTADDR[0-31] Register bit assignments

Bits	Name	Type	Function
[31:0]	VectorAddr 0-31	Read/write	Contains ISR vector addresses.

Note

These registers must only be updated when the relevant interrupts are disabled. Receiving an interrupt while the vector address is being written to can result in unpredictable behavior.

If the system does not support interrupt vector addresses, the VICVECTADDR Registers can be programmed with the numbers of the interrupt source ports they relate to, so that the source of the active interrupt can be easily determined.

3.3.13 Vector Priority Registers, VICVECTPRIORITY[0-31] and VICVECTPRIORITYDAISY

The VICVECTPRIORITY[0-31] and VICVECTPRIORITYDAISY Registers select the interrupt priority level for the 32 vectored interrupt sources, and the daisy chain input. The value can be from 0-15. The default values have all interrupts on the same priority level, 15, which is the lowest. This enables any of the vectored interrupts to be promoted to a higher priority with one simple register write. These registers can be accessed with one wait state.

Table 3-14 shows the bit assignment of the VICVECTPRIORITY[0-31] and VICVECTPRIORITYDAISY Registers.

Table 3-14 VICVECTPRIORITY[0-31] and VICVECTPRIORITYDAISY Register bit assignments

Bits	Name	Type	Function
[31:4]	Reserved	-	Reserved, read as zero, do not modify.
[3:0]	VectPriority	Read/write	Selects vectored interrupt priority level. You can select any of the 16 vectored interrupt priority levels by programming the register with the hexadecimal value of the priority level required, from 0-15.

Note

Hardware priority levels only take effect when multiple interrupts are programmed to have the same priority level and occur at the same time. In this case, vectored interrupt 0 has the highest priority, and interrupt 31 has the lowest priority.

If the VIC is used as part of a daisy-chain configuration in VIC0 mode, ensure that the interrupt service routine does not change the VICVECTPRIORITYDAISY register to a higher priority.

3.3.14 Peripheral Identification Registers, VICPERIPHID0-3

The VICPERIPHID0-3 Registers are four 8-bit registers, that span address locations `0xFE0-0xFEC`. The registers can conceptually be treated as a single 32-bit register. The read-only registers provide the following options of the peripheral:

Part number [11:0]

This identifies the peripheral. The three digit product code `0x192` is used for the PrimeCell VIC.

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 and the value is revision-dependent.

Configuration [31:24]

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

Figure 3-1 shows the bit assignment for the VICPeriphID0-3 Registers.

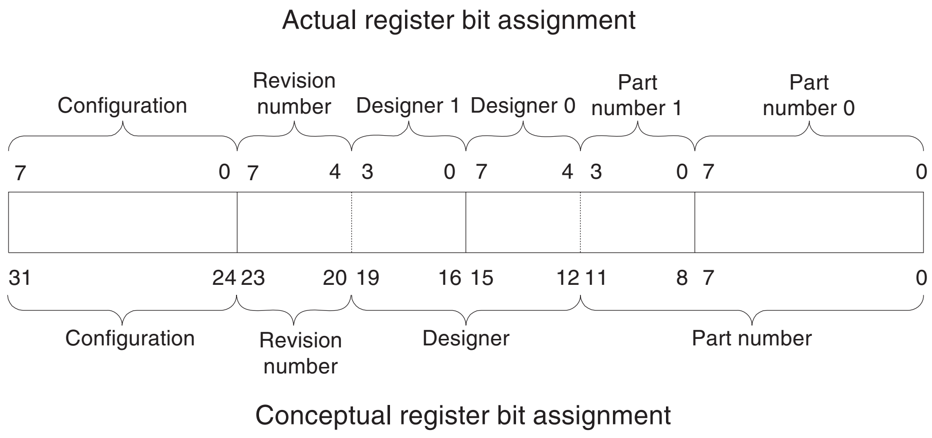

Figure 3-1 Peripheral Identification Register bit assignment

The four 8-bit peripheral identification registers are described in the following sections:

- *VICPERIPHID0 Register*
- *VICPERIPHID1 Register* on page 3-19
- *VICPERIPHID2 Register* on page 3-19
- *VICPERIPHID3 Register* on page 3-20.

VICPERIPHID0 Register

The VICPERIPHID0 Register is hard-coded and the fields within the register determine the reset value. Table 3-15 shows the bit assignment of the VICPERIPHID0 Register.

Table 3-15 VICPERIPHID0 Register bit assignments

Bits	Name	Type	Function
[31:8]	-	-	Reserved, read as zero, do not modify.
[7:0]	Partnumber0	Read	These bits read back as `0x192`.

VICPERIPHID1 Register

The VICPERIPHID1 Register is hard-coded and the fields within the register determine the reset value. Table 3-16 shows the bit assignment of the VICPERIPHID1 Register.

Table 3-16 VICPERIPHD1 Register bit assignments

Bits	Name	Type	Function
[31:8]	-	-	Reserved, read as zero, do not modify.
[7:4]	Designer0	Read	These bits read back as `0x1`.
[3:0]	Partnumber1	Read	These bits read back as `0x1`.

VICPERIPHID2 Register

The VICPERIPHID2 Register is hard-coded and the fields within the register determine the reset value. Table 3-17 shows the bit assignment of the VICPERIPHID2 Register.

Table 3-17 VICPERIPHID2 Register bit assignments

Bits	Name	Type	Function
[31:8]	-	-	Reserved, read as zero, do not modify.
[7:4]	Revision	Read	These bits read back as the revision number, which can be between 0 and 15.
[3:0]	Designer1	Read	These bits read back as `0x4`.

VICPERIPHID3 Register

The VICPERIPHID3 Register is hard-coded and the fields within the register determine the reset value. Table 3-18 shows the bit assignment of the VICPERIPHID3 Register.

Table 3-18 VICPERIPHID3 Register bit assignments

Bits	Name	Type	Function
[31:8]	-	-	Reserved, read as zero, do not modify.
[7:2]	Configuration	Read	These bits read back as `0x0`.
[1:0]	Configuration	Read	Indicates the number of interrupts supported: 00 = 32 (default) 01 = 64 10 = 128 11 = 256.

3.3.15 PrimeCell Identification Registers, VICPCELLID0-3

The VICPCELLID0-3 Registers are four 8-bit registers, that span address locations `0xFF0-0xFFC`. The read-only register can conceptually be treated as a single 32-bit register. The register is used as a standard cross-peripheral identification system. Figure 3-2 shows the bit assignment for the VICPCELLID0-3 Registers.

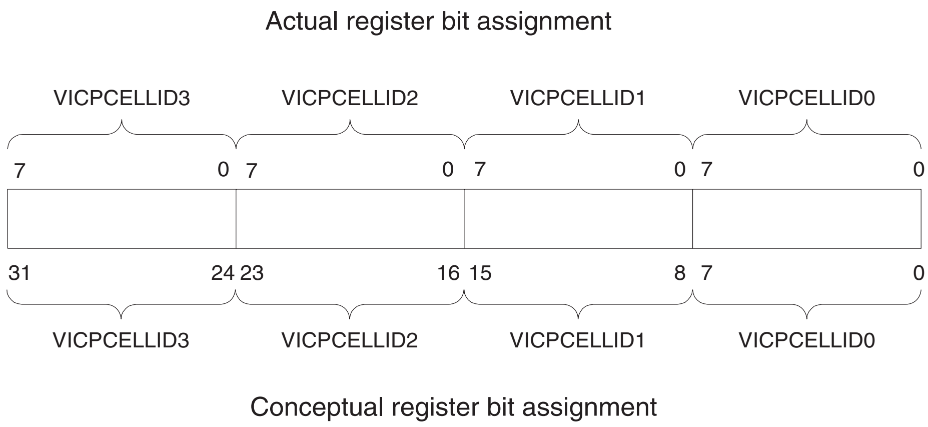

Figure 3-2 PrimeCell Identification Register bit assignment

The four 8-bit registers are described in the following subsections:

- *VICPCELLID0 Register*
- *VICPCELLID1 Register*
- *VICPCELLID2 Register*
- *VICPCELLID3 Register* on page 3-22.

VICPCELLID0 Register

The VICPCELLID0 Register is hard-coded and the fields within the register determine the reset value. Table 3-19 shows the bit assignment of the VICPCELLID0 Register.

Table 3-19 VICPCELLID0 Register bit assignments

Bits	Name	Type	Function
[31:8]	-	-	Reserved, read as zero, do not modify.
[7:0]	VICPCellID0	Read	These bits read back as `0x0D`.

VICPCELLID1 Register

The VICPCELLID1 Register is hard-coded and the fields within the register determine the reset value. Table 3-20 shows the bit assignment of the VICPCELLID1 Register.

Table 3-20 VICPCELLID1 Register bit assignments

Bits	Name	Type	Function
[31:8]	-	-	Reserved, read as zero, do not modify.
[7:0]	VICPCellID1	Read	These bits read back as `0xF0`.

VICPCELLID2 Register

The VICPCELLID2 Register is hard-coded and the fields within the register determine the reset value. Table 3-21 shows the bit assignment of the VICPCELLID2 Register.

Table 3-21 VICPCELLID2 Register bit assignments

Bits	Name	Type	Function
[31:8]	-	-	Reserved, read as zero, do not modify.
[7:0]	VICPCellID2	Read	These bits read back as `0x05`.

VICPCELLID3 Register

The VICPCELLID3 Register is hard-coded and the fields within the register determine the reset value. Table 3-22 shows the bit assignment of the VICPCELLID3 Register.

Table 3-22 VICPCELLID3 Register bit assignments

Bits	Name	Type	Function
[31:8]	-	-	Reserved, read as zero, do not modify.
[7:0]	VICPCellID3	Read	These bits read back as `0xB1`.

3.4 Interrupt latency

Using the VIC port to acknowledge the servicing of an IRQ interrupt reduces the interrupt latency, because the processor does not need to do an AHB access to obtain the address for the ISR. The speed of the handshaking between the VIC and processor depends on the clock frequencies used, and whether the clocks are synchronous or asynchronous. Because the timing for each core differs, see the relevant ARM processor Technical Reference Manual for information on the instruction cycle times. The processor used and its configuration also influence the time for the interrupts to be serviced.

Caution

For accurate timing of interrupts, run code on a cycle-accurate ARM model.

The ARM architecture defines several exceptions to be handled by the processor. The two exceptions that are the main concern are the interrupt and fast interrupt exceptions, triggered by the IRQ and FIQ signals respectively.

The IRQ and FIQ signals are generated by the VIC external to the ARM core to signal real-world events. The speed of response to these signals is known as interrupt latency. The factors influencing interrupt latency are described in this section, using an FIQ as an example.

Interrupt latency is described in more detail in the following sections:

- *Core latency*
- *Memory systems and cycle types* on page 3-24
- *Tightly-coupled memory* on page 3-25
- *Caches* on page 3-26
- *Compiler optimizations* on page 3-26.

Note

For examples of interrupt latency, see *Example interrupt latency calculations* on page 3-27.

3.4.1 Core latency

Several factors influence latency, including the functionality of the ARM core itself, so choosing the ARM core with the right characteristics for a particular application is important. The choice of memory, the partitioning of software and the optimization of that code are also important.

As an example, this section describes an ARM7TDMI and its behavior in response to the FIQ signal.

The FIQ signal is asserted by a device peripheral to the ARM core. The signal is asserted asynchronously to the core and therefore must be synchronized. This requires up to a maximum of four processor cycles. When the signal is synchronized, the core completes the instruction currently in the execution stage of the pipeline, flushes the subsequent instructions and begins loading the instruction pipeline with the instruction located at the FIQ interrupt vector.

The instruction that takes the longest to complete is the *LoaD Multiple* (`LDM`) instruction, when all 16 registers are to be loaded, including the program counter. The instruction takes 20 processor cycles to execute in a system where both instructions and data are read from zero wait-state memory.

This gives a total of 26 cycles as the maximum interrupt latency for FIQ in a system. However, another aspect can extend the latency in this situation. If the memory controller detects accesses to nonexistent memory locations, for example if the `LDM` is attempting to load the register values from nonexistent memory, it can generate an abort signal. This causes the processor to enter the exception processing routine for data aborts before entering the FIQ interrupt processing routine. The data abort exception handler is a higher priority than FIQ, but execution passes to the FIQ handler immediately and returns to the abort handler on exit from the FIQ handler. This enables the data abort context to be preserved with minimum delay for FIQ handling.

The worst case for an ARM7TDMI processor connected to an ideal memory system, and including data abort entry, is 29 processor cycles.

3.4.2 Memory systems and cycle types

The example in *Core latency* on page 3-23 assumes zero wait-state memory as the basis for the calculation. This enables cycles that are internal to the core or access the memory system to be reguarded as they span equal time units. The figures quoted for cores are usually made on this basis.

In most systems it is not practical to implement zero-wait state memory or, at best, to implement zero-wait state memory for only a small part of the total system memory. To understand the effects of different memory types on interrupt response, consider the type of cycles being executed by the processor during execution of an instruction and the way the memory system responds to the different cycle types.

The two cycle types used to access memory are sequential and nonsequential cycles:

- sequential accesses are related to the previous memory access
- nonsequential accesses are not necessarily related to the previous memory access.

Because many memory types are organized internally as a group of cells, sequential accesses can often be performed more efficiently, because most of the decoding for the cell location has already been performed by the nonsequential access.

The `LDM` instruction in the example fetches data from memory to load into the ARM core registers from sequential memory locations. This enables sequential cycles to be used for most of the memory accesses. Memory controllers can use the sequential accesses to fetch the data more efficiently from certain types of memory. However, the nonsequential accesses normally incur wait states. In addition, the events occurring during the execution of the `LDM` cause instructions to be loaded from the interrupt vector locations for the data abort and FIQ handlers. The data for example might be in SDRAM and the exception handlers in Flash memory.

Extend the example to take into account the effects of a real memory system that has a memory controller connected directly to the ARM7TDMI, and consider the number of memory accesses and the type of memory access. The type of access is important, because different access times are possible for sequential and nonsequential accesses.

3.4.3 Tightly-coupled memory

To improve system performance and enable the processor to access data and/or instructions without incurring wait states, there are two possible approaches:

- introduce caches to the system
- include a small amount of zero wait-state memory tightly coupled to the core for use in the most time-critical sections of the system operation.

The inclusion of tightly coupled memory interfaces in ARM cores such as the ARM966E-S simplifies the design of systems in which code and data must be accessed as quickly as possible by the core. In systems that incorporate *Tightly Coupled Memory* (TCM), the system partitioning can be explicit so that those parts of the software that are the most critical to system performance can be placed in memory locations which facilitate efficient execution of the application. For example, the FIQ handler can be placed in TCM so that the worst case interrupt response time is minimized by using zero wait-state memory.

In the sequence of events in processing a FIQ on the ARM966E-S, when the current instruction completes, the write buffer is drained and the FIQ vector is generated to start loading the FIQ handler code into the execution pipeline.

If the FIQ handler is located in program memory attached to the AMBA bus, this sequence of events includes at least one nonsequential memory access and therefore incurs wait states. Additionally, if a data abort exception is generated during the completion of the instruction, before execution of the FIQ handler code, fetching the abort vector code introduces a further nonsequential access. In a system without TCM, the number of nonsequential memory accesses can therefore add significantly to the worst case interrupt latency.

In an embedded system where there is flash memory and SRAM, it might be more efficient to copy the FIQ handler into SRAM and patch the interrupt vector rather than leave it in FlashRAM.

3.4.4 Caches

Caches are local fast storage for the core CPU. They enable systems with a large disparity between core execution speed and memory access speed to have portions of code and data available to the core without the performance penalty of fetching from memory. However, filling the cache incurs a memory overhead.

3.4.5 Compiler optimizations

The longest possible ARM instruction with the added overhead of a data abort is used to indicate one extreme. If an application does not include such an instruction or a data abort does not occur, the worst case for a given application is fewer cycles.

It is possible to reduce the maximum number of registers that a compiler restores using the `LDM` instruction. This reduces the number of cycles before the FIQ instruction handler can begin to load. The side effect is a small increase in code size over the minimum requirement, because more than one `LDM` might be required for a given code sequence. The standard compiler switch `-split_ldm` is used to reduce the number of registers transferred to 4 for `STM` and `LDM`.

3.5 Example interrupt latency calculations

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

- *ARM7TDMI FIQ interrupt latency during LDM*
- *ARM9 FIQ interrupt latency during LDM* on page 3-28
- *ARM10 FIQ interrupt latency during LDM* on page 3-31
- *Interrupts in ARM1026EJ and ARM11 cores* on page 3-31
- *IRQ during IRQ in an example ARMv5 system* on page 3-32
- *IRQ during IRQ in an example ARMv6 system* on page 3-33.

Note

The calculations are based on the following assumptions:

- the ISRs are in zero wait state memory
- no other masters are on the same bus (that is, the master is always granted control of the bus)
- the accesses are from noncached memory
- the processor is running at the same speed as the bus
- large `LDM/STM`s are allowed
- data aborts are allowed
- instruction aborts are allowed.

Changes in these assumptions can affect the resulting interrupt latency.

3.5.1 ARM7TDMI FIQ interrupt latency during LDM

FIQ interrupts have the highest priority in the PrimeCell VIC, and are not nested. In FIQ mode, seven 32-bit registers are banked into the system. This enables the PrimeCell VIC to process the interrupt as quickly as possible.

Table 3-23 on page 3-28 shows the typical worst case cycles for FIQ interrupts. The values in the table are based on the ARM7TDMI rev 4 core, with the following assumptions made:

- FIQ occurs at start of `LDM`
- `LDM` causes a data abort
- memory system consists entirely of zero wait state components

- `LDM` restores 16 registers including PC.

Table 3-23 ARM7TDMI FIQ interrupt latency

Event	Worst case
Request passes through synchronizer.	4 cycles
Longest instruction to complete worst case instruction execution. **Note** If the standard compiler switch `-split_ldm` is used to reduce the number of registers transferred to 4 for `STM` and `LDM`, this can be reduced to 8 cycles.	20 cycles
Data abort entry.	3 cycles
FIQ entry.	2 cycles
Total	29 cycles

Note

For best results, start the FIQ handler at the FIQ vector address, `0x0000001c` or `0xFFFF001c`. For information on compiler switch options, see the *ARM Development Suite Compiler, Linker and Utilities Guide*.

The ARM7TDMI-S core requires two cycles less, because the synchronizer only requires two cycles.

3.5.2 ARM9 FIQ interrupt latency during LDM

Table 3-24 on page 3-29 shows the typical worst case cycles for FIQ interrupts. The values in the table are based on the ARM966E-S rev 2 core, with the following assumptions made:

- FIQ occurs at start of `LDM`
- `LDM` causes a data abort
- FIQ and data abort handlers in zero wait state TCM, `LDM` data in zero wait state TCM

- `LDM` restores 16 registers including PC.

Table 3-24 ARM966E-S Rev 2 FIQ interrupt latency

Event	Worst case
`LDM` completes	16 cycles
`LDM` status transfer	1 cycle
Data abort detection	2 cycles
Data abort entry	2 cycles
Prepare for FIQ entry, fetch, and decode	3 cycles
Total	24 cycles
Additional considerations:	
Write buffer drain	5 cycles
Total for additional consideration	5 cycles

Table 3-25 shows the typical worst case cycles for FIQ interrupts. The values in the table are based on the ARM946E-S rev 1.1 core, with the following assumptions made:

- FIQ occurs at start of `LDM`
- `LDM` causes a data abort
- FIQ and data abort handlers in zero wait state TCM, `LDM` data in zero wait state TCM
- `LDM` restores 16 registers including PC.

Table 3-25 ARM946E-S Rev 1.1 FIQ interrupt latency

Event	Worst case
`LDM` completes	16 cycles
`LDM` status transfer	1 cycle
Data abort detection	2 cycles
Data abort entry	2 cycles
Prepare for FIQ entry, fetch, and decode	3 cycles
Total	24 cycles
Additional considerations:	

Table 3-25 ARM946E-S Rev 1.1 FIQ interrupt latency (continued)

Event	Worst case
Write buffer drain	5 cycles
Cache line fills (worst case 3)	48 cycles
Total for additional consideration	53 cycles

Table 3-26 shows the typical worst case cycles for FIQ interrupts. The values in the table are based on the ARM926EJ-S rev 0.3 core, with the following assumptions made:

- FIQ occurs at start of `LDM`
- `LDM` causes a data abort
- memory system consists entirely of zero wait state components
- `LDM` restores 16 registers including PC.

Table 3-26 ARM926EJ-S FIQ interrupt latency

Event	Worst case
`LDM` completes	16 cycles
`LDM` status transfer	1 cycle
Data abort detection	2 cycles
Data abort entry	2 cycles
Prepare for FIQ entry, fetch, and decode	3 cycles
Total	24 cycles
Additional considerations:	
Write buffer drain	5 cycles
Cache line fills (worst case 3)	48 cycles
Page table walks (worst case 2)	4 cycles
Total for additional consideration	57 cycles

3.5.3 ARM10 FIQ interrupt latency during LDM

Table 3-27 on page 3-31 shows the typical worst case cycles for FIQ interrupts. The values in the table are based on the ARM1026EJ-S rev 1 core, with the following assumptions made:

- FIQ occurs at start of `LDM`
- `LDM` causes a data abort
- memory system consists entirely of zero wait state components
- `LDM` restores 16 registers including PC.

Table 3-27 ARM1026EJ-S FIQ interrupt latency

Event	Worst case
`LDM` completes	16 cycles
`LDM` status transfer	1 cycle
Data abort detection	2 cycles
Data abort entry	2 cycles
Prepare for FIQ entry, fetch, and decode	3 cycles
Total	24 cycles
Additional considerations:	
Write buffer drain 32-bit AHB	8 cycles
Cache line fills (worst case 3) 32-bit AHB	42 cycles
Total for additional consideration	50 cycles

3.5.4 Interrupts in ARM1026EJ and ARM11 cores

Because of the complex inter-instruction dependencies, it is not possible to describe the exact behavior of all the ARM1026EJ/ARM11 instructions in all circumstances. Table 3-23 on page 3-28 to Table 3-27 are accurate in most cases but must not be used instead of running code on a cycle-accurate ARM model.

Two of the performance-enhancing architectural features of the ARM10 and ARM11 cores make it particularly difficult to count the number of cycles an instruction takes:

- branch prediction
- independent load/store unit.

The latencies described here assume the following:

- no outstanding data dependencies between an instruction and a previous instruction
- the instruction does not encounter any resource conflicts
- all data accesses hit the DCache and do not cross protection region boundaries
- all instruction accesses hit in the ICache.

Note

ARM11 (v6 architecture) processors have new instructions to accelerate the handling of exceptions:

- *Store Return State* (`SRS`)
- *Return From Exception* (`RFE`)
- *Change Processor State* (`CPS`).

The ARM11 processors can also be set into a low interrupt latency configuration reducing interrupt latency. See the *ARM1136JF-S Technical Reference Manual* for more information.

This section gives an extended example to show how the combination of new facilities of the ARM11 core improve interrupt latency. The example is not necessarily realistic, but shows the main points.

Timings are roughly consistent with ARM10 core, with the pipeline reload penalty being three cycles. It is assumed that pipeline reloads are combined to execute as quickly as possible and, in particular, that if an interrupt is detected during an instruction that has set a new value for the PC, after that value has been determined and written to the PC but before the resulting pipeline refill is completed, the pipeline refill is abandoned and the interrupt entry sequence started as soon as possible.

Similarly, if a higher-priority IRQ is detected during an exception entry sequence that does not disable IRQs, after the updates to R14, the *Saved Program Status Register* (SPSR), the *Current Program Status Register* (CPSR), and the PC but before the pipeline refill has completed, the pipeline refill is abandoned and the higher level IRQ entry sequence is started as soon as possible.

3.5.5 IRQ during IRQ in an example ARMv5 system

In ARMv5 systems, all IRQ interrupts come through the same vector, at address `0x00000018` or `0xFFFF0018`. The code at this vector must obtain the address of the correct handler from the VIC, branch to it, and store the registers to be overwritten by the interrupt service routine.

Example code to do this is shown below (the cycles relate to ARM1026EJ timing):

```
IRQ2handler  LDR     PC, [R8,#HandlerAddress];(6 cycles / 7 cycles is scaled
register offset used and no conflicts on AHB bus.) (No VIC port - VIC would
supply address thus reducing this latency)...IRQ1handler  ... Include code to
actually process the interrupt ...  STR     R0, [R8,#AckFinished]  SUBS    PC,
R14, #4...IRQ0handler  STMIA   R13, {R0-R3}    ;(4 cycles as R0-R3 used in next
instructions)  MOV     R0, LR ;(1 cycle)  MRS     R1, SPSR ;(1 cycle)  ADD
R2, R13, #8          ;(1 cycle)  MRS     R3, CPSR ;(1 cycle)  BIC     R3, R3,
#0x1F           ;(1 cycle)  ORR     R3, R3, #0x13   ; = Supervisor mode number
(1 cycles)  MSR     CPSR_c, R3 ;(4 cycle)  STMFD   R13!, {R0,R1}   ;(2 cycle)
LDMIA   R2, {R0,R1}     ;(2 cycle)  STMFD   R13!, {R0,R1}   ;(2 cycle)  LDMDB
R2, {R0,R1}          ;(2 cycle)  BIC     R3, R3, #0x80   ; = I bit;(1 cycles)
MSR     CPSR_c, R3        ;(4 cycles)  ... IRQs are now re-enabled, with
original R2, R3, R14, SPSR on stack  ... Include code to stack any more
registers required, process the interrupt  ... and unstack extra registers  ADR
R2, #VICADDRESS  MRS     R3, CPSR  ORR     R3, R3, #0x80    ; = I bit  MSR
CPSR_c, R3  STR     R0, [R2,#AckFinished]  LDR     R14, [R13,#12]   ; Original
SPSR value  MSR     SPSR_fsxc, R14  LDMFD   R13!, {R2,R3,R14}  ADD     R13, R13,
#4  SUBS    PC, R14, #4...
```

The major problem with this is the length of time IRQs are disabled at the start of the lower-priority IRQs. The worst-case interrupt latency for the IRQ0 interrupt occurs if a lower-priority IRQ2 has just fetched its handler address, and is approximately:

- 3 cycles for the pipeline refill after the `LDR` PC instruction fetches the handler address
- + 27 cycles to get to and execute the MSR instruction that re-enables IRQs
- + 3 cycles to re-enter the IRQ exception
- + 6 cycles for the `LDR` PC instruction at IRQ2handler

This gives a total of 39 cycles.

Note

IRQs must be disabled for the final store to acknowledge the end of the handler to the VIC. If not, badly-timed further IRQs, each occurring close to the end of the end of the previous handler, can cause unlimited growth of the locked-down stack.

3.5.6 IRQ during IRQ in an example ARMv6 system

Using the VIC and the new instructions, there is no longer any requirement for everything to go through the single IRQ vector, and the changeover to a different stack occurs much more smoothly. The code is similar to the example shown below:

```
IRQ1handler  ... Include code to actually process the interrupt ...  STR     R0,
[R8,#AckFinished]  SUBS    PC, R14, #4...IRQ2handler  SUB     R14, R14, #4  ;(1
cycle)  SRSFD   R13_svc!      ;(1 cycle)  CPSIE   f, #0x13      ; = Supervisor
mode    (1 cycle)  STMFD   R13!, {R2,R3} ;(2 cycles)  ... IRQs are now
```

```
re-enabled, with original R2, R3, R14, SPSR on stack  ... Include code to stack
any more registers required, process the interrupt  ... and unstack extra
registers  LDMFD   R13!, {R2,R3}  ADR     R14, #VICADDRESS  CPSID   f  STR
R0, [R14,#AckFinished]  RFEFD   R13!...
```

The worst-case interrupt latency for a IRQ1 now occurs if the IRQ1 occurs during an IRQ2 interrupt entry sequence, just after it disables IRQs, and is approximately:

- 3 cycles for the pipeline refill for the IRQ2's exception entry sequence
- + 5 cycles to get to and execute the CPSIE instruction that re-enables IRQs
- + 3 cycles to re-enter the IRQ exception

This gives a total of 11 cycles.

Note

In the ARMv5 system, the potential additional interrupt latency caused by a long `LDM` or `STM` being in progress when the IRQ is detected is only significant because the memory system can stretch its cycles considerably. Otherwise, it is dwarfed by the number of cycles lost because of FIQs being disabled at the start of a lower-priority interrupt handler. In ARMv6, this is still the case, but it is much closer.

3.6 ARM7TDMI IRQ interrupts using AHB

In IRQ mode, interrupt levels can be nested 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 SPSR must be available before further IRQ interrupts can be accepted. This increases the interrupt latency, but provides a scalable nested interrupt system. Table 3-28 shows the typical worst case cycles for IRQ interrupts.

Table 3-28 ARM7TDMI IRQ interrupt latency

Event	Worst case
Interrupt synchronization	4 cycles
Worst case interrupt disable period (sequence LDR, STMFD, MRS, MSR)	20 cycle
Entry to first instruction	2 cycles
Nesting (assuming single-state AHB)	10 cycles
Total	36 cycles

3.7 Interrupt priority

The interrupt priority is regulated by both hardware and software. FIQ interrupts have the highest priority, followed by the vectored interrupts 0-31, and the daisy-chained interrupt has the lowest priority. The priority order of the vectored interrupts is programmable.

To reduce interrupt latency (see *Interrupt latency* on page 3-23), you can re-enable the IRQ interrupts in the processor after the *Interrupt Service Routine* (ISR) is entered. In this case, the current ISR is interrupted and the higher-priority ISR is executed. The PrimeCell VIC then only allows interrupts of a higher priority to interrupt a lower-priority ISR. 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, the interrupts must be disabled, the LR and SPSR reloaded, and the VICADDRESS Register written to (see *Vectored interrupt service routine* on page B-5).

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:

- *PrimeCell VIC test harness overview* on page 4-2
- *Scan testing* on page 4-3
- *Test registers* on page 4-4.

4.1 PrimeCell VIC test harness overview

The additional logic for functional verification and production testing allows:

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

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

AMBA signals

These are tested 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 allowing you to read and write to each intra-chip input/output signal.

These test features are controlled by test registers. This allows you to test the PrimeCell 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 application of test vectors is controlled through the *Test Interface Controller* (TIC) AMBA bus master module.

4.2 Scan testing

The PrimeCell VIC is designed to simplify:

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

This provides an alternative method of manufacturing test.

4.3 Test registers

The PrimeCell VIC test registers are memory-mapped as shown in Table 4-1.

Table 4-1 Test registers memory map

Address	Type	Width	Reset value	Name	Description
VIC base + `0x300`	Read/write	2	`0b00`	VICITCR	Test Control Register
VIC base + `0x304`	Read/write	5	`0x000`	VICITIP1	Test Input Register 1
VIC base + `0x308`	Read/write	32	-	VICITIP2	Test Input Register 2
VIC base + `0x30C`	Read/write	4	`0x000`	VICITOP1	Test Output Register 1
VIC base + `0x310`	Read/write	32	`0x00000000`	VICITOP2	Test Output Register 2
VIC base + `0x314`	Read	32	`0x00000000`	VICINTSSTATUS	Sampled Interrupt Source Status Register
VIC base + `0x318`	Write	32	-	VICINTSSTATUS CLEAR	Sampled Interrupt Source Status Clear Register

The following registers are described in this section:

- *Test Control Register, VICITCR*
- *Integration Test Input Register 1, VICITIP1* on page 4-5
- *Integration Test Input Register 2, VICITIP2* on page 4-6
- *Integration Test Output Register 1, VICITOP1* on page 4-6
- *Integration Test Output Register 2, VICITOP2* on page 4-7
- *Sampled Interrupt Source Status Register, VICINTSSTATUS* on page 4-8
- *Sampled Interrupt Source Status Clear Register, VICINTSSTATUSCLEAR* on page 4-8.

4.3.1 Test Control Register, VICITCR

VICITCR is a 2-bit Test Control Register. The ITEN bit in this register controls the input test multiplexors. The ISS bit controls sampling of the interrupt source inputs. This register must only be used in test mode. The register can be accessed with zero wait states.

Table 4-2 shows the bit assignment of the VICITCR Register.

Table 4-2 VICITCR Register bit assignments

Bits	Name	Type	Description
[31:2]	Reserved	-	Reserved, read as zero, do not modify.
[1]	ISS	Read/write	Interrupt sampled status enable: 0 = normal mode (reset) 1 = interrupt sampling enabled.
[0]	ITEN	Read/write	Integration test enable: 0 = normal mode (reset) 1 = test mode.

4.3.2 Integration Test Input Register 1, VICITIP1

VICITIP1 is a 5-bit register used to control and read the values of the **nVICIRQIN**, **nVICFIQIN**, and **VICIRQACK** inputs. It is also used to read the values set on the **VICIRQREG** and **VICFIQREG** ports. This register must only be used in test mode. The read value in VICITIP1 always takes two clock cycles to update, because of the use of dual flip-flop synchronization logic which avoids metastability problems. For example, when a new value is written into this register, an immediate read from the register returns the old value. After two cycles, a read from this register returns the updated value. The register can be accessed with zero wait states.

Table 4-3 shows the bit assignment of the VICITIP1 Register.

Table 4-3 VICITIP1 Register bit assignments

Bits	Name	Type	Description
[31:11]	Reserved	-	Reserved, read as zero, do not modify.
[10]	VICFIQINREG	Read	This bit is read only and returns the value that is set on the **VICFIQINREG** port.
[9]	VICIRQINREG	Read	This bit is read only and returns the value that is set on the **VICIRQINREG** port.
[8]	VICIRQACK	Read/ write	Read the value of the **VICIRQACK** input when the VICITCR ITEN bit is LOW. Read the value of this field when the VICITCR ITEN bit is HIGH. Write sets input to written value when the VICITCR ITEN bit is HIGH.

Table 4-3 VICITIP1 Register bit assignments (continued)

Bits	Name	Type	Description
[7]	nVICIRQIN	Read/ write	Read the value of the **nVICIRQIN** input when the VICITCR ITEN bit is LOW. Read the value of this field when the VICITCR ITEN bit is HIGH. Write sets input to written value when the VICITCR ITEN bit is HIGH.
[6]	nVICFIQIN	Read/ write	Read the value of the **nVICFIQIN** input when the VICITCR ITEN bit is LOW. Read the value of this field when the VICITCR ITEN bit is HIGH. Write sets input to written value when the VICITCR ITEN bit is HIGH.
[5:0]	Reserved	-	Reserved, read as zero, do not modify.

4.3.3 Integration Test Input Register 2, VICITIP2

VICITIP2 is a 32-bit register used to control and read the value of the **VICVECTADDRIN** input. This register must only be used in test mode. The register can be accessed with zero wait states.

Table 4-4 shows the bit assignment of the VICITIP2 Register.

Table 4-4 VICITIP2 Register bit assignments

Bits	Name	Type	Description
[31:0]	VICVECTADDRIN	Read/ write	Read the value of the **VICVECTADDRIN** input when the VICITCR ITEN bit is LOW. Read the value of this field when the VICITCR ITEN bit is HIGH. Write sets input to written value when the VICITCR ITEN bit is HIGH.

4.3.4 Integration Test Output Register 1, VICITOP1

VICITOP1 is a 4-bit register that is used to control and read the values of the following outputs:

- **nVICIRQ**
- **nVICFIQ**
- **VICVECTADDRV**
- **VICIRQACKOUT**.

This register must only be used in test mode. The read value in VICITOP1 always takes two clock cycles to update, because of the use of dual flip-flop synchronization logic that avoids metastability problems. For example, when a new value is written into this register, an immediate read from the register returns the old value. After two cycles, a read from this register returns the updated value. The register can be accessed with zero wait states.

Table 4-5 shows the bit assignment of the VICITOP1 Register.

Table 4-5 VICITOP1 Register bit assignments

Bits	Name	Type	Description
[31:10]	Reserved	-	Reserved, read as zero, do not modify.
[9]	VICIRQACKOUT	Read/ write	Read the value of the **VICIRQACKOUT** output when the VICITCR ITEN bit is LOW. Read the value of this field when the VICITCR ITEN bit is HIGH. Write sets output to written value when the VICITCR ITEN bit is HIGH.
[8]	VICVECTADDRV	Read/ write	Read the value of the **VICVECTADDRV** output when the VICITCR ITEN bit is LOW. Read the value of this field when the VICITCR ITEN bit is HIGH. Write sets output to written value when the VICITCR ITEN bit is HIGH.
[7]	VICIRQ	Read/ write	Read the value of the internal **VICIRQ** signal when the VICITCR ITEN bit is LOW. This is the pre-inverted version of the final output, and is inverted to create the final **nVICIRQ** output. Read the value of this field when the VICITCR ITEN bit is HIGH. Write sets output to written value when the VICITCR ITEN bit is HIGH.
[6]	VICFIQ	Read/ write	Read the value of the internal **VICFIQ** signal when the VICITCR ITEN bit is LOW. This is the pre-inverted version of the final output, and is inverted to create the final **nVICFIQ** output. Read the value of this field when the VICITCR ITEN bit is HIGH. Write sets output to written value when the VICITCR ITEN bit is HIGH.
[5:0]	Reserved	-	Reserved, read as zero, do not modify.

4.3.5 Integration Test Output Register 2, VICITOP2

VICITOP2 is a 32-bit register that controls the **VICVECTADDROUT** output. This register must only be used in test mode. The register can be accessed with zero wait states.

Table 4-6 shows the bit assignment of the VICITOP2 Register.

Table 4-6 VICITOP2 Register bit assignments

Bits	Name	Type	Description
[31:0]	VICVECTADDROUT	Read/ write	Read the value of the **VICVECTADDROUT** output when the VICITCR ITEN bit is LOW. Read the value of this field when the VICITCR ITEN bit is HIGH. Write sets output to written value when the VICITCR ITEN bit is HIGH.

4.3.6 Sampled Interrupt Source Status Register, VICINTSSTATUS

VICINTSSTATUS is a 32-bit register that shows the sampled status of the raw interrupt source inputs, that can be used to detect interrupt sources that are deasserted before they have been serviced. This register must only be used in test mode. The register captures any interrupts that have occurred since the VICINTSSTATUSCLEAR was last written to. The VICINTSSTATUS Register enables you to determine if an interrupt has been asserted and then deasserted without software managing to service the interrupt. The register only gives the sampled status of the 32 interrupts connected to the VIC, and not the daisy-chained **nVICIRQIN**. Read the VICINTSSTATUS Register of the daisy-chained VIC to determine which interrupts have been active. The register can be accessed with zero wait states.

Table 4-7 shows the bit assignment of the VICINTSSTATUS Register.

Table 4-7 VICINTSSTATUS Register bit assignments

Bits	Name	Type	Description
[31:0]	IntSStatus	Read	Shows the sampled status of the raw interrupt source inputs when the VICITCR ISS bit is HIGH. After reset, or when the VICITCR ISS bit is LOW, returns zero.

4.3.7 Sampled Interrupt Source Status Clear Register, VICINTSSTATUSCLEAR

VICINTSSTATUSCLEAR is a 32-bit register that clears bits in the VICINTSSTATUS Register. This register must only be used in test mode.The register can be accessed with zero wait states.

Table 4-8 shows the bit assignment of the VICINTSSTATUSCLEAR Register.

Table 4-8 VICINTSSTATUSCLEAR Register bit assignments

Bits	Name	Type	Description
[31:0]	IntSStatus	Read	Clears corresponding bits in the VICINTSSTATUS Register when the VICITCR ISS bit is HIGH: 0 = no effect 1 = interrupt status cleared in VICINTSSTATUS Register. Writes of any value have no effect when the VICITCR ISS bit is LOW.

Appendix A
Signal Descriptions

This appendix describes the signals that interface with the ARM PrimeCell Vectored Interrupt Controller (PL192). It contains the following sections:

A.1 AMBA AHB signals

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

Table A-1 AMBA AHB signal descriptions

Name	Type	Source/ destination	Description
HCLK	Input	Clock source	AMBA AHB bus clock, used to time all bus transfers. All signal timings are related to the rising edge of **HCLK**.
HRESETn	Input	Reset controller	AHB bus reset, active LOW.
HADDR[11:2]	Input	Master	System address bus.
HTRANS[1:0]	Input	Master	Transfer type, which can be NONSEQUENTIAL, SEQUENTIAL, IDLE, or BUSY. This signal must be connected to **HTRANS[1]** on the AHB interface. **HTRANS[0]** is not used.
HWRITE	Input	Master	Transfer direction. Indicates a write transfer when HIGH, and a read transfer when LOW.
HSIZE[2:0]	Input	Master	Size of the transfer, which must be word (32-bit) for the VIC (**HSIZE[2:0]** = `0b010`).
HPROT[3:0]	Input	Master	Memory access protection type, which can be User mode (0) or privileged mode (1). This signal must be connected to **HPROT[1]** on the AHB interface. **HPROT[3]**, **HPROT[2]** and **HPROT[0]** are not used.
HWDATA[31:0]	Input	Master	Write data bus, used to transfer data from bus master to bus slaves during write operations.
HSELVIC	Input	Decoder	Slave select signal, which is a combinatorial decode of the address bus. It indicates that the current transfer is intended for the selected slave.
HRDATA[31:0]	Output	Slave	Read data bus, used to transfer data from bus slaves to bus master during read operations.

Table A-1 AMBA AHB signal descriptions (continued)

Name	Type	Source/ destination	Description
HREADYIN	Input	External slave	Transfer done signal, generated by an alternate slave. When HIGH, indicates that a transfer is complete. Can be driven LOW to extend a transfer.
HREADYOUT	Output	Slave	Transfer done signal, generated by the VIC. When HIGH, indicates that a transfer is complete. Can be driven LOW to extend a transfer.
HRESP[1:0]	Input	Slave	Transfer response, which provides additional transfer status information. The response can be OKAY, ERROR, RETRY or SPLIT. The PrimeCell VIC responds with either OKAY or ERROR.

A.2 Interrupt controller signals

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

Table A-2 Interrupt controller signals

Name	Type	Source/ destination	Description
VICINTSOURCE [31:0]	Input	Peripheral interrupt request	Interrupt source input
nVICIRQ	Output	Interrupt controller	Interrupt request to processor
nVICFIQ	Output	Interrupt controller	Fast interrupt request to processor

A.3 Daisy chain signals

The daisy chain signals are used when two or more VICs are daisy-chained (see *Daisy-chained interrupt controller connectivity to processor without VIC port* on page 2-17). Table A-3 shows the daisy chain signals.

Table A-3 Daisy chain signals

Name	Type	Source/ destination	Description
VICVECTADDRIN [31:0]	Input	External interrupt controller	Connects to the **VICVECTADDROUT[31:0]** signal of the previous VIC if daisy chaining is used. Connects to logic 0 if the VIC is not daisy-chained.
VICVECTADDROUT [31:0]	Output	Interrupt controller	Connects to the **VICVECTADDRIN[31:0]** signal of the next VIC if daisy chaining is used. Left unconnected if the VIC is not daisy-chained.
nVICIRQIN	Input	External interrupt controller	Connects to the **nVICIRQ** signal of the previous VIC if daisy chaining is used. Connects to logic 1 if the VIC is the last in the daisy chain, or if VIC is not daisy-chained.
nVICFIQIN	Input	External interrupt controller	Connects to the **nVICFIQ** signal of the previous VIC if daisy chaining is used. Connects to logic 1 if the VIC is the last in the daisy chain, or if VIC is not daisy-chained.
VICFIQINREG	Input	System	Tied HIGH if you wish to register the daisy-chained **nVICFIQIN** input into the VIC. Tied LOW if you do not wish to register the **nVICFIQIN** into the VIC.
VICIRQINREG	Input	System	Tied HIGH if you wish to register the daisy-chained **nVICIRQIN** input into the VIC. Tied LOW if you do not wish to register the **nVICIRQIN** into the VIC.
VICIRQACKOUT	Output	External interrupt controller	Interrupt acknowledge from the processor which has passed through the VICs in the daisy chain. Used during the processor/VIC handshaking after an IRQ is generated.

A.4 VIC port signals

The VIC port signals are shown in Table A-4.

Table A-4 VIC port signals

Name	Type	Source/ destination	Description
VICVECTADDRV	Output	ARM11 or ARM1026EJ processor	Indicates that the **VICVECTADDROUT** bus contains a stable value. Used during the processor/VIC handshaking after an IRQ is generated.
nVICSYNCEN	Input	System	Used to indicate that the VIC port must operate in asynchronous mode when tied LOW (only used with a ARM11 or ARM1026EJ processor which must also be set to asynchronous mode). Standard synchronous operation used when tied HIGH. Tie the port HIGH if the VIC port is not used by the system processor.
VICIRQACK	Input	ARM11 or ARM1026EJ processor	Interrupt acknowledge from the processor. Used during the processor/VIC handshaking after an IRQ is generated.

A.5 Scan test control signals

The internal scan test control signals are shown in Table A-5.

Table A-5 Scan test control signals

Name	Type	Source/ destination	Description
SCANENABLE	Input	Scan controller	Scan enable
SCANINHCLK	Input	Scan controller	Scan data input for **HCLK** domain
SCANOUTHCLK	Output	Scan controller	Scan data output for **HCLK** domain

Appendix B
Example Code

This appendix provides examples of the code required when setting up the ARM PrimeCell Vectored Interrupt Controller (PL192). It contains the following section:

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

B.1 About the example code

The following examples of code are provided in this section:

- *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-3
- *FIQ interrupt handler* on page B-4
- *Vectored interrupt initialization* on page B-4
- *Vectored interrupt service routine* on page B-5
- *Daisy-chained vectored interrupt service routine* on page B-5
- *Highest level vectored IRQ interrupt service routine* on page B-6.

B.1.1 Enable interrupts

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

Example B-1 Enable interrupts

```
    LDR    r0, =IntCntlBase                   ;(where IntCntlBase is a
                                              ;predefined constant, e.g.
                                              ;IntCntlBase EQU 0xFFFFF000)
    MOV    r1, #<interrupt to enable>
    STR    r1, [r0, #IntEnableOffset]
```

B.1.2 Disable interrupts

See Example B-2 for 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

See Example B-3 for 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

See Example B-4 for 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

See Example B-5 for 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, e.g.
                                                ;IntCntlBase EQU 0xFFFFF000)
       MOV    r1, #2
       STR    r1, [r0, #SoftIntClearOffset]
```

B.1.6 FIQ interrupt initialization

See Example B-6 on page B-4 for an example of the FIQ interrupt initialization code.

Example B-6 FIQ interrupt initialization

```
       LDR    r0, =IntCntlBase
       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

See Example B-7 for 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 Vectored interrupt initialization

See Example B-8 for an example of the vectored interrupt initialization code.

Example B-8 Vectored interrupt initialization

```
       LDR    r0, =IntCntlBase
       MOV    r1, #<interrupt_to_enable>
       STR    r1, [r0, #IntEnableClearOffset] ;Disable interrupt
              ;Setup and enable vectored interrupt 15
       MOV    r2, #vector_address             ;Set vector address
       STR    r2, [r0, #VectorAddr15Offset]
       MOV    r2, #vector_priority            ;Set vector priority level
       STR    r2, [r0, #VectorPriority15Offset]
       MOV    r2, #interrupt_source           ;Set interrupt source
       LDR    r2, [r0, #IntSelectOffset]      ;Select IRQ interrupt
       BIC    r2, r2, r1
       STR    r2, [r0, #IntSelectOffset]
       STR    r1, [r0, #IntEnableOffset]      ;Enable interrupt
       MSR    CPSR_c, #(ENABLE_IRQ + MODE_SYS_32) ;Enable IRQ interrupts
```

B.1.9 Vectored interrupt service routine

See Example B-9 for an example of the vectored interrupt service routine code.

Example B-9 Vectored interrupt service routine

```
0x18    LDR pc,    [pc, #-0x120]    ;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

; Add code to clear the interrupt source
; 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 will corrupt lr_sys and r0-r3
; ...

; 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 Vectored IRQ has been serviced
       SUBS pc, lr, #4              ; Return from ISR
```

B.1.10 Daisy-chained vectored interrupt service routine

See Example B-10 on page B-6 for an example of the daisy-chained vectored interrupt service routine code.

Example B-10 Daisy-chained vectored interrupt service routine

```
0x18   LDR    pc, [pc, #-0x120]              ;Load vector into PC

daisy_vector_handler
       ;Code to enable interrupt nesting
       STMFD  sp!, {r12,r14}                 ;Stack workspace
       LDR    r12, VectorAddrDaisyVIC        ;Read VectorAddrDaisyVIC to
                                             ;ensure hardware priority logic is
                                             ;enabled correctly
       LDR    r12,[r12]
       MRS    r12, spsr                      ;Save SPSR into r12
       MSR    cpsr_c, #0x1F                  ;Reenable IRQ, go to system mode

       Interrupt_service_routine

       ;Code to exit handler
       MSR    cpsr_c, #0x52                  ;Disable IRQ, move to IRQ mode
       MSR    spsr, r12                      ;Restore SPSR from r12
       LDMFD  sp!, {r12,r14}                 ;Restore registers
       STR    r0, VectorAddrDaisyVIC
       SUBS   pc, r14, #4                    ;Return from IRQ
```

B.1.11 Highest level vectored IRQ interrupt service routine

See Example B-11 for an example of the highest level vectored IRQ interrupt service routine code.

Example B-11 Highest level vectored IRQ interrupt service routine

```
0x18   LDR    pc, [pc, #-0x120]              ;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
```

Appendix C
Troubleshooting

This appendix describes how to troubleshoot the ARM PrimeCell Vectored Interrupt Controller (PL192). It contains the following section:

C.1 Troubleshooting

Table C-1 lists typical problems and suggested remedies.

Table C-1 Troubleshooting

Problem	Suggested remedy
No interrupts are generated.	Check that the connected interrupt sources have been enabled in the VICINTENABLE register.
No interrupts are generated but VICIRQSTATUS is non-zero.	The priority stack cannot be activated by another interrupt and is not reset at the end of ISR. Ensure that a write is performed to the VICADDRESS Register at the end of the ISR. You can also reset the priority stack by software in the initialization code by writing to the VICADDRESS Register 16 times.
Interrupt nesting is happening by itself.	If using a debugger, this can be set to read a whole bank of registers at a time. If it is reading the VICADDRESS register, the VIC performs interrupt nesting because it thinks the system processor has just started the ISR for the currently active interrupt. The debugger must be set to avoid reading the VICADDRESS register.
An interrupt occurs, but none of the enabled sources are asserted.	The interrupt source might have been asserted for only a short amount of time, enough to set off an IRQ/FIQ, but not long enough to be disabled properly during the ISR. The test mode VICINTSSTATUS register can be used to view a sampled version of the interrupt sources. This has a bit for each of the interrupt source inputs, which are set HIGH when an interrupt is asserted, and can only be set LOW using the VICINTSSTATUSCLEAR register, enabling the detection of interrupts that are deasserted early.

Index