# LogiCORE IP AXI INTC (v1.02a)

DS747 July 25, 2012 **Product Specification**

## Introduction

The LogiCORE™ IP AXI Interrupt Controller (AXI INTC) core concentrates multiple interrupt inputs from peripheral devices to a single interrupt output to the system processor. The registers used for storing interrupt vector addresses, checking, enabling and acknowledging interrupts are accessed through a slave interface for the AMBA® AXI (Advanced eXtensible Interface) specification. The number of interrupts and other aspects can be tailored to the target system. This AXI INTC core is designed to interface with the AXI4-Lite protocol.

## Features

- AXI interface is based on the AXI4-Lite specification
- Each individual interrupt can be configured in Fast Interrupt mode, which improves latency. This mode enables the core to drive the interrupt vector address of the interrupt being processed and uses dedicated interrupt acknowledgement
- Configurable number of (up to 32) interrupts inputs
- Single interrupt output
- Easily cascaded to provide additional interrupt inputs
- Priority between interrupt requests is determined by vector position. The least significant bit (LSB, in this case bit 0) has the highest priority
- Interrupt Enable Register for selectively enabling individual interrupt inputs
- Master Enable Register for enabling interrupts request output
- Each input is configurable for edge or level sensitivity
- Automatic edge synchronization when inputs are configured for edge sensitivity
- Output interrupt request pin is configurable for edge or level generation

| LogiCORE IP Facts Table | |
|---|---|
| **Core Specifics** | |
| Supported Device Family(1) | Zynq™-7000(2), Artix™-7, Virtex®-7, Kintex™-7, Virtex-6, Spartan®-6 |
| Supported User Interfaces | AXI4-Lite |
| Resources | See Table 16 through Table 18. |
| **Provided with Core** | |
| Documentation | Product Specification |
| Design Files | ISE: VHDL<br>Vivado: Encrypted RTL |
| Example Design | Not Provided |
| Test Bench | Not Provided |
| Constraints File | Not Provided |
| Simulation Model | Not Provided |
| Supported S/W Driver(3) | Standalone |
| **Tested Design Tools(4)** | |
| Design Entry Tools | ISE™ Design Suite v14.2<br>Vivado™ Design Suite v2012.2(5) |
| Simulation | Mentor Graphics ModelSim |
| Synthesis Tools | Xilinx Synthesis Technology (XST)<br>Vivado High-Level Synthesis (HLS) |
| **Support** | |
| Provided by Xilinx @ www.xilinx.com/support | |

1. For a complete list of supported derivative devices, please see the Embedded Edition Derivative Device Support.
2. Supported in ISE Design Suite implementations only.
3. Standalone driver details can be found in the EDK or SDK directory (*<install_directory>*/doc/usenglish/xilinx_drivers.htm). Linux OS and driver support information is available from //wiki.xilinx.com.
4. For the supported versions of the tools, see the Xilinx Design Tools: Release Notes Guide.
5. Supports only 7 series devices.

# Functional Description

AXI INTC can be used to expand the number of inputs available to the processor and an option to provide a priority encoding scheme. The output of AXI INTC is intended to be connected to a device that can generate interrupt conditions.

## Capturing, Acknowledging and Enabling Interrupt Conditions

Interrupt conditions are captured by AXI INTC and retained until explicitly acknowledged. Interrupts can be enabled/disabled either globally or individually. The processor is signaled with an interrupt condition when all interrupts are globally enabled, and at least one captured interrupt is individually enabled.

## Edge-Sensitive and Level-Sensitive Capture Modes

Two modes are defined for the capture of interrupt inputs as interrupt conditions:

- Edge-sensitive capture
- Level-sensitive capture

Edge-sensitive capture mode records a new interrupt condition when an active edge occurs on the interrupt input, and an interrupt condition does not already exist. (The polarity of the active edge, rising or falling, is a per-input option.) Figure 1 illustrates three main types of edge generation schemes. In this example all schemes use rising edges for detecting an active edge. The peripheral device generating the interrupt input provides an active edge and later produces an inactive edge to detect another active edge for generating a new interrupt request.

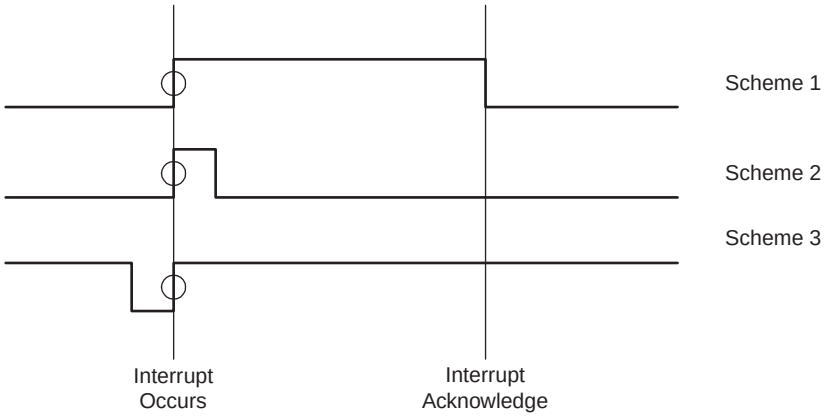



*Figure 1:* **Schemes for Generating Edges**

Level-sensitive capture mode captures an interrupt condition any time the input is at the active level and the interrupt condition does not already exist. (The polarity of the active level, high or low, is as per-input option.)

Observable differences between edge-sensitive and level-sensitive capture are:

- Active level (without subsequent transitions) can produce multiple interrupt conditions
- In edge-sensitive capture, the interrupt input must cycle via an inactive edge and subsequent new active edge to generate an additional interrupt condition

## Interrupt Vector Register (Optional)

If an optional Interrupt Vector Register is configured in the AXI INTC module, then a priority relationship is established between the interrupt inputs (lower number, higher priority). The register returns the number attached

to the individually enabled interrupt with highest priority. This can be used to expedite the speed at which software can select the appropriate interrupt service routine (software vectoring).

### Fast Interrupt Mode Control

Each device connected to the AXI INTC can use either normal or fast interrupt mode, based on the latency requirement. Fast interrupt mode can be chosen for designs requiring better latency. Fast interrupt mode is enabled by setting the corresponding bit in Interrupt Mode register (IMR). The interrupt is acknowledged through PROCESSOR_ACK ports driven by the processor for interrupts configured in fast interrupt mode. The AXI INTC drives the interrupt vector address of the highest priority interrupt along with the IRQ. The IRQ generated is cleared based on the PROCESSOR_ACK signal, and the corresponding IAR bit is updated after acknowledgement is received via PROCESSOR_ACK. The IAR write operation is done internally by the AXI INTC. The processor sends 0b01 on the PROCESSOR_ACK port when the interrupt is acknowledged by the processor (when branching to the interrupt service routine), and sends 0b10 when executing an RTID instruction in the interrupt service routine. For edge-type interrupts, the corresponding bit in the ISR is cleared upon receiving 0b01 on the PROCESSOR_ACK port. For level-type interrupts, the corresponding bit in the ISR will be cleared when 0b10 or 0b11 is seen on the PROCESSOR_ACK port.

Writing to the IAR is not allowed when servicing a fast interrupt. Also, the IVR may not reflect the exact interrupt that is being serviced.

Interrupt Vector Address registers (IVAR) and the Interrupt Mode register (IMR) are generally written during software initialization and must not be written when any interrupt is enabled. Interrupt acknowledgement signals are shown in Table 1.

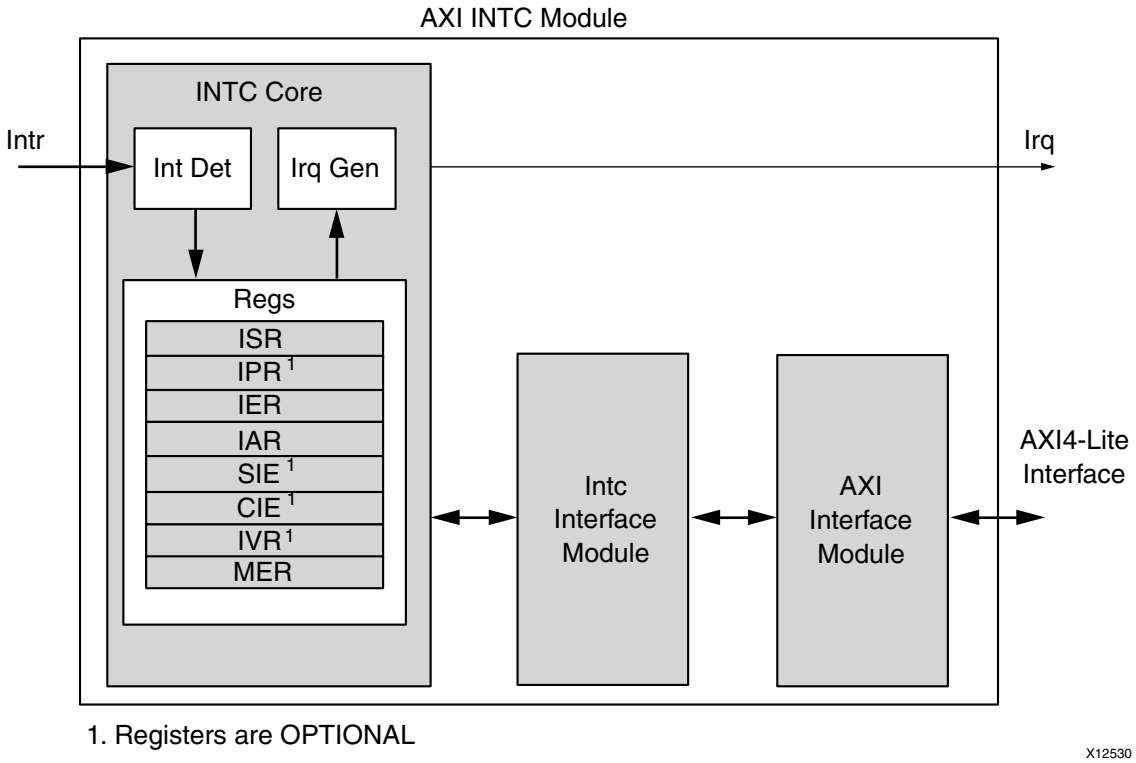
*Table 1:* **Interrupt Acknowledgement Signal Actions**

| PROCESSOR_ACK Pattern | Action Description |
|---|---|
| 0b00 | No interrupt received |
| 0b01 | Set when processor branches to interrupt routine |
| 0b10 | Set when processor returns from interrupt routine by executing RTID |
| 0b11 | Set when processor enables interrupts |

## Module Description

Figure 2 illustrates the main functional units in the AXI INTC module:

- AXI Interface Module: Converts AXI transactions to the internal IPIC interface
- Intc Interface Module: connect the INTC core to the AXI interface in AXI INTC module
- Interrupt Controller (INTC) Core: Consists of the Int Det, Irq Gen, and Regs logical blocks
- Interrupt Detection (Int Det): Detects the valid interrupt from all the interrupt inputs
- Programmer Registers (Regs): Registers used to program and control the operation of interrupt controller
- Interrupt Request Generation (Irq Gen): Generate the interrupt request output
- Fast Interrupt Mode Control: Controls the operation to be either in normal or fast interrupt mode




1. Registers are OPTIONAL

*Figure 2:* **AXI INTC in Normal Mode**

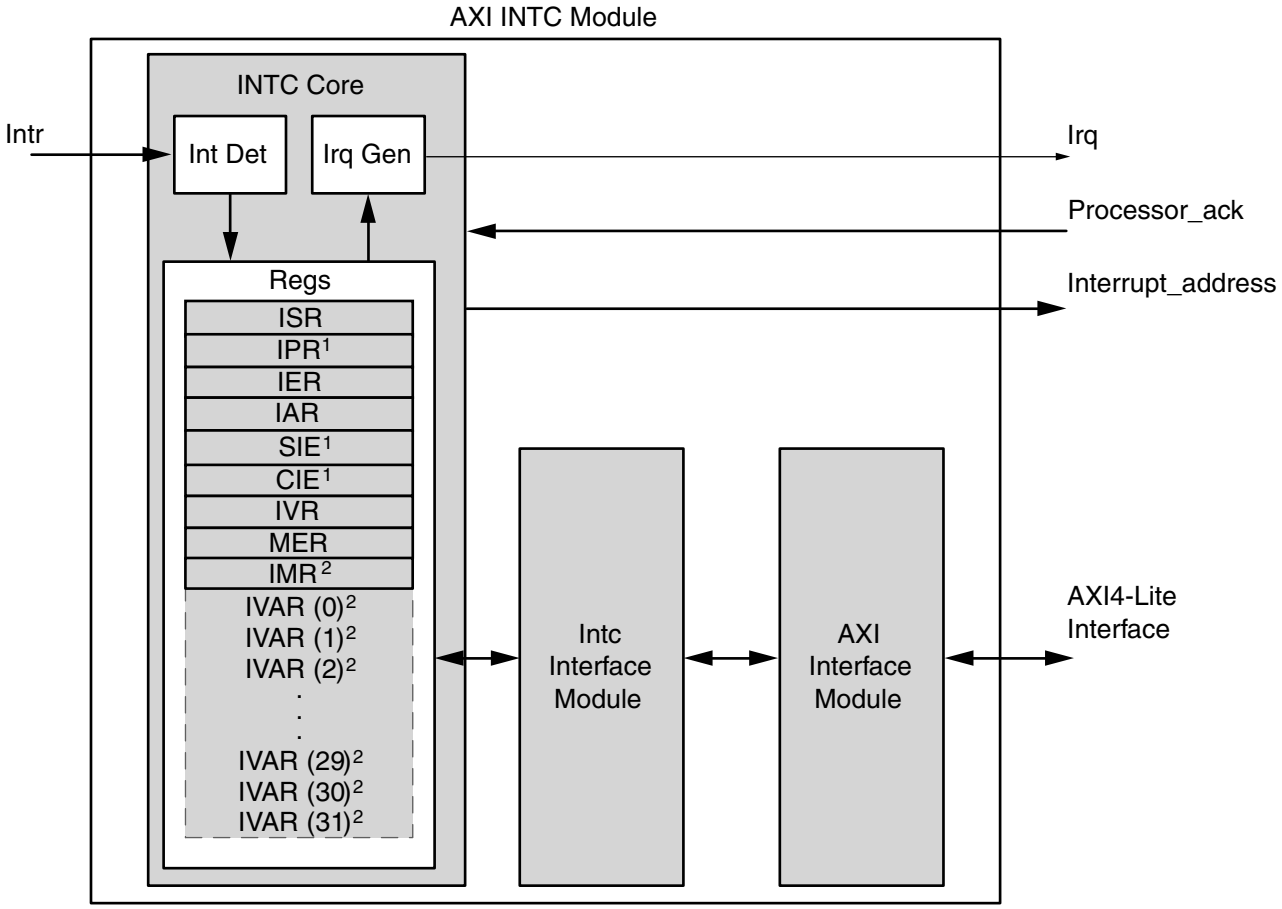



*Figure 3:* **AXI INTC in Fast Interrupt Mode**

## AXI Interface Module

The AXI interface provides a slave interface for transferring data between the INTC and the processor. The AXI INTC registers are mapped into the AXI4-Lite address space. The register addresses are fixed on 4-byte boundaries. All the registers and the data transfers to and from them are always as wide as the data bus. The number of interrupt inputs is configurable up to the width of the data bus, which is set by a configuration parameter. The base address for the registers is also set by a configuration parameter.

## Intc Interface Module

This logical block connects the INTC core to the AXI interface module.

## Interrupt Controller (INTC) Core

Interrupt Controller Core consists of logical blocks as follows:

- Interrupt Detection (Int Det) - For the detection of active input interrupt
- Registers (Regs) - Contains all status and control registers
- Interrupt Request Generation (Irq Gen) - Generates the final output interrupt

### Interrupt Detection (Int Det)

Interrupt detection can be configured for either level or edge detection for each interrupt input. If edge detection is chosen, synchronization registers are included. Interrupt request generation is also configurable as either a pulse output for an edge sensitive request or as a level output that is cleared when the interrupt is acknowledged.

### Programmer Registers (Regs)

The interrupt controller contains programmer accessible registers that allow interrupts to be enabled, queried and cleared under software control. For details refer to:

- Interrupt Status Register (ISR), page 13
- Interrupt Pending Register (IPR), page 14
- Interrupt Enable Register (IER), page 15
- Interrupt Acknowledge Register (IAR), page 16
- Set Interrupt Enables (SIE), page 17
- Clear Interrupt Enables (CIE), page 18
- Interrupt Vector Register (IVR), page 19
- Master Enable Register (MER), page 20
- Interrupt Mode Register (IMR), page 20
- Interrupt Vector Address Register (IVAR), page 21

### Interrupt Request Generation (Irq Gen)

The Irq Gen block generates the final output interrupt from the interrupt controller core. The output interrupt sensitivity is determined by the configuration parameters. Irq Gen checks for IER and MER to enable the interrupt generation. Irq Gen also resets the interrupt after acknowledge. Irq Gen also writes the vector address of the active interrupt in IVR and enables the IPR for pending interrupts.

## I/O Signals

The AXI INTC I/O signals are listed and described in Table 2.

*Table 2:* **I/O Signal Description**

| Port | Signal Name | Interface | I/O | Initial State | Description |
|---|---|---|---|---|---|
| **AXI Global System Signals** | | | | | |
| P1 | S_AXI_ACLK | AXI | Input | - | AXI Clock |
| P2 | S_AXI_ARESETN | AXI | Input | - | AXI Reset, active Low |
| **AXI Write Address Channel Signals** | | | | | |
| P3 | S_AXI_AWADDR[C_S_AXI_ADDR_WIDTH-1:0] | AXI | Input | - | AXI Write address. The write address bus gives the address of the write transaction |
| P4 | S_AXI_AWVALID | AXI | Input | 0x0 | Write address valid. This signal indicates that valid write address and control information are available |
| P5 | S_AXI_AWREADY | AXI | Output | 0x0 | Write address ready. This signal indicates that the slave is ready to accept an address and associated control signals |
| **AXI Write Channel Signals** | | | | | |
| P6 | S_AXI_WDATA[C_S_AXI_DATA_WIDTH - 1: 0] | AXI | Input | - | Write data |
| P7 | S_AXI_WSTB[C_S_AXI_DATA_WIDTH/8-1:0] | AXI | Input | - | Write strobes. This signal indicates which byte lanes to update in memory |
| P8 | S_AXI_WVALID | AXI | Input | - | Write valid. This signal indicates that valid write data and strobes are available |
| P9 | S_AXI_WREADY | AXI | Output | 0x0 | Write ready. This signal indicates that the slave can accept the write data |
| **AXI Write Response Channel Signals** | | | | | |
| P10 | S_AXI_BRESP[1:0] | AXI | Output | 0x0 | Write response. This signal indicates the status of the write transaction<br>“00“- OKAY<br>“10“- SLVERR<br>“11“- DECERR |
| P11 | S_AXI_BVALID | AXI | Output | 0x0 | Write response valid. This signal indicates that a valid write response is available |
| P12 | S_AXI_BREADY | AXI | Input | 0x1 | Response ready. This signal indicates that the master can accept the response information |
| **AXI Read Address Channel Signals** | | | | | |
| P13 | S_AXI_ARADDR[C_S_AXI_ADDR_WIDTH -1:0] | AXI | Input | - | Read address. The read address bus gives the address of a read transaction |

*Table 2:* **I/O Signal Description *(Cont'd)***

| Port | Signal Name | Interface | I/O | Initial State | Description |
|---|---|---|---|---|---|
| P14 | S_AXI_ARVALID | AXI | Input | - | Read address valid. This signal indicates, when HIGH, that the read address and control information is valid and will remain stable until the address acknowledgement signal, S_AXI_ARREADY, is high. |
| P15 | S_AXI_ARREADY | AXI | Output | 0x1 | Read address ready. This signal indicates that the slave is ready to accept an address and associated control signals. |
| **AXI Read Data Channel Signals** | | | | | |
| P16 | S_AXI_RDATA[C_S_AXI_DATA_WIDTH -1:0] | AXI | Output | 0x0 | Read data |
| P17 | S_AXI_RRESP[1:0] | AXI | Output | 0x0 | Read response. This signal indicates the status of the read transfer.<br>"00"- OKAY<br>"10"- SLVERR<br>"11"- DECERR |
| P18 | S_AXI_RVALID | AXI | Output | 0x0 | Read valid. This signal indicates that the required read data is available and the read transfer can complete |
| P19 | S_AXI_RREADY | AXI | Input | 0x1 | Read ready. This signal indicates that the master can accept the read data and response information |
| **INTC Interface Signals** | | | | | |
| P20 | Intr[C_NUM_INTR_INPUTS-1:0](1) | INTC | Input | - | Interrupt inputs |
| P21 | Irq | INTC | Output | 0x1 | Interrupt request output |
| P22 | Interrupt_address[31:0](2) | INTC | Output | 0x0 | Interrupt address output |
| P23 | Processor_ack[1:0] | INTC | Input | - | Interrupt acknowledgement input |
| P24 | Processor_clk | INTC | Input | - | MicroBlaze processor clock |
| P25 | Processor_rst | INTC | Input | - | MicroBlaze processor reset, active |

**Notes:**
1. Intr(0) is always the highest priority interrupt and each successive bit to the left has a corresponding lower interrupt priority.
2. Interrupt_address always drives the vector address of highest priority interrupt.

## Design Parameters

To obtain a AXI INTC that is uniquely tailored, certain features can be parameterized in the AXI INTC design. This allows the user to configure a design that uses only required system resources, and that operates with the best

possible performance. The features that can be parameterized in the AXI INTC core are listed in Table 3.

*Table 3:* **Design Parameters**

| Generic | Feature/Description | Parameter Name | Allowable Values | Default Value | VHDL Type |
|---|---|---|---|---|---|
| **System Parameter** | | | | | |
| G1 | Target FPGA family | C_FAMILY | Zynq-7000, Artix-7, Kintex-7, Virtex-7, Spartan-6 and Virtex-6 | | string |
| **AXI Parameters** | | | | | |
| G2 | AXI address bus width | C_S_AXI_ADDR_WIDTH | 9 | 9 | integer |
| G3 | AXI data bus width | C_S_AXI_DATA_WIDTH | 32 | 32 | integer |
| **INTC Parameters** | | | | | |
| G4 | Number of interrupt inputs | C_NUM_INTR_INPUTS | 1-32 | 2 | integer |
| G5 | Type of interrupt for each input<br>1 = Edge<br>0 = Level | C_KIND_OF_INTR | See(1) | ALL 1s | std_logic_vector |
| G6 | Type of each edge sensitive input<br>1 = Rising<br>0 = Falling<br>Valid if C_KIND_OF_INTR = 1s | C_KIND_OF_EDGE | See(1) | ALL 1s | std_logic_vector |
| G7 | Type of each level sensitive input<br>1= High<br>0= Low<br>Valid if C_KIND_OF_INTR = 0s | C_KIND_OF_LVL | See(1) | ALL 1s | std_logic_vector |
| G8 | Indicates the presence of IPR | C_HAS_IPR | 0 = Not Present<br>1 = Present | 1 | integer |
| G9 | Indicates the presence of SIE | C_HAS_SIE | 0 = Not Present<br>1 = Present | 1 | integer |
| G10 | Indicates the presence of CIE | C_HAS_CIE | 0 = Not Present<br>1 = Present | 1 | integer |
| G11 | Indicates the presence of IVR | C_HAS_IVR | 0 = Not Present<br>1 = Present | 1 | integer |
| G12 | Indicates level or edge active Irq | C_IRQ_IS_LEVEL | 0 = Active Edge<br>1 = Active Level | 1 | integer |
| G13 | indicates the sense of the Irq output | C_IRQ_ACTIVE | 0 = Falling / Low<br>1 = Rising / High | 1 | std_logic |
| G14 | Indicates if processor clock is connected to INTC(7)(8) | C_MB_CLK_NOT_CONNECTED | 0 = Connected<br>1 = Not Connected | 0 | integer |

*Table 3:* **Design Parameters *(Cont'd)***

| Generic | Feature/Description | Parameter Name | Allowable Values | Default Value | VHDL Type |
|---|---|---|---|---|---|
| G15 | Indicates the presence of FAST INTERRUPT logic(8) | C_HAS_FAST | 0 = Not Present<br>1 = Present | 1 | integer |

**Notes:**
1. The interrupt input is a little-endian vector having the width same as the data bus and contain either a '0' or '1' in each position.
2. Synchronizers in the design can be disabled if processor clock and AXI clock are identical. This reduces the core area and latencies introduced by the synchronizers in passing the IRQ to the processor.
3. C_MB_CLK_NOT_CONNECTED should be set to 1 in case the processor clock is not connected to INTC core. When processor clock is not connected, IRQ to the processor is generated on AXI clock. This flexibility is given for backward compatibility of the core. The synchronizers in the design are disabled when the processor clock is not connected.
4. When processor_clk is connected, a DRC error will be generated if the same clock is not connected to both the processor and AXI INTC.

## Dependencies between Parameters and I/O Signals

The dependencies between the AXI INTC core design parameters and I/O signals are described in Table 4. In addition, when certain features are parameterized out of the design, the related logic will no longer be a part of the design. The unused input signals and related output signals are set to a specified value.

*Table 4:* **Parameter-I/O Signal Dependencies**

| Generic or Port | Name | Affects | Depends | Relationship Description |
|---|---|---|---|---|
| **Design Parameters** | | | | |
| G2 | C_S_AXI_ADDR_WIDTH | P3, P13 | - | Defines the width of the ports |
| G3 | C_S_AXI_DATA_WIDTH | P6, P7, P16 | - | Defines the width of the ports |
| G15 | C_HAS_FAST | P22, P23 | - | Ports will be valid if the generic C_HAS_FAST = 1 |
| G15 | C_HAS_FAST | G14 | - | C_MB_CLK_NOT_CONNECTED has to be 0 when C_HAS_FAST is |
| **I/O Signals** | | | | |
| P3 | S_AXI_AWADDR[C_S_AXI_ADDR_WIDTH-1:0] | - | G2 | Port width depends on the generic C_S_AXI_ADDR_WIDTH |
| P6 | S_AXI_WDATA[C_S_AXI_DATA_WIDTH-1:0] | - | G3 | Port width depends on the generic C_S_AXI_DATA_WIDTH |
| P7 | S_AXI_WSTB[C_S_AXI_DATA_WIDTH/8-1:0] | - | G3 | Port width depends on the generic C_S_AXI_DATA_WIDTH |
| P13 | S_AXI_ARADDR[C_S_AXI_ADDR_WIDTH -1:0] | - | G2 | Port width depends on the generic C_S_AXI_ADDR_WIDTH |
| P16 | S_AXI_RDATA[C_S_AXI_DATA_WIDTH -1:0] | - | G3 | Port width depends on the generic C_S_AXI_DATA_WIDTH |
| P22 | Interrupt_address[31:0] | - | G15 | Port will be valid if the generic C_HAS_FAST = 1 |
| P23 | Processor_ack[1:0] | - | G15 | Port will be valid if the generic C_HAS_FAST = 1 |

## Register Descriptions

All AXI INTC registers are accessed through the AXI interface. Each register is 32 bits although some bits may be unused and is accessed on a 4-byte boundary.

Because AXI addresses are byte addresses, AXI INTC register offsets are located at integral multiples of four. Table 5 illustrates the registers. The AXI INTC registers are read as little-endian data.

The eight registers visible to the programmer are shown in Table 5 and described in this section. These points should be considered when reading and writing to registers:

- Writes to a read-only register returns an OKAY response with no register changes.
- Reads to a write-only register returns zero with an OKAY response.
- All registers are defined for 32-bit access only; any partial-word accesses (byte, half word) have undefined results and return a bus error (SLVERR).
- Unless stated otherwise, any register bits that are not mapped to inputs successfully returns zero when read and successfully returns with no register change when written.
- All registers uses C_NUM_INTR_INPUTS to determine its width except MER, which is always 2 bits wide and IVR which is always 32 bits wide.

*Table 5:* **Register Descriptions** (1)

| Address (hex) | Register Name | Access Type | Default Value (hex) | Description |
|---|---|---|---|---|
| 0x0 | ISR | Read / Write | 0x0 | Interrupt Status Register |
| 0x4 | IPR | Read | 0x0 | Interrupt Pending Register |
| 0x8 | IER | Read / Write | 0x0 | Interrupt Enable Register |
| 0xC | IAR | Write | 0x0 | Interrupt Acknowledge Register |
| 0x10 | SIE | Write | 0x0 | Set Interrupt Enable Register |
| 0x14 | CIE | Write | 0x0 | Clear Interrupt Enable Register |
| 0x18 | IVR | Read | 0xFFFF | Interrupt Vector Register |
| 0x1C | MER | Read / Write | 0x0 | Master Enable Register |
| 0x20 | IMR | Read / Write | 0x0 | Interrupt Mode Register |

*Table 5:* **Register Descriptions *(Cont'd)***[1]

| Address (hex) | Register Name | Access Type | Default Value (hex) | Description |
|---|---|---|---|---|
| 0x100 | IVAR[2] | Read/Write | 0x10 | Interrupt Vector Address Register |
| 0x104 | | | 0x10 | |
| 0x108 | | | 0x10 | |
| 0x10C | | | 0x10 | |
| 0x110 | | | 0x10 | |
| 0x114 | | | 0x10 | |
| 0x118 | | | 0x10 | |
| 0x11C | | | 0x10 | |
| 0x120 | | | 0x10 | |
| 0x124 | | | 0x10 | |
| 0x128 | | | 0x10 | |
| 0x12C | | | 0X10 | |
| 0x130 | | | 0X10 | |
| 0x134 | | | 0X10 | |
| 0x138 | | | 0X10 | |
| 0x13C | | | 0X10 | |
| 0x140 | | | 0X10 | |
| 0x144 | | | 0X10 | |
| 0x148 | | | 0X10 | |
| 0x14C | | | 0X10 | |
| 0x150 | | | 0X10 | |
| 0x154 | | | 0X10 | |
| 0x158 | | | 0X10 | |
| 0x15C | | | 0X10 | |
| 0x160 | | | 0X10 | |
| 0x164 | | | 0X10 | |
| 0x168 | | | 0X10 | |
| 0x16C | | | 0X10 | |
| 0x170 | | | 0X10 | |
| 0x174 | | | 0X10 | |
| 0x178 | | | 0X10 | |
| 0x17C | | | 0X10 | |

**Notes:**

1. If the number of interrupt inputs is less than the data bus width the inputs will start with INT0. INT0 maps to the LSB of the ISR, IPR, IER, IAR, SIE, CIE and additional inputs corresponding sequentially to successive bits to the left.
2. IVAR(0) is at (0x100) and IVAR(31) at (0x17C).

## Interrupt Status Register (ISR)

When read, the contents of this register indicate the presence or absence of an active interrupt signal. Each bit in this register that is set to a '1' indicates an active interrupt signal on the corresponding interrupt input. Bits that are '0' are not active. The bits in the ISR are independent of the interrupt enable bits in the IER. See the Interrupt Enable Register (IER), page 15 for the interrupt status bits that are masked by disabled interrupts.

The ISR register is writable by software only until the Hardware Interrupt Enable (HIE) bit in the MER has been set. Given these restrictions, when this register is written to, any data bits that are set to '1' will activate the corresponding interrupt just as if a hardware input became active. Data bits that are zero have no effect.

This allows software to generate interrupt to test purposes until the HIE bit has been set. Once HIE has been set (enabling the hardware interrupt inputs), then writing to this register does nothing. If there are fewer interrupt inputs than the width of the data bus, writing a '1' to a non-existing interrupt input does nothing and reading it will return '0'. The Interrupt Status Register (ISR) is shown in Figure 4 and the bits are described in Table 6.

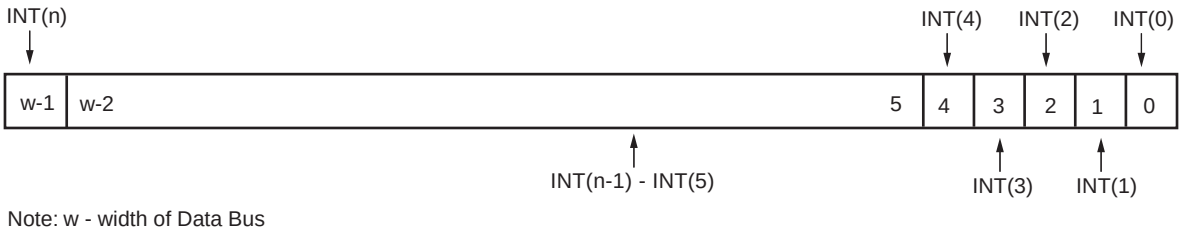



*Figure 4:* **Interrupt Status Register (ISR)**

*Table 6:* **Interrupt Status Register Bit Definitions**

| Bit(s) | Name | Core Access | Reset Value | Description |
|---|---|---|---|---|
| (w[1]-1):0 | INT(n)-INT(0)<br>(n<= w-1) | Read / Write | 0x0 | Interrupt Input (n) - Interrupt Input (0)<br>'0' - Not Active<br>'1' - Active |

**Notes:**
1. w - Width of Data Bus

## Interrupt Pending Register (IPR)

This is an optional read only register in the AXI INTC and can be parameterized out by setting C_HAS_IPR = 0. Reading the contents of this register indicates the presence or absence of an active interrupt signal that is also enabled. This register is used to reduce interrupt processing latency by reducing the number of reads of the INTC by one.

Each bit in this register is the logical AND of the bits in the ISR and the IER. If there are fewer interrupt inputs than the width of the data bus, reading a non-existing interrupt input will return zero. The Interrupt Pending Register (IPR) is shown in Figure 5 and the bits are described in Table 7.

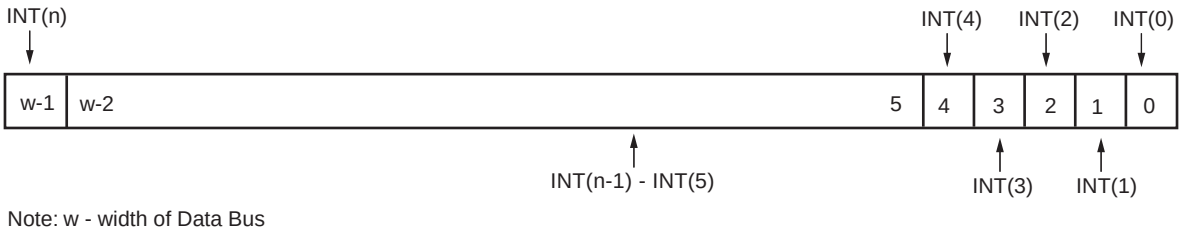



*Figure 5:* **Interrupt Pending Register**

*Table 7:* **Interrupt Pending Register Bit Definitions**

| Bit(s) | Name | Core Access | Reset Value | Description |
|---|---|---|---|---|
| (w[(1)]-1):0 | INT(n)-INT(0)<br>(n<= w-1) | Read / Write | 0x0 | Interrupt Input (n) - Interrupt Input (0)<br>'0' - Not Active<br>'1' - Active |

**Notes:**

1. w - Width of Data Bus

## Interrupt Enable Register (IER)

This is a read / write register. Writing a '1' to a bit in this register enables the corresponding ISR bit to cause assertion of the INTC output. An IER bit set to '0' does not inhibit an interrupt condition for being captured, just reported. Writing a '0' to a bit disables, or masks, the generation of interrupt output for the corresponding interrupt input signal. Note however, that disabling an interrupt input is not the same as clearing it. Disabling an active interrupt prevents that interrupt from reaching the IRQ output. When it is re-enabled, the interrupt immediately generates a request on the IRQ output.

An interrupt must be cleared by writing to the Interrupt Acknowledge Register, as described below. Reading the IER indicates which interrupt inputs are enabled; where a '1' indicates the input is enabled and a '0' indicates the input is disabled.

If there are fewer interrupt inputs than the width of the data bus, writing a '1' to a non-existing interrupt input returns a '0'. The Interrupt Enable Register (IER) is shown in Figure 6 and the bits are described in Table 8.

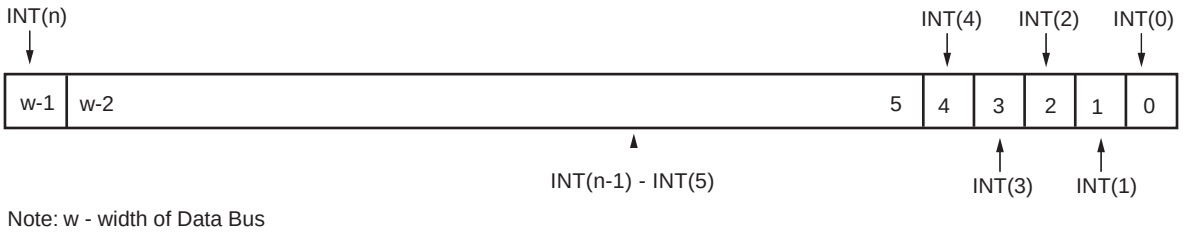



*Figure 6:* **Interrupt Enable Register**

*Table 8:* **Interrupt Enable Register Bit Definitions**

| Bit(s) | Name | Core Access | Reset Value | Description |
|---|---|---|---|---|
| (w[1]-1):0 | INT(n)-INT(0)<br>(n<= w-1) | Read / Write | 0x0 | Interrupt Input (n) - Interrupt Input (0)<br>'0' - Not Active<br>'1' - Active |

**Notes:**

1. w - Width of Data Bus

## Interrupt Acknowledge Register (IAR)

The IAR is a write-only location that clears the interrupt request associated with selected interrupt inputs. Note that writing one to a bit in IAR clears the corresponding bit in ISR, and also clears the same bit itself in IAR.

In fast interrupt mode, bits in IAR are cleared by the sensing the acknowledgement pattern over the PROCESSOR_ACK port. In normal interrupt mode, IAR is cleared by the acknowledgement received over the AXI interface.

Writing a '1' to a bit location in the IAR will clear the interrupt request that was generated by the corresponding interrupt input. An interrupt input that is active and masked by writing a '0' to the corresponding bit in the IER will remain active until cleared by acknowledging it. Unmasking an active interrupt causes an interrupt request output to be generated (if the ME bit in the MER is set).

Writing 0s has no effect as does writing a '1' to a bit that does not correspond to an active input or for which an interrupt input does not exist. The IAR is shown in Figure 7 and the bits are described in Table 9.

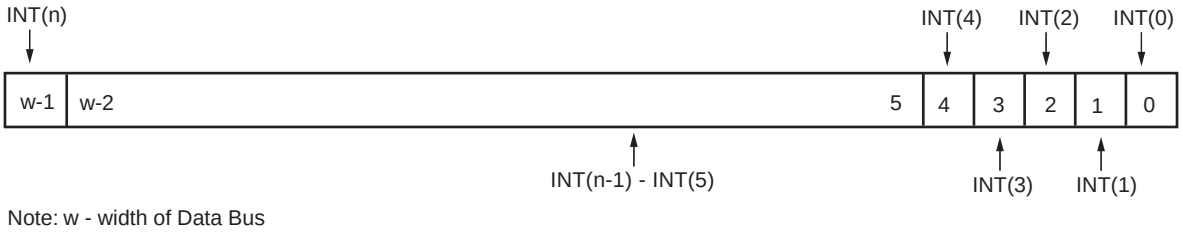



*Figure 7:* **Interrupt Acknowledge Register**

*Table 9:* **Interrupt Acknowledge Register Bit Definitions**

| Bit(s) | Name | Core Access | Reset Value | Description |
|---|---|---|---|---|
| (w[1]-1):0 | INT(n)-INT(0)<br>(n<= w-1) | Read / Write | 0x0 | Interrupt Input (n) - Interrupt Input (0)<br>'0' - Not Active<br>'1' - Active |

**Notes:**

1. w - Width of Data Bus

## Set Interrupt Enables (SIE)

SIE is a location used to set IER bits in a single atomic operation, rather than using a read / modify / write sequence. Writing a '1' to a bit location in SIE will set the corresponding bit in the IER. Writing 0s does nothing, as does wiring a '1' to a bit location that corresponds to a non-existing interrupt input. The SIE is optional in the AXI INTC and can be parametrized out by setting C_HAS_SIE = 0. The Set Interrupt Enable (SIE) register is shown in Figure 8 and the bits are described in Table 10.

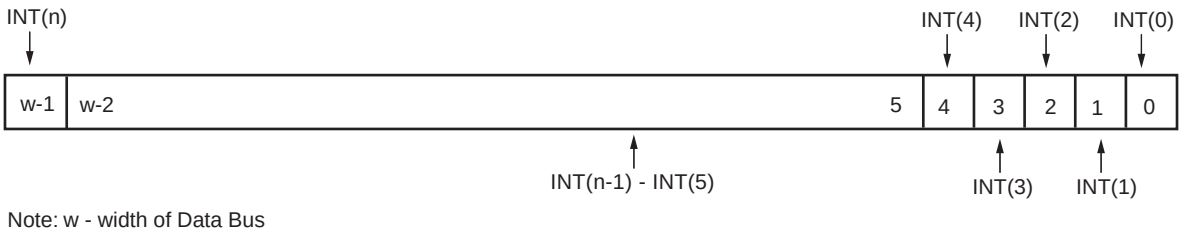



*Figure 8:* **Set Interrupt Enable (SIE) Register**

*Table 10:* **Set Interrupt Enable (SIE) Register Bit Definitions**

| Bit(s) | Name | Core Access | Reset Value | Description |
|---|---|---|---|---|
| (w[(1)]-1):0 | INT(n)-INT(0)<br>(n<= w-1) | Read / Write | 0x0 | Interrupt Input (n) - Interrupt Input (0)<br>'0' - Not Active<br>'1' - Active |

**Notes:**

1. w - Width of Data Bus

## Clear Interrupt Enables (CIE)

CIE is a location used to clear IER bits in a single atomic operation, rather than using a read / modify / write sequence. Writing a '1' to a bit location in CIE will clear the corresponding bit in the IER. Writing 0s does nothing, as does writing a '1' to a bit location that corresponds to a non-existing interrupt input. The CIE is also optional in the AXI INTC and can be parameterized out by setting C_HAS_CIE = 0. The Clear Interrupt Enables (CIE) register is shown in Figure 9 and the bits are described in Table 11.

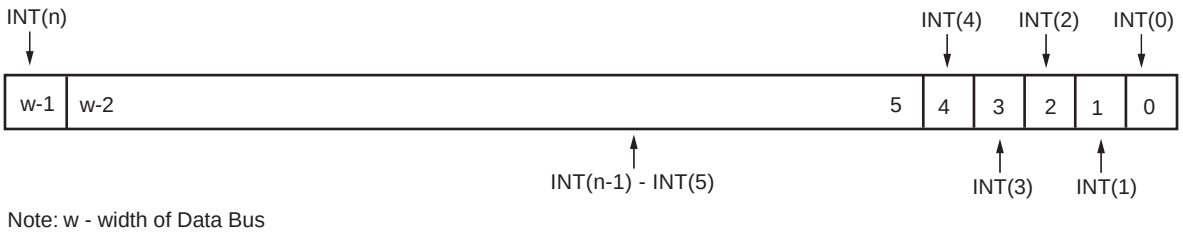



*Figure 9:* **Clear Interrupt Enable (CIE) Register**

*Table 11:* **Clear Interrupt Enable Register Bit Definitions**

| Bit(s) | Name | Core Access | Reset Value | Description |
|---|---|---|---|---|
| (w[(1)]-1):0 | INT(n)-INT(0)<br>(n<= w-1) | Read / Write | 0x0 | Interrupt Input (n) - Interrupt Input (0)<br>'0' - Not Active<br>'1' - Active |

**Notes:**

1. w - Width of Data Bus

## Interrupt Vector Register (IVR)

The IVR is a read-only register and contains the ordinal value of the highest priority, enabled, and active interrupt input. INT0 (always the LSB) is the highest priority interrupt input. Each successive input (to the left) has a corresponding lower interrupt priority.

If no interrupt inputs are active, the IVR contains all 1s. This IVR acts as an index for giving the correct Interrupt Vector Address along with IRQ. The Interrupt Vector Register (IVR) is shown in Figure 10 and described in Table 12.

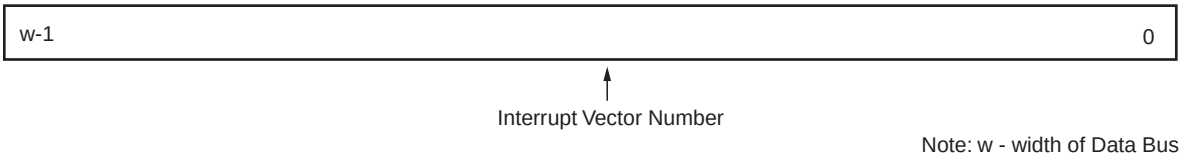



*Figure 10:* **Interrupt Vector Register (IVR)**

*Table 12:* **Interrupt Vector Register (IVR) Bit Definitions**

| Bit(s) | Name | Core Access | Reset Value | Description |
|---|---|---|---|---|
| (w[(1)]-1):0 | Interrupt Vector Number | Read | 0x0 | Ordinal of highest priority, enabled, active interrupt input |

**Notes:**

1. w - Width of Data Bus

## Master Enable Register (MER)

This is a 2-bit, read / write register. The two bits are mapped to the two least significant bits of the location. The least significant bit contains the Master Enable (ME) bit and the next bit contains the Hardware Interrupt Enable (HIE) bit. Writing a '1' to the ME bit enables the IRQ output signal. Writing a '0' to the ME bit disables the Irq output, effectively masking all interrupt inputs.

The HIE bit is a write-once bit. At reset, this bit is reset to '0', allowing the software to write to the ISR to generate interrupts for testing purposes, and disabling any hardware interrupt inputs. Writing a '1' to this bit enables the hardware interrupt inputs and disables software generated inputs. Writing a '1' also disables any further changes to this bit until the device has been reset. Writing 1s or 0s to any other bit location does nothing. When read, this register will reflect the state of the ME and HIE bits. All other bits will read as 0s. The Master Enable Register (MER) is shown in Figure 11 and is described in Table 13.

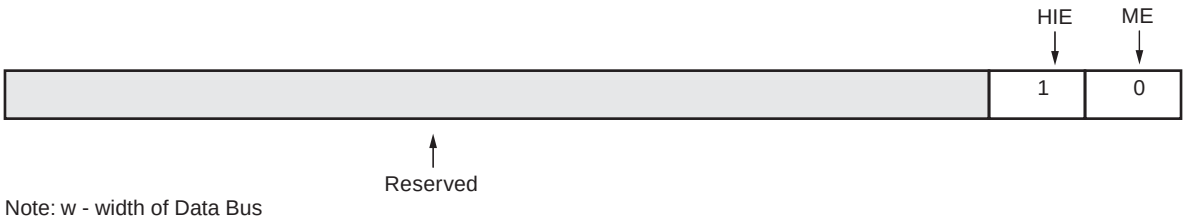



*Figure 11:* **Master Enable Register (MER)**

*Table 13:* **Master Enable Register Bit Definitions**

| Bit(s) | Name | Core Access | Reset Value | Description |
|---|---|---|---|---|
| (w[(1)]-1):2 | Reserved | N/A | 0x0 | Reserved |
| 1 | HIE | Read / Write | '0' | Hardware Interrupt Enable<br>'0' = Read - SW interrupts enabled<br>Write - No effect<br>'1' = Read - HW interrupts enabled<br>Write - Enable HW interrupts |
| 0 | ME | Read / Write | '0' | Master IRQ Enable<br>'0' = Irq disabled - All interrupts disabled<br>'1' = Irq enabled - All interrupts can be enabled |

**Notes:**

1. w - Width of Data Bus

## Interrupt Mode Register (IMR)

This register exists only if parameter C_HAS_FAST is set to '1'. IMR register is used to set the interrupt mode of the connected interrupts. All the interrupts can be set in any mode by setting the corresponding interrupt bit position in IMR. Writing '0' to any bit position will process the corresponding interrupt in normal interrupt mode. Writing '1' to any bit position will process the corresponding interrupt in fast interrupt mode. Unused bit positions in IMR register will return zero.

The Interrupt Mode Register (IMR) is shown in Figure 12 and is described in Figure 14.

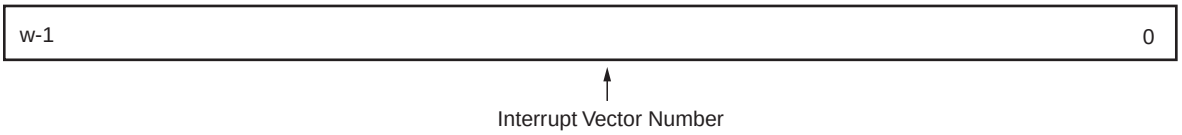



*Figure 12:* **Interrupt Mode Register (IMR)**

*Table 14:* **Interrupt Mode Register (IMR) Bit Definitions**

| Bit(s) | Name | Core | Reset Value | Description |
|---|---|---|---|---|
| (w(1)-1):0 | INT(n)-INT(0)<br>(n<= w-1) | Read / Write | 0x0 | Interrupt Input (n) - Interrupt Input (0)<br>'0' – Normal Interrupt mode<br>'1' – Fast Interrupt mode |

**Notes:**

1. w - Width of Data Bus

## Interrupt Vector Address Register (IVAR)

These are 32-bit wide read/write registers and are C_NUM_INTR_INPUTS in number. Each interrupt connected to the Interrupt controller will have a unique Interrupt vector address that the processor jumps to for servicing that particular interrupt. In normal interrupt mode of operation (IMR(i) = 0), the interrupt vector addresses are taken by the drivers/application. In fast interrupt mode (IMR(i) = 1), the service routine address will be driven by the interrupt controller along with the IRQ. IVAR registers are programmed with the corresponding peripheral interrupt vector address during software initialization.

These registers store the interrupt vector addresses of all the C_NUM_INTR_INPUTS interrupts. Address of the interrupt with highest priority is transmitted out along with IRQ.

If not all the 32 interrupts are used, any read on the unused register address returns zero. Writing 1s or 0s to any bit location does nothing.

The IVAR can be accessed through the AXI interface. IVAR registers are in the AXI clock domain and are used in the processor clock domain to give the INTERRUPT_ADDRESS along with IRQ. These are not synchronized to the processor clock domain. So, it is expected that writing to these registers should be done when ME is '0' (MER(0) = '0').

The Interrupt Vector Address Register (IVAR) is shown in Figure 13 and is described in Table 15.

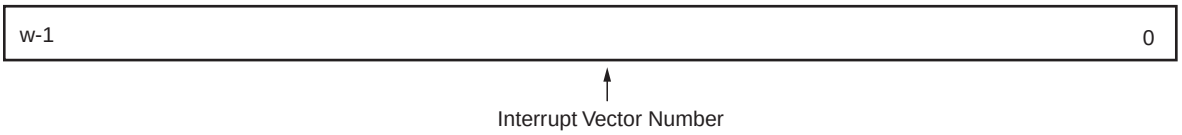



*Figure 13:* **Interrupt Vector Address Register (IVAR)**

*Table 15:* **Interrupt Mode Register (IMR) Bit Definitions**

| Bit(s) | Name | Core | Reset Value | Description |
|---|---|---|---|---|
| (w(1)-1):0 | Interrupt Vector Address | Read / Write | 0x0 | Interrupt vector address of the active interrupt with highest priority |

**Notes:**

1. w - Width of Data Bus

## Software Considerations

This section provides an overview of software initialization and communication with an AXI INTC. The number of interrupt inputs that a AXI INTC has is set by the C_NUM_INTR_INPUTS generic described in Table 3. The first input is always INT0 and is mapped to each LSB of each register (except for IVR and MER).

A valid interrupt input signal is any signal that provides the correct polarity and type of interrupt input. Valid hardware interrupt inputs are rising edges, falling edges, high levels, and low levels. Valid interrupts can be generated if the HIE bit has not been set. The polarity and type of each hardware interrupt input is specified in the AXI INTC generics C_KIND_OF_INTR, C_KIND_OF_EDGE and C_KIND_OF_LVL (see Table 4).

Software interrupts do not have any polarity or type associated with them, so, until HIE has been set, they are always valid. Any valid interrupt input signal that is applied to an enabled interrupt input will generate a corresponding interrupt request within the AXI INTC.

All interrupt requests are combined (an OR function) to form a single interrupt request output. Interrupts are enabled individually by dedicated interrupt enable bits and the external Irq output signal is then gated by a Master Enable (ME) bit.

During power-up or reset, the AXI INTC is put into a state where all interrupt inputs and the interrupt request output are disabled. In order for the AXI INTC to accept interrupts and request service, the following steps are required:

1. Each bit in the IER corresponding to an interrupt input must be set to '1'. This allows AXI INTC to begin accepting interrupt input signals. INT0 has the highest priority, and it corresponds to the least significant bit (LSB) in the IER.
2. The MER must be programmed based on the intended use of the AXI INTC. There are two bits in the MER: the Hardware Interrupt Enable (HIE) and the Master IRQ Enable (ME). The ME bit must be set to enable the interrupt request output.
3. If software testing is to be performed, The HIE bit must remain at its reset value of '0'. Software testing can now proceed by writing a '1' to any bit position in the ISR that corresponds to an existing interrupt input. A corresponding interrupt request is generated if that interrupt is enabled, and interrupt handling proceeds normally.
4. After software testing has been completed, or if software testing is not performed, a '1' is written to the HIE bit, which enables the hardware interrupt inputs and disables any further software generated interrupts.
5. After a '1' has been written to the HIE bit, any further writes to this bit have no effect. This feature prevents stray pointers from accidentally generating unwanted interrupt requests, while still allowing self-test software to perform system tests at power-up or after a reset.

Reading the ISR indicates which inputs are active. If present, the IPR indicates which enabled inputs are active. Reading the IVR provides the ordinal value of the highest priority interrupt that is enabled and active. In fast interrupt mode, that is when C_HAS_FAST = 1, reading any IVAR provides the interrupt vector address of that particular interrupt.

For example, if a valid interrupt signal has occurred on the INT3 interrupt input and nothing is active on INT2, INT1 and INT0, reading the IVR will provide a value of 3. If INT0 becomes active, then reading the IVR provides a value of 0.

If no interrupts are active or it is not present, reading the IVR returns all 1s.

Acknowledging an interrupt is achieved by writing a '1' to the corresponding bit location in the IAR. Disabling an interrupt by masking it (writing a '0' to the IER) does not clear the interrupt. That interrupt will remain active but blocked until it is unmasked or cleared. Interrupts in fast interrupt mode are acknowledged through the PROCESSOR_ACK port, which writes to the corresponding bit in IAR register.

An interrupt acknowledge bit clears the corresponding interrupt request. However, if a valid interrupt signal remains on that input (another edge occurs or an active level still exists on the corresponding interrupt input), a new interrupt request output is generated.

Also, all interrupt requests are combined to form the Irq output so any remaining interrupt requests that have not been acknowledged will cause a new interrupt request output to be generated.

The software can disable the interrupt request output at any time by writing a '0' to the ME bit in the MER. This effectively masks all interrupts for that AXI INTC. Alternatively, interrupt inputs can be selectively masked by writing a '0' to each bit location in the IER that corresponds to an input that is to be masked.

If present, SIE and CIE provide a convenient way to enable or disable (mask) and interrupt input without having to read, mask off, and then write back the IER. Writing a '1' to any bit location(s) in the SIE set(s) the corresponding bit(s) in the IER without affecting any other IER bits.

Writing a '1' to any bit location(s) in the CIE clear(s) the corresponding bit(s) in the IER without affecting any other IER bits.

## Timing Diagrams

The timing diagrams in this section depict the functionality of the core.

- Configured Input Interrupt (Intr) for Edge sensitive (rising), Output Interrupt Request (Irq) to Level sensitive (active High), disabled fast interrupt logic (C_HAS_FAST = 0). Timing diagram is shown in Figure 14.

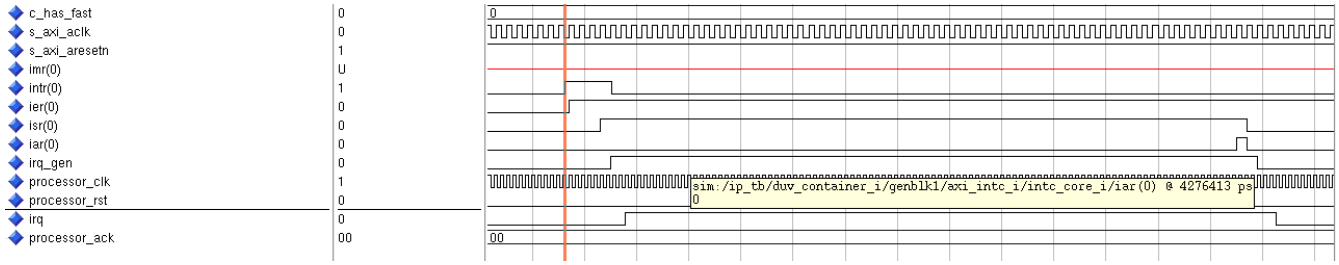



*Figure 14:* **Input Intr - Edge Sensitive (Rising), Output Irq - Level Sensitive (High), C_HAS_FAST = 0**

- Configured Input Interrupt (Intr) for Edge sensitive (rising), Output Interrupt Request (Irq) to Edge sensitive (Rising), disabled fast interrupt logic (C_HAS_FAST = 0). Timing diagram is shown in Figure 15.

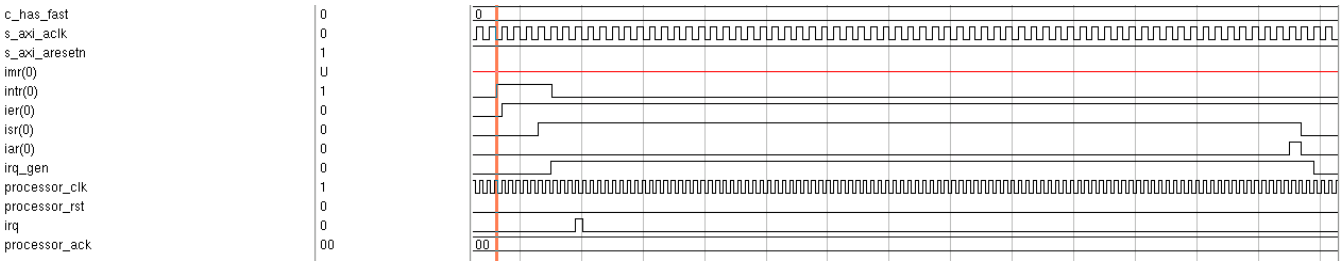



*Figure 15:* **Input Intr - Edge Sensitive (Rising), Output Irq - Edge Sensitive (Rising), C_HAS_FAST = 0**

- Configured Input Interrupt (Intr) for Level sensitive (active High), Output Interrupt Request (Irq) to Level sensitive (active High), disabled fast interrupt logic (C_HAS_FAST = 0). Timing diagram is shown in Figure 16.

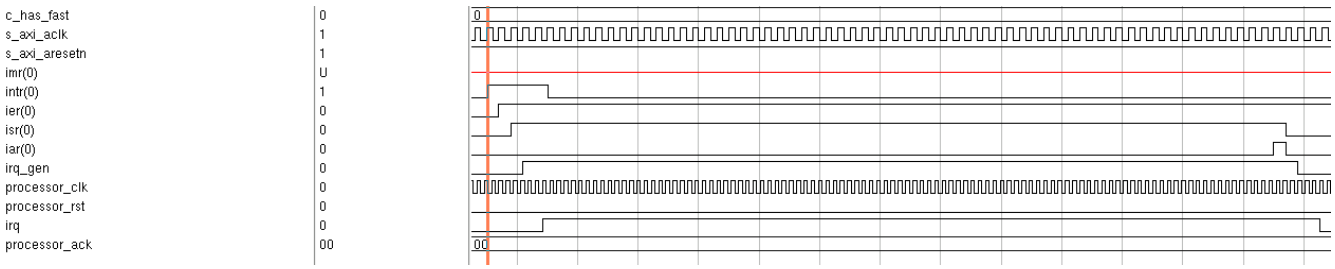



*Figure 16:* **Input Intr - Level Sensitive (High), Output Irq - Level Sensitive (High), C_HAS_FAST = 0**

- Configured Input Interrupt (Intr) for Level sensitive (active High), Output Interrupt Request (Irq) to Edge sensitive (Rising), disabled fast interrupt logic (C_HAS_FAST = 0). Timing diagram is shown in Figure 17.

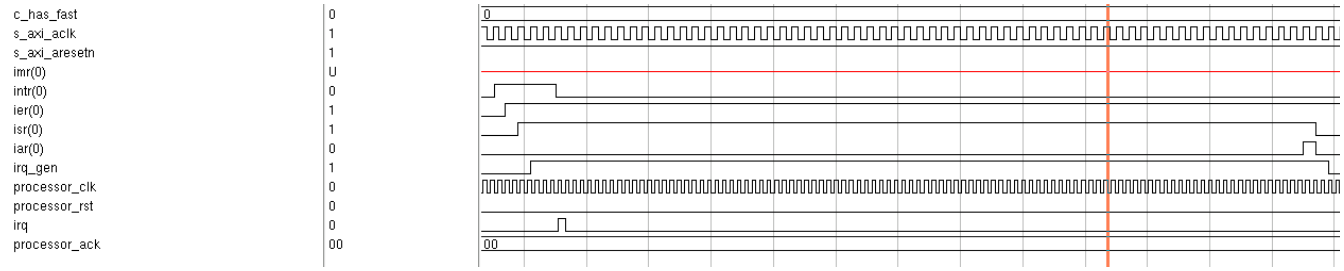



*Figure 17:* **Input Intr - Level Sensitive (High), Output Irq - Edge Sensitive (Rising), C_HAS_FAST = 0**

- Configured Input Interrupt (Intr) for Edge sensitive (rising), include fast logic to High (C_HAS_FAST = 1) and mode set to fast interrupt mode (IMR(i) = 1). Timing diagram is shown in Figure 18.

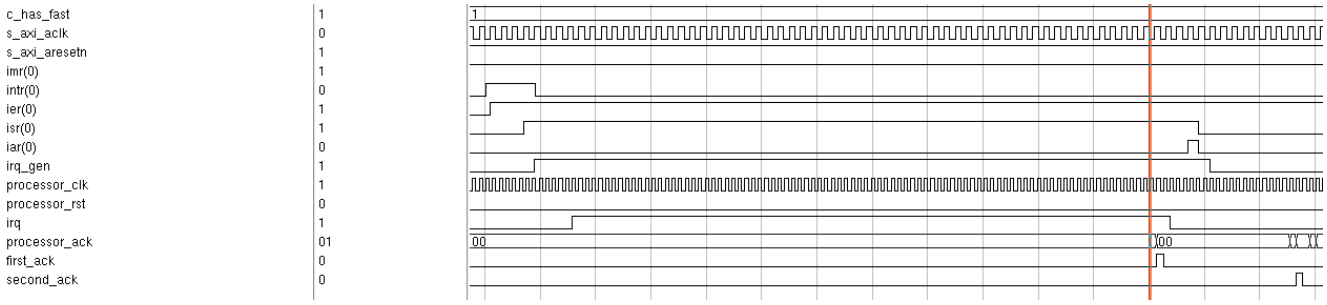



*Figure 18:* **Input Intr - Edge Sensitive (Rising), C_HAS_FAST = 1 and IMR(i) = '1'**

- Configured Input Interrupt (Intr) for Level sensitive (active High), include fast Logic to High (C_HAS_FAST = 1) and mode set to fast interrupt mode (IMR(i) = 1). Timing diagram is shown in Figure 19.

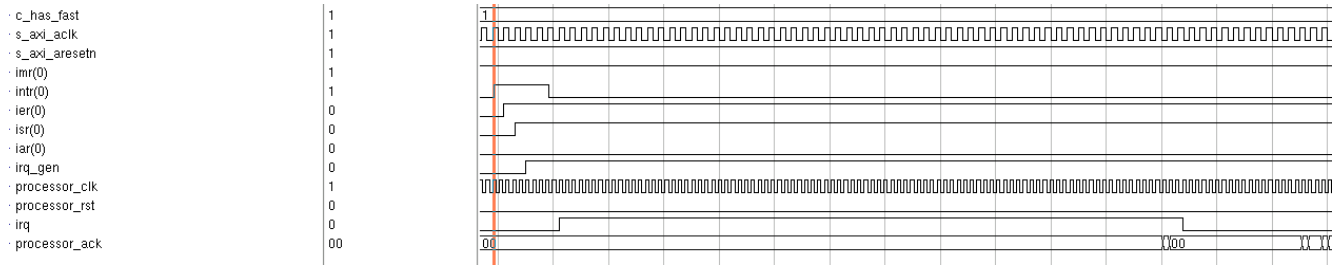



*Figure 19:* **Input Intr - Level Sensitive (active High), C_HAS_FAST = 1 and IMR(i) = '1'**

# Design Implementation

## Target Technology

The intended target technology is Zynq-7000, 7 series, Spartan-6, and Virtex-6 family FPGAs.

## Device Utilization and Performance Benchmarks

### Core Performance

Because the AXI INTC core will be used with other design modules in the FPGA, the utilization and timing numbers reported in this section are estimates only. When the AXI INTC core is combined with other designs in the system, the utilization of FPGA resources and timing of the AXI INTC design will vary from the results reported here.

The AXI INTC resource utilization for various parameter combinations measured with a Virtex-6 device as the target is detailed in Table 16.

*Table 16:* **Performance and Resource Utilization Benchmarks on Virtex-6 (XC6VLX240T-3-FF1156)**

| Parameter Values (other parameters at default value) | | | | | Device Resources | | | Performance |
|---|---|---|---|---|---|---|---|---|
| C_NUM_INTR_INPUTS | C_HAS_IPR | C_HAS_SIE | C_HAS_CIE | C_HAS_IVR | Slices | Slice Flip-Flops | LUTs | $F_{MAX}$ (MHz) |
| 32 | 1 | 1 | 1 | 1 | 352 | 533 | 585 | 209.118 |
| 8 | 1 | 1 | 1 | 1 | 219 | 367 | 402 | 218.531 |
| 8 | 1 | 1 | 1 | 1 | 219 | 367 | 402 | 218.531 |
| 8 | 1 | 1 | 1 | 1 | 318 | 469 | 576 | 217.675 |
| 16 | 1 | 1 | 1 | 1 | 237 | 431 | 434 | 232.072 |
| 16 | 1 | 1 | 1 | 1 | 237 | 431 | 434 | 232.072 |
| 8 | 1 | 1 | 1 | 1 | 318 | 469 | 576 | 217.675 |
| 32 | 1 | 1 | 1 | 1 | 237 | 431 | 434 | 232.072 |
| 16 | 1 | 1 | 1 | 1 | 318 | 469 | 576 | 217.675 |
| 8 | 1 | 1 | 1 | 1 | 237 | 431 | 434 | 232.072 |
| 16 | 1 | 1 | 1 | 1 | 318 | 469 | 576 | 217.675 |
| 8 | 1 | 1 | 1 | 1 | 237 | 431 | 434 | 232.072 |

**Notes:**

1. The above numbers are reported by tools for clock period constraint of 5 ns.

The AXI INTC resource utilization for various parameter combinations measured with a Spartan-6 device as the target is detailed in Table 17.

*Table 17:* **Performance and Resource Utilization Benchmarks on Spartan-6 (XC6SLX75-3-FGG676)**

| Parameter Values (other parameters at default value) | | | | | Device Resources | | | Performance |
|---|---|---|---|---|---|---|---|---|
| C_NUM_INTR_INPUTS | C_HAS_IPR | C_HAS_SIE | C_HAS_CIE | C_HAS_IVR | Slices | Slice Flip-Flops | LUTs | $F_{MAX}$ (MHz) |
| 32 | 1 | 1 | 1 | 1 | 296 | 533 | 617 | 146.327 |
| 8 | 1 | 1 | 1 | 1 | 239 | 369 | 437 | 147.842 |
| 32 | 1 | 1 | 1 | 1 | 256 | 433 | 461 | 144.739 |
| 32 | 1 | 1 | 1 | 1 | 270 | 469 | 593 | 146.284 |
| 16 | 1 | 1 | 1 | 1 | 256 | 433 | 461 | 144.739 |
| 32 | 1 | 1 | 1 | 1 | 270 | 469 | 593 | 146.284 |
| 16 | 1 | 1 | 1 | 1 | 296 | 533 | 617 | 146.327 |
| 16 | 1 | 1 | 1 | 1 | 296 | 533 | 617 | 146.327 |
| 8 | 1 | 1 | 1 | 1 | 270 | 469 | 593 | 146.284 |
| 8 | 1 | 1 | 1 | 1 | 270 | 469 | 593 | 146.284 |
| 8 | 1 | 1 | 1 | 1 | 239 | 369 | 437 | 147.842 |
| 8 | 1 | 1 | 1 | 1 | 256 | 433 | 461 | 144.739 |

**Notes:**

1. The above numbers are reported by tools for clock period of constraint of 6.25 ns.

The AXI INTC resource utilization for various parameter combinations measured with a Kintex-7 device as the target is detailed in Table 18.

***Note:*** Resources numbers for Zynq-7000 devices are expected to be similar to 7 series device numbers.

*Table 18:* **Performance and Resource Utilization on Kintex-7 (XC7K410T-3-FFG676) and Zynq-7000**

| Parameter Values (other parameters at default value) | | | | | Device Resources | | | Performance |
|---|---|---|---|---|---|---|---|---|
| C_NUM_INTR_INPUTS | C_HAS_IPR | C_HAS_SIE | C_HAS_CIE | C_HAS_IVR | Slices | Slice Flip-Flops | LUTs | $F_{MAX}$ (MHz) |
| 32 | 1 | 1 | 1 | 1 | 386 | 588 | 751 | 265.463 |
| 8 | 1 | 1 | 1 | 1 | 308 | 422 | 587 | 224.316 |

*Table 18:* **Performance and Resource Utilization on Kintex-7 (XC7K410T-3-FFG676) and Zynq-7000 *(Cont'd)***

| Parameter Values (other parameters at default value) | | | | | Device Resources | | | Performance |
|---|---|---|---|---|---|---|---|---|
| C_NUM_INTR_INPUTS | C_HAS_IPR | C_HAS_SIE | C_HAS_CIE | C_HAS_IVR | Slices | Slice Flip-Flops | LUTs | $F_{MAX}$ (MHz) |
| 8 | 1 | 1 | 1 | 1 | 386 | 588 | 751 | 265.463 |
| 8 | 1 | 1 | 1 | 1 | 324 | 486 | 611 | 215.564 |
| 32 | 1 | 1 | 1 | 1 | 324 | 486 | 611 | 215.564 |
| 8 | 1 | 1 | 1 | 1 | 386 | 588 | 751 | 265.463 |
| 32 | 1 | 1 | 1 | 1 | 386 | 588 | 751 | 265.463 |
| 32 | 1 | 1 | 1 | 1 | 324 | 486 | 611 | 215.564 |
| 8 | 1 | 1 | 1 | 1 | 324 | 486 | 611 | 215.564 |
| 16 | 1 | 1 | 1 | 1 | 324 | 486 | 611 | 215.564 |
| 16 | 1 | 1 | 1 | 1 | 324 | 486 | 611 | 215.564 |
| 16 | 1 | 1 | 1 | 1 | 324 | 486 | 611 | 215.564 |

**Notes:**

1. The above numbers are reported by tools for clock period of constraint of 5 ns.

## System Performance

To measure the system performance ($F_{MAX}$) of this core, this core was added to a system using a Virtex-6 FPGA and a system using a Spartan-6 FPGA as the device under test (DUT).

Because the AXI Interrupt Controller core will be used with other design modules in the FPGA, the utilization and timing numbers reported in this section are estimates only. When this core is combined with other designs in the system, the design's FPGA resources and timing usage will vary from the results reported here.

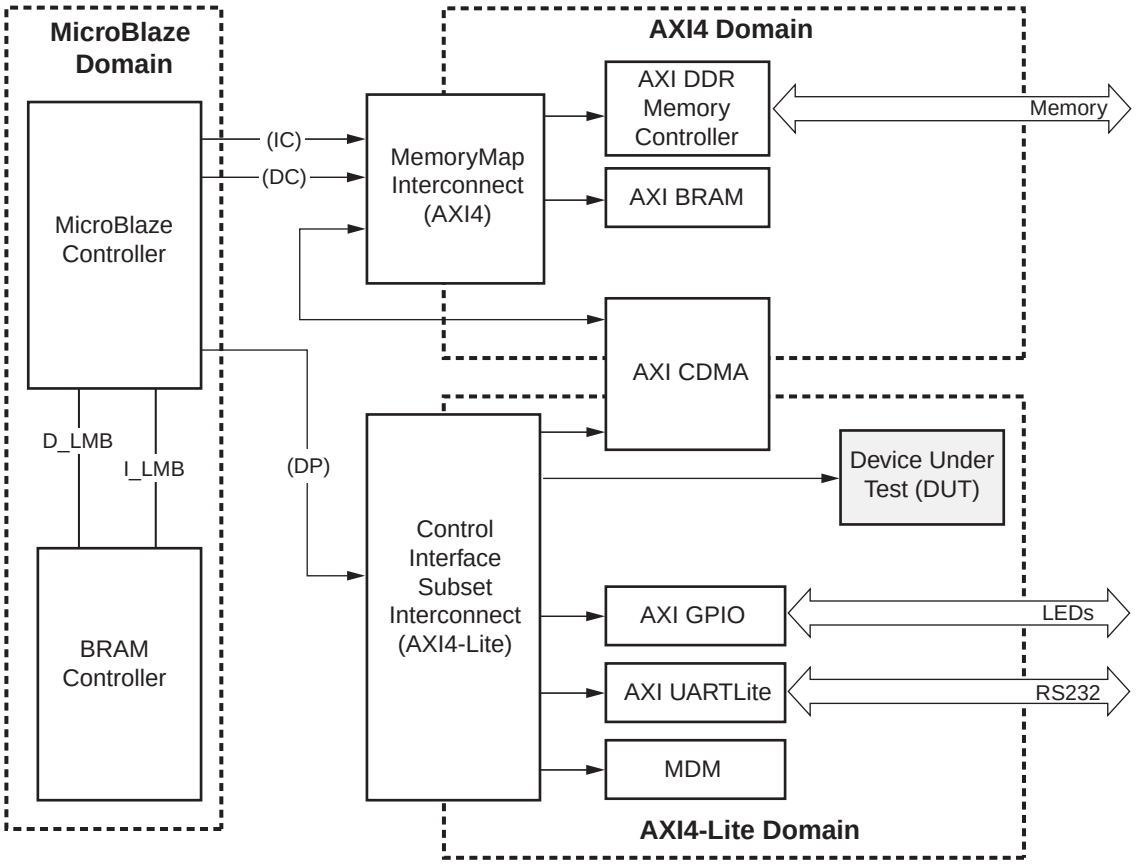



*Figure 20:* **Virtex-6 and Spartan-6 Devices $F_{MAX}$ Margin System**

The target FPGA was filled with logic to drive the LUT and block RAM usage to approximately 70% and the I/O usage to approximately 80%. Using the default tool options and the slowest speed grade for the target FPGA, the resulting target $F_{MAX}$ numbers are shown in Table 19.

*Table 19:* **AXI INTC System Performance**

| Target FPGA | Target $F_{MAX}$ (MHz) |
|---|---|
| Kintex-7 | 200 |
| Virtex-7 | 200 |
| Virtex-6 | 150 |
| Spartan-6 | 133 |

The target $F_{MAX}$ is influenced by the exact system and is provided for guidance. It is not a guaranteed value across all systems.

## Support

Xilinx provides technical support for this LogiCORE IP product when used as described in the product documentation. Xilinx cannot guarantee timing, functionality, or support of product if implemented in devices that are not defined in the documentation, if customized beyond that allowed in the product documentation, or if changes are made to any section of the design labeled *DO NOT MODIFY*.

## Ordering Information

This Xilinx LogiCORE IP module is provided at no additional cost with the Xilinx Vivado Design Suite and ISE Design Suite Embedded Edition software under the terms of the Xilinx End User License.

Information about this and other Xilinx LogiCORE IP modules is available at the Xilinx Intellectual Property page. For information on pricing and availability of other Xilinx LogiCORE modules and software, please contact your local Xilinx sales representative.

## Revision History

The following table shows the revision history for this document:

| Date | Version | Description of Revisions |
|---|---|---|
| 09/21/10 | 1.0 | Initial Xilinx release. |
| 12/14/10 | 2.0 | Updated for core v1.01a and ISE Design Suite v12.4. |
| 06/22/11 | 3.0 | Updated for ISE Design Suite v13.2. Added support for Artix-7, Virtex-7, and Kintex-7 devices. |
| 04/24/12 | 4.0 | Updated for core v1.02a and ISE Design Suite v14.1. Added Fast Interrupt Mode Control and new registers Interrupt Mode Register (IMR) and Interrupt Vector Address Register (IVAR). C_HAS_FAST parameter added. |
| 07/25/12 | 5.0 | Added support for Vivado Design Suite. Added support for Zynq-7000 devices. Removed BASEADDR and HIGHADDR parameters. |

## Notice of Disclaimer