DCR Interrupt Controller (v2.00a)

DS429 April 24, 2009 **Product Specification**

Introduction

A DCR (Device Control Register Bus v29) Interrupt Controller (INTC) core is composed of a bus-centric wrapper containing the INTC core and a DCR interface. The INTC core is a simple, parameterized interrupt controller that, along with the appropriate bus interface, attaches to the DCR Bus.

In this document INTC and DCR INTC are used interchangeably to refer to functionality or interface signals that are common to all variations of the Interrupt Controller. When the discussion switches to a bus centric version, then the interrupt controller will be referred to as DCR INTC.

Features

- DCR v2.0 bus interface (IBM SA-14-2525-00 32-bit DCR Bus Architecture Specifications, v2.9)
- Supports address bus width of 10-bits and data bus width of 32-bits for DCR interface
- Configurable number of (up to 32) interrupt inputs
- Single interrupt output
- Can be 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 disabling individual interrupt inputs
- Master Enable Register for disabling interrupt request output

LogiCORE™ Facts		
Core Specifics		
Supported Device Family	See EDK Supported Device Families.	
Version of Core	dcr_intc	v2.00a
Resources Used		
	Min	Max
Slices	See Table 13 and Table 14.	
FFs		
LUTs		
Block RAMs	N/A	
Provided with Core		
Documentation	Product Specification.	
Design File Formats	VHDL	
Constraints File	N/A	
Verification	N/A	
Instantiation Template	N/A	
Reference Designs	None	
Design Tool Requirements		
Xilinx Implementation Tools	See Tools for requirements.	
Verification		
Simulation		
Synthesis		
Support		
Provided by Xilinx, Inc.		

Features (contd)

- Each input is configurable for edge or level sensitivity; edge sensitivity can be configured for rising or falling; level sensitivity can be active-high or low
- Automatic edge synchronization when inputs are configured for edge sensitivity
- Output interrupt request pin is configurable for edge or level generation — edge generation configurable for rising or falling; level generation configurable for active-high or -low

Functional Description

Interrupt Controller Overview

Most CPUs provide one or more interrupt request input pins that allow external devices to request service. DCR INTC can be used to expand the number of interrupt inputs available to the CPU and, optionally, to provide a priority encoding scheme. The output of DCR INTC is intended to be connected to a CPU interrupt input and each of the interrupt inputs to DCR 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 DCR INTC and retained until explicitly cleared (also referred to as acknowledged). Facilities are also provided to enable or disable interrupts, globally and individually. If all interrupts are globally enabled and if at least one captured interrupt is individually enabled, an interrupt condition is signaled upstream to the CPU.

Edge and Level-Sensitive Capture Modes

Three modes are defined for the capture of interrupt inputs as interrupt conditions.

The first, edge-sensitive capture, captures a new interrupt condition any time 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 the three main types of edge generation schemes, using rising edges for the active edge in this example. In all three schemes, the device generating the interrupt provides an active edge and some time later it produces an inactive edge in preparation for generating a new interrupt request.

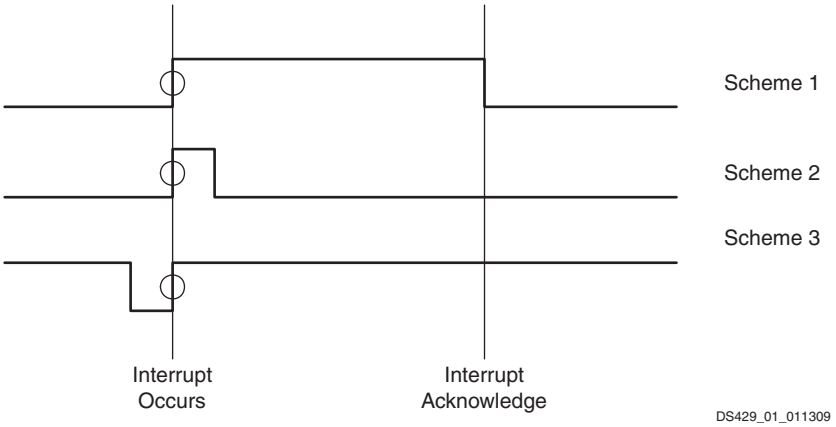

Figure 1: **Schemes for Generating Edges**

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

The third scheme shows the inactive edge occurring immediately before the active edge. All three schemes are possible and should be detected by the interrupt detection circuitry without missing an interrupt or causing spurious interrupts.

An observable difference between edge-sensitive and level-sensitive capture is:

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

Timing Requirement

There is a timing requirement on interrupt input signaling. After an edge for an edge-sensitive interrupt input signal or after the establishment of the active level for a level-sensitive interrupt input signal, the signal must remain stable for a minimum interval to guarantee capture. If the device driving the interrupt input is on the same clock domain as DCR INTC, then this interval is one clock period with standard synchronous setup and hold requirements. If the driving device is on an unrelated clock, the interval must be at least an DCR INTC clock period plus a suitable margin. The margin is chosen by specification to be 20% of the DCR INTC clock period.

Interrupt Vector Register

The interrupt controller described in this document is intended for use in a hard vector interrupt system. It does not directly provide an auto vectoring capability. It does however, provide a vector number that could be used in a software based vectoring scheme.

If an optional Interrupt Vector Register is opted in the DCR INTC, 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).

In non-auto vectoring interrupt designs, it may be necessary for the software interrupt handler to service the highest priority interrupt and then check the status for any additional interrupts that may have arrived before returning from the interrupt handler.

DCR Interrupt Controller Organization

A block diagram of the DCR INTC is shown in Figure 2. The DCR INTC is organized into three main functional units:

- Interrupt detection and request generation
- Registers
- A bus interface

Interrupt Detection

Interrupt detection can be configured for either level or edge detection for each individual interrupt input. If edge detection is chosen, synchronization registers are also included. Interrupt request

generation is also configurable as either a pulse output for an edge sensitive request or as a level output that is reset when the interrupt is acknowledged.

Registers

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

- <RD Red>"Interrupt Status Register (ISR)" on page 9
- <RD Red>"Interrupt Pending Register (IPR)" on page 10
- <RD Red>"Interrupt Enable Register (IER)" on page 10
- <RD Red>"Interrupt Acknowledge Register (IAR)" on page 11
- <RD Red>"Set Interrupt Enables (SIE)" on page 12
- <RD Red>"Clear Interrupt Enables (CIE)" on page 12
- <RD Red>"Interrupt Vector Register (IVR)" on page 13
- <RD Red>"Master Enable Register (MER)" on page 14

Bus Interface

Provides an interface to the DCR Bus. See Figure 2.

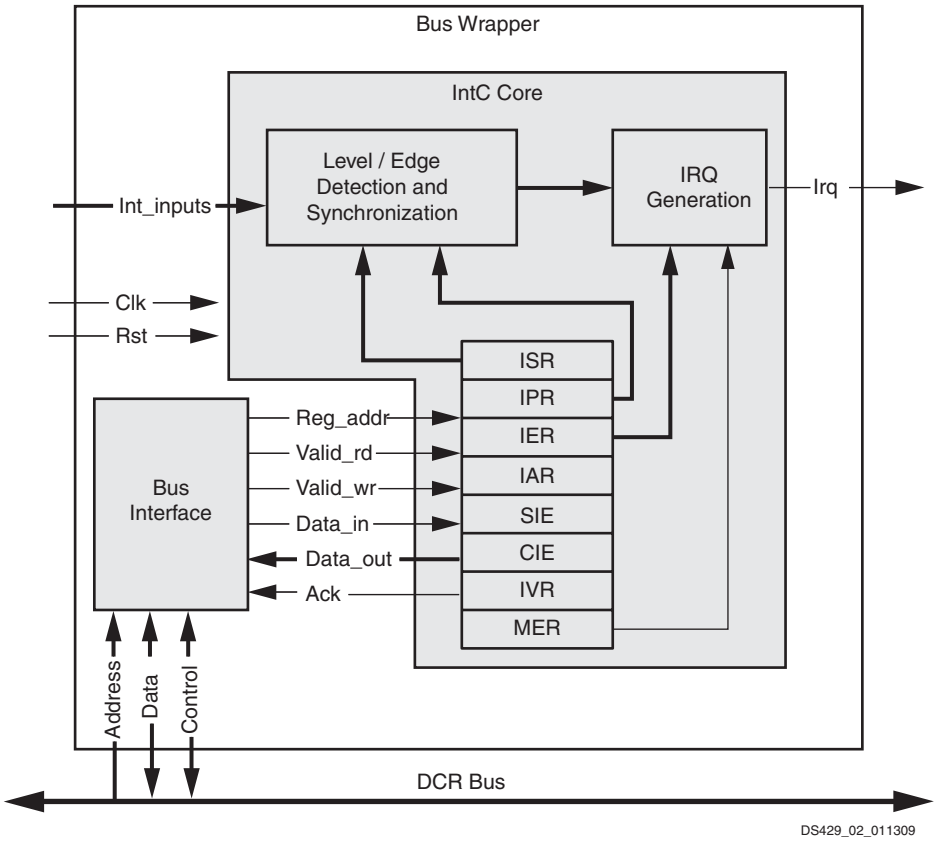

Figure 2: **DCR Interrupt Controller Block Diagram**

DCR INTC I/O Signals

The DCR INTC signals are listed and described in Table 1.

Table 1: **DCR INTC I/O Signal Description**

Port	Signal Name	Interface	I/O	Initial State	Description
System Signals					
P1	DCR_Clk	DCR	I	-	DCR clock
P2	DCR_Rst	DCR	I	-	DCR reset, active high
DCR Interface Signals					
P3	DCR_ABus[0 to C_DCR_AWIDTH – 1]	DCR	I	-	DCR address bus
P4	DCR_Read	DCR	I	-	DCR read input signal
P5	DCR_Write	DCR	I	-	DCR write input signal
P6	DCR_DBus[0 to C_DCR_DWIDTH – 1]	DCR	I	-	DCR data bus (write data input)
P7	Intc_dcrDBus[0 to C_DCR_DWIDTH – 1]	DCR	O	All 0's	INTC data bus (read data output)
P8	Intc_dcrAck	DCR	O	0	INTC data transfer acknowledge
INTC Interface Signals					
P9	Intr[(C_NUM_INTR_INPUTS-1) : 0]<RD Red><SP Superscript>(1)	INTC	I	-	Interrupt inputs
P10	Irq	INTC	0	0	Interrupt request output

Notes:
1. Intr(0) is always the highest priority interrupt and each successive bit to the left has a corresponding lower interrupt priority

DCR INTC Design Parameters

To allow the user to create a DCR INTC that is uniquely tailored for the user's system, certain features are parameterizable in the DCR INTC design. This allows the user to have a design that utilizes only the resources required by the system and runs at the best possible performance. The features that are parameterizable in the DCR INTC core are as shown in <RD Red>Table 2.

Table 2: **DCR INTC Design Parameters**

Generic	Feature / Description	Parameter Name	Allowable Values	Default Value	VHDL Type
System Parameters					
G1	DCR INTC Base Address	C_BASEADDR<RD Red><SP Superscript>(1)	Valid Address<RD Red><SP Superscript>(2)	None<RD Red><SP Superscript>(3)	std_logic_vector

Table 2: **DCR INTC Design Parameters *(Cont'd)***

Generic	Feature / Description	Parameter Name	Allowable Values	Default Value	VHDL Type
G2	DCR INTC High Address	C_HIGHADDR<RD Red><SP Superscript>(1)	Valid Address<RD Red><SP Superscript>(4)	None<RD Red><SP Superscript>(3)	std_logic_vector
		DCR Parameters			
G3	DCR address width	C_DCR_AWIDTH	32	32	integer
G4	DCR data width	C_DCR_DWIDTH	32, 64, 128	32	integer
		INTC Parameters			
G5	Number of interrupt inputs	C_NUM_INTR_INPUTS	1 - C_SPLB_NATIVE_DWIDTH	2	integer
G6	Type of interrupt for each input X = None 1 = Edge 0 = Level	C_KIND_OF_INTR	See<RD Red><SP Superscript>(5)	ALL 1's	std_logic_vector
G7	Type of each edge sensitive input X = N/A 1 = Rising 0 = Falling Valid if C_KIND_OF_INTR = 1's	C_KIND_OF_EDGE	See<RD Red><SP Superscript>(5)	ALL 1's	std_logic_vector
G8	Type of each level sensitive input X = N/A 1 = High 0 = Low Valid if C_KIND_OF_INTR = 0's	C_KIND_OF_LVL	See<RD Red><SP Superscript>(5)	ALL 1's	std_logic_vector
G9	Indicates the presence of IPR	C_HAS_IPR	0 = Not Present 1 = Present	1	integer
G10	Indicates the presence of SIE	C_HAS_SIE	0 = Not Present 1 = Present	1	integer
G11	Indicates the presence of CIE	C_HAS_CIE	0 = Not Present 1 = Present	1	integer
G12	Indicates the presence of IVR	C_HAS_IVR	0 = Not Present 1 = Present	1	integer
G13	Indicates whether the Irq output uses level (or edge) generation.	C_IRQ_IS_LEVEL	0 = edge generation 1 = level generation	1	Integer

Table 2: **DCR INTC Design Parameters *(Cont'd)***

Generic	Feature / Description	Parameter Name	Allowable Values	Default Value	VHDL Type
G14	Indicates the sense of the Irq output	C_IRQ_ACTIVE	'0' = falling / low '1' = rising / high	1	std_logic

Notes:

1. C_BASEADDR and C_HIGHADDR give the first and last addresses, respectively, of the address range assigned to the DCR INTC. The size in bytes, C_HIGHADDR - C_BASEADDR + 1, and the alignment of the address range must be a power of two (to allow simple decoding) and greater than or equal to 32 (to accommodate the eight 32-bit DCR INTC registers)
2. C_BASEADDR must be aligned to the address range size. In other words, it must be a multiple of the address range size
3. No default value is specified to ensure that the actual value is set i.e. if the value is not set, a compiler error will be generated
4. If the size and alignment rules are met then C_HIGHADDR = C_BASEADDR+2**p-1 for some p >= 5; the low-order p bits of C_BASEADDR are '0'; the low-order p bits of C_HIGHADDR are '1'; and the remaining corresponding bit positions are equal. As the value of p increases the resources and time needed to decode the address range decrease and the amount of the overall system address map consumed by DCR INTC increases. For example:
   C_BASEADDR = 0x70800000
   C_HIGHADDR = 0x7080001F
   provides the maximum address decode resolution, requiring the upper 27 address bits to be decoded.
   Conversely,
   C_BASEADDR = 0x70000000
   C_HIGHADDR = 0x7FFFFFFF
   will reduce the address decoding logic for an DCR INTC (only the 4 upper address bits), resulting in a smaller and faster address decode, which consumes one sixteenth of the system address map
5. A little-endian vector of width same as the data bus containing a 0 or 1 in each position corresponding to an interrupt input

DCR INTC Parameter - Port Dependencies

The dependencies between the DCR INTC core design parameters and I/O signals are described in Table 3. 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 3: **DCR INTC Parameter-Port Dependencies**

Generic or port	Name	Affects	Depends	Relationship Description
Design Parameters				
G3	C_DCR_AWIDTH	P3	-	Affects number of bits in address bus
G4	C_DCR_DWIDTH	P6, P7	-	Affects number of bits in data buses
G5	C_NUM_INTR_INPUTS	P9	G4	Affects number of bits in Intr. C_NUM_INTR_INPUTS <= C_DCR_DWIDTH
I/O Signals				
P9	Intr[(C_NUM_INTR_INPUTS-1) : 0]	-	G5	Width varies with the parameter C_NUM_INTR_INPUTS

Programming Model

Register Data Types and Organization

All DCR INTC registers are accessed through the DCR bus interface. The base address for these registers is provided by the configuration parameter, C_BASEADDR. Each register is 32 bits although some bits may be unused and is accessed on a 4-byte boundary offset from the base address.

Because DCR addresses are byte addresses, DCR INTC register offsets are located at integral multiples of four from the base address. Table 4 illustrates the registers and their offsets from the base address. The DCR INTC registers are read as big-endian data.

Though all the INTC registers are in little-endian form, bus will always read these registers as big-endian data. The bit and byte labeling for big-endian data types is shown below in Figure 3.

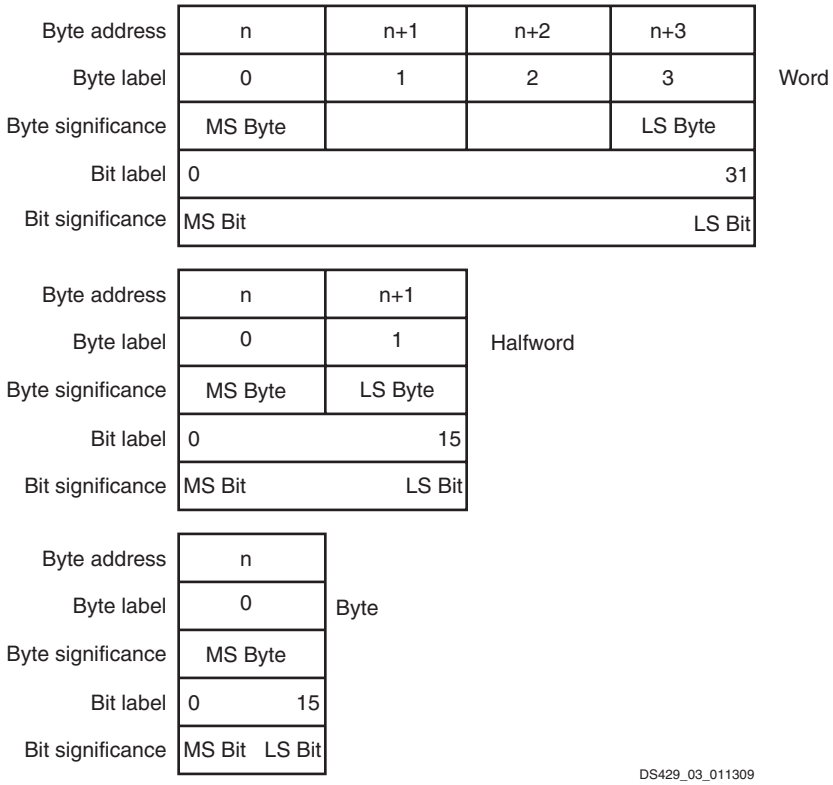

Figure 3: **DCR Big-Endian Data Types**

Registers

The eight registers visible to the programmer are shown in Table 4 and described in this section.

These points should be considered when reading and writing to registers:

- Write of a read-only register has no affect
- Read of a write-only register returns zero
- All registers are defined for 32-bit access only; any partial-word accesses (byte, halfword) have undefined results and return a bus error

- Unless stated otherwise, any register bits that are not mapped to inputs return zero when read and do nothing when written
- All registers use C_NUM_INTR_INPUTS except MER which is always 2-bit wide and IVR which is always 32-bit wide

Table 4: **DCR INTC Registers and Base Address Offsets**

Register Name	Base Address + Offset (Hex)	Access Type	Abbreviation	Reset Value
Interrupt Status Register	C_BASEADDR + 0x0	Read / Write	ISR	All Zeros
Interrupt Pending Register	C_BASEADDR + 0x4	Read only	IPR	All Zeros
Interrupt Enable Register	C_BASEADDR + 0x8	Read / Write	IER	All Zeros
Interrupt Acknowledge Register	C_BASEADDR + 0xC	Write only	IAR	All Zeros
Set Interrupt Enable Bits	C_BASEADDR + 0x10	Write only	SIE	All Zeros
Clear Interrupt Enable Bits	C_BASEADDR + 0x14	Write only	CIE	All Zeros
Interrupt Vector Register	C_BASEADDR + 0x18	Read only	IVR	All Ones
Master Enable Register	C_BASEADDR + 0x1C	Read / Write	MER	All Zeros

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 correspond sequentially to successive bits to the left

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 <RD Red>"Interrupt Enable Register (IER)" on page 10 for interrupt status bits that is masked by disabled interrupts.

The ISR register is writable by software until the Hardware Interrupt Enable (HIE) bit in the MER has been set. Once that bit has been set, software can no longer write to the ISR. 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 interrupts for 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 zero. The Interrupt Status Register (ISR) is shown in Figure 4 and the bits are described in Table 5.

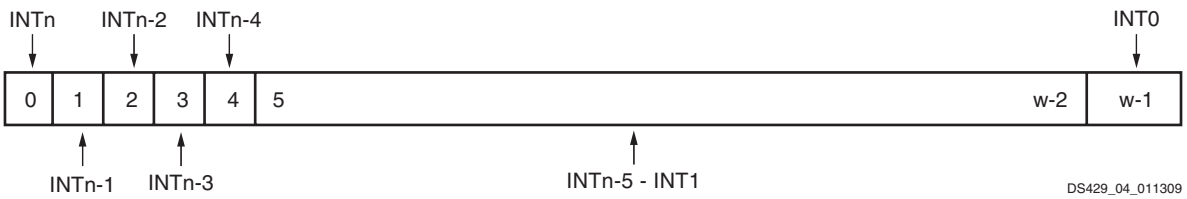

Figure 4: **Interrupt Status Register (ISR)**

Table 5: **Interrupt Status Register**

Bits	Name	Description	Reset Value
0 : (w<RD Red><SP Superscript>(1) –1)	INT(n) – INT(0) ($n \leq w - 1$)	**Interrupt Input (n) – Interrupt Input (0)** '0' = Not active '1' = Active	All Zeros

Notes:
1. w - Width of Data Bus

Interrupt Pending Register (IPR)

This is an optional read only register in the DCR 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.

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 6.

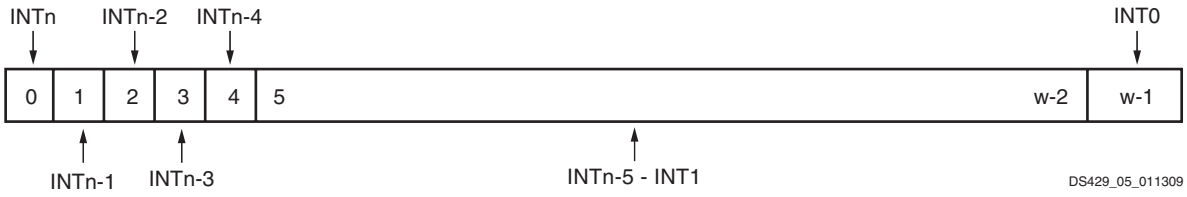

Figure 5: **Interrupt Pending Register (IPR)**

Table 6: **Interrupt Pending Register**

Bits	Name	Description	Reset Value
0 : (w<RD Red><SP Superscript >(1) – 1)	INT(n) – INT(0) ($n \leq w - 1$)	**Interrupt Input (n) – Interrupt Input (0)** '0' = Not active '1' = Active	All Zeros

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 interrupt input signal. Writing a 0 to a bit disables (or masks) the corresponding interrupt input signal. Note however, that disabling an interrupt input is not the same as clearing it. Disabling an active interrupt blocks that interrupt from reaching the IRQ output, but as soon as it is re-enabled the interrupt will immediately generate a request on the IRQ output. An interrupt must be cleared by writing to the Interrupt Acknowledge Register as described below. Reading this register indicates which interrupt inputs are enabled, where a one indicates the input is enabled and a zero 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 does nothing and reading it will return zero. The Interrupt Enble Register (IER) is shown in the Figure 6 and the bits are described in Table 7.

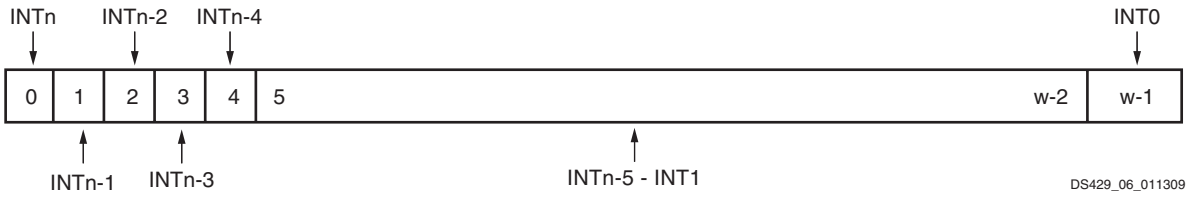

Figure 6: **Interrupt Enable Register (IER)**

Table 7: **Interrupt Enable Register**

Bits	Name	Description	Reset Value
0 : (w<RD Red><SP Superscript >(1) – 1)	INT(n) – INT(0) ($n \leq w - 1$)	**Interrupt Input (n) – Interrupt Input (0)** '1' = Interrupt enabled '0' = Interrupt disabled	All Zeros

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 a one to a bit location in the IAR will clear the corresponding bit in ISR and also clears the same bit itself in IAR.

Writing a one 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 will cause an interrupt request output to be generated (if the ME bit in the MER is set).

Writing zeros does nothing as does writing a one to a bit that does not correspond to an active input or for which an interrupt input does not exist. The Interrupt Acknowledge Register (IAR) is shown in Figure 7 and the bits are described in Table 8.

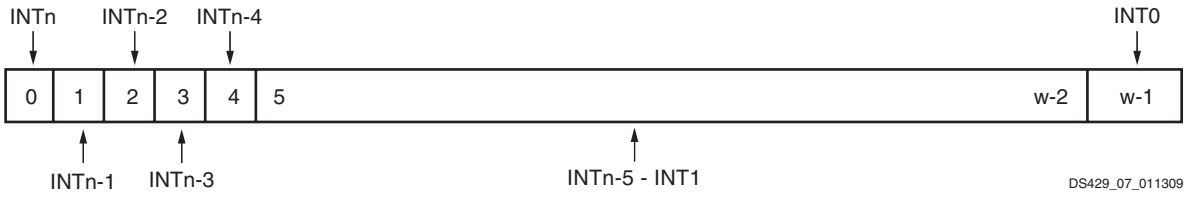

Figure 7: **Interrupt Acknowledge Register (IAR)**

Table 8: **Interrupt Acknowledge Register**

Bits	Name	Description	Reset Value
0 : (w<RD Red><SP Superscript >(1) – 1)	INT(n) – INT(0) ($n \leq w - 1$)	**Interrupt Input (n) – Interrupt Input (0)** '1' = Clear Interrupt '0' = No action	All Zeros

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 one to a bit location in SIE will set the corresponding bit in the IER. Writing zeros does nothing, as does writing a one to a bit location that corresponds to a non-existing interrupt input.

The SIE is optional in the DCR INTC and can be parameterized out of the implementation by setting C_HAS_SIE = 0.

The Set Interrupt Enables (SIE) register is shown in Figure 8 and the bits are described in Table 9.

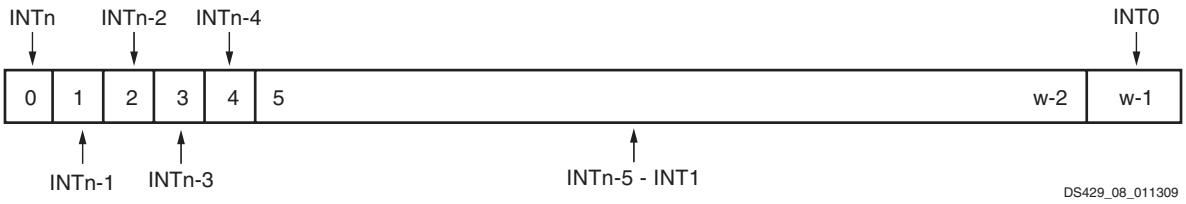

Figure 8: **Set Interrupt Enables (SIE)**

Table 9: **Set Interrupt Enables**

Bits	Name	Description	Reset Value
0 : (w<RD Red><SP Superscript> (1) – 1)	INT(n) – INT(0) ($n \leq w - 1$)	**Interrupt Input (n) – Interrupt Input (0)** '1' = Set IER bit '0' = No action	All Zeros

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 one to a bit location in CIE will clear the corresponding bit in the IER. Writing zeros does nothing, as does writing a one to a bit location that corresponds to a non-existing interrupt input.

The CIE is also optional in the DCR Into and can be parameterized out of the implementation by setting C_HAS_CIE = 0. The Clear Interrupt Enables (CIE) register is shown in Figure 9 and the bits are described in Table 10.

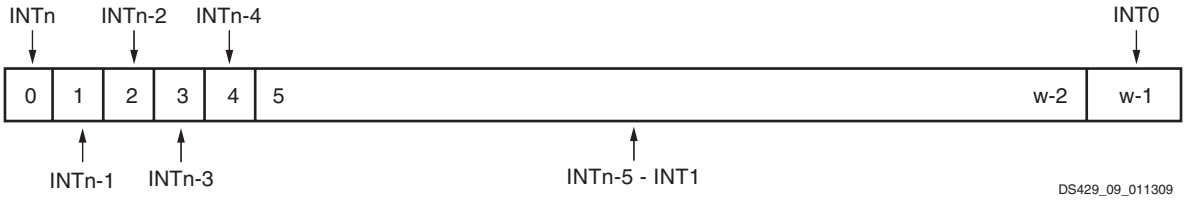

Figure 9: **Clear Interrupt Enables (CIE)**

Table 10: **Clear Interrupt Enables**

Bits	Name	Description	Reset Value
0 : (w<RD Red><SP Superscript >(1) – 1)	INT(n) – INT(0) ($n \leq w - 1$)	**Interrupt Input (n) – Interrupt Input (0)** '1' = Clear IER bit '0' = No action	All Zeros

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, active interrupt input.

INT0 (always the LSB) is the highest priority interrupt input and each successive input to the left has a correspondingly lower interrupt priority.

If no interrupt inputs are active then the IVR will contain all ones. The IVR is optional in the DCR INTC and can be parameterized out of the implementation by setting C_HAS_IVR = 0. The Interrupt Vector Register (IVR) is shown inFigure 10 and described in Table 11.

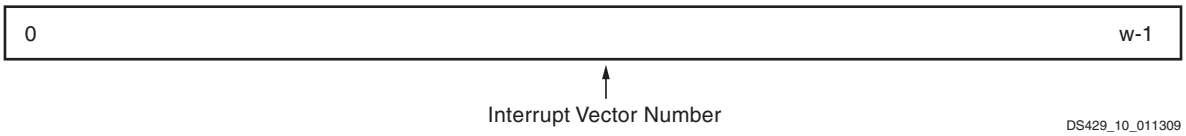

Figure 10: **Interrupt Vector Register (IVR)**

Table 11: **Interrupt Vector Register**

Bits	Name	Description	Reset Value
0 : (w<RD Red><SP Superscript> (1) – 1)	Interrupt Vector Number	Ordinal of highest priority, enabled, active interrupt input	All Ones

Notes:
1. w - Width of Data Bus

Master Enable Register (MER)

This is a two 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 zero, allowing software to write to the ISR to generate interrupts for testing purposes, and disabling any hardware interrupt inputs.

Writing a one to this bit enables the hardware interrupt inputs and disables software generated inputs. Writing a one also disables any further changes to this bit until the device has been reset.

Writing ones or zeros 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 zeros. The Master Enable Register (MER) is shown in Figure 11 and is described in Table 11.

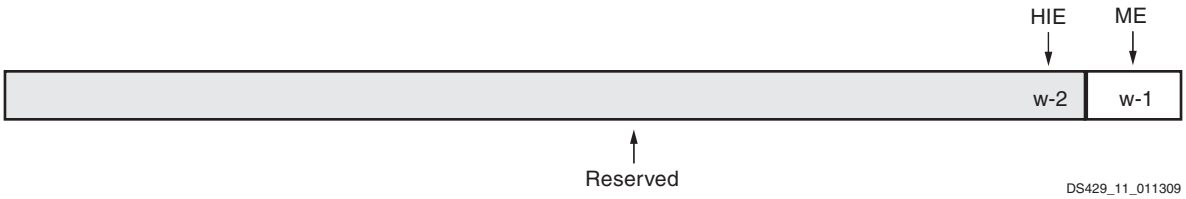

Figure 11: **Master Enable Register (MER)**

Table 12: **Master Enable Register**

Bits	Name	Description	Reset Value
0 : (w<RD Red><SP Superscript>(1) – 3)	Unused	**Not used**	All Zeros
(w<RD Red><SP Superscript>(1) – 2)	HIE	Hardware Interrupt Enable '0' = Read – SW interrupts enabled Write – No effect '1' = Read – HW interrupts enabled Write – Enable HW interrupts	'0'
(w<RD Red><SP Superscript>(1) – 1)	ME	Master IRQ Enable '0' = IRQ disabled – All interrupts disabled '1' = IRQ enabled – All interrupts enabled	'0'

Notes:
1. w - Width of Data Bus

Programming DCR INTC

This section provides an overview for software initialization and communication with an DCR INTC.

The number of interrupt inputs that an Into has is set by the generic C_NUM_INTR_INPUTS described in Table 2. The first input is always Int0 and is mapped to the LSB of the registers (except IVR and MER).

A valid interrupt input signal is any signal that provides the correct polarity and type of interrupt input. Examples of valid interrupt inputs are rising edges, falling edges, high levels, and low levels (hardware interrupts), or software interrupts if HIE has not been set. Each interrupt input can be selectively enabled or disabled (masked). The polarity and type of each hardware interrupt input is specified in the DCR INTC generics C_KIND_OF_INTR, C_KIND_OF_EDGE, and C_KIND_OF_LVL (see Table 2).

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 DCR 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 collectively by a master interrupt enable bit.

During power-up or reset, an Into is put into a state where all interrupt inputs and the interrupt request output are disabled. In order for the Into 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 a one. This allows the DCR 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 DCR 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 zero. Software testing can now proceed by writing a one 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. Once software testing has been completed, or if software testing is not performed, a one is written to the HIE bit, which enables the hardware interrupt inputs and disables any further software generated interrupts.
5. After a one 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 will indicate which inputs are active. If present, the IPR will indicate which enabled inputs are active. Reading the optional IVR provides the ordinal value of the highest priority interrupt that is enabled and active.

For example, if the IVR is present, and 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 three. If Int0 becomes active then reading the IVR will provide a value of zero. If no interrupts are active or it is not present, reading the IVR will return all ones.

Acknowledging an interrupt is achieved by writing a one to the corresponding bit location in the IAR. Note that disabling an interrupt by masking it (writing a zero to the IER) does not clear the interrupt. That interrupt will remain active but blocked until it is unmasked or cleared.

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 zero to the ME bit in the MER. This effectively masks all interrupts for that DCR INTC. Alternatively, interrupt inputs can be selectively masked by writing a zero 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) an interrupt input without having to read, mask off, and then write back the IER. Writing a one to any bit location(s) in the SIE sets the corresponding bit(s) in the IER without affecting any other IER bits.

Writing a one to any bit location(s) in the CIE clears the corresponding bit(s) in the IER without affecting any other IER bits.

Timing Diagram

Figure 12 shows write and read transaction on DCR INTC.

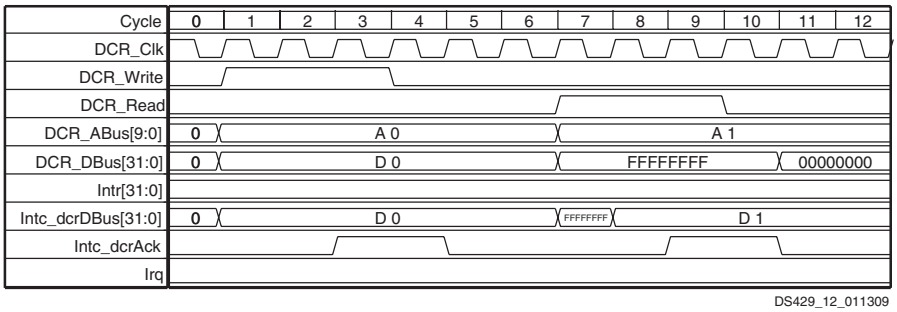

Figure 12: **Write and Read transaction on DCR INTC**

Core Usage in EDK

To use this core with the EDK design tools, see solution record SR18804 for details

Design Implementation

The target technology is an FPGA listed in EDK Supported Device Families.

Device Utilization and Performance Benchmarks

Core Performance

Since the DCR INTC core will be used with other design modules in the FPGA, the utilization and timing numbers reported in this section are estimates only, and will vary from the results reported here.

The DCR INTC resource utilization for various parameter combinations measured with Virtex®-4 as the target device is detailed in Table 13.

Table 13: **Performance/Resource Utilization Benchmarks Virtex-4 FPGA (xc4vfx60-11-ff1152)**

Parameter Values					Device Resources			f_{MAX}
C_NUM_INTR_INPUTS	C_HAS_IPR	C_HAS_SIE	C_HAS_CIE	C_HAS_IVR	Slices	Slice Flip-flops	LUTs	f_{MAX} (MHZ)
1	0	0	0	0	76	52	69	259
1	1	0	0	0	77	53	73	256
1	0	1	0	0	76	52	68	297
1	0	0	1	0	78	52	75	250

Table 13: **Performance/Resource Utilization Benchmarks Virtex-4 FPGA (xc4vfx60-11-ff1152)**

1	0	0	0	1	75	54	70	286
1	1	1	1	1	73	55	75	286
2	1	1	1	1	88	60	90	224
2	0	0	0	0	87	56	83	290
4	1	1	1	1	135	76	130	144
4	0	0	0	0	121	68	115	175
8	1	1	1	1	174	108	168	140
8	0	0	0	0	141	92	139	216
16	1	1	1	1	234	174	295	186
16	0	0	0	0	219	140	246	131
32	1	1	1	1	400	300	523	130
32	0	0	0	0	351	236	426	130

Notes:
1. Above numbers are reported by tool for clock of period constraint of 10ns

The DCR INTC resource utilization for various parameter combinations measured with Virtex-5 as the target device is detailed in Table 14.

Table 14: **Performance/Resource Utilization Benchmarks Virtex-5 FPGA (xc5vfx70t-ff1136-1)**

Parameter Values					Device Resources			f_{MAX}
C_NUM_INTR_INPUTS	C_HAS_IPR	C_HAS_SIE	C_HAS_CIE	C_HAS_IVR	Slices	Slice Flip-flops	LUTs	f_{MAX} (MHZ)
1	0	0	0	0	30	52	62	361
1	1	0	0	0	68	53	63	173
1	0	1	0	0	28	52	62	307
1	0	0	1	0	28	52	62	307
1	0	0	0	1	28	54	64	362
1	1	1	1	1	30	55	64	315
2	1	1	1	1	58	60	69	288
2	0	0	0	0	43	56	65	305
4	1	1	1	1	78	76	87	133
4	0	0	0	0	58	68	83	199
8	1	1	1	1	89	108	137	218
8	0	0	0	0	103	92	119	112

Table 14: **Performance/Resource Utilization Benchmarks Virtex-5 FPGA (xc5vfx70t-ff1136-1)**

16	1	1	1	1	136	172	212	193
16	0	0	0	0	126	140	177	148
32	1	1	1	1	278	300	378	131
32	0	0	0	0	190	236	305	184

Notes:
1. Above numbers are reported by tool for clock of period constraint of 10ns

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
04/11/01	1.0	Initial Xilinx release.
05/04/01	2.0	Changed to reflect EA INTC programmer interface
05/10/01	2.1	Added HIE to MER and changed functionality of ISR to match
09/17/01	2.2	Fixed errors in figures; added I/O and parameterization tables
10/04/01	2.3	Changed C_BASE_NUM_BITS to C_HIGHADDR and added note
11/07/01	2.4	Added Programming the INTC section
12/04/01	3.0	Changed to a core INTC with bus centric wrappers (opb_intc v1.00b and dcr_intc v1.00a); separate figures and tables for OPB and DCR
12/07/01	3.1	Changed DCR signal names to match internal standardization; added type and range columns to I/O tables and type and valid values to generics tables.
05/20/02	3.2	Update for EDK 1.0.
07/22/02	3.3	Add XCO parameters for System Generator
07/31/02	3.4	Add additional text for xco parameters; add additional text for IAR; fixed typos
11/05/02	3.5	Update for EDK SP2
01/08/03	3.6	Update for EDK SP3
07/30/03	3.7	Changed spec to be DCR specific (removed OPB interface information. See OPB_Intc spec for OPB interface information)
12/05/03	3.8	Add notes to table 10 per CR 179857
01/22/04	3.8.1	Update page 1 and add Core Usage in EDK section for CR 182552
08/20/04	3.9	Updated for Gmm; updated trademarks usage and supported device family listing.
01/11/04	4.0	Converted to new DS template; updated images to Xilinx graphic standards; removed all conditional text; removed all references to the OPB interface
05/15/06	4.1	Updated for Virtex-5 support
09/18/06	4.2	Fixed CR #415521
05/18/07	4.3	Fixed CR #422720
03/13/08	4.4	Fixed CR #443710
08/25/08	5.0	Revision rolled to v2.00.a; removed unused parameters C_X, C_Y & C_U_SET; removed support for Virtex2p.
09/24/08	6.0	Removed unused parameter C_FAMILY; converted new DS template; deleted information for other interrupt controllers.
09/26/08	6.1	Updated for review comments.
10/17/08	6.2	Updated the resource utilization numbers.
4/24/09	6.3	Replaced references to supported device families and tool name(s) with hyperlink to PDF file.

Notice of Disclaimer