OPB IPIF (v2.00h)

DS414 November 3, 2006 **Product Specification**

Introduction

The OPB IPIF is a continuation of the Xilinx family of IBM CoreConnect™ compatible LogiCORE products. It provides a bi-directional interface between a User IP core and the OPB 32-bit bus standard. The OPB is a peripheral bus for the Embedded PPC405 processor featured in the Xilinx Virtex-II Pro product line and for the MicroBlaze™ processor, a soft core that can be implemented in many Xilinx FPGA families.

Features

- Compatible with IBM CoreConnect 32-bit OPB.
- Supports User IP data widths from 8 bits to 32 bits with automatic byte steering.
- Extensive User customizing support via HDL parameterization. User optioned services include:
  - Local IP Interrupt collection with User S/W programmable enables/disables.
  - User S/W triggered reset generator for localized reset of User's core.
  - Burst transfer support.
  - DMA function with optional Scatter/Gather mechanization.
  - User configured WrFIFO with optional IP packet support.
  - User configured RdFIFO with optional IP packet support.

LogiCORE™ Facts		
Core Specifics		
Supported Device Family	Spartan™-II, Spartan-IIE, Spartan-3, Spartan-3E, Virtex™, Virtex-II, Virtex-II Pro, Virtex-E, Virtex-4	
Version of Core	opb_ipif	v2.00h
Resources Used		
	Min	Max
Slices		
LUTs		
FFs		
Block RAMs		
Provided with Core		
Documentation	Product Specification	
Design File Formats	VHDL	
Constraints File	None	
Verification	EDK Simulation Support	
Instantiation Template	N/A	
Reference Designs	N/A	
Design Tool Requirements		
Xilinx Implementation Tools	Xilinx EDK 6.2 Xilinx IISE 6.2	
Verification	ModelSim SE 5.6d or later	
Simulation	ModelSim SE 5.6d or later	

NOTICE OF DISCLAIMER: Xilinx is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Xilinx makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Xilinx expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

Functional Description

The OPB IPIF is designed to provide a User with a quick to implement and highly adaptable interface between the IBM OPB Bus and a User IP core. Through the use of VHDL generics, the design provides various services and features that can be optioned in or out, depending on the User requirements. The back end interface standard, the Xilinx IPIC, is common between the OPB IPIF (V2_00_a and later) and the PLB IPIF. This allows IP blocks using the IPIC to be adapted for either the OPB or the PLB. Figure 1 shows a block diagram of the OPB IPIF. The diagram also indicates the module's ports as they relate to the IPIF services. The port references and groupings are detailed in Table 9 on page 21.

The OPB IPIF provides several services for the User, most of which can be optioned in or out. The base element of the design is the Slave Attachment. This block provides the basic functionality for OPB Slave operation. It implements the protocol and timing translation between the OPB Bus and the IPIC. Optionally, the Slave Attachment can be enhanced with burst transfer support. This feature provides higher data transfer rates for OPB *sequential-address* accesses.

The Byte Steering function is responsible for steering data bus information onto the correct byte lanes. This block supports both read and write directions and is required when a target address space in the IPIF or the User IP has a data width smaller than the OPB Bus width (currently 32 bits).

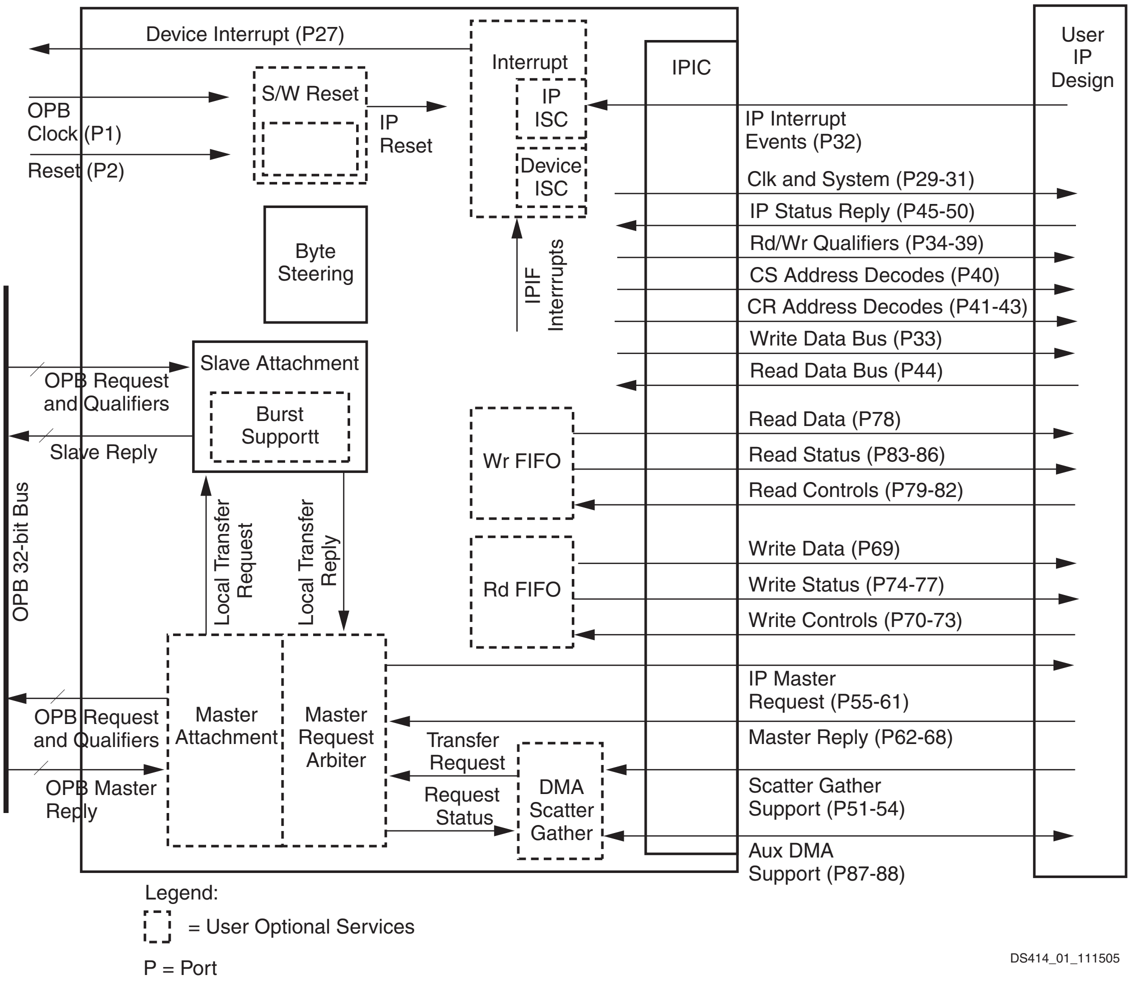


Figure 1: **OPB IPIF Block Diagram**

The User may option in a software Reset service. This service allows the system microprocessor to perform a local reset of the User IP by writing a data key value to the User assigned address space for the Reset Service. This Reset Service is in addition to the hardware reset provided by the OPB Reset input. When activated by the write, a reset pulse is generated that is sent to the User IP and the various internal IPIF elements (excluding the Slave Attachment which has to complete the write timing). When optioned in, the Reset Service can also provide an IPIF Module Information Register (MIR). The MIR is read only and provides information about the IPIF. The information in the register is detailed in the register description section of this document.

An Interrupt Service is provided in the OPB IPIF. This module collects interrupts from the User IP and internal IPIF interrupt sources. It then coalesces them into a single interrupt output signal that is available for connection to a system interrupt controller or the microprocessor. The block contains User accessible capture registers and enable/disable registers. This module is described more thoroughly in the Xilinx LogiCORE ***DS-413 OPB IPIF Interrupt*** Product Specification.

The OPB IPIF provides read and write FIFO Services. They may be independently optioned in or out of the IPIF implementation. These FIFOs are synchronous to the OPB clock and have User parameterized depth, width, and packet support features. The operation of the FiFO's are described in the Xilinx LogiCORE ***DS415 On-Chip Peripheral Bus IP Interface Packet FIFO*** Product Specification.

The OPB IPIF provides an optional Direct Memory Access (DMA) Service. This service automates the movement of large amounts of data between the USER IP or IPIF FIFOs and other peripherals (such as System Memory or a Bridge device). The inclusion of the DMA Service automatically includes the Master Attachment Service. The DMA Service requires access to the OPB as a Master device. The DMA Service may be optionally enhanced with Scatter/Gather mechanization that allows the User to automate DMA operations via Buffer Descriptors stored in System Memory. The operation of the DMA/SG is detailed in the Xilinx LogiCORE ***DS-416 Direct Memory Access and Scatter Gather*** Product Specification.

The User may option in a OPB Master Attachment Service. This is needed when the User IP needs to access the OPB as a Master device.

OPB IPIF Parameters

The OPB IPIF provides for User interface tailoring via VHDL Generic parameters. These parameters are detailed in Table 1. In the table, related parameters are associated into functional groupings. Detailed descriptions of the individual parameters are given in the paragraphs following Table 1.

Table 1: **OPB IPIF Design Parameters**

Generic	Feature / Description	Parameter Name	Allowable Values	Default Value	VHDL Type
IPIF Decoder Address Range Definition					
G1	Array of Address Range Identifiers	C_ARD_ID_ARRAY	See "OPB IPIF Parameter Detailed Descriptions" on page 7 for description.	User must set values.	INTEGER_ARRAY_TYPE [1]
G2	Array of Base Address / High Address Pairs for each Address Range	C_ARD_ADDR_RANGE_ARRAY	See "OPB IPIF Parameter Detailed Descriptions" on page 7 for description.	User must set values.	SLV64_ARRAY_TYPE [1]
G3	Array of data widths for each target address range	C_ARD_DWIDTH_ARRAY	See "OPB IPIF Parameter Detailed Descriptions" on page 7 for description.	User must set values.	INTEGER_ARRAY_TYPE [1]
G4	Array of the desired number of chip enables for each address range	C_ARD_NUM_CE_ARRAY	See "OPB IPIF Parameter Detailed Descriptions" on page 7 for description.	User must set values.	INTEGER_ARRAY_TYPE [1]
G5	Array of unique properties for each address range specified	C_ARD_DEPENDENT_PROPS_ARRAY	See "OPB IPIF Parameter Detailed Descriptions" on page 7 for description.	User must set values.	DEPENDENT_PROPS_ARRAY_TYPE [1]
IPIF Module Identification Register Service					
G6	User assigned Device Block ID	C_DEV_BLK_ID	0 to 255	90	integer
G7	Device Module Information Register Enable	C_DEV_MIR_ENABLE	0 = disabled 1 = enabled	0 (Disabled)	integer
IPIF Slave Attachment Features					
G8	OPB Burst Support Enable	C_DEV_BURST_ENABLE	0 = Burst Support Disabled 1 = Burst Support Enabled	0 (Disabled)	integer
G9	OPB Maximum burst size (in bytes) to be used during DMA fixed burst operations	C_DEV_MAX_BURST_SIZE	2 to 64[2]	64 bytes (16 Words)	integer
IPIF Interrupt Service					
G10	Include IPIF Interrupt Source Controller	C_INCLUDE_DEV_ISC	0 = Omit the IPIF ISC 1 = Include the IPIF ISC	0	integer

Table 1: **OPB IPIF Design Parameters *(Contd)***

Generic	Feature / Description	Parameter Name	Allowable Values	Default Value	VHDL Type
G11	Include the IPIF ISC Priority Encoder service	C_INCLUDE_DEV_PENCODER	0 = Omit the Priority Encoder Service 1 = Include the Priority encoder service	0	integer
G12	Specifies the Number and type of interrupts for the IP Interrupt Status Controller.	C_IP_INTR_MODE_ARRAY	See "OPB IPIF Parameter Detailed Descriptions" on page 7 for description.	User must set.	INTEGER_ARRAY_TYPE (1)
IP Master Service					
G13	Specifies inclusion / omission of a OPB Master Service for the IP	C_IP_MASTER_PRESENT	0 = Omit IP Master Service 1 = Include IP Master Service.	0	integer
G14	If both DMA and an IP master are present, specifies which, if either, will have priority if both masters request simultaneously	C_MASTER_ARB_MODEL	0 = FAIR 1 = DMA_PRIORITY 2 = IP_PRIORITY	0	integer
IPIF DMA/SG Service					
G15	DMA channel type specification Note: The number of array entries specifies number of DMA channels (currently 2 channels supported)	C_DMA_CHAN_TYPE_ARRAY	0 = Simple DMA 1 = Simple Scatter Gather 2 = Tx Scatter Gather with Packet Mode Support 3 = Rx Scatter Gather with Packet Mode Support	(2,3)	INTEGER_ARRAY_TYPE (1)
G16	Specifies the maximum width in bits for DMA transfer byte counters. Note: The number of array entries must match the number of DMA channels	C_DMA_LENGTH_WIDTH_ARRAY	8 to 32	(11,11)	INTEGER_ARRAY_TYPE (1)
G17	Specifies the address assignment for the Length FIFOs used in Scatter Gather operation. Note: The number of array entries must match the number of DMA channels	C_DMA_PKT_LEN_FIFO_ADDR_ARRAY	(x"00000000_00000000",x"00000000_FFFFFFFF")	(x"00000000_00000000",x"00000000_00000000")	SLV64_ARRAY_TYPE(1)

Table 1: **OPB IPIF Design Parameters *(Contd)***

Generic	Feature / Description	Parameter Name	Allowable Values	Default Value	VHDL Type
G18	Specifies the address assignment for the Status FIFOs used in Scatter Gather operation. Note: The number of array entries must match the number of DMA channels	C_DMA_PKT_STAT_FIFO_ADDR_ARRAY	(x"00000000_00000000",x"00000000_FFFFFFFF")	(x"00000000_00000000",x"00000000_00000000")	SLV64_ARRAY_TYPE[(1)]
G19	Specifies for each DMA channel, the interrupt coalescing value. Note: The number of array entries must match the number of DMA channels	C_DMA_INTR_COALESCE_ARRAY	0 = Interrupt coalescing disabled for the channel 1 = Interrupt coalescing enabled for the channel	(0,0)	INTEGER_ARRAY_TYPE [(1)]
G20	DMA burst size in units of C_OPB_DWIDTH. If set to 1, DMA will use single transactions, disabling burst. [(2)]	C_DMA_BURST_SIZE	A power of two less than or equal to 16.	16	integer
G21	Allow DMA to use burst transfers	C_DMA_SHORT_BURST_REMAINDER	0 = Use single transactions once the amount left in a DMA operation (the remainder) is less than C_DMA_BURST_SIZE. 1 = Use a short burst for the remainder.	0	integer
G22	Applies if interrupt coalescing is included. Gives the base time unit in ns for the packet-wait-bound function	C_DMA_PACKET_WAIT_UNIT_NS	Expected to be left at default but may take any value that makes sense for the application (e.g. lower value for simulations)	1000000 (1 ms)	integer
OPB I/O Specification					
G23	Width of the OPB Address Bus	C_OPB_AWIDTH	32	32	integer
G24	Width of the OPB Data Bus	C_OPB_DWIDTH	32	32	integer
G25	OPB Clock Period in picoseconds	C_OPB_CLK_PERIOD_PS	up to 10000 (100MHz)	10000	integer
IPIF Bus Specification					
G26	IPIF Data Bus Width (bits)	C_IPIF_DWIDTH	Set equal to **G24**	64	integer

Table 1: **OPB IPIF Design Parameters *(Contd)***

Generic	Feature / Description	Parameter Name	Allowable Values	Default Value	VHDL Type
FPGA Family Type					
G27	Xilinx FPGA Family	C_FAMILY	virtex2, virtex2p, virtexe, virtex, spartan3	virtex2	string

Notes:

1. This Parameter VHDL type is a custom type defined in the ipif_pkg.vhd.
2. If C_DEV_BURST_ENABLE = 0, then C_DMA_BURST_SIZE is disregarded and the DMA burst size is set to 1 to disable DMA bursting.

OPB IPIF Parameter Detailed Descriptions

Address Range Definition Arrays

One of the primary functions of the OPB IPIF is to provide address decoding, Chip Enable/Chip Select control signal generation, and data byte steering. This is a complex task when the diversity of decoding possibilities and data bus widths that may be required by numerous and quite different client peripherals is considered. The solution to the problem has been to make the OPB IPIF address decoding function highly customizable by the peripheral designer via parameterization.

The OPB IPIF employs VHDL Generics that are defined as unconstrained arrays as the method for customizing address space decoding. These parameters are called the Address Range Definition (ARD) arrays. The OPB IPIF has five such arrays that are used for address space definition. They can be recognized by the "C_ARD" prefix of the Generic name. The ARD Generics are:

- · C_ARD_ID_ARRAY
- · C_ARD_ADDR_RANGE_ARRAY
- · C_ARD_DWIDTH_ARRAY
- · C_ARD_NUM_CE_ARRAY
- · C_ARD_DEPENDENT_PROPS_ARRAY

One of the big advantages of using unconstrained arrays for address space description is that it allows the User to specify as few or as many unique and non-contiguous OPB Bus address spaces as the peripheral design needs. The IPIF decoding logic will be optimized to recognize and respond to only the those defined address spaces during active OPB Bus transaction requests. Since the number of entries in the arrays can grow or shrink based on each User Application, the IPIF is designed to analyze the User's entries in the arrays and then automatically add or remove services, resources, and interconnections based on the arrays' contents. Unfortunately, top level VHDL ports cannot come and go as services are added or removed. The ports on unused services can only be minimally sized and deactivated. This requires the User wrapper to ignore unused service outputs from the IPIF IPIC and tie low unused service inputs to the IPIC.

The ordering of a set of address space entries within the ARD arrays is not important. Each address space is processed independently from any of the other address space entries. **However, once an ordering is established in any one of the arrays, that ordering of the entries must be maintained across all of the ARD arrays**. That is, the first entry in C_ARD_ID_ARRAY will be associated with the first address pair in C_ARD_ADDR_RANGE_ARRAY which is associated with the first data width entry in the C_ARD_DWIDTH_ARRAY and so on.

C_ARD_ID_ARRAY

The C_ARD_ID_ARRAY is used to assign an integer identifier to an Address space definition. Predefined identifiers are declared as VHDL constants in the ipif_pkg.vhd file located in the ipif_common library. Table 2 lists the currently predefined identifiers. The identifiers are separated into two categories; IPIF Mandatory and User Optional.

The first category is the IPIF Mandatory service identifiers. These are used by the IPIF elaboration processing to include or remove services provided internally by the IPIF design. The IPIF will include a service only if it's corresponding identifier is present in the C_ARD_ID_ARRAY. If the identifier is not found in the array, the service and the associated resources are not implemented. The User's use of these identifiers is mandatory when specifying address spaces for IPIF services. It should be noted that Xilinx has reserved a block of integer values for future IPIF service identifiers.

The second category of identifier is the User Optional category. Seventeen identifiers have been predefined for User convenience. They can be used to identify the address spaces in the User's peripheral. Optionally, Users may define their own identifiers that have more descriptive names for the address space they are defining. For example, the following definitions could be made in a User VHDL package or in the HDL design file hosting the IPIF component instantiation.

· Constant Control_Regs : integer := 120;

· Constant Status_Regs : integer := 121;

· Constant Data_Buffer : integer := 122;

Once declared, these identifiers can then be entered into the C_ARD_ID_ARRAY to provide a more descriptive identifier for User address spaces. When defining their own identifiers, Users must ensure that the assigned integer value does not conflict with any of the Xilinx predefined or reserved identifier values. In addition, any User defined identifier values must also be unique with respect to one another.

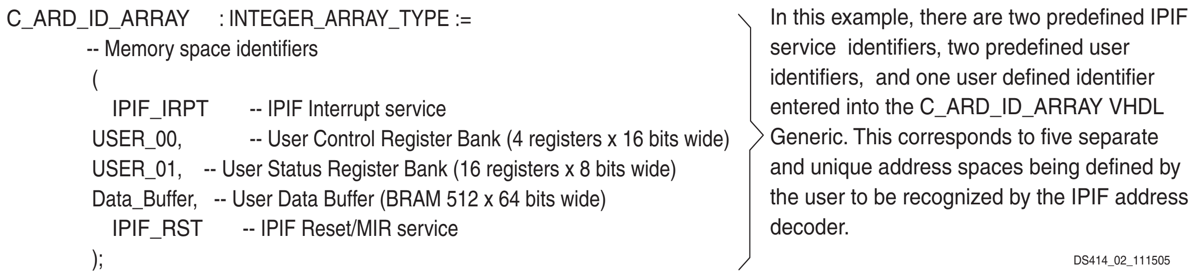
Table 2: **Xilinx Predefined Address Space Identifiers**

Identifier Name	Identifier VHDL Type	Identifier Value	Identifier Description
IPIF Mandatory Category			
IPIF_INTR	Integer	1	The presence of this identifier in the C_ARD_ID_ARRAY specifies the inclusion of the IPIF Interrupt Service in the IPIF.
IPIF_RST	Integer	2	The presence of this identifier in the C_ARD_ID_ARRAY specifies the inclusion of the IPIF S/W Reset /MIR Service in the IPIF.
IPIF_DMA_SG	Integer	4	The presence of this identifier in the C_ARD_ID_ARRAY specifies the inclusion of the DMA Service in the IPIF. The inclusion of DMA will also cause the IPIF to automatically include a OPB Master Interface Service.
IPIF_WRFIFO_REG	Integer	5	The presence of this identifier in the C_ARD_ID_ARRAY specifies the inclusion of the IPIF WrFIFO register Service.
IPIF_WRFIFO_DATA	Integer	6	The presence of this identifier in the C_ARD_ID_ARRAY specifies the inclusion of the IPIF WrFIFO Service.

Table 2: **Xilinx Predefined Address Space Identifiers *(Contd)***

Identifier Name	Identifier VHDL Type	Identifier Value	Identifier Description
IPIF_RDFIFO_REG	Integer	7	The presence of this identifier in the C_ARD_ID_ARRAY specifies the inclusion of the IPIF RdFIFO register Service.
IPIF_RDFIFO_DATA	Integer	8	The presence of this identifier in the C_ARD_ID_ARRAY specifies the inclusion of the IPIF RdFIFO Service.
Reserved IPIF	Integer	9 - 99	These identifier values are reserved for IPIF services that may be added in the future.
User Optional Category			
USER_00	Integer	100	User Address Space 0
USER_01	Integer	101	User Address Space 1
USER_02	Integer	102	User Address Space 2
USER_03	Integer	103	User Address Space 3
USER_04	Integer	104	User Address Space 4
USER_05	Integer	105	User Address Space 5
USER_06	Integer	106	User Address Space 6
USER_07	Integer	107	User Address Space 7
USER_08	Integer	108	User Address Space 8
USER_09	Integer	109	User Address Space 9
USER_10	Integer	110	User Address Space 10
USER_11	Integer	111	User Address Space 11
USER_12	Integer	112	User Address Space 12
USER_13	Integer	113	User Address Space 13
USER_14	Integer	114	User Address Space 14
USER_15	Integer	115	User Address Space 15
USER_16	Integer	116	User Address Space 16

An example C_ARD_ID_ARRAY is shown in Figure 2. This example will be carried through each of the ARD Array descriptions.



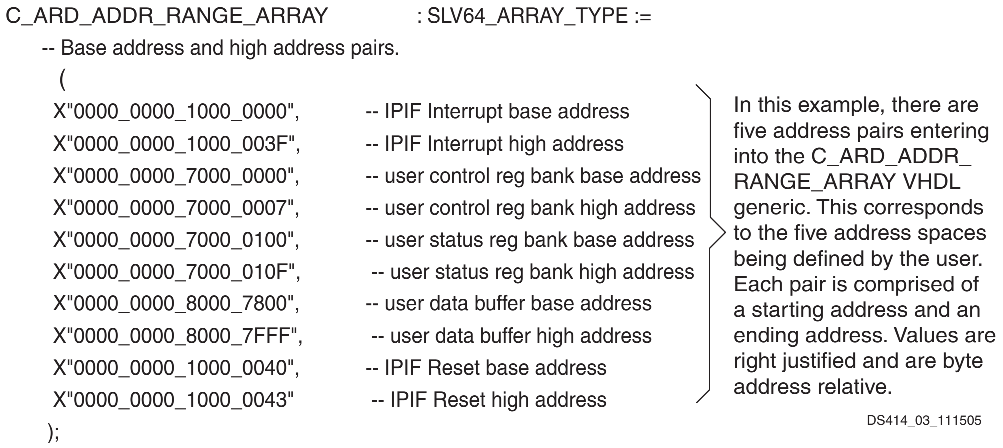
```
C_ARD_ID_ARRAY     : INTEGER_ARRAY_TYPE :=
           -- Memory space identifiers
            (
               IPIF_IRPT       -- IPIF Interrupt service
            USER_00,           -- User Control Register Bank (4 registers x 16 bits wide)
            USER_01,     -- User Status Register Bank (16 registers x 8 bits wide)
            Data_Buffer,   -- User Data Buffer (BRAM 512 x 64 bits wide)
               IPIF_RST       -- IPIF Reset/MIR service
            );
```

In this example, there are two predefined IPIF service identifiers, two predefined user identifiers, and one user defined identifier entered into the C_ARD_ID_ARRAY VHDL Generic. This corresponds to five separate and unique address spaces being defined by the user to be recognized by the IPIF address decoder.

Figure 2: **Example Address Space Identifier Entries.**

C_ARD_ADDR_RANGE_ARRAY

The actual address range for an address space definition is entered in this array. Each address space is by definition a contiguous block of addresses as viewed from the host microprocessor's total addressable space. It's specification requires a pair of entries in this array. The first entry of the pair is the Base Address (starting address) of the block, the second entry is the High Address (ending address) of the block. These addresses are byte relative addresses. The array elements are defined as std_logic_vector(0 to 63) in the ipif_pkg.vhd file in IPIF Common library. Currently, the biggest address bus used on the PLB and OPB buses is 32 bits. However, 64 bit values have been allocated for future growth in address bus width.



```
C_ARD_ADDR_RANGE_ARRAY             : SLV64_ARRAY_TYPE :=
    -- Base address and high address pairs.
     (
     X"0000_0000_1000_0000",        -- IPIF Interrupt base address
     X"0000_0000_1000_003F",        -- IPIF Interrupt high address
     X"0000_0000_7000_0000",        -- user control reg bank base address
     X"0000_0000_7000_0007",        -- user control reg bank high address
     X"0000_0000_7000_0100",        -- user status reg bank base address
     X"0000_0000_7000_010F",         -- user status reg bank high address
     X"0000_0000_8000_7800",        -- user data buffer base address
     X"0000_0000_8000_7FFF",         -- user data buffer high address
     X"0000_0000_1000_0040",        -- IPIF Reset base address
     X"0000_0000_1000_0043"         -- IPIF Reset high address
    );
```

In this example, there are five address pairs entering into the C_ARD_ADDR_ RANGE_ARRAY VHDL generic. This corresponds to the five address spaces being defined by the user. Each pair is comprised of a starting address and an ending address. Values are right justified and are byte address relative.

DS414_03_111505

Figure 3: **Address Range Specification Example.**

The User must follow several rules when assigning values to the address pairs. These rules assure that the address range will be correctly decoded in the IPIF. First, the User must decide the required address range to be defined. The block size (in bytes) must be a power of 2 (i.e. 2, 4, 8,16,32,64,128,256 and so on). Secondly, the Base Address must start on an address boundary that is a multiple of the chosen block size. For example, an address space is needed that will include 2048 bytes (0x800 hex) of the system memory space. Valid Base Address entries are 0x00000000, 0x00000800, 0xFFFFF000, 0x90001000, etc. A value of 0x00000120 is not valid because it is not a multiple of 0x800 (2048). Thirdly, the High Address entry is equal to the assigned Base Address plus the block size minus 1. Continuing the example of a 2048 byte block size, a Base Address of 0x00000000 yields a High Address of 0x000007FF; a Base Address of 0x00000800 would require a corresponding High Address value of 0x00000FFF.

The IPIF predefined services also have predefined address space block sizes. Please refer to Table 4 for the values. These sizes must be used to formulate the Base_Address and High_Address entries in the C_ARD_ADDR_RANGE ARRAY if the services are utilized. Note that there is not a restriction on where the address space is assigned in memory space.

Table 3: **Minimum Required Address Block for IPIF Predefined Services.**

Predefined Identifier	Minimum Address Block Size Required (bytes)
IPIF_INTR	64 (40$_H$)
IPIF_RST	4 (04$_H$)
IPIF_DMA_SG	64 (40$_H$) per DMA Channel Specified

Table 3: **Minimum Required Address Block for IPIF Predefined Services. *(Contd)***

Predefined Identifier	Minimum Address Block Size Required (bytes)
IPIF_WRFIFO_REG	8 (08_H)
IPIF_WRFIFO_DATA	This should be set to the smallest power of 2 space that will encompass the largest contiguous burst size (in bytes) used to write to the FIFO by the OPB.
IPIF_RDFIFO_REG	8 (08_H)
IPIF_RDFIFO_DATA	This should be set to the smallest power of 2 space that will encompass the largest contiguous burst size (in bytes) used to read from the FIFO by the OPB.
Reserved IPIF	Not Yet Defined

C_ARD_DWIDTH_ARRAY

The IPIF allows a User to connect a peripheral that has bus accessible resources with data widths smaller than that of the host bus data width. The IPIF is able to mechanize this via information contained in the C_ARD_DWIDTH_ARRAY. The User enters the integer value of the data width (in bits) of each memory space defined. The allowed values are 8, 16, 32 for a 32-bit OPB bus. Note that if multiple data widths are present in a peripheral, the User must define a unique address space for each unique data width. When an address space decode is active in the IPIF decoder logic, the data width value for the target resource, the active Byte Enables, and the byte address are used to *steer* the data to/from the 32-bit OPB Bus byte lanes from/to the target's data port byte lanes. When connecting a peripheral to the IPIF that has data widths less than the Bus data width, the User must connect to the IPIF read/write data ports starting with the most significant byte lane (lane 0) to the least significant byte lane (lane 3). Figure 5 shows an example of connecting a multi-data width peripheral to the IPIF data buses. An example of how this array is populated is shown in Figure 4.

The IPIF predefined services must also have predefined data widths. These data widths must be entered in the C_ARD_DWIDTH ARRAY if the services are utilized. The data width values are shown in Table 4.

Table 4: **Required Data Width Entry for IPIF Predefined Services.**

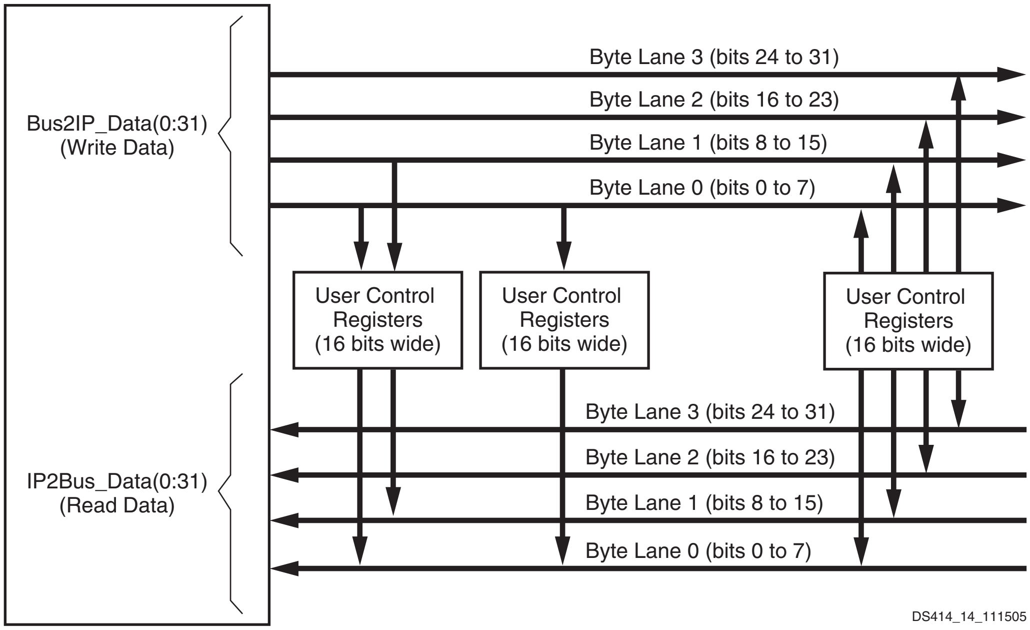
Predefined Identifier	Required C_ARD_DWIDTH_ARRAY Entry
IPIF_INTR	32
IPIF_RST	32
IPIF_DMA_SG	32
IPIF_WRFIFO_REG	32
IPIF_WRFIFO_DATA	32
IPIF_RDFIFO_REG	32
IPIF_RDFIFO_DATA	32
Reserved IPIF	Not Yet Defined

```
C_ARD_DWIDTH_ARRAY      : INTEGER_ARRAY_TYPE :=
           -- Memory space data width definition (in bits)
            (
              32     -- IPIF Interrupt service (32 interrupts needed from User IP)
            16,      -- User Control Register Bank (4 registers x 16 bits wide)
            8,    -- User Status Register Bank (16 registers x 8 bits wide)
            64,   -- User Data Buffer (BRAM 512 x 64 bits wide)
              32    -- IPIF Reset/MIR service (always 32 bits)
            );
```

In this example, the User defines the data widths of the various address spaces being defined. The IPIF Address Decoder furnishes this information to the Byte Steering module for appropriate data routing and mirroring that is dynamically tailored for each address space as it is accessed by the PLB.

DS414_04_111505

Figure 4: **Data Width Array Example.**



Bus2IP_Data(0:31)
(Write Data)
Byte Lane 3 (bits 24 to 31)
Byte Lane 2 (bits 16 to 23)
Byte Lane 1 (bits 8 to 15)
Byte Lane 0 (bits 0 to 7)
User Control Registers (16 bits wide)
User Control Registers (16 bits wide)
User Control Registers (16 bits wide)
IP2Bus_Data(0:31)
(Read Data)
Byte Lane 3 (bits 24 to 31)
Byte Lane 2 (bits 16 to 23)
Byte Lane 1 (bits 8 to 15)
Byte Lane 0 (bits 0 to 7)
DS414_14_111505

Figure 5: **Example Byte Lane Connections of User Functions**

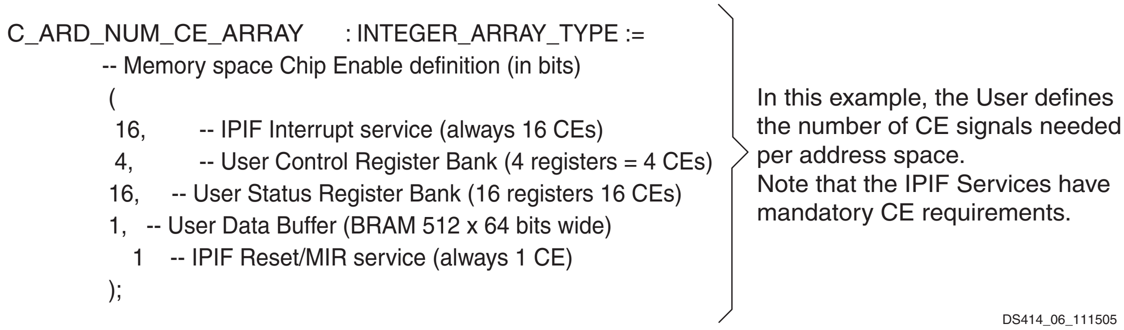
C_ARD_NUM_CE_ARRAY

The IPIF decoding logic provides the User the ability to generate multiple chip enables within a single address space. This is primarily used to support a bank of registers that need an individual chip enable for each register. The User enters the desired number of chip enables for an address space in the C_ARD_NUM_CE_ARRAY. The values entered are positive integers. Each address space must have at least 1 chip enable specified. The address space range will be subdivided and sequentially assigned a chip enable based on the data width entered into the C_ARD_DWIDTH_ARRAY. This supports register widths from 8 bits up to the width of the OPB. The User Application activates the chip enable by using an address within the defined address space that is aligned in accordance with the corresponding DWIDTH entry in the C_ARD_DWIDTH_ARRAY.

The User must ensure that the address space for a group of chip enables is greater than or equal to the specified width of the memory space in bytes (C_ARD_DWIDTH_ARRAY entry / 8) times the number of desired chip enables. IPIF service address spaces have a predefined number of chip enables that must be used when included in the address space list. These are shown in Table 5.

Table 5: **Required Chip Enable Entry for IPIF Predefined Services.**

Predefined Identifier	Required C_ARD_NUM_CE_ARRAY Entry
IPIF_INTR	16
IPIF_RST	1
IPIF_DMA_SG	1
IPIF_WRFIFO_REG	2
IPIF_WRFIFO_DATA	1
IPIF_RDFIFO_REG	2
IPIF_RDFIFO_DATA	1
Reserved IPIF	Not Yet Defined



```
C_ARD_NUM_CE_ARRAY      : INTEGER_ARRAY_TYPE :=
            -- Memory space Chip Enable definition (in bits)
             (
              16,       -- IPIF Interrupt service (always 16 CEs)
              4,        -- User Control Register Bank (4 registers = 4 CEs)
             16,    -- User Status Register Bank (16 registers 16 CEs)
             1,   -- User Data Buffer (BRAM 512 x 64 bits wide)
                1    -- IPIF Reset/MIR service (always 1 CE)
             );
```

In this example, the User defines the number of CE signals needed per address space.
Note that the IPIF Services have mandatory CE requirements.

Figure 6: **Chip Enable Specification Example**

C_ARD_DEPENDENT_PROPS_ARRAY

The C_ARD_DEPENDENT_PROPS_ARRAY is used to provide additional parameters, if needed, that are specific to the type of Address Range. That is, it is used to further define *properties* that are *dependent* on the type of Address Range. The number of dependent properties varies from Address Range to Address Range and may be zero. Whether or not there are dependent properties, an entry in C_ARD_DEPENDENT_PROPS_ARRAY is required for each Address Range in the system. This keeps the ARD arrays aligned, a requirement that was mentioned previously. In what follows, it will be explained how this required entry can be kept small, never larger than needed for the amount of dependent-property parameterization actually used.

Dependent-property parameterization is based upon a two-dimensional array construct. The "outer" array is unconstrained and, as mentioned, must be populated with one entry per Address Range, just as with the other ARD-array parameters. The "inner" array is globally constrained by the system to some size, but the actual size is not relevant to the user, beyond the assurance that it will always be large enough to accommodate the needs of any of the pre-defined Address Ranges.

By implementation, the dependent-property parameter is actually an index into the inner array and the value at that index is the value of the parameter. Any indices that are not required are defaulted to the value of zero. To keep things readable and adaptable for future expansion, the user refers to the parameterization indices only by symbolic names defined in the ipif_pkg package[1]. The symbolic name of an index is just the name desired to be given to the corresponding parameter. When defining the symbolic name for a dependent-property parameter, if possible, the name is chosen so that an assigned value of zero implies the desired default state for the parameter.

The VHDL *aggregate* construct can be used as the "user interface" to this type of parameterization, as shown next, where the "OTHERS =>" association is used to set all dependent properties to zero, which, since none of the Address Ranges of the running example actually uses dependent properties, effectively gives them "don't care" values.

```
C_ARD_DEPENDENT_PROPS_ARRAY : DEPENDENT_PROPS_ARRAY_TYPE
        := ( 0 => (others => 0),        -- IPIF_IRPT
             1 => (others => 0),        -- USER_00
             2 => (others => 0),        -- USER_01
             3 => (others => 0),        -- User Data Buffer
             4 => (others => 0)         -- IPIF_RST
           );
```

Note that explicit index assignment using *named association* is used in the example. In theory, *positional association* as in the previous unconstrained-array examples could alternatively be used. However, some VHDL tools have not proven to be robust when positional association was used for population of dependent-properties-array aggregates.

In the version of the OPB IPIF presented in this document, the only IPIF-resident address ranges that have dependent properties are the write and read FIFOs (See Table 6 for an enumeration). Each FIFO type actually has two Address-Range IDs, but the dependent properties are attached to only one of these, namely to either the IPIF_WFIFO_DATA or the IPIF_RFIFO_DATA Address Range, as appropriate. (As an aside, note also that the other IPIF resident services such as the Reset/MIR service and the interrupt management service *could have* been parameterized using dependent properties. The option to do so is being considered for future IPIFs.)

Next, for the purpose of illustration, we extend the running example to show how dependent-property parameterization would apply for a system that also has a Read FIFO and a Write FIFO. The four additional Address Ranges are added after the Address Ranges that have been part of the running example. This gives an expanded system containing nine Address Ranges. Of course, the other four ARD arrays would also have to be complemented with corresponding expanded values to have a valid nine-address-range system, but we only show the full expanded system for the presently discussed C_ARD_DEPENDENT_PROPS_ARRAY parameter, since the purpose of the expanded system is to illustrate dependent-property parameterization.

```
C_ARD_DEPENDENT_PROPS_ARRAY : DEPENDENT_PROPS_ARRAY_TYPE
        := ( 0 => (others => 0),        -- IPIF_IRPT
             1 => (others => 0),        -- USER_00
             2 => (others => 0),        -- USER_01
             3 => (others => 0),        -- User Data Buffer
```

[1] This VHDL package contains constants and functions that support the declaration and usage of Xilinx IPIFs. In essence, the package is part of IPIF declarations. For the OPB IPIF described herein, it is compiled into VHDL library ipif_common_v1_00_d from the source code in ipif_pkg.vhd.

```
  4 => (others => 0),             -- IPIF_RST
  5 => (others => 0),             -- IPIF_WRFIFO_REG
  6 => (FIFO_CAPACITY_BITS => 16384, -- IPIF_WRFIFO_DATA
        WR_WIDTH_BITS => 32,
        RD_WIDTH_BITS => 32,
        EXCLUDE_VACANCY => 1,
        OTHERS => 0
       ),
  7 => (others => 0),             -- IPIF_RDFIFO_REG
  8 => (FIFO_CAPACITY_BITS => 16384, -- IPIF_RDFIFO_DATA
        WR_WIDTH_BITS => 32,
        RD_WIDTH_BITS => 32,
        EXCLUDE_VACANCY => 1,
        OTHERS => 0
   )
);
```

For the expanded example, each FIFO has another available parameter--with symbolic name EXCLUDE_PACKET_MODE. However, since the default value of this parameter (0, for "packet mode *not* to be excluded") is the desired characteristic, this parameter is not treated explicitly and instead defaulted. However, explicit treatment is allowed, if, for example, the User wanted to explicitly show all parameters.

Taken together these concepts and mechanisms allow to achieve the goals of folding away unnecessary parameter details, of having readable dependent-parameter associations, and of leaving open the possibility of expanding the pre-defined set of Address Range types[1] without changing the static generic interface of the IPIF.

[1] The expansion could also be with respect to adding parameters to an already defined Address Range type.

Table 6: **Packet FIFO Dependent Properties Parameters.**

FIFO Parameter	Predefined Index Alias	Description
FIFO Capacity (bits)	FIFO_CAPACITY_BITS	This parameter specifies the capacity of the FIFO (in bits). This is defined as the number of storage locations desired multiplied by the width of the storage location (in bits). [(1)]
Write Port Width (bits)	WR_WIDTH_BITS	This parameter specifies the bit width of the Write port for the FIFO. For a read FIFO, this is the port accessible by the User IP via the IPIC. For a Write FIFO, this the port accessible by the OPB Bus.The port providing access to the OPB Bus cannot exceed the width of the OPB Bus. [(2)]
Read Port Width (bits)	RD_WIDTH_BITS	This parameter specifies the bit width of the Read port for the FIFO. For a write FIFO, this is the port accessible by the User IP via the IPIC. For a Read FIFO, this is the port accessible by the OPB Bus. The port providing access to the OPB Bus cannot exceed the width of the OPB Bus. [(2)]
Packet Mode Exclusion/ Inclusion	EXCLUDE_PACKET_MODE	If it is not needed, this parameter allows the User to save FPGA resources by specifying the exclusion the Packet Mode feature in the FIFO. 0 = Retain the Packet Mode feature. 1 = Exclude the Packet Mode Feature.
Vacancy Calculation Exclusion/ Inclusion	EXCLUDE_VACANCY	If it is not needed, this parameter allows the User to save FPGA resources by specifying the exclusion the Vacancy calculation feature in the FIFO. 0 = Retain the Vacancy feature. 1 = Exclude the Vacancy Feature.
Reserved	N/A	Reserved

Notes:

1. The maximum FIFO capacity is dependent upon the target FPGA family specified in the C_FAMILY parameter. Virtex™ and Virtex™E Packet FIFO implementations have a maximum capacity is 4096 x 64 or 262,144 bits per FIFO. Virtex™II and Virtex™II Pro device implementations have a maximum Packet FIFO capacity of 16384 x 64 or 1,048,576 bits.
2. The current implementation of the RdFIFO and the WrFIFO require the read port and the write port to be the same width.

C_DEV_BLK_ID

This integer parameter has a range from 0 to 255. It is used to uniquely identify a device within a User system. It has no functional effect on the IPIF. It is intended to support hardware and software integration of the User's microprocessor system by providing a unique and constant data value that can be read via the OPB. It's value is echoed in the IPIF MIR. This parameter is only used when the C_DEV_MIR_ENABLE is set to 1.

C_DEV_MIR_ENABLE

This integer parameter has a range of 0 to 1. Setting it to 1 enables the Module Information Record (MIR) for the IPIF to be read via the OPB. This parameter is used only if the Reset Service is included in the IPIF.

C_DEV_BURST_ENABLE

This integer parameter has a range of 0 to 1. A setting of 1 enables the inclusion of additional logic needed to support OPB sequential-address transfers as bursts.

C_DEV_MAX_BURST_SIZE

This integer parameter defines the maximum number of bytes that are allowed to be transferred in a single burst operation and is used to size certain buses in the IPIF.

C_INCLUDE_DEV_ISC

This integer parameter has a range of 0 to 1 and is used only if the IPIF Interrupt Service is included. When set to 1, it specifies that a Device Interrupt Source Controller (ISC) for internal IPIF generated interrupts is required. This device ISC is used to collect WrFIFO and RdFIFO deadlock interrupts (an error that may occur in Packet Mode Operation), DMA / SG interrupts, and the transfer Error interrupt. Additional information on the Interrupt Service in the IPIF can be found in the Xilinx LogiCORE *DS-413 OPB IPIF Interrupt* Product Specification.

C_INCLUDE_DEV_PENCODER

This integer parameter has a range of 0 to 1 and is used only if the IPIF Interrupt Service is included and the C_INCLUDE_DEV_ISC parameter is set to 1. This parameter, when set to 1, includes a priority encoder function needed to generate the value of the Interrupt ID register in the Device Interrupt Source Controller. A priority encoder is useful in aiding the User interrupt service routine to resolve the source of an interrupt within a OPB device incorporating an IPIF. Additional information on the Interrupt Service in the IPIF can be found in the Xilinx LogiCORE *DS-413 OPB IPIF Interrupt* Product Specification.

C_IP_INTR_MODE_ARRAY

This parameter allows the User to define the number and capture type of interrupt events from the User IP to the IP ISC located in the IPIF Interrupt Service. This parameter is defined as an unconstrained array of integers in the range of 1 to 6. Xilinx has predefined constant identifiers that provide a descriptive name in place of the integer value. These identifiers are detailed in Table 7. The User must make a capture mode entry in this parameter array for each input interrupt event needed from the User IP to the IPIF Interrupt Service.

There are two limits to how many IP interrupts can be specified by the User. The first and upper limit is the data bus width of the OPB. This can be limited further if the User elects to enter a bit width less than the OPB bus width in the C_ARD_DWIDTH_ARRAY entry corresponding to the IPIF Interrupt Service. If the entered value is less than 64 (which could save FPGA resources), then the User's IP interrupt number will be capped at that entered width value.

The number of entries in this array determines the size of the IP2Bus_IntrEvent bus which conveys the User IP interrupts to the IPIF. The entry position in the array corresponds directly with bit index of the IP2Bus_IntrEvent bus. That is, entry position 0 in the array corresponds to IP2Bus_IntrEvent(0), entry position 1 in the array corresponds to IP2Bus_IntrEvent(1), and so on. User Application access of Device and IP Interrupt Source Controller registers is detailed in the register description section of this document Additional interrupt processing insight can be sought in the Xilinx LogiCORE *DS-413 OPB IPIF Interrupt* Product Specification.

Table 7: **Xilinx Predefined Interrupt Capture Mode Identifiers**

Identifier Name	Identifier VHDL Type	Identifier Value	Identifier Description
Pass Through			
INTR_PASS_THRU	Integer	1	The input interrupt from the IP has no additional capture processing applied to it. It is immediately sent to the IP ISC Interrupt Enable gating logic.
INTR_PASS_THRU_INV	Integer	2	The input interrupt from the IP is logically inverted but has no additional capture processing applied to it. The inverted interrupt level is immediately sent to the IP ISC Interrupt Enable gating logic.
Sample and Hold Logic High			
INTR_REG_EVENT	Integer	3	The IP ISC Status Register will sample the IP Interrupt input at the rising edge of each OPB clock pulse. If a logic high is sampled, the bit of the IP Interrupt Status Register corresponding to the input interrupt position will stay high until the User Application (interrupt service routine) clears the interrupt.
INTR_REG_EVENT_INV	Integer	4	This capture mode is the same as the INTR_REG_EVENT mode except that the IP Interrupt input is logically inverted before it enters the Sample and Hold logic of the IP Interrupt Status Register.
Registered Edge Detect			
INTR_POS_EDGE_DETECT	Integer	5	The IP ISC Status Register will sample the IP Interrupt input at the rising edge of each OPB clock pulse. A one OPB clock delayed sample will also be maintained. The new sample and the delayed sample will be compared. If the new sample is logic high and the old sample is logic low (a rising edge event), the IP Interrupt Status Register will latch and hold a logic 1 for the interrupt bit position. Once latched, the bit of the IP Interrupt Status Register corresponding to the input interrupt position will stay high until the User Application (interrupt service routine) clears the interrupt.
INTR_NEG_EDGE_DETECT	Integer	6	The IP ISC Status Register will sample the IP Interrupt input at the rising edge of each OPB clock pulse. A one OPB clock delayed sample will also be maintained. The new sample and the delayed sample will be compared. If the new sample is logic low and the old sample is logic high (a falling edge event), the IP Interrupt Status Register will latch and hold a logic 1 for the interrupt bit position. Once latched, the bit of the IP Interrupt Status Register corresponding to the input interrupt position will stay high until the User Application (interrupt service routine) clears the interrupt.

C_IP_MASTER_PRESENT

This integer parameter has a range of 0 to 1. If the User's IP requires access to the OPB as a Master, this parameter must be set to a value of 1. This will cause the IPIF to include the Master Attachment Service in its implementation and the IP Master interface signals on the IPIC will become active.

C_IMASTER_ARB_MODEL

This integer parameter has a range of 0 to 2. It applies only if both a master function is used in the IP core (C_IP_MASTER_PRESENT = 1) and the DMA service is requested by including an Address Range of type IP_DMA_SG in the C_ARD_ID_ARRAY address-range allocation parameter. In this case, arbitration is required to regulate access in the face of simultaneous requests from the two master functions. If the C_MASTER_ARB_MODEL parameter is set to 0, *fair* arbitration is used. If the parameter is set to 1, the DMA controller will always have priority and, similarly, if the parameter is set to 2, the IP master will always have priority.

C_DMA_CHAN_TYPE_ARRAY

This parameter array applies directly to the DMA/SG Service within the IPIF. Please refer to the Xilinx LogiCORE *DS-416 Direct Memory Access and Scatter Gather* Product Specification for a description of the service. This IPIF parameter is assigned to the **C_DMA_CHAN_TYPE** parameter of the DMA/SG Service. This parameter is only used if the DMA Service is activated in the IPIF via the **C_ARD_ID_ARRAY**.

C_DMA_LENGTH_WIDTH_ARRAY

This parameter array applies directly to the DMA/SG Service within the IPIF. Please refer to the Xilinx LogiCORE *DS-416 Direct Memory Access and Scatter Gather* Product Specification for a description of the service. This IPIF parameter is assigned to the **C_DMA_LENGTH_WIDTH** parameter of the DMA/SG service. This parameter is only used if the DMA Service is activated in the IPIF via the **C_ARD_ID_ARRAY** and a Scatter Gather channel type is specified in the **C_DMA_CHAN_TYPE_ARRAY** parameter.

C_DMA_PKT_LEN_FIFO_ADDR_ARRAY

This parameter array applies directly to the DMA/SG service within the IPIF. Please refer to the Xilinx LogiCORE *DS-416 Direct Memory Access and Scatter Gather* Product Specification for a description of the service. This IPIF parameter is assigned to the **C_LEN_FIFO_ADDR** parameter of the DMA/SG service. This parameter is only used if the DMA Service is activated in the IPIF via the **C_AR**

D_ID_ARRAY and a Scatter Gather channel type is specified in the **C_DMA_CHAN_TYPE_ARRAY** parameter.

C_DMA_PKT_STAT_FIFO_ADDR_ARRAY

This parameter array applies directly to the DMA/SG service within the IPIF. Please refer to the Xilinx LogiCORE *DS-416 Direct Memory Access and Scatter Gather* Product Specification for a description of the service. This IPIF parameter is assigned to the **C_STAT_FIFO_ADDR** parameter of the DMA/SG service. This parameter is only used if the DMA Service is activated in the IPIF via the **C_ARD_ID_ARRAY** and a Scatter Gather channel type is specified in the **C_DMA_CHAN_TYPE_ARRAY** parameter.

C_DMA_INTR_COALESCE_ARRAY]

This parameter array applies directly to the DMA/SG service within the IPIF. Please refer to the Xilinx LogiCORE *DS-416 Direct Memory Access and Scatter Gather* Product Specification for a description of the service. This IPIF parameter is assigned to the **C_INTR_COALESCE** parameter of the DMA/SG service. This parameter is only used if the DMA Service is activated in the IPIF via the **C_ARD_ID_ARRAY** and a Scatter Gather channel type of 2 or 3 is specified in the **C_DMA_CHAN_TYPE_ARRAY** parameter.

C_DMA_BURST_SIZE

This integer parameter applies directly to the DMA/SG service within the IPIF. Please refer to the Xilinx LogiCORE *DS-416 Direct Memory Access and Scatter Gather* Product Specification for a description of the service. This IPIF parameter is assigned a value that is a power of two between one and 16, inclusively. It communicates to the DMA controller the number of data-word transfers that will make up a burst. If set to one, this parameter effectively disables bursting by the DMA controller.

C_DMA_SHORT_BURST_REMAINDER

This integer parameter applies directly to the DMA/SG service within the IPIF. Please refer to the Xilinx LogiCORE *DS-416 Direct Memory Access and Scatter Gather* Product Specification for a description of the service. This parameter is used to determine how data is transferred if there is no longer sufficient data to use the burst size specified by the previous parameter. Two values are allowed. If set to 0, the parameter causes the completion of the DMA operation to occur as a series of single-beat transactions. If set to 1, the parameter causes the completion of the DMA operation to occur as a "short burst", just long enough to transfer the remaining data.

C_DMA_PACKET_WAIT_UNIT_NS

This parameter array applies directly to the DMA/SG service within the IPIF. Please refer to the Xilinx LogiCORE *DS-416 Direct Memory Access and Scatter Gather* Product Specification for a description of the service. This IPIF parameter is assigned to the **C_PACKET_WAIT_UNIT_NS** parameter of the DMA/SG service. This parameter is only used if the DMA Service is activated in the IPIF via the **C_ARD_ID_ARRAY** and a Scatter Gather channel type of 2 or 3 is specified in the **C_DMA_CHAN_TYPE_ARRAY** parameter.

C_OPB_AWIDTH

This integer parameter is used by the OPB IPIF to size OPB address related components within the IPIF. This value should be set to 32, the current address width of the OPB.

C_OPB_DWIDTH

This integer parameter is used by the OPB IPIF to size OPB data bus related components within the IPIF. This value should be set to 32, the current data width of the OPB.

C_OPB_CLK_PERIOD_PS

This integer parameter specifies the period of the OPB clock in picoseconds. It is used by the DMA/SG service for timing packet-wait bounds (if interrupt coalescing is optioned in). Please refer to the Xilinx LogiCORE *DS-416 Direct Memory Access and Scatter Gather* Product Specification if more detail is desired.

C_IPIF_DWIDTH

This integer parameter is used by the OPB IPIF to size IPIF data bus related components within the IPIF and it's interface to the User IP. This parameter is included for future IPIF optimization. It should be set equal to the OPB data bus width which is 32.

C_FAMILY

This parameter is defined as a string. It specifies the target FPGA technology for implementation of the OPB IPIF. This parameter is required by the Packet FIFO designs which utilize Xilinx BRAM primitives. The size and configuration of these primitives can vary from one FPGA technology family to another.

Table 8: **Xilinx Predefined Identifiers for FPGA Families.**

OPB IPIF Category	Xilinx Identifier	Defined VHDL type	Assigned Value	Family Description
Unsupported by this OPB IPIF	any	String	"any"	Any FPGA Family
	x4k	String	"x4k"	4K
	x4ke	String	"x4ke"	4K-E
	x4kl	String	"x4kl"	4K-L
	x4kex	String	"x4kex"	4K-EX
	x4kxl	String	"x4kxl"	4k-XL
	x4kxv	String	"x4kxv"	4K-XV
	x4kxla	String	"x4kxla"	4K-XLA
	spartan	String	"spartan"	Spartan
	spartanxl	String	"spartanxl"	Spartan-XL
	spartan2	String	"spartan2"	Spartan-II
	spartan2e	String	"spartan2e"	Spartan-IIE
OPB IPIF Supported Families	spartan3	String	"spartan3"	Spartan-III
	virtex	String	"virtex"	Virtex
	virtexe	String	"virtexe"	Virtex-E
	virtex2	String	"virtex2"	Virtex-II
	virtex2p	String	"virtex2p"	Virtex-II Pro

Allowable Parameter Combinations

See individual parameter descriptions.

OPB IPIF I/O Signals

The OPB IPIF has two major interfaces. These are the OPB Bus and the IP Interconnect (IPIC). The device also generates an interrupt for use by a processing element. The I/O signals for the design are listed in Table 9. The interfaces referenced in this table are shown in Figure 1 in the OPB IPIF block diagram.

Table 9: **OPB IPIF I/O Signals**

Port	Signal Name	Interface	I/O	Description
		OPB System Signals		
P1	OPB_clk	OPB	I	OPB main bus clock. See table note 1.
P2	Reset	OPB	I	OPB main bus reset. See table note 1.
P3	Freeze	OPB	I	Reserved for debug freeze signal.
		OPB Slave Request Signals		
P4	OPB_ABus(0:**C_OPB_AWIDTH**-1)	OPB	I	See table note 1.
P5	OPB_DBus(0:**C_OPB_DWIDTH**-1)	OPB	I	See table note 1.

Table 9: **OPB IPIF I/O Signals** ***(Contd)***

Port	Signal Name	Interface	I/O	Description
P6	OPB_BE(0:[**C_OPB_DWIDTH**/8]-1)	OPB	I	See table note 1.
P7	OPB_Select	OPB	I	See table note 1.
P8	OPB_RNW	OPB	I	See table note 1.
P9	OPB_seqAddr	OPB	I	See table note 1.
		OPB Slave Reply Signals		
P10	Sln_DBus(0:**C_OPB_DWIDTH**-1)	OPB	O	See table note 1.
P11	Sln_xferAck	OPB	O	See table note 1.
P12	Sln_errAck	OPB	O	See table note 1.
P13	Sln_retry	OPB	O	See table note 1.
P14	Sln_toutSup	OPB	O	See table note 1.
		OPB Master Request Signals		
P15	Mn_request	OPB	O	See table note 1.
P16	Mn_busLock	OPB	O	See table note 1.
P17	Mn_ABus(0:**C_OPB_AWIDTH**-1)	OPB	O	See table note 1.
P18	Mn_BE(0:**C_OPB_DWIDTH**/8-1)	OPB	O	See table note 1.
P19	Mn_select	OPB	O	See table note 1.
P20	Mn_RNW	OPB	O	See table note 1.
P21	Mn_seqAddr	OPB	O	See table note 1.
		OPB Master Reply Signals		
P22	OPB_MnGrant	OPB	I	See table note 1.
P23	OPB_xferAck	OPB	I	See table note 1.
P24	OPB_errAck	OPB	I	See table note 1.
P25	OPB_retry	OPB	I	See table note 1.
P26	OPB_timeout	OPB	I	See table note 1.
		Interrupt Controller Device Interrupt Output		
P27	IP2INTC_Irpt	System Interrupt Controller	O	Device interrupt output to microprocessor interrupt input or system interrupt controller. Registered level type, asserted active high.
		IPIC System Signals		
P28	IP2Bus_Clk **(2)**	User IP	I	Input clock from the IP. This is currently not used by any IPIF services.
P29	Bus2IP_Clk	User IP	O	IPIC clock pass-through from OPB_Clk.
P30	Bus2IP_Reset	User IP	O	Active high reset for use by the User IP.
P31	Bus2IP_Freeze	User IP	O	Active high signal requesting an operational freeze of the User IP.

Table 9: **OPB IPIF I/O Signals** ***(Contd)***

Port	Signal Name	Interface	I/O	Description
IPIC User Interrupt Inputs				
P32	IP2Bus_IntrEvent(0:**C_IP_ INTR_MODE_ARRAY**'length -1)	User IP	I	User IP interrupt signals to be captured in the IPIF IP ISC. The number and capture properties are set by the C_IP_INTR_MODE_ARRAY VHDL Generic.
IPIC IP Slave Interface Request and Qualifier Signals				
P33	Bus2IP_Data(0:**C_IPIF_ DWIDTH**-1)	User IP	O	Write data bus to the User IP. Write data is accepted by the IP during a write operation by assertion of the IP2Bus_WrAck signal and the rising edge of the Bus2IP_Clk.
P34	Bus2IP_Addr(0:**C_ OPB_AWIDTH**-1)	User IP	O	Address bus indicating the desired address of the requested read or write operation.
P35	Bus2IP_RNW	User IP	O	This signal indicates the sense of a requested operation with the User IP. High is a read, low is a write.
P36	Bus2IP_BE(0:(**C_IPIF_ DWIDTH**/8)-1)	User IP	O	Byte enable qualifiers for the requested read or write operation with the User IP. Bit 0 corresponds to Byte lane 0, Bit 1 to Byte lane 1, and so on.
P37	Bus2IP_Burst	User IP	O	Active high signal indicating that the active read or write operation with the User IP is utilizing bursting protocol. This signal is asserted for each data beat of the burst transfer.
P38	Bus2IP_WrReq	User IP	O	Active high signal indicating the initiation of a write operation with the IP. It is asserted for 1 Bus2IP_Clk during single data beat transactions and remains high to completion on burst write operations.
P39	Bus2IP_RdReq	User IP	O	Active high signal indicating the initiation of a read operation with the IP. It is asserted for 1 Bus2IP_Clk during single data beat transactions and remains high to completion on burst write operations.
IPIC IP Slave Interface Address Decode Service				
P40	Bus2IP_CS(0:**C_ARD_ID_ ARRAY** 'length - 1)	User IP	O	Active High chip select bus. Each bit of the bus corresponds to an entry in the C_ARD_ID_ARRAY. Assertion of a chip select indicates a active transaction request to the chip select's target address space.
P41	Bus2IP_CE(0: see note 3)	User IP	O	Active high chip enable bus. Chip enables are assigned per the Users entries in the C_ARD_NUM_CE_ARRAY. Chip enables are asserted during active transaction requests with the target address space and in conjunction with the corresponding sub-address within the space.

Table 9: **OPB IPIF I/O Signals *(Contd)***

Port	Signal Name	Interface	I/O	Description
P42	Bus2IP_RdCE(0: see note 3)	User IP	O	Active high chip enable bus. Chip enables are assigned per the User's entries in the C_ARD_NUM_CE_ARRAY. These chip enables are asserted only during active read transaction requests with the target address space and in conjunction with the corresponding sub-address within the space.
P43	Bus2IP_WrCE(0: see note 3)	User IP	O	Active high chip enable bus. Chip enables are assigned per the Users entries in the C_ARD_NUM_CE_ARRAY. These chip enables are asserted only during active write transaction requests with the target address space and in conjunction with the corresponding sub-address within the space.
IPIC IP Slave Interface Reply Signals				
P44	IP2Bus_Data(0:**C_ IPIF_DWIDTH** -1)	User IP	I	Input Read Data bus from the User IP. Data is qualified with the assertion of IP2Bus_RdAck signal and the rising edge of the Bus2IP_Clk.
P45	IP2Bus_WrAck	User IP	I	Active high Write Data qualifier. Write data on the Bus2IP_Data Bus is deemed accepted by the User IP at the rising edge of the Bus2IP_Clk and IP2Bus_WrAck asserted high by the User IP.
P46	IP2Bus_RdAck	User IP	I	Active high read data qualifier. Read data on the IP2Bus_Data Bus is deemed valid at the rising edge of Bus2IP_Clk and the assertion of the IP2Bus_RdAck signal by the User IP.
P47	IP2Bus_Retry	User IP	I	Active high signal indicating the User IP is requesting a retry of an active operation.
P48	IP2Bus_Error	User IP	I	Active high signal indicating the User IP has encountered an error with the requested operation. This signal is asserted in conjunction with IP2Bus_RdAck or the IP2Bus_WrAck.
P49	IP2Bus_ToutSup	User IP	I	Active high signal requesting suppression of the transaction time-out function in the IPIF for the active read or write operation.
P50	IP2Bus_PostedWrInh	User IP	I	Active high signal requesting full handshake transfer protocol for all write transactions to the User IP. This is generally used to slow down writes into a User IP FIFO if it is nearing a possible overrun condition.
IPIC IP to DMA Support				
P51	IP2DMA_RxLength_Empty	User IP	I	Active high signal indicating that the User IP's RxLength FIFO is empty [(4)]

Table 9: **OPB IPIF I/O Signals *(Contd)***

Port	Signal Name	Interface	I/O	Description
P52	IP2DMA_RxStatus_Empty	User IP	I	Active high signal indicating that the User IP's RxStatus FIFO is empty **(4)**
P53	IP2DMA_TxLength_Full	User IP	I	Active high signal indicating that the User IP's TxLength FIFO is full **(4)**
P54	IP2DMA_TxStatus_Empty	User IP	I	Active high signal indicating that the User IP's TxStatus FIFO is empty **(4)**
IPIC IP Master Request				
P55	IP2Bus_Addr(0:**C_OPB_AWIDTH**-1)	User IP	I	Address bus from the User IP used to convey the desired address to be output on the OPB Bus during IP Master read or write operations. The address value must be valid during the assertion of the IP2Bus_MstWrReq or IP2Bus_MstRdReq signal.
P56	IP2Bus_MstBE(0:(**C_IPIF_DWIDTH**/8) -1)	User IP	I	Byte Enable bus from the User IP used to convey the desired Byte Enables to be output on the OPB Bus during IP Master single data beat read or write operations. The byte-enable value must be valid during the assertion of the IP2Bus_MstWrReq or IP2Bus_MstRdReq signal.
P57	IP2IP_Addr(0:**C_OPB_AWIDTH**-1)	User IP	I	Address bus from the User IP used to convey the desired address to be output on the IPIF Local Bus during IP Master read or write operations. The address value must be valid during the assertion of the IP2Bus_MstWrReq or IP2Bus_MstRdReq signal.
P58	IP2Bus_MstWrReq	User IP	I	Active high signal initiating a Master write transaction.
P59	IP2Bus_MstRdReq	User IP	I	Active high signal initiating a Master read transaction.
P60	IP2Bus_MstBurst	User IP	I	Active high signal indicating that the corresponding Master transaction be completed using burst protocol. This IPIF implementation supports IP Master bursts only of fixed-size eight.
P61	IP2Bus_MstBusLock	User IP	I	Active high signal requesting a OPB Bus Lock assertion during the corresponding Master transaction request.
IPIC IP Status Reply to IP Master				
P62	Bus2IP_MstWrAck	User IP	O	Active high signal indicating that a write data beat of the requested write transaction will be completed at the next rising edge of the Bus2IP_Clk.

Table 9: **OPB IPIF I/O Signals *(Contd)***

Port	Signal Name	Interface	I/O	Description
P63	Bus2IP_MstRdAck	User IP	O	Active high signal indicating that a read data beat of the requested read transaction will be completed at the next rising edge of the Bus2IP_Clk.
P64	Bus2IP_MstRetry	User IP	O	Active high signal indicating that the Master is requesting a retry on the active Master transaction request.
P65	Bus2IP_MstError	User IP	O	Active high signal indicating that the Master has encountered an error on the active Master transaction request.
P66	Bus2IP_MstTimeout	User IP	O	Active high signal indicating that the Master has encountered a bus time-out on the active Master transaction request.
P67	Bus2IP_MstLastAck	User IP	O	Active high signal. When concurrent with ’MstWrAck’ or ’MstRdAck’, indicates that the last data beat (either read or write) of the requested transaction will be completed at the next rising edge of the Bus2IP_Clk. When concurrent with 'Bus2IP_MstRetry', indicates that there is no data buffered in the IPIF that could not be stored before the retry is reported. When concurrent with ’Bus2IP_MstError’ indicates an error completion of the master transaction.
P68	Bus2IP_IPMstTrans	User IP	O	Active high signal indicating that the current read or write operation on the User IP’s slave port (with the IPIF) is being initiated by the User IP’s Master transaction request interface.
IPIC Read Packet FIFO Signals				
P69	IP2RFIFO_Data(0: See note 6)	User IP	I	Write data from the User IP to the RdFIFO write port is transmitted on this bus. Data present on the bus is written when IP2RFIFO_WrReq is high, RFIFO2IP_WrAck is high, and a rising edge of the Bus2IP_Clk occurs. [(7)]
P70	IP2RFIFO_WrReq	User IP	I	Active high signal indicating that the IP is attempting to write the data on the IP2RFIFO_Data bus to the User IP side of the RdFIFO. The transaction is not completed until the RdFIFO responds with an active high assertion on the RFIFO2IP_WrAck signal and a corresponding rising edge of the Bus2IP_Clk signal occurs. [(7)]
P71	IP2RFIFO_WrMark	User IP	I	Active high signal commanding the RdFIFO to perform a "Mark" operation. [(7)]
P72	IP2RFIFO_WrRelease	User IP	I	Active high signal commanding the RdFIFO to perform a "Release" operation. [(7)]

Table 9: **OPB IPIF I/O Signals *(Contd)***

Port	Signal Name	Interface	I/O	Description
P73	IP2RFIFO_WrRestore	User IP	I	Active high signal commanding the RdFIFO to perform a "Restore" operation. [(7)]
P74	RFIFO2IP_WrAck	User IP	O	Active high signal indicating that the data write request will complete at the next rising edge of the Bus2IP_Clk signal. [(7)]
P75	RFIFO2IP_AlmostFull	User IP	O	Active high signal indicating that the RdFIFO can accept only one more data write. [(7)]
P76	RFIFO2IP_Full	User IP	O	Active high signal indicating that the RdFIFO is full and cannot accept data. The RFIFO2IP_WrAck signal assertion will be suppressed until the FIFO is no longer full. [(7)]
P77	RFIFO2IP_Vacancy(0: See note 8)	User IP	O	Status bus indicating the available locations for writing in the RdFIFO. [(7)]
IPIC Write Packet FIFO Signals				
P78	WFIFO2IP_Data(0: See note 6)	User IP	O	Read data from the WrFIFO to the User IP is transmitted on this bus. Data present on the bus is valid when IP2WFIFO_RdReq is high, WFIFO2IP_RdAck is high, and a rising edge of the Bus2IP_Clk occurs. [(7)]
P79	IP2WFIFO_RdReq	User IP	I	Active high signal indicating that the IP is attempting to read data from the WrFIFO. The transaction is not completed until the WrFIFO responds with an active high assertion on the WFIFO2IP_RdAck signal and a corresponding rising edge of the Bus2IP_Clk signal occurs. [(7)]
P80	IP2WFIFO_RdMark	User IP	I	Active high signal commanding the WrFIFO to perform a "Mark" operation. [(7)]
P81	IP2WFIFO_RdRelease	User IP	I	Active high signal commanding the WrFIFO to perform a "Release" operation. [(7)]
P82	IP2WFIFO_RdRestore	User IP	I	Active high signal commanding the WrFIFO to perform a "Restore" operation. [(7)]
P83	WFIFO2IP_RdAck	User IP	O	Active high signal asserted in response to a User IP read request of the WrFIFO. Data on the WFIFO2IP_Data bus is valid for reading when this signal is asserted in conjunction with the rising edge of the Bus2IP_Clk. [(7)]
P84	WFIFO2IP_AlmostEmpty	User IP	O	Active high signal indicating that the WrFIFO can provide only one more data read. [(7)]
P85	WFIFO2IP_Empty	User IP	O	Active high signal indicating that the WrFIFO is empty and cannot provide data. The WFIFO2IP_RdAck signal assertion will be suppressed until the FIFO is no longer empty. [(7)]
P86	WFIFO2IP_Occupancy(0: See Note 8)	User IP	O	Status bus indicating the available locations for reading in the WrFIFO. [(7)]

Table 9: **OPB IPIF I/O Signals *(Contd)***

Port	Signal Name	Interface	I/O	Description
		IPIC Auxiliary DMA Support Signals		
P87	IP2Bus_DMA_Req [(2)]	User IP	I	Reserved for future growth.
P88	Bus2IP_DMA_Ack [(2)]	User IP	O	Reserved for future growth.

Notes:

1. This signal's function and timing is defined in the IBM® **64-Bit On-Chip Peripheral Bus Architecture Specification Version 2.1.**
2. Greyed-out signals are not used by the OPB IPIF design but are reserved for future support.
3. The size of the Bus2IP_CE, Bus2IP_RdCE, and the Bus2IP_WrCE buses is the sum of the integer values entered in the **C_ARD_NUM_CE_ARRAY**.
4. Scatter/Gather compliancy requires the User IP to include a TxLength FIFO and TxStatus FIFO for Tx DMA/SG channel interfaces and an Rx Length FIFO and an Rx Status FIFO for Rx DMA/SG Channel interfaces.
5. The left-most bit index of the IP2BUS_MstNum bus is equal to log2(**C_DEV_MAX_BURST_SIZE**/(**C_OPB_DWIDTH**/8)).
6. When included in the IPIF implementation, the width of the FIFO data port is set by the User via the **C_ARD_DEPENDENT_PROPS_ARRAY**. If the FIFO is not included in the implementation, the port defaults to 32 bits wide.
7. The behavior of the Read Packet FIFO and the Write Packet FIFO is discussed in the Product Specification **DS415 On-Chip Peripheral Bus IP Interface Packet FIFO**.
8. The left-most bit index of the RFIFO2IP_Vacancy and WFIFO2IP_Occupancy bus is determined by log2(FIFO Depth). The FIFO Depth is a separate User parameter for the RdFIFO and the WrFIFO. The depths are set in the C_ARD_DEPENDENT_PROPS_ARRAY.

Parameter - Port Dependencies

The OPB IPIF parameterization has effects on many of it's I/O port sizes. These are indicated in the port definitions presented in Table 9. There are cases where IPIF ports are rendered unused based on parameter settings. These dependencies are summarized in Table 10. Unused IPIF Input Ports need to be tied to logic 0.

Table 10: **Parameter/Port Dependencies**

Port	Name or Group Description	Depends on Parameter	Relationship Description
P15 thru P26	OPB IPIF Master Interface to OPB	G1, G13	The OPB IPIF Master port signals are used only if the ID '**IPIF_DMA_SG**' is present in the **C_ARD_ID_ARRAY** parameter or if the **C_IP_MASTER_PRESENT** parameter is set to 1.
P32	IP2Bus_IntrEvent	G1	If the ID '**IPIF_INTR**' is not present in the **C_ARD_ID_ARRAY** parameter, then input IP2Bus_IntrEvent(0) is connected directly to output port Dev_Intr_Out (**P71**).
P51	IP2DMA_RxLength_ Empty	**G1**, **G15**	This port signal is used only if the ID '**IPIF_DMA_SG**' is present in the **C_ARD_ID_ARRAY** parameter and a value of 3 is present in the **C_DMA_CHAN_TYPE_ARRAY** parameter.
P52	IP2DMA_RxStatus_ Empty	**G1**, **G15**	This port signal is used only if the ID '**IPIF_DMA_SG**' is present in the **C_ARD_ID_ARRAY** parameter and a value of 3 is present in the **C_DMA_CHAN_TYPE_ARRAY** parameter.
P53	IP2DMA_TxLength_ Full	**G1**, **G15**	This port signal is used only if the ID '**IPIF_DMA_SG**' is present in the **C_ARD_ID_ARRAY** parameter and a value of 2 is present in the **C_DMA_CHAN_TYPE_ARRAY** parameter.

Table 10: **Parameter/Port Dependencies *(Contd)***

Port	Name or Group Description	Depends on Parameter	Relationship Description
P54	IP2DMA_TxStatus_ Empty	**G1**, **G15**	This port signal is used only if the ID ’**IPIF_DMA_SG**’ is present in the **C_ARD_ID_ARRAY** parameter and a value of 2 is present in the **C_DMA_CHAN_TYPE_ARRAY** parameter.
P55 thru P68	IP Master Interface Ports	G13	The IP Master ports are only used if the **C_IP_MASTER_PRESENT** parameter is set to 1.
P69 thru P77	RdFIFO Interface Ports	G1	These ports are used only if the IDs ’**IPIF_RDFIFO_REG**’ and ’**IPIF_RDFIFO_DATA**’ are present in the **C_ARD_ID_ARRAY** parameter.
P71 thru P73	RdFIFO Interface Packet Mode Control Ports	**G1**, **G5**	These ports are used only if the IDs ’**IPIF_RDFIFO_REG**’ and ’**IPIF_RDFIFO_DATA**’ are present in the **C_ARD_ID_ARRAY** parameter and Packet Mode Features are enabled for the RdFIFO via the setting in **C_ARD_DEPENDENT_PROPS_ARRAY**.
P78 thru P86	WrFIFO Interface Ports	G1	These ports are used only if the IDs ’**IPIF_WRFIFO_REG**’ and ’**IPIF_WRFIFO_DATA**’ are present in the **C_ARD_ID_ARRAY** parameter.
P80 thru P82	WrFIFO Interface Packet Mode Control Ports	**G1**, **G5**	These ports are used only if the IDs ’**IPIF_WRFIFO_REG**’ and ’**IPIF_WRFIFO_DATA**’ are present in the **C_ARD_ID_ARRAY** parameter and Packet Mode Features are enabled for the WrFIFO via the setting in **C_ARD_DEPENDENT_PROPS_ARRAY**.

Register-Interface Descriptions for OPB IPIF Services

The following section discusses the User Application interface to the various service registers provided by the IPIF. Depending on the User Parameter assignments, these registers may or may not exist in the User’s implementation

.

Table 11: **OPB IPIF Services Register Summary**

IPIF Service Name	Register Name	OPB Address Offset from Service's Base Address Assignment	Allowed Access	Optioning Parameter(s)
OPB IPIF Interrupt Service	**Device Interrupt Source Controller**			
	Device Interrupt Status Register	+ 0x0	Read/ Toggle on Write[1]	**G1-G5** 'AND' **G10**
	Device Interrupt Pending Register	+ 0x4	Read	**G1-G5** 'AND' **G10**
	Device Interrupt Enable Register	+ 0x8	Read/Write	**G1-G5** 'AND' **G10**
	Device Interrupt ID Register (Priority Encoder)	+ 0x18	Read	**G1-G5** 'AND' **G10** 'AND' **G11**
	Device Global Interrupt Enable Register	+ 0x1C	Read/Write	G1-G5
	IP Interrupt Source Controller			
	IP Interrupt Status Register	+ 0x20	Read/ Toggle on Write[1]	**G1-G5** 'AND' **G12**
	IP Interrupt Enable Register	+ 0x28	Read/Write	**G1-G5** 'AND' **G12**
OPB IPIF Reset/MIR Service	IPIF Software Reset Register	+ 0x0	Write[2]	G1-G5
	IPIF Module Identification Register	+ 0x0	Read[2]	**G1-G5** 'AND' **G7** (**G8** provides Block ID)
OPB IPIF Read Packet FIFO Register Service	RdFIFO Reset	+ 0x0	Write[2]	G1-G5
	RdFIFO Module Identification Register	+ 0x0	Read[2]	G1-G5
	RdFIFO Status	+ 0x4	Read	G1-G5
OPB IPIF Read Packet FIFO Data Service	RdFIFO Data	+ 0x0	Read	G1-G5
OPB IPIF Write Packet FIFO Register Service	WrFIFO Reset	+ 0x0	Write[2]	G1-G5
	WrFIFO Module Identification Register	+ 0x0	Read[2]	G1-G5
	WrFIFO Status	+ 0x4	Read	G1-G5
OPB IPIF Write Packet FIFO Data Service	WrFIFO Data	+ 0x0	Read	G1-G5

Table 11: **OPB IPIF Services Register Summary**

IPIF Service Name	Register Name	OPB Address Offset from Service's Base Address Assignment	Allowed Access	Optioning Parameter(s)
OPB IPIF DMA Service (one register set per channel, 2 channels allowed numbered 0 to1)	DMA & Scatter Gather Module Identification Record / Channel Reset	+ (chan_num*64) + 0x0	Read/ Write[2]	**G1-G5** 'AND' **G15-G22**
	DMA & Scatter Gather Control Register	+ (chan_num*64) + 0x4	Read/Write	**G1-G5** 'AND' **G15-G22**
	DMA Source Address	+ (chan_num*64) + 0x8	Read/Write	**G1-G5** 'AND' **G15-G22**
	DMA Destination Address	+ (chan_num*64) + 0xC	Read/Write	**G1-G5** 'AND' **G15-G22**
	DMA Start/Length	+ (chan_num*64) + 0x10	Read/Write	**G1-G5** 'AND' **G15-G22**
	DMA/SG Status Register	+ (chan_num*64) + 0x14	Read	**G1-G5** 'AND' **G15-G22**
OPB IPIF Scatter Gather Service	SG Buffer Descriptor Address	+ (chan_num*64) + 0x18	Read/Write	**G1-G5** 'AND' **G15-G22**
	SG Software Control Register	+ (chan_num*64) + 0x1C	Read/Write	**G1-G5** 'AND' **G15-G22**
	SG Unserviced Packet Count	+ (chan_num*64) + 0x20	Read/Write	**G1-G5** 'AND' **G15-G22**
	SG Gather Packet Count Threshold	+ (chan_num*64) + 0x24	Read/Write	**G1-G5** 'AND' **G15-G22**
	SG Packet Wait Bound	+ (chan_num*64) + 0x28	Read/Write	**G1-G5** 'AND' **G15-G22**
DMA/SG Service Shared Interrupt Source Control	DMA/SG Interrupt Status Register	+ (chan_num*64) + 0x2C	Read/Toggle on Write	**G1-G5** 'AND' **G15-G22**
	DMA/SG Interrupt Enable Register	+ (chan_num*64) + 0x30	Read/Write	**G1-G5** 'AND' **G15-G22**
OPB IPIF DMA/SG Service Internal Processing Support	SG Packet Length (internal use; do not access)	+ (chan_num*64) + 0x3C	N/A Not for User access	**G1-G5** 'AND' **G15-G22**

Notes:

1. Toggle each bit position to which a "1" is written.
2. Reads the Module Identification Record. Reset if written with 0xA.

IPIF Interrupt Service Registers

The following section details the register compliment of the IPIF Interrupt Service. Parameterization of the IPIF will affect which registers are present and the composition of the bits within the registers that are present.

Device Interrupt Status Register (DISR)

The Device Interrupt Status Register shown in Figure 7 gives the interrupt status for the device (IPIF + User Interrupts). Each bit within this register represents a major function within the device. The bits are detailed in Table 12. This register is fixed at 32 bits wide and each utilized bit within the register is set to 1 whenever the corresponding interrupt input has met the interrupt capture criteria. Unlike the IP Interrupt Status Register, the interrupt capture mode for this register is fixed. The **TERR** bit is captured with a 'sample and hold high' mode. This simply means that if the input interrupt is sampled to be 1 at a rising edge of a OPB Clock pulse, the register bit is set to a 1 and 'held' until the User Interrupt Service Routine clears it to a 0. The remaining bits within the register (**IPIR**, **DMA0**, **DMA1**, **WFDL**, and **RFDL**) are pass through. Once asserted, they are 'held' by the source of the interrupt (IP ISR, DMA, or FIFO) and therefore an additional sample and hold operation is not necessary in this register. These interrupts must be cleared at the source function.

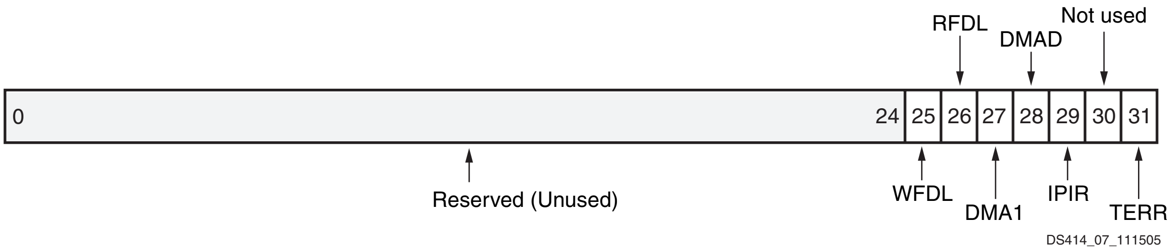


Figure 7: **Device Interrupt Status Register**

Table 12: **Device Interrupt Status Register Description**

Bit(s)	Name	Core Access	Reset Value	Description
31	TERR	Read/TOW (1)	'0'	**Transaction Error.** This interrupt indicates that a function within the IPIF or the User IP responded to a Read or Write transaction request with the assertion of the IP2Bus_Error Reply signal. • '0' = No transaction error detected. • '1' = Transaction Error detected.
30	Unused	Read	'0'	• **Unused.** Reserved.
29	IPIR	Read	'0'	**IP Interrupt Request**. This interrupt indicates that a User IP interrupt input on the IP2Bus_IntrEvent bus has been captured in the IP Interrupt Status Register and is enabled via the IP Interrupt Enable Register. • '0' = No enabled interrupt is captured • '1' = IP interrupt is captured and enabled.
28	DMA0	Read	'0'	**DMA CH0 Interrupt Request**. This interrupt indicates that DMA CH0 is asserting it's interrupt request. • '0' = No DMA CH0 Interrupt. • '1' = DMA CH0 interrupt asserted.
27	DMA1	Read	'0'	**DMA CH1 Interrupt Request**. This interrupt indicates that DMA CH1 is asserting it's interrupt request. • '0' = No DMA CH1 Interrupt. • '1' = DMA CH1 interrupt asserted.

Table 12: **Device Interrupt Status Register Description *(Contd)***

Bit(s)	Name	Core Access	Reset Value	Description
26	RFDL	Read	'0'	**Read FIFO Deadlock Interrupt Request**. This interrupt indicates that the Read FIFO is asserting it's Deadlock interrupt request. [2] • '0' = No Read FIFO Deadlock Interrupt. • '1' = Read FIFO Deadlock interrupt asserted.
25	WFDL	Read	'0'	**Write FIFO Deadlock Interrupt Request**. This interrupt indicates that the Write FIFO is asserting it's Deadlock interrupt request. [2] • '0' = No Write FIFO Deadlock Interrupt. • '1' = Write FIFO Deadlock interrupt asserted.
0-24	Unused	Read	zeroes	**Unused** bit positions. Reserved.

Notes:

1. TOW is defined as Toggle On Write. Writing a 1 to a bit position within the register causes the corresponding bit position in the register to 'toggle' state. This mechanism avoids the requirement on the User Interrupt Service routine to perform a Read/Modify/Write operation to clear a single bit within the register. Read Modify/Write operations can lead to inadvertent clearing of interrupts captured in the time period between the Read and the Write operations.
2. Deadlock is a condition that only occurs if the Read and Write FIFOs are in Packet Mode Operation. A Deadlock is defined as the FIFO being both Full and Empty at the same time. This can occur if a single packet size is bigger than the capacity of the FIFO. Sizing FIFO correctly avoids this condition.

Device Interrupt Pending Register (DIPR)

The Device Interrupt Pending Register shown in Figure 8 is a read-only value that is the logical AND of the Device Interrupt Status Register and the Device Interrupt Enable Register (see below) on a bit-by-bit basis. Therefore, the Interrupt Pending Register will report only captured interrupts that are also enabled by the corresponding bit in the Interrupt Enable Register.

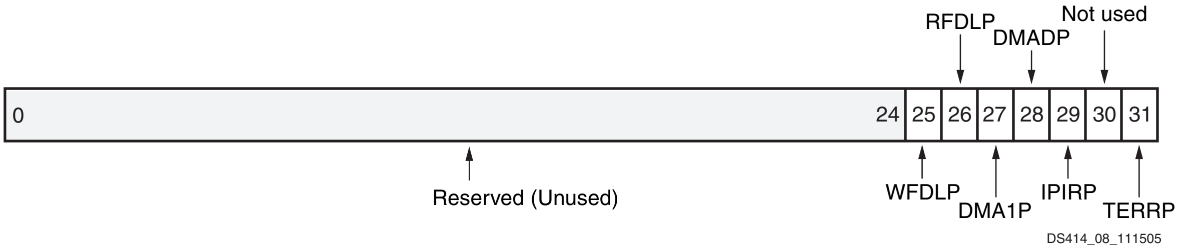


Figure 8: **Device Interrupt Pending Register**

Table 13: **Device Interrupt Pending Register Description**

Bit(s)	Name	Core Access	Reset Value	Description
31	TERRP	Read	'0'	Transaction Error Pending. This bit is the logical 'AND' of the TERR bit in the DISR and the corresponding bit in the DIER • '0' = No Transaction Error pending. • '1' = Transaction Error captured and enabled.
30	Unused	Read	'0'	• **Unused.** Reserved.
29	IPIRP	Read	'0'	**IP Interrupt Request Pending**. This bit is the logical 'AND' of the IPIR bit in the DISR and the corresponding bit in the DIER. • '0' = No IP interrupt pending • '1' = IP interrupt is pending.
28	DMA0P	Read	'0'	**DMA CH0 Interrupt Request Pending**. This bit is the logical 'AND' of the DMA0 bit in the DISR and the corresponding bit in the DIER. • '0' = No DMA CH0 Interrupt pending. • '1' = DMA CH0 interrupt pending.
27	DMA1P	Read	'0'	**DMA CH1 Interrupt Request Pending**. This bit is the logical 'AND' of the DMA1 bit in the DISR and the corresponding bit in the DIER. • '0' = No DMA CH1 Interrupt pending. • '1' = DMA CH1 interrupt pending.
26	RFDLP	Read	'0'	**Read FIFO Deadlock Interrupt Request Pending**. This bit is the logical 'AND' of the RFDL bit in the DISR and the corresponding bit in the DIER. • '0' = No Read FIFO Deadlock Interrupt pending. • '1' = Read FIFO Deadlock interrupt pending.
25	WFDLP	Read	'0'	**Write FIFO Deadlock Interrupt Request Pending**. This bit is the logical 'AND' of the WFDL bit in the DISR and the corresponding bit in the DIER. • '0' = No Write FIFO Deadlock Interrupt pending. • '1' = Write FIFO Deadlock interrupt pending.
0-24	Unused	Read	zeroes	**Unused** bit positions. Reserved.

Device Interrupt Enable Register (DIER)

The Device Interrupt Enable Register shown in Figure 9 determines which interrupt sources in the Device Interrupt Status Register are allowed to generate interrupts to the system.

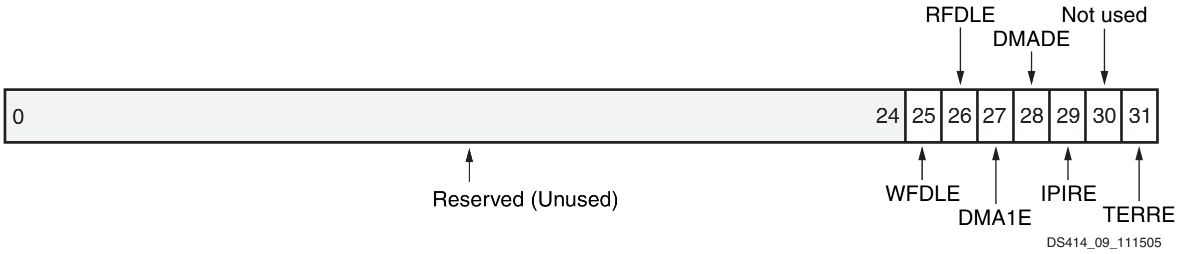


Figure 9: **Device Interrupt Enable Register**

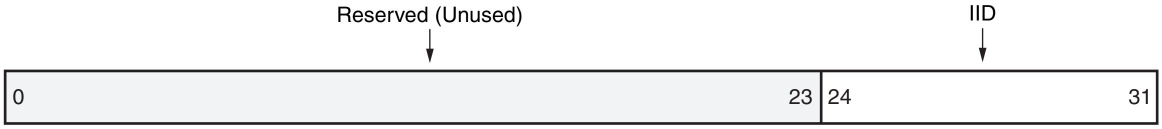
Table 14: **Device Interrupt Enable Register Description**

Bit(s)	Name	Core Access	Reset Value	Description
31	TERRE	Read/Write	'0'	Transaction Error Enable. This bit is the interrupt enable for the TERR bit in the DISR. • '0' = Mask Interrupt. • '1' = Enable Interrupt.
30	Unused	Read	'0'	• **Unused.** Reserved.
29	IPIRE	Read/Write	'0'	**IP Interrupt Request Enable**. This bit is the interrupt enable for the IPIR bit in the DISR. • '0' = Mask Interrupt. • '1' = Enable Interrupt.
28	DMA0E	Read/Write	'0'	DMA CH0 Interrupt Request Enable This bit is the interrupt enable for the DMA0 bit in the DISR. • '0' = Mask Interrupt. • '1' = Enable Interrupt.
27	DMA1E	Read/Write	'0'	**DMA CH1 Interrupt Request Enable**. This bit is the interrupt enable for the DMA1 bit in the DISR. • '0' = Mask Interrupt. • '1' = Enable Interrupt.
26	RFDLE	Read/Write	'0'	**Read FIFO Deadlock Interrupt Request Enable**. This bit is the interrupt enable for the RFDL bit in the DISR. • '0' = Mask Interrupt. • '1' = Enable Interrupt.
25	WFDLE	Read/Write	'0'	**Write FIFO Deadlock Interrupt Request Enable**. This bit is the interrupt enable for the WFDL bit in the DISR. • '0' = Mask Interrupt. • '1' = Enable Interrupt.
0-24	Unused	Read	zeroes	**Unused** bit positions. Reserved.

Device Interrupt ID Register (DIID)

The Device Interrupt ID Register shown in Figure 10 is an ordinal value output of a priority encoder. The value indicates which interrupt source, if any, has a pending interrupt. A value of 0x80 indicates that there are no pending interrupts, otherwise, the value gives the bit position in the DIPR of the highest priority interrupt that is pending.

The priority is highest for the interrupt bit in the LSB position (bit 31), which reports as ID value 0x00, and decreases in priority (and increases in the reported ID value) for each successively more significant position (i.e. going left).



DS414_10_111505

Figure 10: **Device Interrupt ID Register**

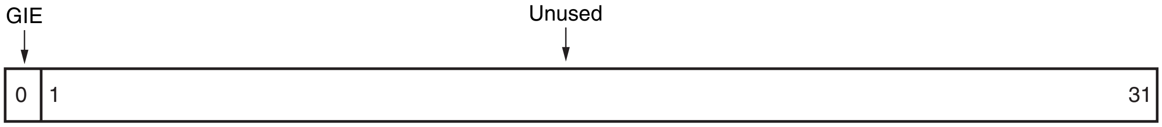
Table 15: **Device Interrupt ID Register Description**

Bit(s)	Name	Core Access	Reset Value	Description
0-23	Reserved	Read	Zeros	Reserved. Unused.
24-31	IID	Read	0x80	Interrupt ID. • 0x80 - The DIPR has no pending interrupts. • Otherwise - The ordinal ID of the highest-priority pending interrupt in the DIPR.

Device Global Interrupt Enable Register (DGIE)

The Device Global Interrupt Enable Register shown in Figure 11 has a single defined bit, in the high-order position, that is used to globally enable the final interrupt output from the IPIF Interrupt service to the Dev_Intr_Out (**P71**) output port.

If interrupts are globally disabled (the GIE it is set to 0, there will be no interrupt from the device under any circumstances. Otherwise, there may be interrupts generated for interrupt sources that are active and not disabled in the Device ISR and the IP ISR.



DS414_11_111505

Figure 11: **Device Global Interrupt Enable Register**

Table 16: **Device Global Interrupt Enable Register Description**

Bit(s)	Name	Core Access	Reset Value	Description
0	GIE	Read/Write	0	Global Interrupt Enable. • 0 - Interrupts disabled; no interrupt possible from this device. • 1 - Device Interrupt enabled.
1-31		Read	zeros	Unused

IP Interrupt Status Register (IPISR)

The IP Interrupt Status Register shown in Figure 12 is the interrupt capture register for the User IP. It is part of the IPIF Interrupt Service. The IPISR captures interrupts input from the User IP on the IP2Bus_IntrEvent (**P76**) input port. The number of active bits in the IPISR (and the capture mode for each) is determined by the User entries for the parameter C_IP_INTR_MODE_ARRAY (**G12**).

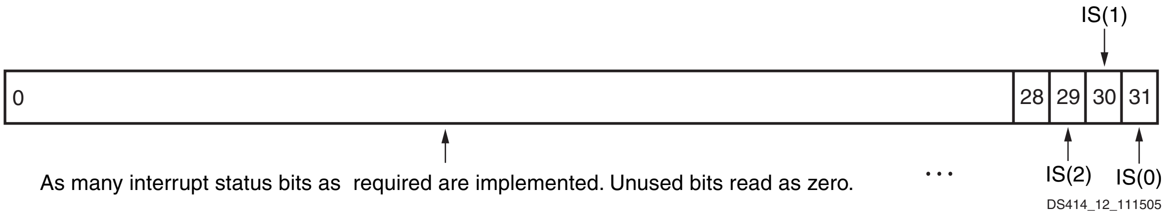

IS(1)
0
28
29
30
31
As many interrupt status bits as required are implemented. Unused bits read as zero.
. . .
IS(2)
IS(0)
DS414_12_111505

Figure 12: **IP Interrupt Status Register**

Table 17: **IP Interrupt Status Register Description**

Bit(s)	Name	Core Access	Reset Value	Description
0 thru 31	IS(i)	Read/TOW (1)	zeros	**Interrupt Status(i).** Used to signal an active interrupt condition reported by the IP via the IP2Bus_IntrEvent bus. Bits of IP2Bus_IntrEvent are assigned in increasing order, starting with 0, to decreasing bit positions in the status register, starting with 31. • 1 - active • 0 - not active

Notes:

1. TOW is defined as Toggle On Write. Writing a 1 to a bit position within the register causes the corresponding bit position in the register to 'toggle' state. This mechanism avoids the requirement on the User Interrupt Service routine to perform a Read/Modify/Write operation to clear a single bit within the register. Read Modify/Write operations can lead to inadvertent clearing of interrupts captured in the time period between the Read and the Write operations.

IP Interrupt Enable Register

The IP Interrupt Enable Register shown in Figure 13 has an enable bit for each defined bit of the IP Interrupt Status Register.

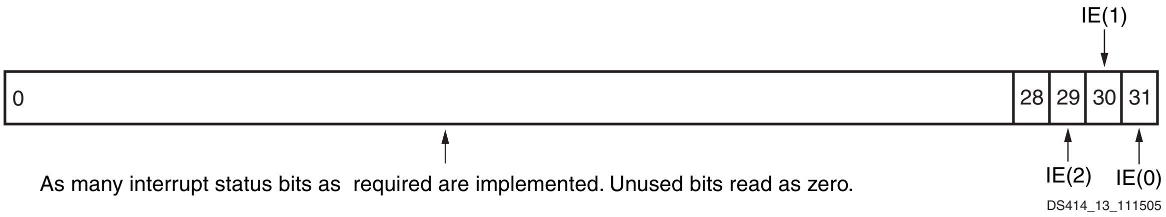

IE(1)
0
28
29
30
31
As many interrupt status bits as required are implemented. Unused bits read as zero.
IE(2)
IE(0)
DS414_13_111505

Figure 13: **IP Interrupt Enable Register**

Table 18: **IP Interrupt Enable Register Description**

Bit(s)	Name	Core Access	Reset Value	Description
0-31	IE(i)	Read/Write	zeros	**Interrupt Enable(i).** • 1 - enabled • 0 - disabled

IPIF Software Reset/MIR Service Registers

The section details the register compliment for the IPIF Software Reset and Module Information Register Service. Parameterization of the IPIF will affect which registers are present.

IPIF Software Reset Register

The Software Reset Register shown in Figure 14 is not an actual register, but rather, a write-only address, which when written with a specific value, generates a reset for the device.

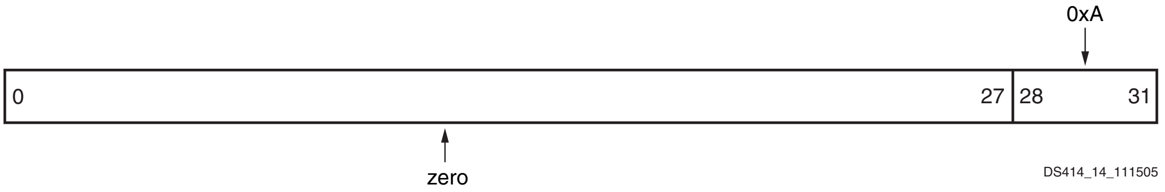


Figure 14: **IPIF Software Reset Register**

IPIF Software Reset Register

Table 19: **IPIF Software Reset Register Description**

Bit(s)	Name	Core Access	Reset Value	Description
0-31	Reset Key	Write	n/a	**Reset write value.** • ’0x0000000A’ - Generate a device reset (the implementation is allowed to check just the four least-significant bits for the value 0xA). • others - No effect.

IPIF Module Identification Register

The Module Identification Register, Figure 15, is a read-only value that can be used by software to verify the system addressability of the IPIF and to verify that the system-unique Block Identifier and the IPIF version, as reported in the fields of the Module Information Register, are as expected. This information can be used to verify the correctness of a system build.

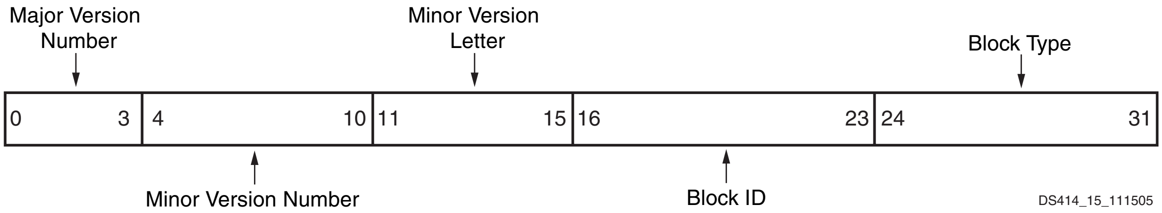


Figure 15: **IPIF Module Identification Register**

Table 20: **IPIF Module Identification Register Description**

Bit(s)	Name	Core Access	Reset Value	Description
0-3	Major Version Number	Read	n/a	Module Major Version Number.
4-10	Minor Version Number	Read	n/a	Module Minor Version Number.
11-15	Minor Version Letter	Read	n/a	Module Minor Version Letter. Letters a-z are encoded as 00000 - 11001
16-23	Block ID	Read	n/a	Module Block ID.
24-31	Block Type	Read	n/a	Module Block Type.

IPIF Read FIFO Service Registers

The following section details the register compliment of the IPIF Read FIFO Service. Parameterization of the IPIF will affect whether these registers are present or not.

RdFIFO Reset Register

A reset of the RdFIFO occurs when this register address is written with the value 0x0000000A. See Figure 16. This reset re-initializes the RdFIFO logic. Previously stored data in the FIFO is lost.

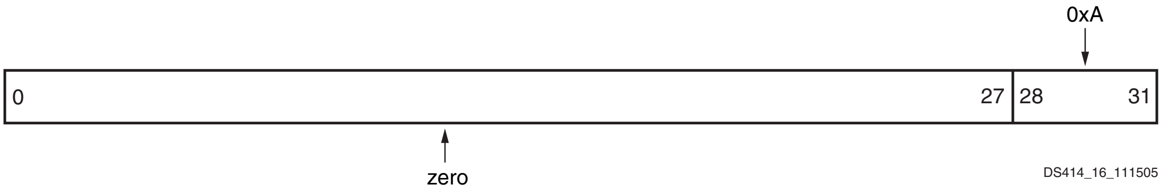


Figure 16: **RdFIFO Reset Register**

Table 21: **RdFIFO Reset Register Description**

Bit(s)	Name	Core Access	Reset Value	Description
0-31		Write	n/a	**Reset write value.** • ’0x0000000A’ - Generate a FIFO reset (the implementation is allowed to check just the four least-significant bits for the value 0xA). • others - No effect.

RdFIFO Module Identification Register

The RdFIFO Module Identification Register shown in Figure 17 is a read-only value that can be used by software to verify the addressability of the RdFIFO and to verify that the system-unique Block Identifier and the RdFIFO version, as reported in the fields of the Module Information Register, are as expected. This information can be used to verify the correctness of a system build.

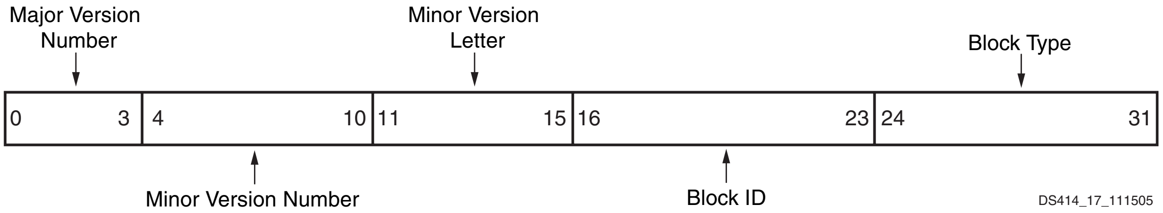


Figure 17: **RdFIFO Module Identification Register**

Table 22: **RdFIFO Model Identification Register Description**

Bit(s)	Name	Core Access	Reset Value	Description
0-3	Major Version Number	Read	n/a	Module Major Version Number.
4-10	Minor Version Number	Read	n/a	Module Minor Version Number.
11-15	Minor Version Letter	Read	n/a	Module Minor Version Letter. Letters a-z are encoded as 00000 - 11001
16-23	Block ID	Read	n/a	Module Block ID.
24-31	Block Type	Read	n/a	Module Block Type.

RdFIFO Status Register

The RdFIFO Status Register shown in Figure 18 is a read-only register that gives the occupancy status of the RdFIFO.

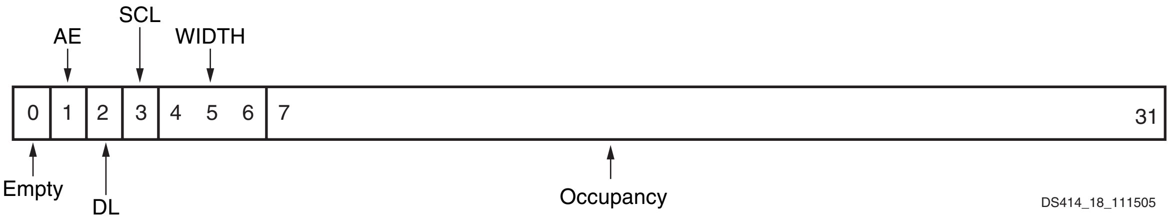


Figure 18: **RdFIFO Status Register Modified Bit Layout**

Table 23: **RdFIFO Status Register Bit Definitions**

Bit(s)	Name	Core Access	Reset Value	Description
7 to 31	Occupancy	Read	Zero [(1)]	RdFIFO Occupancy. • This is an unsigned value reflecting a current snapshot of the number of locations occupied with data in the RdFIFO memory core.
4 to 6	Width	Read	FIFO Width [(2)]	Encoded FIFO Data Port Width. This field reflects the parametrized width of the FIFO data port. • ’000’ = 32 bits (legacy PFIFO default) • ’001’ = 8 bits • ’010’ = 16 bits • ’011’ = 32 bits • ’100’ = 64 bits • ’101’ = 128 bits (not currently used) • ’110’ = 256 bits (not currently used) • ’111’ = 512 bits (not currently used)
3	SCL	Read	See Note [(3)]	Occupancy Scaling Enabled. • ”0’ = The Occupancy value is not scaled. • ’1’ = The Occupancy value is scaled to fit in the available Status Register space.
2	DL	Read	’0’	FIFO DeadLock Condition. • ’0’ = The FIFO is not in DeadLock. • ’1’ = The FIFO is in DeadLock (Simultaneously Full and Empty due to Packet Mode Operations).

Table 23: **RdFIFO Status Register Bit Definitions *(Contd)***

Bit(s)	Name	Core Access	Reset Value	Description
1	AE	Read	'0'	FIFO Almost Empty Condition. • '0' = If FIFO is not Empty, then at least two data values are available for reading. • '1' = FIFO is Almost Empty (only one more data value is available for reading).
0	Empty	Read	'1'	FIFO Empty Condition. • '0' = FIFO is not Empty. • '1' = FIFO is Empty and cannot provide anymore data.

Notes:

1. The FIFO Occupancy value defaults to 0. The max value that can be set is the FIFO depth which is set via User parameterization.
2. The FIFO data port width is set via User parameterization.
3. If the FIFO is part of a system where the Bus data width is less than 32 bits, it may be necessary to scale the Vacancy value down to fit in the available space in the reduced width Status Register. See the Packet FIFO IPSPEC.

RdFIFO Data Register

The RdFIFO Data Register shown in Figure 19 is a read-only register that is used to read data from the FIFO.

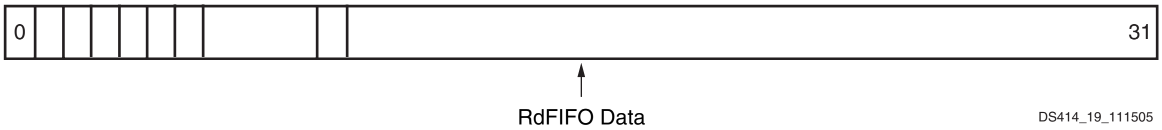


Figure 19: **RdFIFO Data Register**

Table 24: **RdFIFO Data Register Description**

Bit(s)	Name	Core Access	Reset Value	Description
0 to 31	RdFIFO Data	Read	n/a	RdFIFO Data

IPIF WrFIFO Service Registers

The following section details the register compliment of the IPIF Write FIFO Service. Parameterization of the IPIF will affect whether these registers are present or not.

WrFIFO Reset Register

A reset of the WrFIFO occurs when this register address is written with the value 0x0000000A. See Figure 20.

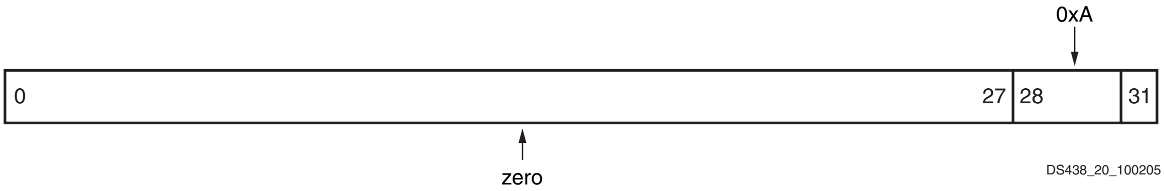


Figure 20: **WrFIFO Reset Register**

Table 25: **WrFIFO Reset Register Description**

Bit(s)	Name	Core Access	Reset Value	Description
0-31	Reset Key	Write	n/a	**Reset write value.** • '0x0000000A' - Generate a FIFO reset (the implementation is allowed to check just the four least-significant bits for the value 0xA). • others - No effect.

WrFIFO Module Identification Register

The WrFIFO Module Identification Register shown in Figure 21 is a read-only value that can be used by software to verify the addressability of the WrFIFO and to verify that the system-unique Block Identifier and the WrFIFO version, as reported in the fields of the Module Information Register, are as expected. This information can be used to verify the correctness of a system build.

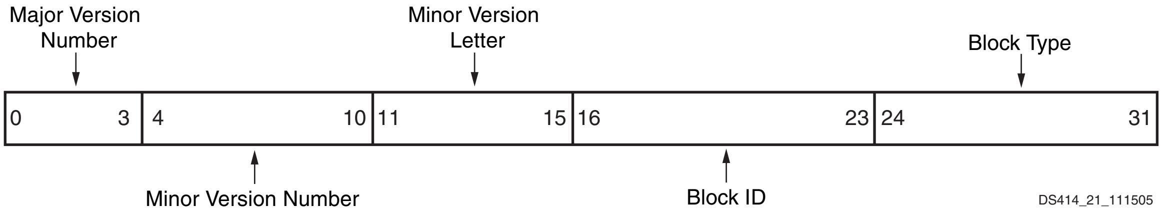


Figure 21: **WrFIFO Module Identification Register**

Table 26: **WrFIFO Model Identification Register Description**

Bit(s)	Name	Core Access	Reset Value	Description
0-3	Major Version Number	Read	n/a	Module Major Version Number.
4-10	Minor Version Number	Read	n/a	Module Minor Version Number.

Table 26: **WrFIFO Model Identification Register Description *(Contd)***

Bit(s)	Name	Core Access	Reset Value	Description
11-15	Minor Version Letter	Read	n/a	Module Minor Version Letter. Letters a-z are encoded as 00000 - 11001
16-23	Block ID	Read	n/a	Module Block ID.
24-31	Block Type	Read	n/a	Module Block Type.

WrFIFO Status Register

The WrFIFO Status Register shown in Figure 22 is a read-only register that gives the vacancy status of the WrFIFO.

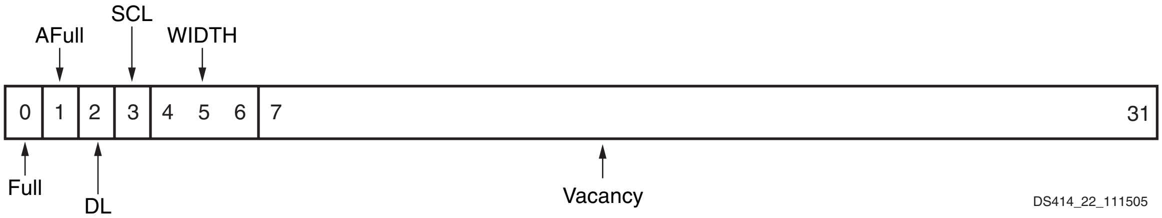


Figure 22: **WrFIFO Status Register Modified Bit Layout**

Table 27: **WrFIFO Status Register Bit Definitions**

Bit(s)	Name	Core Access	Reset Value	Description
7 to 31	Vacancy	Read	FIFO Depth [(1)]	WrFIFO Vacancy. • This is an unsigned value reflecting a current snapshot of the number of locations available for data storage in the WrFIFO memory core.
4 to 6	Width	Read	FIFO Width [(2)]	Encoded FIFO Data Port Width. • '000' = 32 bits (legacy PFIFO default) • '001' = 8 bits • '010' = 16 bits • '011' = 32 bits • '100' = 64 bits • '101' = 128 bits (not currently used) • '110' = 256 bits (not currently used) • '111' = 512 bits (not currently used)
3	SCL	Read	See Note [(3)]	Vacancy Scaling Enabled. • '0' = The Vacancy value is not scaled. • '1' = The Vacancy value is scaled to fit in the available Status Register space.

Table 27: **WrFIFO Status Register Bit Definitions *(Contd)***

Bit(s)	Name	Core Access	Reset Value	Description
2	DL	Read	’0’	FIFO DeadLock Condition. • ’0’ = The FIFO is not in DeadLock. • ’1’ = The FIFO is in DeadLock (Simultaneously Full and Empty due to Packet Mode Operations).
1	AF	Read	’0’	FIFO Almost Full Condition. • ’0’ = If FIFO is not Full, then at least two locations are available for writing. • ’1’ = FIFO is Almost Full (only one more location available for writing).
0	Full	Read	’0’	FIFO Full Condition. • ’0’ = FIFO is not Full. • ’1’ = FIFO is Full and cannot accept anymore data.

Notes:

1. The FIFO Vacancy value defaults to the depth of the FIFO. The FIFO depth is set via User parameterization.
2. The FIFO data port width is set via User parameterization.
3. If the FIFO is part of a system where the Bus data width is less than 32 bits, it may be necessary to scale the Vacancy value down to fit in the available space in the reduced width Status Register. See the Packet FIFO IPSPEC.

WrFIFO Data Register

The WrFIFO Data Register Figure 23 is a write-only register that is used to write data to the FIFO.

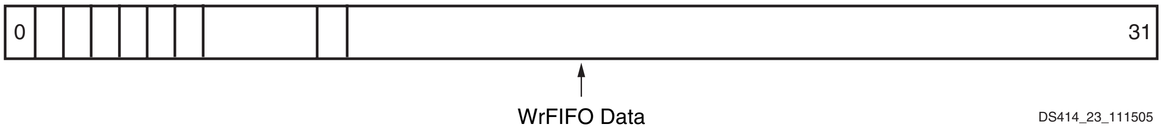


Figure 23: **WrFIFO Data Register**

Table 28: **WrFIFO Data Register Description**

Bit(s)	Name	Core Access	Reset Value	Description
0 to 31	WrFIFO Data	Write	n/a	Wr FIFO Data

IPIF DMA/Scatter Gather Service Registers

The following section documents the DMA/SG Service’s register set. The registers described here are repeated for each DMA/SG channel specified via User parameterization. The registers are defined for 32-bit accesses via the OPB.

RST Register (RST)

This register provides the User Application the ability to initiate a commanded reset of the DMA/SG. A data key value is written to the register's address to initiate the reset. The value to be written to the address is shown in Figure 24 and detailed in Table 29.

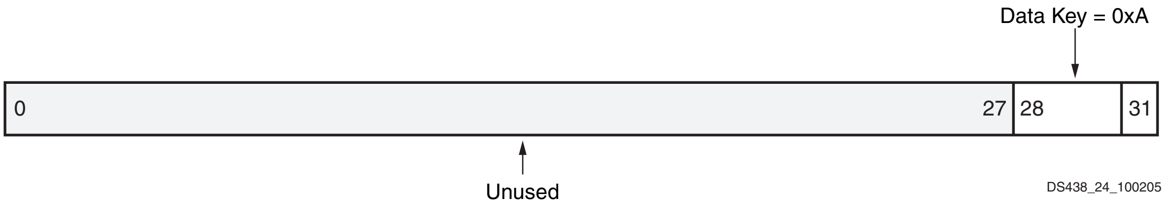


Figure 24: **DMA/SG RST Register Bit Layout**

Table 29: **RST/MIR Register Description**

Bit(s)	Field Name	Core Access	Reset Value	Description
28-31	Data Key	Write	n/a	**Reset write value.** • '0xA' - Generate a FIFO reset (the implementation is allowed to check just the four least-significant bits for the value 0xA). • others - No effect.
0-27	Unused	Write	n/a	**Unused bits.** • Not used. User Application should set to zeros

DMA/SG Module Identification Register (MIR)

The DMA/SG Module Identification Register shown in Figure 25 is a read-only value (not actually a register) that can be used by software to verify the OPB access to the DMA/SG Channel. This register provides read information that is constant and unique. This information can be used to verify the correctness of a system build and the User Application system addressing model.

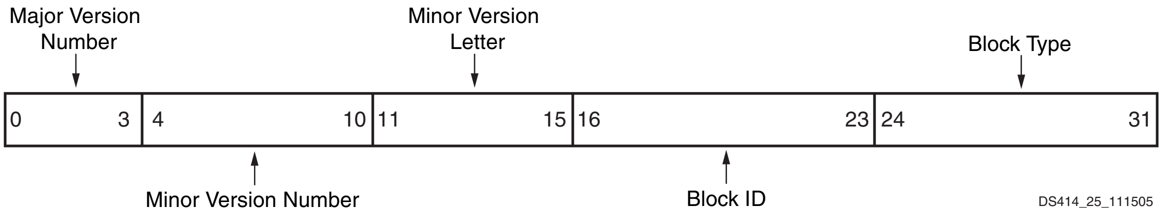


Figure 25: **DMA/SG Module Identification Register**

Table 30: **DMA/SG Module Identification Register Description**

Bit(s)	Name	Core Access	Reset Value	Description
0-3	Major Version Number	Read	Major Version of DMA/SG	Module Major Version Number. Binary Encoded value from 1 to 15. Value 0 is reserved for Xilinx Engineering Development.
4-10	Minor Version Number	Read	Minor Version of DMA/SG	Module Minor Version Number. Binary Encoded value from 0 to 127.
11-15	Minor Version Letter	Read	Revision of DMA/SG	Module Revision Letter. Letters a-z are binary encoded as 00000 - 11001
16-23	Block ID	Read	User Assigned Device Block ID	Module Block ID. This is an echo of the User Assigned Device Block ID set per OPB IPIF parameter **G8**.
24-31	Block Type	Read	User Assigned DMA/SG Channel Type	Module Block Type. 0x04 = Simple DMA 0x05 = Scatter/Gather 0x06 = Tx Channel 0x07 = Rx Channel

DMA/SG Control Register (DMACR)

The DMA/SG Control Register shown in Figure 26 provides control information to the DMA/SG Service. The register is setup by the User Application as part of the DMA transfer initialization. The register is detailed in Table 31.

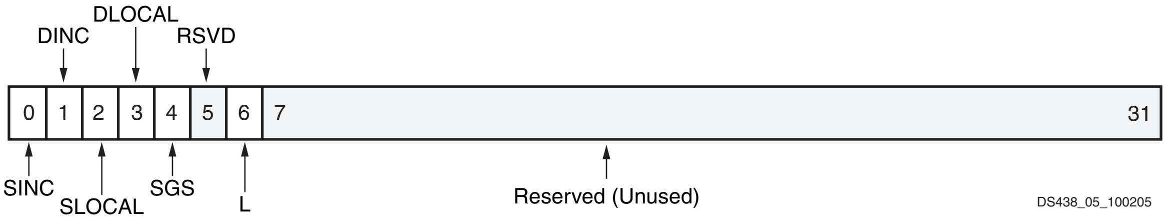


Figure 26: **DMA/SG Control Register Bit Layout**

Table 31: **DMA/SG Control Register Description**

Bit(s)	Field Name	Core Access	Reset Value	Description
0	SINC	R/W	'1'	Source Address Increment. This bit controls the Source Address generation during an active DMA operation. • '0' = Hold Source Address Constant. • '1' = Increment Source Address every completed DMA data beat by the number of bytes transferred in that data beat. (For a 32-bit OPB, the increment value is 4 bytes.)
1	DINC	R/W	'0'	Destination Address Increment. This bit controls the Destination Address generation during an active DMA operation. • '0' = Hold the Destination Address Constant. • '1' = Increment the Destination Address every completed DMA data beat by the number of bytes transferred in that data beat. (For a 32-bit OPB, the increment value is 8 bytes)
2	SLOCAL	R/W	'0'	Source Address is Local. (1) This bit indicates the geography of the Source Address relative to the host IPIF. • '0' = The Source Address is not local to the IPIF. It must be accessed via a OPB Master transaction read request from the IPIF Master Attachment. • '1' = The Source Address is local to the host IPIF and must be accessed using IPIF internal protocol mechanizations via the IPIF Slave Attachment.
3	DLOCAL	R/W	'1'	Destination Address is Local. (1) This bit indicates the geography of the Destination Address relative to the host IPIF. • '0' = The Destination Address is not local to the IPIF. It must be accessed via a OPB Master transaction write request from the IPIF Master Attachment. • '1' = The Destination Address is local to the host IPIF and must be accessed using IPIF internal protocol mechanizations via the IPIF Slave Attachment.
4	SGS	R/W	'1'	Scatter Gather Stop. Scatter Gather Stop. SGS is used to tell the Scatter Gather automation to stop as soon as a stop condition occurs. For simple SG channels, the stop condition is established by completion of the buffer descriptor for which the SGS bit is set. For packet SG channels, the start condition is established by completion of all partially completed packets.This bit is relevant during Scatter Gather operation. • '0' = Do not stop. • '1' = Stop when completion criteria is met.

Table 31: **DMA/SG Control Register Description *(Contd)***

Bit(s)	Field Name	Core Access	Reset Value	Description
5	RSVD	N/A	'0'	Reserved (not used)
6	L	R/W	'0'	Last. The DMA data transfer associated with the Buffer Descriptor setting this bit represents the last of a data packet. This bit is relevant during Scatter Gather operation. • '0' = This is not the last transfer. • '1' = This is the last DMA transfer needed to complete the current data packet.
7-31	RSVD	N/A	zeros	Reserved (not used)

Notes:

1. OPB IPIF design restricts the assertion of SLOCAL and DLOCAL to be one or the other but not both simultaneously.

Source Address Register (SA)

The Source Address Register shown in Figure 27 is used to specify the starting system address for the source of the data to be moved in a DMA operation. The identification of the address as either local or non-local relative to the host IPIF is set in the DMA Control register bit 2. The register is detailed in Table 32.

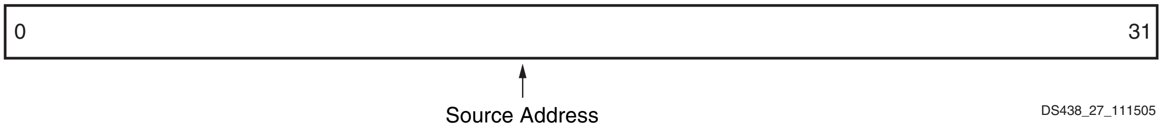


Figure 27: **Source Address Register Bit Layout**

Table 32: **Source Address Register Description**

Bit(s)	Field Name	Core Access	Reset Value	Description
0-31	Source Address	R/W	Zeros	Source Address. This 32-bit value specifies the starting system address for the source device of data that is to be moved during a DMA operation.

Destination Address Register (DA)

The Destination Address Register shown in Figure 28 is used to specify the starting system address for the destination of the data to be moved in a DMA operation. The identification of the address as either local or non-local relative to the host IPIF is set in the DMA Control register bit 3. The register is detailed in Table 33.

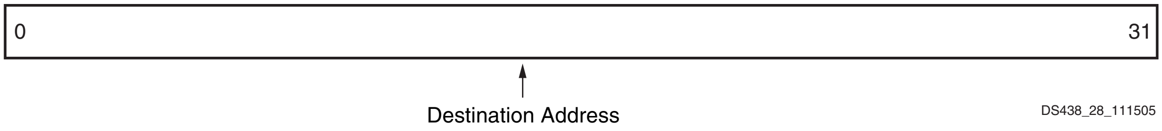


Figure 28: **Destination Address Register Bit Layout**

Table 33: **Destination Address Register Description**

Bit(s)	Field Name	Core Access	Reset Value	Description
0-31	Destination Address	R/W	Zeros	Destination Address. This 32-bit value specifies the starting system address for the destination target of data that is to be moved during a DMA operation.

Length Register (LENGTH)

The Length Register shown in Figure 29 specifies the number of bytes to transfer during a DMA operation. The number of bits allowed in the length specification is set by OPB IPIF Parameter **G16** (C_DMA_LENGTH_WIDTH_ARRAY). Using as few bits as possible will save FPGA resources. The Length value is right-justified in the 32 bit field. The register is detailed in Table 34.

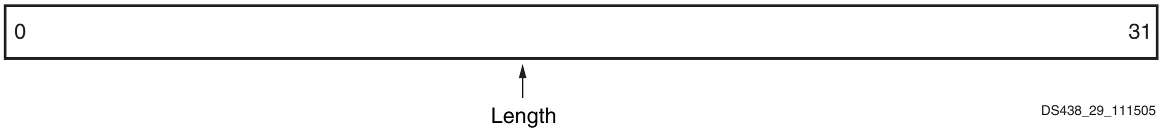


Figure 29: **Length Register Bit Layout**

Table 34: **Length Register Description**

Bit(s)	Field Name	Core Access	Reset Value	Description
0-31	Length	R/W	Zeros	Length Register. This register passes information into and out of a DMA operation. Prior to the operation it is loaded by the User Application to the required number of bytes to transfer. After the DMA operation completes, it represents the number, if any, of the requested bytes that did not transfer. For OPB packet-receive channels, the loaded value must be a multiple of 4 (word alignment). Otherwise, a loaded length that is not a multiple of 4 results 1 to 3 trailing "pad" bytes to be transferred, the actual number be that required to make the completed transfer a multiple of 4. Pad bytes, if any, are not, however, reflected in the final value of Length. **Note:** **Writing to this register starts the DMA operation. The User Application should write to this register as the last operation when setting up a DMA operation.**

DMA Status Register (DMASR)

The DMA Status Register shown in Figure 30 provides feedback to the User Application regarding DMA and Scatter Gather operations. The register is detailed in Table 35.

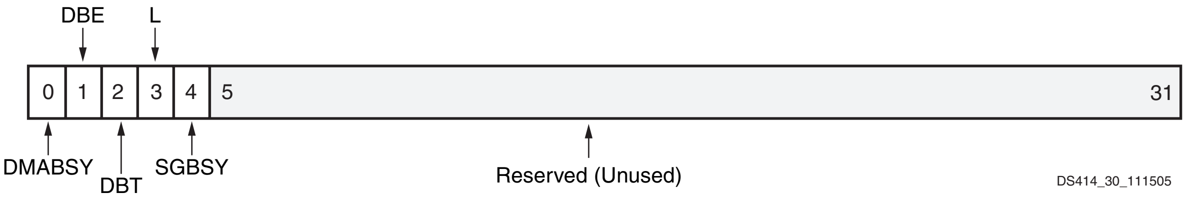


Figure 30: **Length Register Bit Layout**

Table 35: **DMA Status Register Description**

Bit(s)	Field Name	Core Access	Reset Value	Description
0	DMABSY	R	'0'	DMA Busy. This DMA status bit reports the operational status of this DMA channel. • '0' = DMA Idle • '1' = DMA is 'Busy'.
1	DBE	R	'0'	DMA Bus Error. This DMA status bit reports if a bus error was encountered by this DMA channel on the last attempted transfer. • '0' = No Bus Error Replies received. • '1' = A Bus Error Reply was received during an attempted DMA operation.
2	DBT	R	'0'	DMA Bus Time-out. This DMA status bit reports if a bus time-out error was encountered by this DMA channel on the last attempted transfer. • '0' = No Bus Time-out error occurred. • '1' = A Bus Time-out error occurred during an attempted DMA operation.
3	L	R	'0'	Scatter Gather Last. This Scatter Gather status bit is asserted if the current Scatter Gather Buffer Descriptor being executed (or just completed) is the last of a series of descriptors needed to DMA a data packet (i.e. The packet is completed). • '0' = Not the last buffer descriptor (packet transfer is not finished). • '1' = This is the last Buffer Descriptor needed to complete a packet transfer.
4	SGBSY	R	'0'	Scatter Gather Busy. This Scatter Gather status bit reports the operational status of this Scatter Gather channel. • '0' = Scatter Gather Idle • '1' = Scatter Gather is 'Busy'.
5-31	Reserved	N/A	zeros	

Buffer Descriptor Address Register (BDA)

The Buffer Descriptor Address Register Figure 31 is used to hold the system address of the first word of the next Buffer Descriptor to be executed by the Scatter Gather Service. Typically, Buffer Descriptors are stored in a system memory device and are written and updated by the User Application. The register is detailed in Table 36.

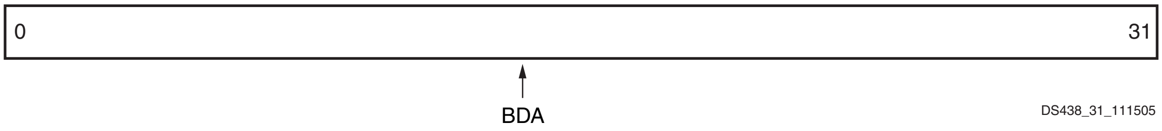


Figure 31: **Buffer Descriptor Address Register Bit Layout**

Table 36: **Buffer Descriptor Address Register Description**

Bit(s)	Field Name	Core Access	Reset Value	Description
0-31	BDA	R/W	Zeros	Buffer Descriptor Address. The address of the first word of the next Buffer Descriptor to be executed by the Scatter Gather Service of this channel.

Software Control Register (SWCR)

The Software Control Register shown in Figure 32 is used for User Application control of the Scatter Gather Service of this DMA channel. The register is detailed in Table 37.

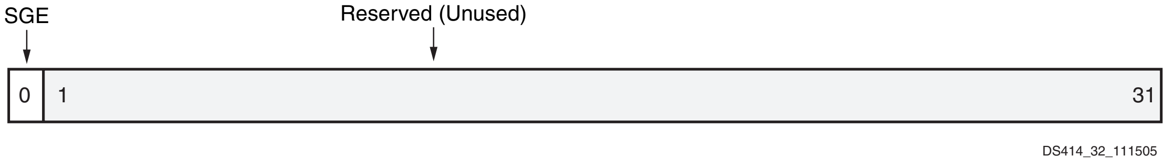


Figure 32: **Software Control Register Bit Layout**

Table 37: **Software Control Register Description**

Bit(s)	Field Name	Core Access	Reset Value	Description
0	SGE	R/W	'0'	Scatter Gather Enable. SG automation does not begin unless SGE=1 and DMACR.SGS = 0. SG automation can be stopped while active by setting SGE=0. For simple SG channels, the stopping condition is after the completion of any Buffer Descriptor that may have started. For the packet SG channels, the stopping condition is the completion (including writing of the SR value) of any packet that may have started. • '0' = Scatter Gather Disabled. • '1' = Scatter Gather Enabled.
1-31	Reserved	N/A	Zeros	Reserved. Not Used

Unserviced Packet Count Register (UPC)

The Unserviced Packet Count Register shown in Figure 33 is a Scatter Gather status register that provides a count of the number of data packets transferred by the Scatter Gather Service that have not been 'acknowledged' by the User Application. The register is detailed in Table 38.

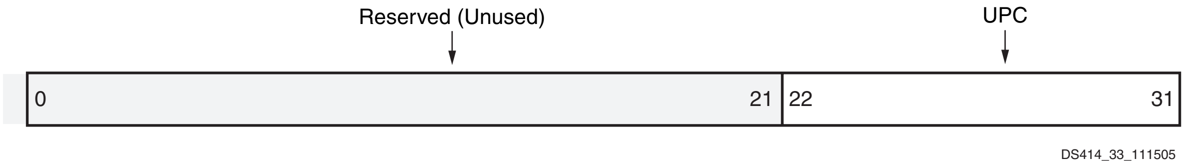


Figure 33: **Unserviced Packet Count Register Bit Layout**

Table 38: **Unserviced Packet Count Register Description**

Bit(s)	Field Name	Core Access	Reset Value	Description
0-21	Reserved	N/A	Zeros	Reserved. Not Used
22-31	UPC	R/W [1]	0	Unserviced Packet Count [2]. The unserviced packet count, with the cooperation of User Application, gives an accurate count for the channel of the number of packets that have been handled by the device but not "acknowledged" by the User Application. An accurate count is maintained when the User Application and Scatter Gather logic access the register concurrently.

Notes:

1. The only write operation that is defined is a write to the register with a data value of 0x00000001. This causes the UPC to decrement by one. This is how the User Application "acknowledges" the servicing of a packet for purposes of maintaining the count.
2. The UPC, PCT and PWB registers are provided to facilitate the Interrupt Coalescing feature of the DMA/SG Service.

Packet Count Threshold Register (PCT)

The Packet Count Threshold Register shown in Figure 34 is used to hold a value written by the User Application that specifies the an interrupt trigger threshold for the UPC. The register is detailed in Table 39.

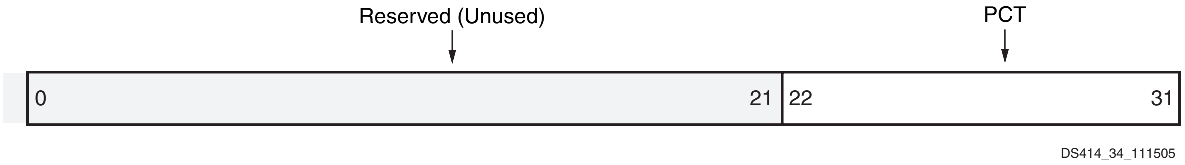


Figure 34: **Packet Count Threshold Register Bit Layout**

Table 39: **Packet Count Threshold Register Description**

Bit(s)	Field Name	Core Access	Reset Value	Description
0-21	Reserved	N/A	Zeros	Reserved. Not Used
22-31	PCT	R/W	0	Packet Count Threshold [1]. Whenever the Unserviced Packet Count, UPC, meets or exceeds this threshold, a PCTR (Packet Count Threshold Reached) interrupt is generated. The value 0 is a special value that disables the interrupt from being generated.

Notes:

1. The UPC, PCT and PWB registers are provided to facilitate the Interrupt Coalescing feature of the DMA/SG Service.

Packet Wait Bound Register (PWB)

The Packet Wait Bound Register shown in Figure 35 is used to place an upper bound on the time that a packet might have to wait until its reception or transmission is made visible to the User Application through an interrupt. The register is detailed in Table 40.

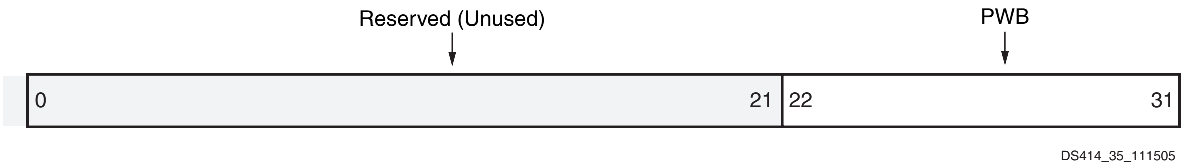


Figure 35: **Packet Wait Bound Register Bit Layout**

Table 40: **Packet Wait Bound Register Description**

Bit(s)	Field Name	Core Access	Reset Value	Description
0-21	Reserved	N/A	Zeros	Reserved. Not Used
22-31	PWB	R/W	0	Packet Wait Bound. [1]. This field specifies the PWB in units of milliseconds (plus or minus 33%). Whenever there is no pending interrupt for the channel and the Unserviced Packet Count (UPC) is non-zero, a timer starts counting toward time-out at the PWB value (with accuracy interval of plus 1 or minus 0 units). If the time-out is reached, a PWBR (Packet Wait Bound Reached) interrupt is generated. The value 0 is a special value that disables the interrupt.

Notes:

1. The UPC, PCT and PWB registers are provided to facilitate the interrupt coalescing feature of the DMA/SG Service.

Interrupt Status Register (ISR)

The Interrupt Status Register shown in Figure 36 is the source of operational interrupts to the User Application for a DMA/SG channel. The interrupt bits in this register are masked by the Enable Register and OR'd together to form a single interrupt signal to the IPIF level ISR (see Device ISR). Thus, the IPIF ISR will have a corresponding interrupt bit for each DMA/SG channel that has been specified. The register is detailed in Table 41.

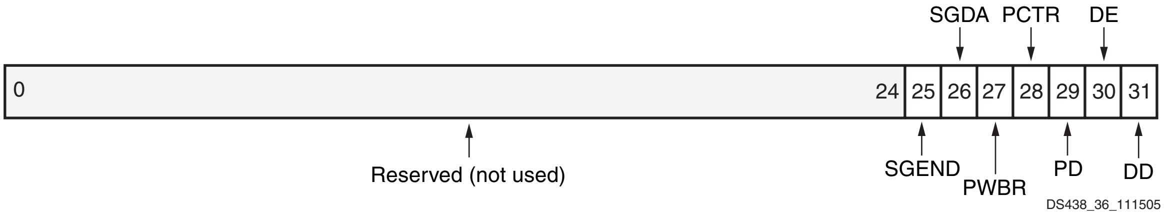


Figure 36: **Interrupt Status Register Bit Layout**

Table 41: **Interrupt Status Register Description**

Bit(s)	Field Name	Core Access	Reset Value	Description
0-24	Reserved	N/A	zeroes	Reserved. Not Used
25	SGEND	R/TOW (1)	'0'	Scatter Gather End. SG operation has stopped as a result of finishing the processing of a Buffer Descriptor where the SGS bit was set in the Descriptor's DMACR word. This bit is valid in both Simple Scatter Gather mode and Packet Scatter Gather mode. • '0' = Scatter Gather End not reached. • '1' = Scatter Gather End reached.
26	SGDA	R/TOW (1)	'0'	SG Disable Acknowledge. SG operation has stopped as a result of the User Application stopping operation by forcing DMASWCR.SGE=0 (see DMA Software Control Register). This bit is valid in both Simple Scatter Gather mode and Packet Scatter Gather mode. In Simple SG mode, this bit will be set at the completion of the Buffer Descriptor being executed when the SGE bit was cleared. In Packet SG mode, the bit will get set at the completion of the packet being transferred when the SGE bit is cleared. • '0' = Scatter Gather Disable not encountered. • '1' = Scatter Gather Disable has been acknowledged and operations terminated.
27	PWBR	R/TOW (1)	'0'	Packet Wait Bound Reached. The Scatter Gather Packet Wait Bound timer has reached its time-out value (see Packet Wait Bound Register). This bit is valid in Packet Scatter Gather mode only. • '0' = Scatter Gather End not reached. • '1' = Scatter Gather End reached.

Table 41: **Interrupt Status Register Description *(Contd)***

Bit(s)	Field Name	Core Access	Reset Value	Description
28	PCTR	R/TOW [(1)]	'0'	Packet Count Threshold Reached. The Scatter Gather Packet Count has reached its Threshold value (see Packet Count Threshold Register). This bit is valid in Packet Scatter Gather mode only. • '0' = Scatter Gather Packet Count below threshold. • '1' = Scatter Gather Packet Count is at or above Threshold.
29	PD	R/TOW [(1)]	'0'	Packet Done. A packet transfer has completed and its status has been updated in the SR field of the packet's first Buffer Descriptor. This bit is valid in Packet Scatter Gather mode only. • '0' = Packet not completed. • '1' = Packet transfer has completed.
30	DE	R/TOW [(1)]	'0'	DMA Error. This interrupt indicates whether or not a DMA error occurred during DMA operation. The result is fatal. The channel and any associated FIFOs or devices must be re-initialized. This bit is valid in all modes. • '0' = No Error. • '1' = Error encountered.
31	DD	R/TOW [(1)]	'0'	**DMA Done.** Set whenever a DMA transfer operation finishes either normally or with an error. This bit is valid in all modes. During Scatter Gather operation, this bit is set at the completion of each Buffer Descriptor. • '0' = Not Done. • '1' = DMA Transfer Done.

Notes:

1. TOW refers to a Toggle on Write operation. Writing a 1 to a bit position in the register causes the value at that bit position to toggle state. Writing a 0 has no effect. During normal operation, interrupts that are asserted (= 1) can be cleared by the User Application without the need for a Read/Modify/Write operation. A Read/Modify/Write can inadvertently clear interrupts captured in the time period between the Read and Write operations.

Interrupt Enable Register (IER)

The Interrupt Enable Register shown in Figure 37 provides the User Application the ability to mask or enable the interrupts captured in the DMA/SG ISR. The register is detailed in Table 42.

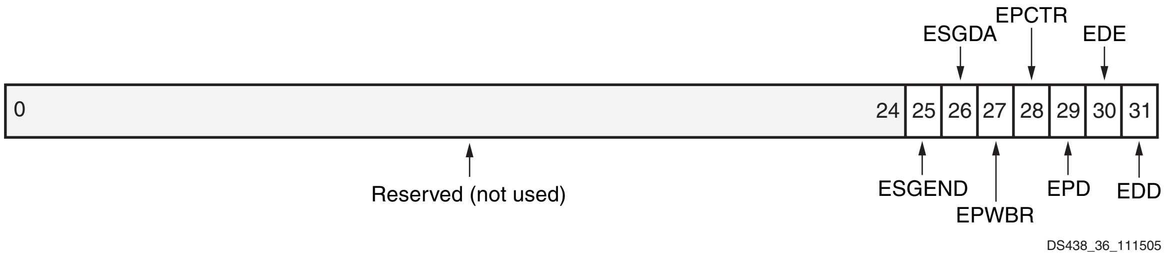


Figure 37: **Interrupt Enable Register Bit Layout**

Table 42: **Interrupt Enable Register Description**

Bit(s)	Field Name	Core Access	Reset Value	Description
0-24	Reserved	N/A	zeroes	Reserved. Not Used
25	ESGEND	R/W	'0'	Enable Scatter Gather End. • '0' = Interrupt Masked. • '1' = Interrupt Enabled.
26	ESGDA	R/W	'0'	Enable SG Disable Acknowledge. • '0' = Interrupt Masked. • '1' = Interrupt Enabled.
27	EPWBR	R/W	'0'	Enable Packet Wait Bound Reached. • T'0' = Interrupt Masked. • '1' = Interrupt Enabled.
28	EPCTR	R/W	'0'	Enable Packet Count Threshold Reached. • '0' = Interrupt Masked. • '1' = Interrupt Enabled.
29	EPD	R/W	'0'	Enable Packet Done. • '0' = Interrupt Masked. • '1' = Interrupt Enabled.
30	EDE	R/W	'0'	Enable DMA Error. • '0' = Interrupt Masked. • '1' = Interrupt Enabled.
31	EDD	R/W	'0'	**Enable DMA Done.** • '0' = Interrupt Masked. • '1' = Interrupt Enabled.

Usage Protocols for OPB IPIF Services

Using the IP Slave Interface

The Slave Interface allows modules connected to the IPIC to respond to commands to read and write data.

Register Model

A simple slave IP may take full advantage of the IPIF address-decoding service to generate CE decodes for all of the individual addresses of interest within a given Address Range. These CE decodes come in three versions, Bus2IP_CE, Bus2IP_RdCE and Bus2IP_WrCE, conveying, respectively that an individual address has been decoded for read/write, read, or write[1]. An application that uses the CE decodes is sometimes referred to as a *register model*.

Figure 38 shows examples of read transactions as seen from a register model and Figure 39 shows similar write transactions.

The appropriate request signal, Bus2IP_RdReq or Bus2IP_WrReq, asserts for the first cycle of a single transaction. For reads, the IP may drive non-zero data to IP2Bus_Data whenever one of its register-read decodes is active and it must drive valid data during the cycle that it asserts its acknowledgement. At all other times the IP must drive zero.

The acknowledgement indicates the transaction's completion, therefore the duration of the transaction is controlled by the IP, subject to timeout. (Transactions may be made as short as one cycle by returning the acknowledgement in the same cycle as the request, if the operating frequency permits.)

If the IP is able to respond to the transaction, but not within 8 cycles, the first cycle being the one on which Bus2IPRdReq or Bus2IPWrReq is asserted, it can suppress the timeout by asserting IP2Bus_Toutsup within 8 cycles and hold it until it responds to the transaction.

If the transaction is not successful, error completion can be indicated by asserting IP2Bus_Error to qualify the acknowledgement. If the transaction cannot be completed successfully, but may succeed if retried, the IP2Bus_Retry response is asserted by the IP as a mutually exclusive one-cycle alternative to the acknowledgement.

To summarize response signalling, the transaction response signals IP2Bus_RdAck, IP2Bus_WrAck, IP2Bus_Error, IP2Bus_Retry and IP2Bus_Toutsup must be driven to zero when the IP is not addressed. When the IP is addressed, either a one-cycle acknowledgement or a one-cycle retry assertion—but never both—terminates the transaction. IP2Bus_Error must be valid during a cycle in which an acknowledgement is asserted and otherwise is a "don't care".

[1] The application connects to the CE, RdCE or WrCE that is most appropriate for its use and can expect synthesis and implementation tools to optimize away any signals that are unused. This even extends more generally to the use of the CS, RNW and Addr signals, as well. The user can use full decodes or do custom decoding without expecting a penalty for unused signals.

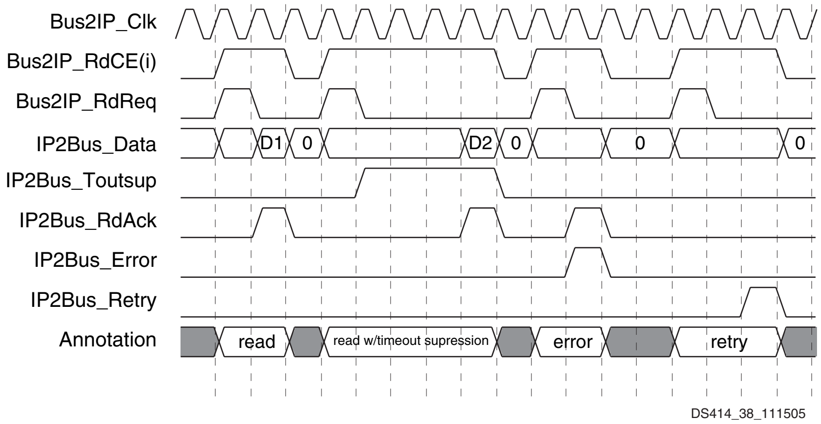


Figure 38: **Slave Read Transactions, Register Model**

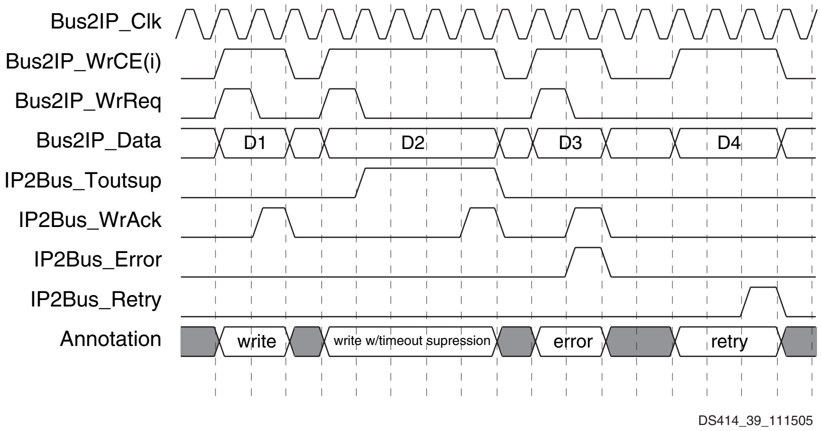


Figure 39: **Slave Read Transactions, Register Model**

Memory Model

Another type of IP might require an address-range CS decode, a read/write signal and address bits. This is sometimes referred to as the *memory model* or *SRAM model*.

Figure 44 and Figure 41 show examples of read and write transactions under the memory model.

The timing and protocol for the memory model is identical to that for the register model. There is just one slave protocol and the existence of a register model or a memory model is from the perspective of the IP in deciding which decode and control signals to interpret.

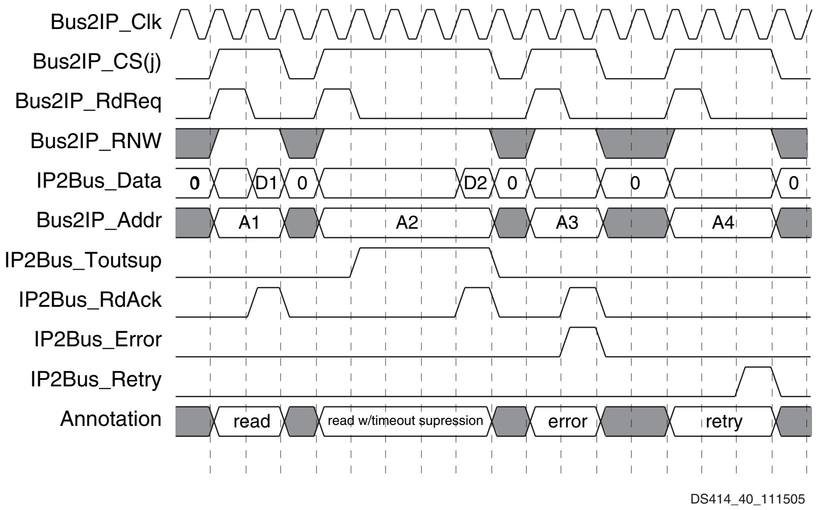


Figure 40: **Slave Read Transactions, Memory Model**

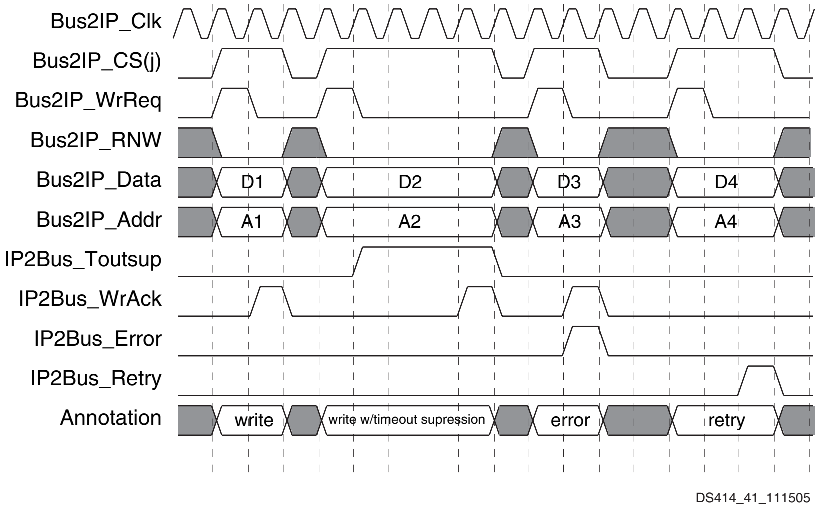


Figure 41: **Slave Read Transactions, Memory Model**

Agents that Activate the Slave Interface

It should be obvious that transactions that originate from a remote OPB master and that target an address space of the local IPIF will result in corresponding local IPIC transactions. Less obvious, but also true, is that if either or both of the possible local master agents has been optioned into the IPIF, then IPIC slave transactions will also occur as a result of *locally mastered* transactions.

The two possible local-master agents are the DMA controller—an optional service of the IPIF—and a master capability that might be implemented in the IP and interfaced to the IPIF utilizing the master interface of the IPIC. Further description of these local-master agents and the master interface is found in "Using the IPIF DMA/Scatter Gather Service" on page 67 and "Using the IP Master Interface" on page 74. To be emphasized in the present discussion, however, is that locally mastered write transactions to the OPB will be preceded by a local IPIC read transactions to collect the data to be written and that locally mastered read transactions will use IPIC write transactions to put away the data collected from OPB reads.

Posted Write Transactions

Locally mastered read transactions have a challenging aspect when flow-control is considered. Under the OPB protocol, the remote slave has total control of when data is returned (or wait states injected). It may return data on every clock and, in fact, for burst (i.e. sequential-address) transactions, it should take pains to do so.

Hence, the IPIF must be prepared to absorb data on any clock. This, in turn, means either that the IPIC target must be able to absorb data on any clock or that the IPIF must buffer data, at least until it can discontinue its OPB read transaction.

This version of the IPIF uses the first solution and requires that any address space that can be the put-away target of a locally mastered read also be able to absorb write data in the cycle that it is presented—without the opportunity to delay or refuse it. Such an IPIC write transaction is called a *posted write*.

A related situation arises for remotely mastered write bursts (no local master involved). Here, since the OPB_seqAddr signal is asserted, the goal is to absorb write data each cycle. Even though the OPB protocol would allow for a round-trip acknowledge from the IP, this privilege is sacrificed in the interest of performance. In the end, the requirement and solution is the same. Either the IP absorbs posted writes or the IPIF provides buffering.

Inhibiting Posted Writes

The strict requirement that address spaces that can be targeted by local read transactions or by remote burst write transactions be able to accept posted writes is a requirement that can be relieved in some cases through use of the IP2Bus_PostedWrInh signal. An application can disable the IPIF mechanism that optimizes incoming, remotely mastered, sequential-address write transactions into bursts by asserting IP2Bus_PostedWrInh no later than the cycle on which the transaction starts on the OPB. This first cycle becomes a decision point on whether to handle the OPB sequential-address transaction at the IPIC as a burst of posted writes or as a sequence of slower, round-trip, fully acknowledged writes.

This same technique cannot be used to disable posted writes that are due to locally mastered reads. Nevertheless, the IP2Bus_PostedWrInh signal is given an additional role in allowing the IP to gain some relief from posted writes that are due to local DMA reads. If the signal is asserted, the IPIF will not accept a new DMA request for a read and the IP will at most receive the posted writes for a DMA read that has already been accepted by the IPIF.

Using the IPIF Interrupt Service

Most microprocessor systems require peripheral devices to request the attention of the microprocessor through the assertion of interrupt signals. Generally, a central interrupt controller is used to collect the interrupts from various sources and then apply prioritization and masking functions to them per User Application programming. The OPB IPIF provides the User a Service that is a simple interrupt controller function that is used to collect interrupts from within a User Device. These will be generated by IPIF Services and the User IP. The Interrupt Service captures and coalesces these various interrupt signals into a single interrupt output signal that is sent to the microprocessor's System Interrupt Controller. The Service also provides local registers that the User Application can utilize to read interrupt status, set up masking criteria, and perform interrupt clearing for the individual interrupts. These registers are detailed in the Register-Interface Descriptions for OPB IPIF Services section of this document.

Interrupts may be generated within a Device by the User IP and/or IPIF Services. The number of User IP interrupts that need to be captured depends on the function of the IP and is generally quite different from IP to IP. Rather than attempting to accommodate this variable number of interrupt bits into a single register, a hierarchical interrupt capture and reporting scheme is used that is coupled with User parameterization.

Table 43 summarizes the possible interrupt sources that a device may require. Depending upon the configuration, some or all may be absent in simple devices.

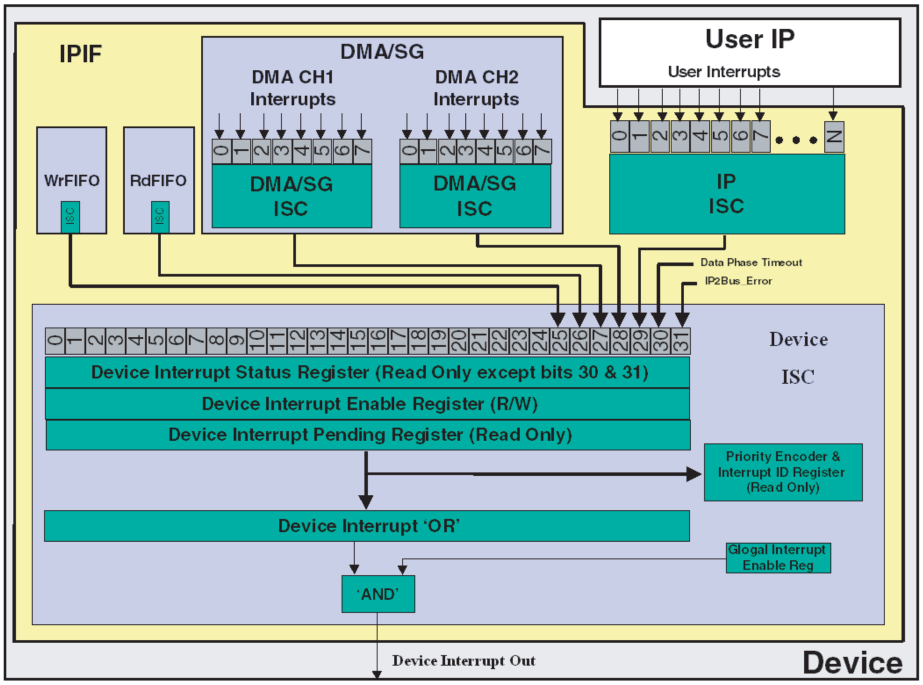
Table 43: **OPB IPIF Interrupt Source Summary**

Interrupt	Source	Description
Transaction Error	IPIF	IPIC transaction acknowledged with an error (IP2Bus_Error).
IP [1]	User IP	Interrupt conditions specific to the IP. 0 to 32 allowed
DMA/SG [1]	IPIF (DMA/SG)	Interrupt conditions on a per-channel basis. Up to 8 interrupts per channel.
Read FIFO Packet Mode Deadlock [1]	IPIF (Read FIFO)	The Deadlock bit in the Read FIFO status register has been set. One per FIFO.
Write Packet Mode Deadlock [1]	IPIF (Write FIFO)	The Deadlock bit in the Write FIFO status register has been set. One per FIFO.

Notes:
1. These are parameterization dependent.

The hierarchical interrupt reporting structure is based on the *Interrupt Source Controller*, sometimes referred to simply as an ISC. An Interrupt Source Controller is a function that captures a number of interrupt input signals and, using masking and logic, coalesces the captured interrupts into a single output signal that is sent to the next higher level of interrupt hierarchy.

An interrupt-event signal is a transient condition that needs to be captured by an Interrupt Source Controller and held until there is an explicit acknowledgement (actually a clear operation) by the User Application. An Interrupt-active signal is defined as an a interrupt signal that is captured and held (until activity cleared) by a *lower-level* ISC. Interrupt active signals do not need to be recaptured at the next higher level of interrupt hierarchy. Figure 42 shows the OPB IPIF interrupt structure for an example User device that is maximally populated with interrupts.



DS414_42_111505

Figure 42: **Example User Device Interrupt Hierarchy**

In the lower half of the diagram, the *Device Interrupt Source Controller* is shown. The Device ISC receives a Transaction Error interrupt event and a single interrupt-active signal from each of the five lower-level ISCs. It is the function of the Device ISC to output the single Device Interrupt signal to the microprocessor's System Interrupt Controller via the Dev_Intr_Out output port. Note that it is the highest level of interrupt hierarchy for the Device. It's registers also provide control and status information that are used to mask and/or discover the source of interrupts within the Device.

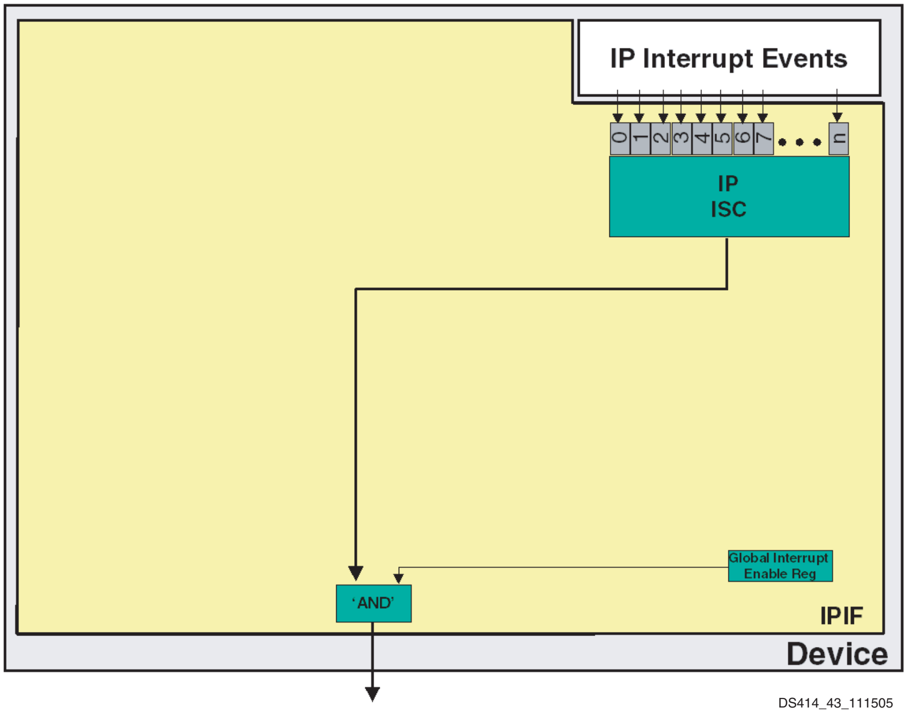
User interrupts are generally (but not required to be) captured and controlled in the IP ISC. The IP ISC is implemented in the IPIF as part of the Interrupt Service for the User IP. It captures interrupt events directly from the User IP per the capture mode specified by the C_IP_INTR_MODE_ARRAY parameter. The number of User IP interrupts needed (N) is inferred from the number of entries in the C_IP_INTR_MODE_ARRAY parameter. The IP ISC then coalesces the IP interrupts into a single interrupt active signal that is output to the Device ISC.

IPIF ISCs include one for each DMA channel enabled and one for each Packet FIFO. The Packet FIFO interrupt is only used when the FIFO has been parameterized with Packet Mode support enabled.

There two IPIF configurations for which part of the interrupt hierarchy is optimized away:

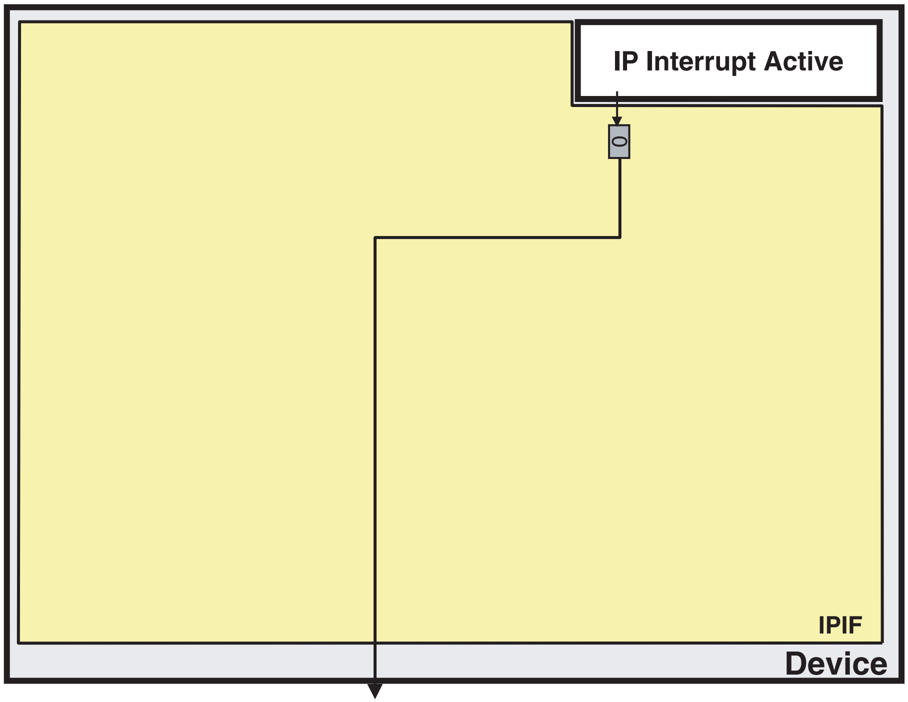
- For IPIF configurations where there are no FIFO or DMA/SG Services enabled, all of the Device ISC except the Global Interrupt Enable Register may be optimized away by the User setting the **C_INCLUDE_DEVICE_ISC** parameter to 0. In this case, the only source of interrupts is the IP ISC. However, the Transaction Error and Dataphase Time-out interrupts are not available. This option, which reduces hardware cost and software accesses, is shown in Figure 43.
- For IPIF configurations where there are no FIFO or DMA/SG modules and the IP has a self contained interrupt solution, the single interrupt-active signal from the IP is passed by the IPIF directly from IP2Bus_IntrEvent(0) input to the Dev_Intr_Out output signal. This implementation is

generated when the **ARD Arrays** have no entry for the IPIF_INTR identifier. This is graphically shown in Figure 44.



DS414_43_111505

Figure 43: **OPB IPIF Interrupt Service Structure with Device ISC Removed**



DS414_44_111505

Figure 44: **OPB IPIF Interrupt Structure with Interrupt Service Removed.**

Using the IPIF FIFO Service

If data buffering is needed by the IP, the IPIF can be parameterized to include either a read FIFO, a write FIFO, or both. A read FIFO is read from the OPB and written to by the IP. In a data-communications application, it would typically be connected to the receive channel. A write FIFO is written to from the OPB and read by the IP and would typically be connected to the transmit channel.

The read or write FIFO service is made available to the IP through a read or write FIFO interface. The interfaces to the read and write FIFO are essentially the same, differing only in data direction and whether the FIFO flags are with respect to vacancy (full, almost full, or count of vacant locations) or occupancy (empty, almost empty, or count of occupied locations).

In some applications, the ability to "back up" in the FIFO is useful. That is, data is *provisionally* read or written but not *committed* until later. As an example of an application that can benefit from provisional-data capability, consider a transmitter reading from a Write FIFO, where the packet transmission protocol requires that in some situations—for example a collision—the packet transmission must be aborted and retried. Another example is a receiver feeding a Read FIFO where the receiver filters incoming data and discards packets that fail a CRC check. It is with applications such as these in mind that the provisional-data capability is referred to as the *packet mode* and the FIFOs as *packet FIFOs*.

Provisional operations are enabled by setting a *mark* in the FIFO. If it is later decided to discard the uncommitted operations, a *restore* command returns the FIFO to its marked position. If, on the other hand, it is later decided to commit the uncommitted operations, a new mark command commits the provisional operations and sets a new mark, or a *release* command commits the provisional operations and clears the mark, which means that future operations—until a new mark is set—will be committed operations.

Provisional data operations lead to some FIFO conditions not encountered with simple FIFOs. Uncommitted writes to the read FIFO are reflected immediately in the vacancy seen by the IP but are not reflected in the occupancy seen on the opposite, OPB, side until and if they are committed. Similarly, uncommitted reads of the write FIFO are reflected immediately in the occupancy seen by the IP but are not reflected in the vacancy seen on the opposite side. Therefore, the OPB side may see a quantum increase in occupancy or a quantum decrease in vacancy (depending on whether it is a read or write FIFO) when operations are committed and the IP side may see a quantum decrease in vacancy or increase in occupancy when uncommitted operations are discarded.

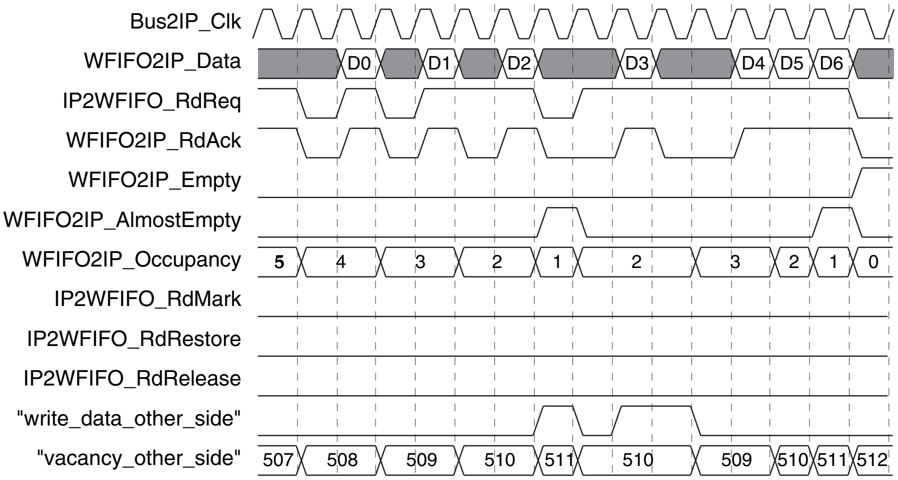
A corollary of this is that the number of possible uncommitted operations is limited by the depth of the FIFO. Further, if the number of uncommitted operations becomes equal to the depth of the FIFO, the FIFO appears to be full at one of its ports and empty at the other! This apparent "deadlock" can only be broken by committing or discarding the uncommitted operations. This condition is most likely an indication of a system error where the FIFOs are not sized correctly for the application. Therefore, the FIFOs generate an interrupt if this condition occurs.

Simultaneous control-signal assertions are restricted or interpreted as follows. The assertion of any one of mark, restore or release must be exclusive of the other two. If a mark, restore or release command occurs on a cycle in which the FIFO also acknowledges a read or write data operation, first the data operation completes, then the command is applied.

The transfer of data to or from a FIFO is done under a request-acknowledge protocol. The number of clock cycles between a request by the IP and the acknowledgement by the FIFO should be considered to be variable and may be as small as zero, meaning that the FIFO may acknowledge combinatorially on the same cycle in which the request is asserted. The FIFO and IP take data and update status on a clock edge where the request and acknowledge are both active.

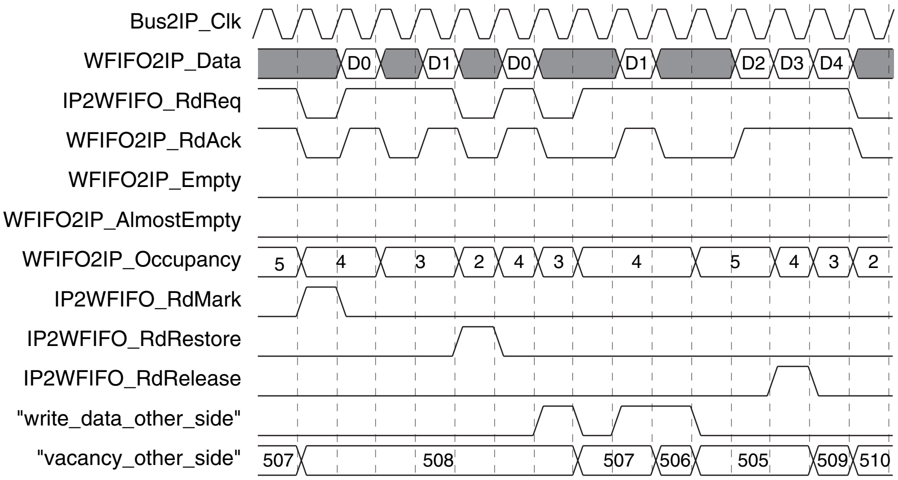
The mark, restore and release commands take affect on the clock edge that they are asserted and are not acknowledged.

The figures, Figure 45 through Figure 48, give examples of read and write FIFO operation. These examples are for FIFOs with a capacity of 512 items.



DS414_45_111505

Figure 45: **Reading the Write FIFO**



DS414_46_111505

Figure 46: **Reading the Write FIFO, Including Uncommitted Reads Using Mark, Restore, and Release**

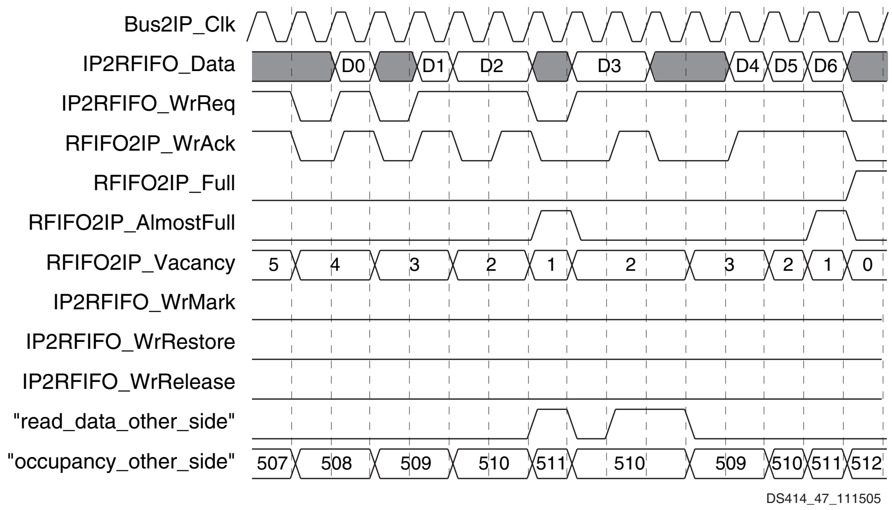


Figure 47: **Writing the Read FIFO**

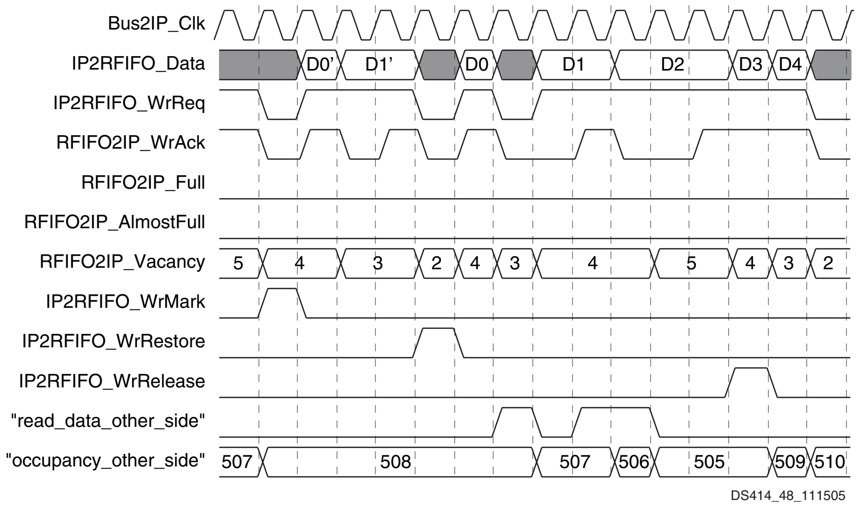


Figure 48: **Writing the Read FIFO, Including Uncommitted Writes Using Mark, Restore, and Release**

Using the IPIF DMA/Scatter Gather Service

The OPB IPIF offers a DMA Service that is useful for automating large data transfers in and out of the User Device via the OPB. When the DMA Service is enabled through the IPIF parameterization, a Master Attachment Service is automatically enabled. This is required by the DMA Service since it must initiate OPB transfers. Thus, a OPB Master signal interface with the OPB is necessary and the OPB Device becomes a Master-Slave combination. The User should refer to Xilinx Document ***DS-416 Direct Memory Access and Scatter Gather*** for detailed DMA and Scatter Gather operation description.

Simple DMA Basics

At a high level, a Simple DMA transfer is comprised of two separate operations. These are reading data from a memory mapped source address and then writing the data to a memory mapped destination address. These two operations always have to be performed in that order, read then write. From the IPIF perspective (of which the DMA is a part) each read and write operation is further classified as a local operation or a Bus operation. A local operation is defined as a read or write transaction where the memory mapped target address is for an element within the same IPIF as the DMA or with User IP elements that are attached to that IPIF. Conversely, the Bus operation is a read or write operation to/from a memory mapped target address that is not local and must be accessed via the OPB Bus. The Bus operation is sometimes referred to as a 'remote' operation due to the fact the target address is for an element that is not part of the IPIF and User IP initiating the transaction and is therefore 'remote' to the Device.

The User Application is required to set up four registers within the DMA for a Simple DMA operation. These are: the DMA Control Register, the DMA Source Address Register, the DMA Destination Address Register, and the DMA Length Register. If interrupts are to be used, the User Application must also enable the 'Done' interrupt in the DMA Interrupt Enable Register as well as set-up the IPIF Interrupt Service and the System Interrupt Controller. The DMA registers are detailed in the section "IPIF DMA/Scatter Gather Service Registers" on page 45 and the IPIF Interrupt Service Registers are detailed in the section "IPIF Interrupt Service Registers" on page 31. The User Application must write the transfer length to the Length Register as the last operation. This is due to the fact that writing to Length Register also initiates the DMA transfer.

The current DMA implementation segments the requested byte transfer loaded in the Length Register into multiple Fixed Length bursts of 16 double-words followed by a final cleanup transaction that can be a single data beat request or a Fixed Length burst of 2 to 16 data beats. The final cleanup transaction is necessary only if the total transfer count is not an even multiple of 16 double-words. At the completion of the transfer, the DMA will assert a 'Done' bit in it's status register. The User Application can poll this register until this happens to determine when the transfer has completed. Optionally, the User Application can use interrupt protocol to track DMA completion. When the DMA asserts the 'Done' bit, it also asserts an interrupt. This interrupt can be used to signal the User Application that the DMA transfer is complete.

Example DMA Programming Using Interrupt Protocol

The following sequence of operations shows operations the User Application must do to Enable the DMA transfer done interrupt, initiate a simple DMA transfer, and perform completion housekeeping.

1. Setup the microprocessor's System Interrupt Controller (Enable the User Device interrupt).
2. Enable the DMA Service interrupt in the IPIF Device Interrupt Enable Register.
3. Enable the Global Interrupt Enable in the IPIF Global Interrupt Enable Register.
4. Enable the DMA 'Done' Interrupt in the DMA Interrupt Enable Register.
5. Write the control mode to the DMA Control Register.
6. Write the source address to the DMA Source Address Register.
7. Write the destination address to the DMA Destination Address Register.
8. Write the transfer length (in bytes) to the Length Register. This starts the transfer.
9. Wait for Device Interrupt to occur in the System Interrupt Controller.
10. Read the IPIF Device Interrupt Status Register and verify that the source of the Interrupt within the Device is the IPIF DMA.
11. Read and verify the transfer status from the DMA Status Register has no errors reported.
12. Clear the 'Done' Interrupt in the DMA Interrupt Status Register.
13. Clear the Device Interrupt in the System Interrupt Controller.
14. Ready for next transfer.

Scatter Gather and Buffer Descriptors

Scatter/Gather adds additional automation to the IPIF DMA service. The function allows the User Application to setup automatic DMA operations via control and status data structures called Buffer Descriptors. The composition of a Buffer Descriptor is shown in Table 44. The last entry in each Buffer Descriptor is an address for the next Buffer Descriptor to be executed. Thus, they can be arranged as 'linked lists'. The list continues to be executed in sequence by the Scatter Gather function until a Buffer Descriptor is encountered that has the 'L' bit in the DMACR word set to'1'. When the Buffer Descriptor is finished, the Scatter Gather function terminates and waits for the User Application to restart it. It should be noted that a Scatter Gather Buffer Descriptor chain can link back to itself without termination. This will cause the Scatter Gather function to run continuously (once started) until it is aborted by the User Application.

Table 44: **Scatter Gather Buffer Descriptor Layout**

Entry Name	Descriptor Address Offset	Description
SR	+00	**Device Status Register Read Value**. This word provides device status to the User Application. This word in the Buffer Descriptor is overwritten, by the associated Scatter Gather automation, with the value read from the Device's associated Status FIFO. The bit layout of this word is up to the User IP and User Application. Note: This word is only overwritten if the execution of this Buffer Descriptor resulted in the completion of the Packet transfer. This is indicated by the 'L' bit in the DMACR word of this Buffer Descriptor being set to '1'.
DMACR	+04	**DMA Control Register Load Value**. The value to load into the associated DMA Channel's Control Register for the data transfer controlled by this Buffer Descriptor. The bit format matches DMA Control Register register definition.
SA	+08	**DMA Source Address Register Load Value**. The value to load into the associated DMA Channel's Source Address Register for the data transfer controlled by this Buffer Descriptor. The bit layout matches DMA Source Address Register register definition.
DA	+0C	**DMA Destination Address Load Register Value**. The value to load into the associated DMA Channel's Destination Address Register for the data transfer controlled by this Buffer Descriptor. The bit layout matches DMA Destination Address Register register definition.
LENGTH	+10	**DMA Length Register Load Value**. The value to load into the associated DMA Channel's Length Register for the data transfer controlled by this Buffer Descriptor. The value represents bytes and the format matches DMA Length Register definition.
DMASR	+14	**DMA Status Register Read Value**. This word provides DMA Channel status to the User Application. It is overwritten by the Scatter Gather automation at the completion of the data transfer controlled by this Buffer Descriptor with the value read from the associated DMA Channel's Status Register. The bit layout matches DMA Status Register definition.
BDA	+18	**Next Buffer Descriptor Address**. This value designates the system address of the next Buffer Descriptor to be executed by the associated Scatter Gather channel. The format matches DMA BDA Register definition.

Generally, a Buffer Descriptor is a set of values corresponding to DMA registers. When the Buffer Descriptor is executed, the Scatter Gather function fetches the values from the Buffer Descriptor storage address (usually in Main Memory) and loads the values into the associated DMA registers and starts the DMA transfer. When the transfer is completed, the Scatter Gather function reads status information from the DMA Status Register and writes it into the DMA status word (DMASR) in the Buffer Descriptor. The User Application can then retrieve this information to determine the success or failure of the DMA operations associated with the Buffer Descriptor. If the Buffer Descriptor is not the last one in a sequence, the Scatter Gather function fetches the BDA word from the Descriptor (the address of the next Buffer Descriptor to execute) and then sets up execution of the new Descriptor. When a Buffer Descriptor signals the last of a transfer sequence (the 'L' bit in the DMACR word set to'1' for Packet mode, the 'SGS' bit set to 1 in Simple Scatter Gather mode)), the Scatter Gather automation terminates at the completion of the Descriptor. The User Application can also terminate Scatter Gather operation by clearing the SGE bit in the DMA SWCR register. When this occurs, simple Scatter Gather will complete the current Buffer Descriptor and stop. In Packet Mode, Buffer Descriptors continue to be processed until the current packet has be transferred. Termination is signaled to the User Application with the assertion of the SGDA interrupt bit in the DMA Interrupt Status register.

Simple Scatter Gather

Simple Scatter Gather is defined as the automation of sequential simple DMA transfers via Buffer Descriptors. This operation mode is selected via the **C_DMA_CHAN_TYPE** parameter. An entry of 1 in this parameter configures a corresponding DMA channel for this type of operation. Once started, the Simple Scatter Gather Channel keeps fetching and executing the assigned Buffer Descriptor list until it is stopped by the User Application or it encounters a Buffer Descriptor with the SGS bit set in the DMACR word. Operation is stopped when the Buffer Descriptor's DMA data transfer (in progress) is completed. The User has to ensure when using simple Scatter Gather controlled DMA operations that source data is always available and the destination can always accept the data (at least until the sequence of Buffer Descriptors has completed).

The following sequence of operations shows operations the User Application must do to Enable the Scatter Gather End interrupt, initiate a simple Scatter Gather transfer, and perform completion housekeeping.

1. Setup the microprocessor's System Interrupt Controller (Enable the User Device interrupt).
2. Enable the DMA Service interrupt(s) in the IPIF Device Interrupt Enable Register (DMA0 and or DMA1).
3. Enable the Global Interrupt Enable (GIE) in the IPIF Global Interrupt Enable Register.
4. Enable the Scatter Gather 'End' Interrupt (ESGEND) in the DMA Interrupt Enable Register.
5. Set up the Buffer Descriptor list in system memory.
6. Write the system address of the first word of the first Buffer Descriptor to be executed into the DMA BDA register.
7. Clear the SGS bit in the DMA Control register.
8. Set the SGE bit in the DMA Software Control register. This will start the SG operation.
9. Wait for Device Interrupt to occur in the System Interrupt Controller.
10. Read the IPIF Device Interrupt Status Register and verify that the source of the Interrupt within the Device is the IPIF DMA channel of interest.
11. Read and verify that the SGEND interrupt is set in the DMA Interrupt Status register.
12. Read and verify the transfer status from the DMA Status Register has no errors reported.
13. Clear the SGEND Interrupt in the DMA Interrupt Status Register.

14. Clear the Device Interrupt in the System Interrupt Controller.
15. Ready for next Buffer Descriptor sequence.

Packet Mode Scatter Gather and Hardware Architecture Implications

Packet Scatter Gather is used to provide further automation to applications that are data packet based such as Ethernet. Currently, the IPIF supports one or two packet channels. The most common usage is a two channel approach where one channel is parameterized as a Transmit Scatter Gather channel (Tx) and the other is a Receive Scatter Gather channel (Rx). This operational behavior is selected via the **C_DMA_CHAN_TYPE** parameter with an entry value of 2 or 3 respectively for the corresponding DMA channel**.** Each channel requires it's own set of Buffer Descriptors to control operation. In addition, the nature of Tx and Rx data transfers requires different operational procedure between Tx and Rx Scatter Gather channels.

In order to understand these differences, it is necessary to be familiar with the hardware architecture that is required by the Packet Scatter Gather Service. A two channel Device is shown in Figure 49. The key architectural features are the presence of the WrFIFO and the RdFIFO in the IPIF and Status and Length elements (usually small SRL based FIFOs but it can be a register) in the User IP. Special note should be taken of the associated status signals to the DMA/Scatter Gather Service from these elements. These are required for Packet Scatter Gather operation.

Each Scatter Gather channel requires a length element. The length element provides two functions. First, it stores information about the length of each data packet stored in the IPIF FIFOs that needs to be processed. The length value represents the number of bytes comprising a data packet. It is only written into the length element after the completion of the data transfer of the full packet into the associated IPIF FIFO. Second, the length element provides a handshake mechanism (via the Empty flag) by which packet transfers are initiated between the USER IP and the Scatter Gather Service. The presence of a value (length FIFO is not 'Empty') in the length element is used to initiate transfer activity by the 'Read' logic. For the Tx direction, the 'Read' logic is the User IP. It must read the length value from the Tx Length FIFO (when one is present) and then Read the associated packet data from the IPIF WrFIFO. In the Rx direction, the Scatter Gather Rx channel reads the Rx Length value from the Rx Length FIFO (when one is present) and then activates the packet data transfer from the IPIF RdFIFO to the Destination by Buffer Descriptor control of the Rx DMA channel.

From the point of view of the Scatter Gather Service, the status signal from the Length element is different based on whether it is part of a Tx channel or an Rx channel. The Tx Length element transfers information written by the Tx Scatter Gather function (via the IPIC interface) to the User IP logic. It therefore provides a 'Full' indication to the Scatter Gather Tx channel to throttle writing if it becomes full. The Rx Length element transfers information from the User IP logic to the Scatter Gather function (also via the IPIC interface). This element provides an 'Empty' status to the Rx Scatter Gather channel to throttle additional reading when it is empty or to initiate servicing when it is not Empty. The Scatter Gather channel also utilizes the WrFIFO Vacancy status to throttle DMA operations writing the WrFIFO if it should become Full. This is an internal IPIF connection.

Paired with each length element is a status element. The status value is User defined and it is always written or provided by the User IP and is Read by the associated Scatter Gather channel. In the case of a Tx channel, the Status is written when the associated Tx Packet data has been transmitted and the status of the transmission is known. The Tx Scatter Gather channel will wait for the Tx Status to be present before the Tx Packet transfer is considered completed and a new one can start. In the Rx direction, the Rx status is written just prior to the Rx Length value being written into the Length element. In either case, the Scatter Gather reads the value and then writes it to the pertinent Buffer

Descriptor SA word. It is up to the User Application to retrieve it from the Buffer Descriptor and process it.

The following sequence of operations shows operations the User Application must do to set up and run a Packet Mode Scatter Gather operation. In this case, Scatter Gather interrupt coalescing is not utilized (Packet Wait Bound and Packet Count Threshold are not used).

1. Setup the microprocessor's System Interrupt Controller (Enable the User Device interrupt).
2. Enable the DMA Service interrupt(s) in the IPIF Device Interrupt Enable Register (DMA0 and or DMA1).
3. Enable the Global Interrupt Enable (GIE) in the IPIF Global Interrupt Enable Register.
4. Enable the Packet Done Interrupt (PD) in the DMA Interrupt Enable Register of the Tx channel DMA and the Rx channel DMA.
5. Set up the Buffer Descriptor list in system memory for the Packet transmission. A separate list is required for the Tx channel and the Rx channel.
6. Write the system address of the first word of the first Buffer Descriptor to be executed into the associated DMA BDA register. One required for the Tx channel and one for the Rx channel.
7. Clear the SGS bit in the DMA Control register of each channel.
8. Set the SGE bit in the DMA Software Control register of each channel. This will start the SG operation. The Tx channel will begin processing the first Buffer Descriptor and shuttling data from the source address space to the WrFIFO. The Rx channel waits for the Rx Length element to have an entry in it before transfer activity occurs.
9. Wait for Device Interrupt to occur in the System Interrupt Controller.
10. Read the IPIF Device Interrupt Status Register and verify that the source of the Interrupt within the Device is the IPIF DMA channel of interest (Tx or Rx).
11. Read and verify that the Packet Done (PD) interrupt is set in the DMA Interrupt Status register.
12. Read and verify the transfer status of the Packet. This is done by reading the First word of the first Buffer Descriptor for the associated channel. The status value will be an echo of the value the User IP wrote into the Status element for the associated Scatter Gather channel.
13. Clear the PD Interrupt in the DMA Interrupt Status Register for the channel.
14. Clear the Device Interrupt in the System Interrupt Controller.
15. Ready for next packet done interrupt.

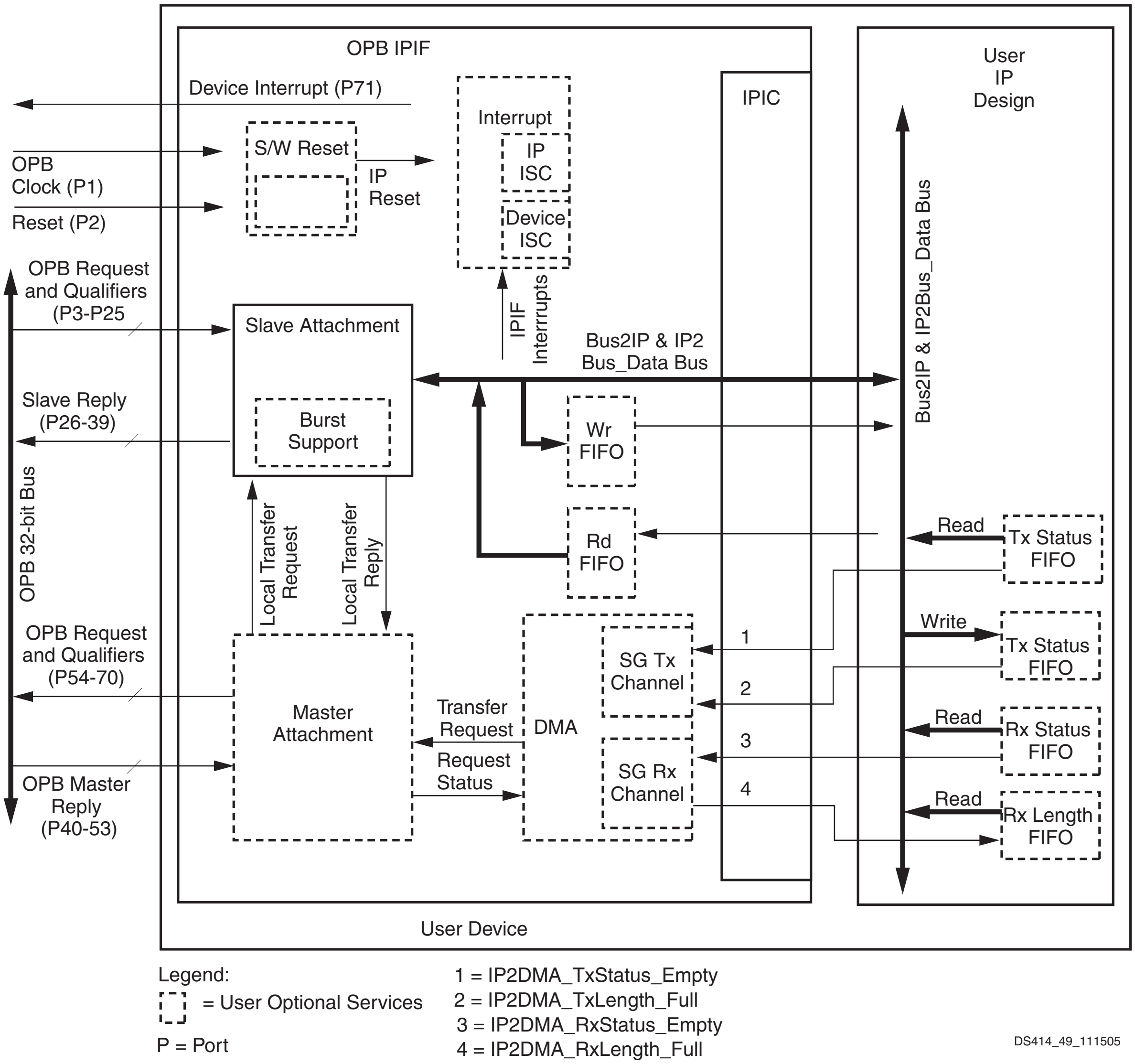


Figure 49: **Device Architecture for Packet Scatter Gather Service Support**

Scatter Gather Interrupts

The Scatter Gather Service is the most complex and dynamic element of the IPIF. Even though it provides DMA transfer automation to off load the User Application, it still requires management and servicing by the User Application. Service requests and status queues are relayed to the User Application via interrupts. This mechanism utilizes the Interrupt Service built into the OPB IPIF and the DMA/Scatter Gather Service. The User should refer to Xilinx Document *DS-416 Direct Memory Access and Scatter Gather* for information on interrupt setup and use during Scatter Gather operation. The User should also note that the use of the Scatter Gather Service does not preclude the User IP from generating it's own interrupts that can provide additional synchronization and status to the User Application operations.

Using the IP Master Interface

The OPB IPIF IPIC interface provides a mechanism for the User IP to initiate OPB read and write operations. The portion of the IPIC used for this Service is the IP Master ports group. When the User specifies that an IP Master interface is required via the C_IP_MASTER_PRESENT parameter, the OPB IPIF will be generated with a Master Attachment Service and the IP Master ports made operational.

IP Master Operations

The IP Master interface to the OPB IPIF is simple request/transfer acknowledge protocol. The signal set is detailed in the port descriptions in Table 9, "OPB IPIF I/O Signals," on page 21 in the IP Master Signals group. The IP Master requests an operation by raising the desired request signal (IP2Bus_MstRdReq for reads or IP2Bus_MstWrReq for writes). Simultaneously with the assertion of the request, the IP Master must also drive the desired transfer qualifiers for the transaction (address, byte enables, burst, bus lock, etc.) until the transfer is completed.

The interface is built upon the dual access transfer mechanism of the OPB IPIF Master Attachment and Slave Attachment. The User should notice that the IP Master signal set does not include any data buses. That is because the data transfer utilizes the IPIC data buses and the Master Attachment/Slave Attachment interfaces. Since this transfer approach is comprised of a local operation and a Bus operation, the IP Master is required to provide an address for the local resource (IP2IP_Addr) and one for the Bus resource (IP2Bus_Addr). The User must also provide byte enable signals (BE) for the Bus address for single beat transactions. The local transfer BE is derived from the Bus BE (IP2Bus_BE) and the local address (IP2IP_Addr).

In the case of an IP Master Read request, the Master Attachment schedules a Read transfer with the OPB Bus and a Write transfer with the local Slave Attachment. The data is read from the Bus target at the address specified by the IP2Bus_Addr signal bus. The read data comes into the IPIF via its OPB data bus and is handed off to the Slave Attachment. The Slave Attachment then initiates a local write transfer via the IPIC interface and protocol to the address specified by the IP2IP_Addr bus. The write address output on the IP2IP_Addr by the IP Master must be a valid system address for the local target memory space.

An IP Master Write request is the opposite. The Master Attachment first schedules a Read transfer with the local Slave Attachment. The data is read by the Slave Attachment (via the IPIC signals and protocol) from the local address specified by the IP2IP_Addr signal bus and is made ready to present to the OPB. The Master Attachment then initiates the Write transfer to the Bus address target specified by the IP2Bus_Addr bus. As mentioned earlier, local address (the read address in this case) set on the IP2IP_Addr by the IP Master must be a valid system address for the local source memory space.

For this version of the IPIF, only master read transactions with 32-bit data are supported. For master write transactions, the local IPIC read is always at 32 bits, but the master's IP2Bus_MstBE value is transmitted to the bus to qualify the data being written.

Master Request Arbiter

If a DMA/SG Service is enabled along with IP Master support, an arbitration function is automatically included in the OPB IPIF. The arbitration function is omitted if the DMA is enabled without the IP Master or vice versa. The Arbiter is needed to regulate access between the DMA and the IP Master should simultaneous requests be present. By default, the Arbiter uses a fair arbitration algorithm that transfers the next priority away from the just-serviced requester. Alternatively, the C_MASTER_ARB_MODEL parameter can be used to statically fix the priority to either requester.

The activation of master reply signals to the IP Master is the indication that its request has been granted access to the OPB.

Master Transactions

Figure 50, Figure 51, and Figure 52 give examples of various master transactions.

The master starts a transaction by asserting a read or write request signal (IP2Bus_MstRdReq or IP2Bus_MstWrReq). Each transfer (there will be more than one if the transaction is a burst) is acknowledged by either Bus2IP_MstRdAck or Bus2IP_MstWrAck. This *transfer acknowledgement* indicates that the data has reached its final destination.

The last acknowledgement (the only one for a single transaction) is accompanied by the *transaction acknowledgement* signal, Bus2IP_MstLastAck.

Exceptional transaction completion can occur as an *error*, *retry* or *timeout*. An **error** is indicated by a cycle in which both Bus2IP_MstError and Bus2IP_MstLastAck are asserted. A **retry** is indicated by a one-cycle assertion of Bus2IP_MstRetry, during which, Bus2IP_MstLastAck serves to differentiate two situations that may exist at the point of retry. The first of these situations, *clean retry* (Bus2IP_MstLastAck=1), exists when all data read from the IPIC—although it is not all of the data requested—has been written to the OPB. The second situation, *dirty retry* (Bus2IP_MstLastAck=0), exists when some IPIC data that has been read has not been written. A **timeout** is indicated by a single cycle assertion of Bus2IP_MstTimeout.

Two cases are defined for the IP core's continuation behavior after it receives a retry response. In the first case, called *retained-state-retry continuation*, it continues with the same transaction. The OPB IPIF will retain the state that it may have built up to support the transaction. It will re-arbitrate for the OPB and continue the transaction from the point that the retry was returned.

In the second case, called *abort-on-retry continuation*, the transaction is aborted and any state built up to support the transaction is discarded. The cases are differentiated by the value of the request signal, whether IP2Bus_MstRdReq or IP2Bus_MstWrReq, on the cycle immediately after the OPB IPIF returns Bus2IP_MstRetry. If the request remains asserted in this cycle, the transaction is continued using the retained-state retry mechanism, otherwise it is aborted.

Typically, retained-state-retry continuation is appropriate under dirty-retry status—particularly if the IPIC reads are destructive (e.g. from a FIFO) and if it is okay to continue to block access to the OPB for the IPIF-resident DMA master, if one is present. On the other hand, abort-on-retry continuation is required under clean-retry status. This in turn means that the IP master generally should use the transfer acknowledgements to track the progress of the master transaction to keep a consistent view of the amount of data that has been transferred.

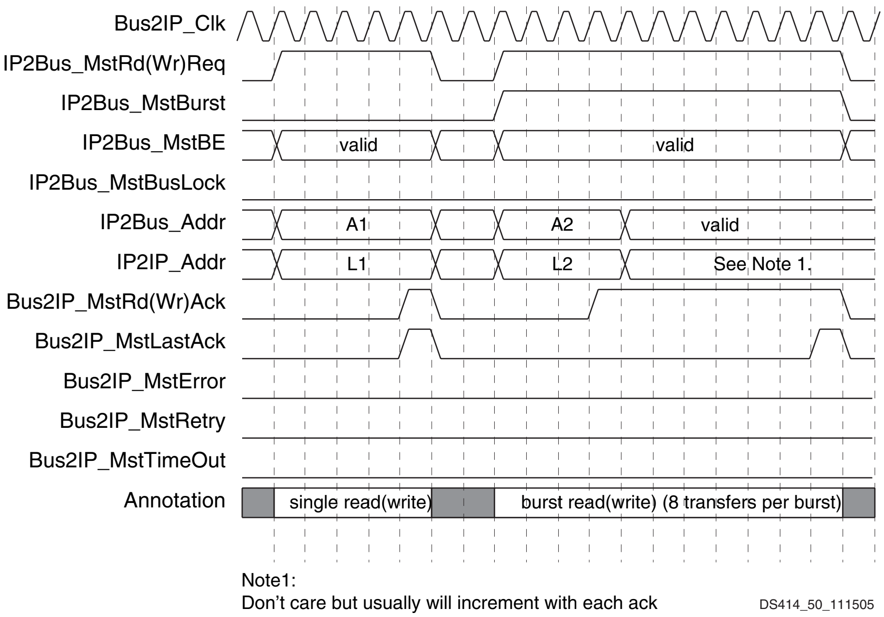


Figure 50: **Master Single and Burst Transactions**

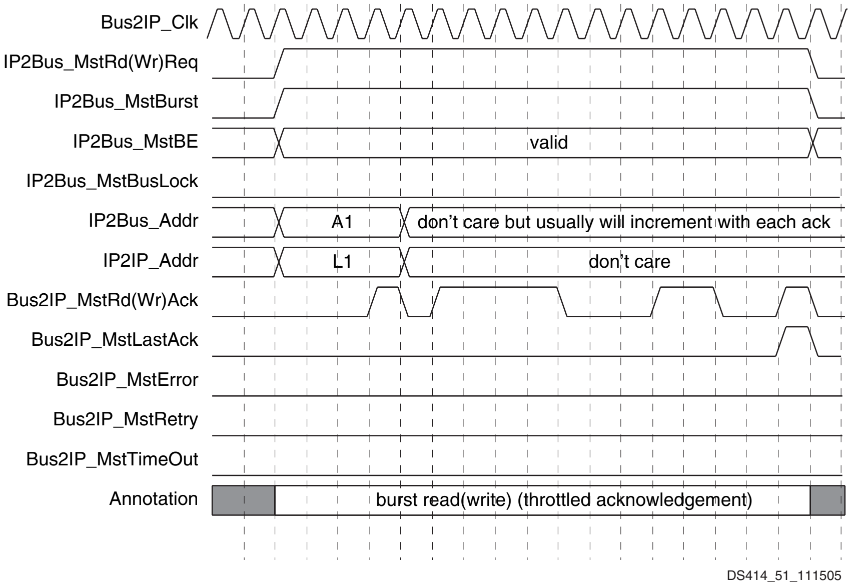


Figure 51: **Master Burst Transaction with Throttled Acknowledgments**

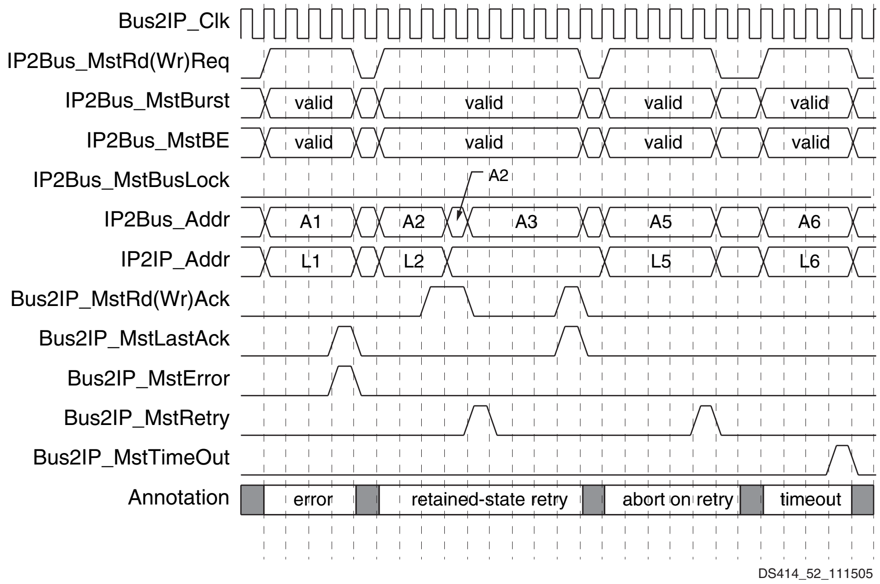


Figure 52: **Master Transactions with Exceptional Responses of Error, Retry, and Timeout**

IP Master Bus Locking

IP Master bus locking is not supported by this version of the OPB IPIF. The active-high IP2Bus_MstBusLock signal should not be asserted by the User.

The lock is also recognized by the Master Request Arbiter if it is present. The Arbiter will go into a locked state when it gives access to the IP Master. If a DMA access is in progress at the time of the Lock request, the DMA access will be allowed to complete before the Arbiter locks out the DMA requests. The DMA will continue to be locked out until the IP2Bus_MstBusLock signal is de-asserted by the IP Master.

Bus2IP_IPMstTrans Signal

The dual access nature of the OPB IPIF Master Attachment can sometimes pose an operational dilemma to the IP Master. There are scenarios when the IP Master needs to know when a read or write transaction request appearing on the IPIC interface (with the User IP) is caused as a result of its own master request or that of some remote OPB Master. This information is conveyed with the Bus2IP_IPMstTrans signal. It is output by the IPIF for use by the User's IP. When it is asserted, it is indicating that a local IPIC operation is in progress and it is part of the IP Master's request execution.

User Application Topics

Understanding and Using IPIC Chip Selects and Chip Enables.

Implementing Chip Select (CS) and Chip Enable (CE) signals is a common design task that is needed within microprocessor based systems to qualify the selection of registers, ports, and memory via an address decoding function. The OPB IPIF implements a flexible technique for providing these signals to Users via the ARD parameters. As such, the User must understand the relationship between the population of the ARD array parameters and the Bus2IP_CS, the Bus2IP_CE, the Bus2IP_RdCE, and the Bus2IP_WrCE buses that are available to the User at the IPIC interface with the IPIF. An example of ARD Array population and the resulting CS and CE bus generation is shown in Figure 53. The timing characteristics of these signals are shown in the section titled "Using the IP Slave Interface" on page 58.

The signal set to use for User IP functions is up to the User and the design requirements. Unused CE and CS signals and associated generation logic will be ’trimmed’ during synthesis and PAR phases of FPGA development.

Chip Select Bus (Bus2IP_CS(0:n)

A single Chip Select signal is assigned to each address space defined by the User in the ARD arrays. The Chip Select is asserted (active high) whenever a valid access (Read or Write) is requested of the address space. It remains asserted until the transfer between the IPIF Slave Attachment and the addressed target has completed. The User is provided the Bus2IP_CS port as part of the IPIF’s IPIC signal set. The Bus2IP_CS bus has a one to one correlation to the number and ordering of entries in the C_ARD_ID_ARRAY parameter. For example, if the C_ARD_ID_ARRAY has 5 entries in it, the Bus2IP_CS bus will be sized as 0 to 4. Bus2IP_CS(0) will correspond to the first address space, Bus2IP_CS(1) to the second address space, and so on. Note that this implementation will result in the Bus2IP_CS bus having chip selects that will be assigned to any enabled IPIF internal service address spaces. The User will have to take this into account when connecting to the bus. The nature of the Chip Select bus requires the User IP to provide any additional qualification with Bus2IP_Addr, Bus2IP_RNW, Bus2IP_WrReq or Bus2IP_RdReq that might be needed.

Chip Enable Bus (Bus2IP_CE(0:y)

Each address space defined in the ARD arrays is allowed to have 1 or more Chip Enable signals assigned to it. Chip Enables are used for subdividing an address space into smaller spaces that are each less than or equal to the OPB Bus width. Generally this is useful for selecting registers and ports during read or write transactions. The OPB IPIF allows the User to do this via parameters entered in the C_ARD_NUM_CE_ARRAY. For each defined address space, the User enters the number of desired Chip Enable signals to be generated for each space. The data width of the space as defined in the C_ARD_DWIDTH_ARRAY determines the size of the address slice assigned to each CE signal for the address space.

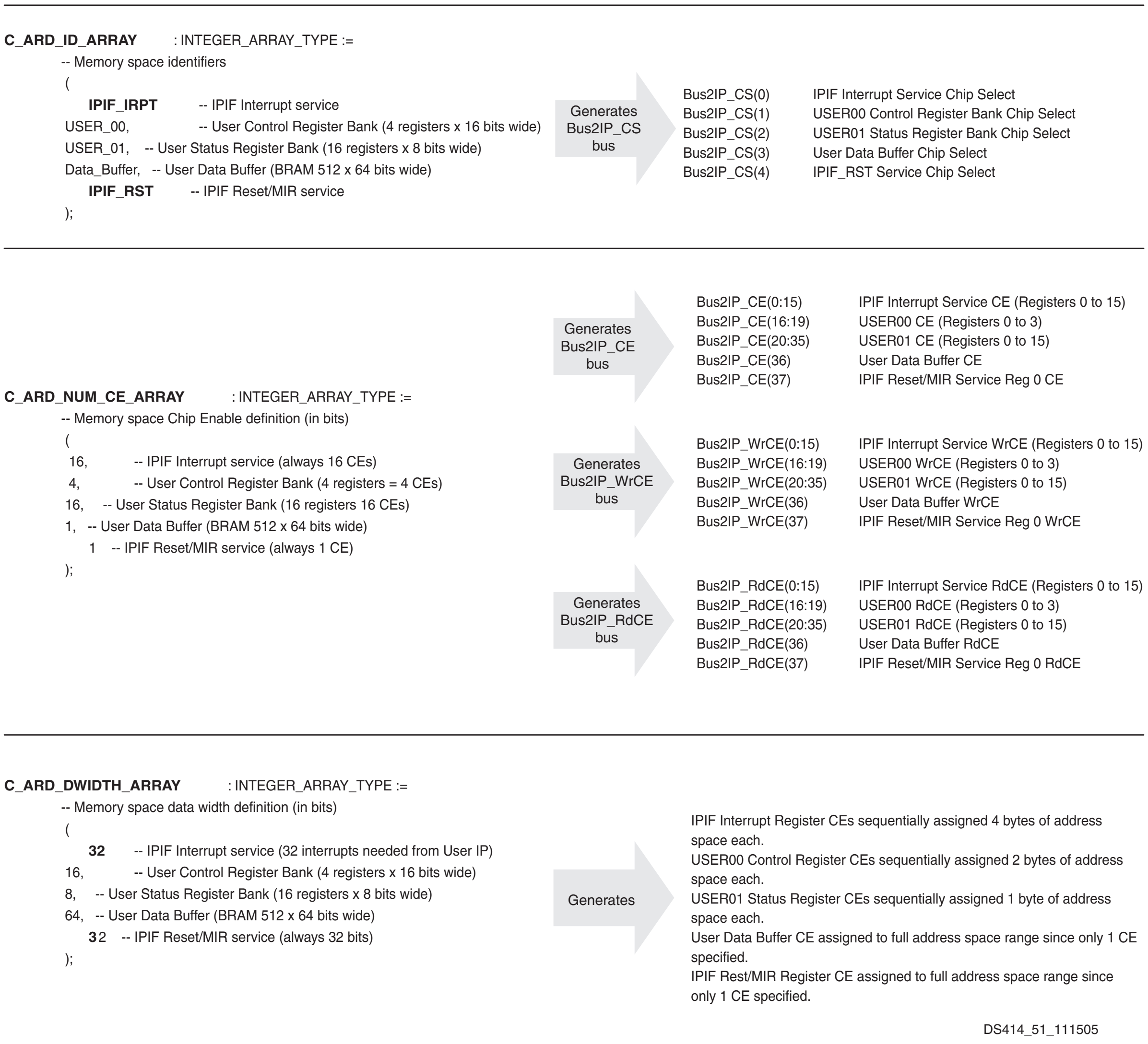
A value of at least 1 is required for each address space’s C_ARD_NUM_CE_ARRAY entry and if this minimal value is used the single Chip Enable signal becomes identical in function to the Chip Select signal for the address space to which the Chip Enable belongs. This represents a special case of a general aliasing rule in which all addresses of the form *BA* + *i***N* will activate the same Chip Enable, where *N* is the smallest power of two that is less than or equal to the number of Chip Enables requested, *BA* is the "base address" for the Chip Enable, and *i* ranges between zero and the largest value that allows the aliased address to still fall within the address space.

Read Chip Enable Bus (Bus2IP_RdCE(0:y)

The Bus2IP_RdCE bus is the same as the Bus2IP_CE bus except that the Bus2IP_RdCE signals are only asserted if the requested transaction is a read.

Write Chip Enable Bus (Bus2IP_WrCE(0:y)

The Bus2IP_WrCE bus is the same size as the Bus2IP_CE bus except that the Bus2IP_WrCE signals are only asserted if the requested transaction is a write.



```
C_ARD_ID_ARRAY        : INTEGER_ARRAY_TYPE :=
        -- Memory space identifiers
        (
           IPIF_IRPT        -- IPIF Interrupt service
        USER_00,            -- User Control Register Bank (4 registers x 16 bits wide)
        USER_01,   -- User Status Register Bank (16 registers x 8 bits wide)
        Data_Buffer,  -- User Data Buffer (BRAM 512 x 64 bits wide)
           IPIF_RST        -- IPIF Reset/MIR service
        );
```

Bus2IP_CS(0)	IPIF Interrupt Service Chip Select
Bus2IP_CS(1)	USER00 Control Register Bank Chip Select
Bus2IP_CS(2)	USER01 Status Register Bank Chip Select
Bus2IP_CS(3)	User Data Buffer Chip Select
Bus2IP_CS(4)	IPIF_RST Service Chip Select

```
C_ARD_NUM_CE_ARRAY        : INTEGER_ARRAY_TYPE :=
        -- Memory space Chip Enable definition (in bits)
        (
         16,          -- IPIF Interrupt service (always 16 CEs)
         4,           -- User Control Register Bank (4 registers = 4 CEs)
        16,     -- User Status Register Bank (16 registers 16 CEs)
        1,   -- User Data Buffer (BRAM 512 x 64 bits wide)
           1   -- IPIF Reset/MIR service (always 1 CE)
        );
```

Bus2IP_CE(0:15)	IPIF Interrupt Service CE (Registers 0 to 15)
Bus2IP_CE(16:19)	USER00 CE (Registers 0 to 3)
Bus2IP_CE(20:35)	USER01 CE (Registers 0 to 15)
Bus2IP_CE(36)	User Data Buffer CE
Bus2IP_CE(37)	IPIF Reset/MIR Service Reg 0 CE

Bus2IP_WrCE(0:15)	IPIF Interrupt Service WrCE (Registers 0 to 15)
Bus2IP_WrCE(16:19)	USER00 WrCE (Registers 0 to 3)
Bus2IP_WrCE(20:35)	USER01 WrCE (Registers 0 to 15)
Bus2IP_WrCE(36)	User Data Buffer WrCE
Bus2IP_WrCE(37)	IPIF Reset/MIR Service Reg 0 WrCE

Bus2IP_RdCE(0:15)	IPIF Interrupt Service RdCE (Registers 0 to 15)
Bus2IP_RdCE(16:19)	USER00 RdCE (Registers 0 to 3)
Bus2IP_RdCE(20:35)	USER01 RdCE (Registers 0 to 15)
Bus2IP_RdCE(36)	User Data Buffer RdCE
Bus2IP_RdCE(37)	IPIF Reset/MIR Service Reg 0 RdCE

```
C_ARD_DWIDTH_ARRAY        : INTEGER_ARRAY_TYPE :=
        -- Memory space data width definition (in bits)
        (
           32       -- IPIF Interrupt service (32 interrupts needed from User IP)
        16,         -- User Control Register Bank (4 registers x 16 bits wide)
        8,    -- User Status Register Bank (16 registers x 8 bits wide)
        64,   -- User Data Buffer (BRAM 512 x 64 bits wide)
           32   -- IPIF Reset/MIR service (always 32 bits)
        );
```

IPIF Interrupt Register CEs sequentially assigned 4 bytes of address space each.
USER00 Control Register CEs sequentially assigned 2 bytes of address space each.
USER01 Status Register CEs sequentially assigned 1 byte of address space each.
User Data Buffer CE assigned to full address space range since only 1 CE specified.
IPIF Rest/MIR Register CE assigned to full address space range since only 1 CE specified.

Figure 53: **ARD Arrays and CS/CE Relationship Example.**

Available Support Functions for Automatic Ripping of CE and CS Buses.

The User may find it convenient to use some predefined functions developed by Xilinx to automatically rip signals from the Bus2IP_CS, Bus2IP_CE, Bus2IP_WrCE, and Bus2IP_RdCE buses. These functions facilitate bus ripping regardless of order or composition and mix of IPIF Services and User functions in the ARD Arrays. This is extremely useful if User parameterization adds or removes IPIF services and User IP functions (which changes the size and ordering of the CS and CE buses). Table 45 lists and details these functions. These functions are declared and defined in the ipif_pkg.vhd source file that is located in the Xilinx EDK at the following path:

\EDK\hw\iplib\pcores\ipif_common_v1_00_b\hdl\vhdl\ipif_pkg.vhd.

The following library declaration must appear in the User's VHDL source:

library ipif_common_v1_00_b;

use ipif_common_v1_00_b.ipif_pkg.all;

An example of how these functions are used is shown in Figure 54.

Table 45: **IPIF Support VHDL Functions.**

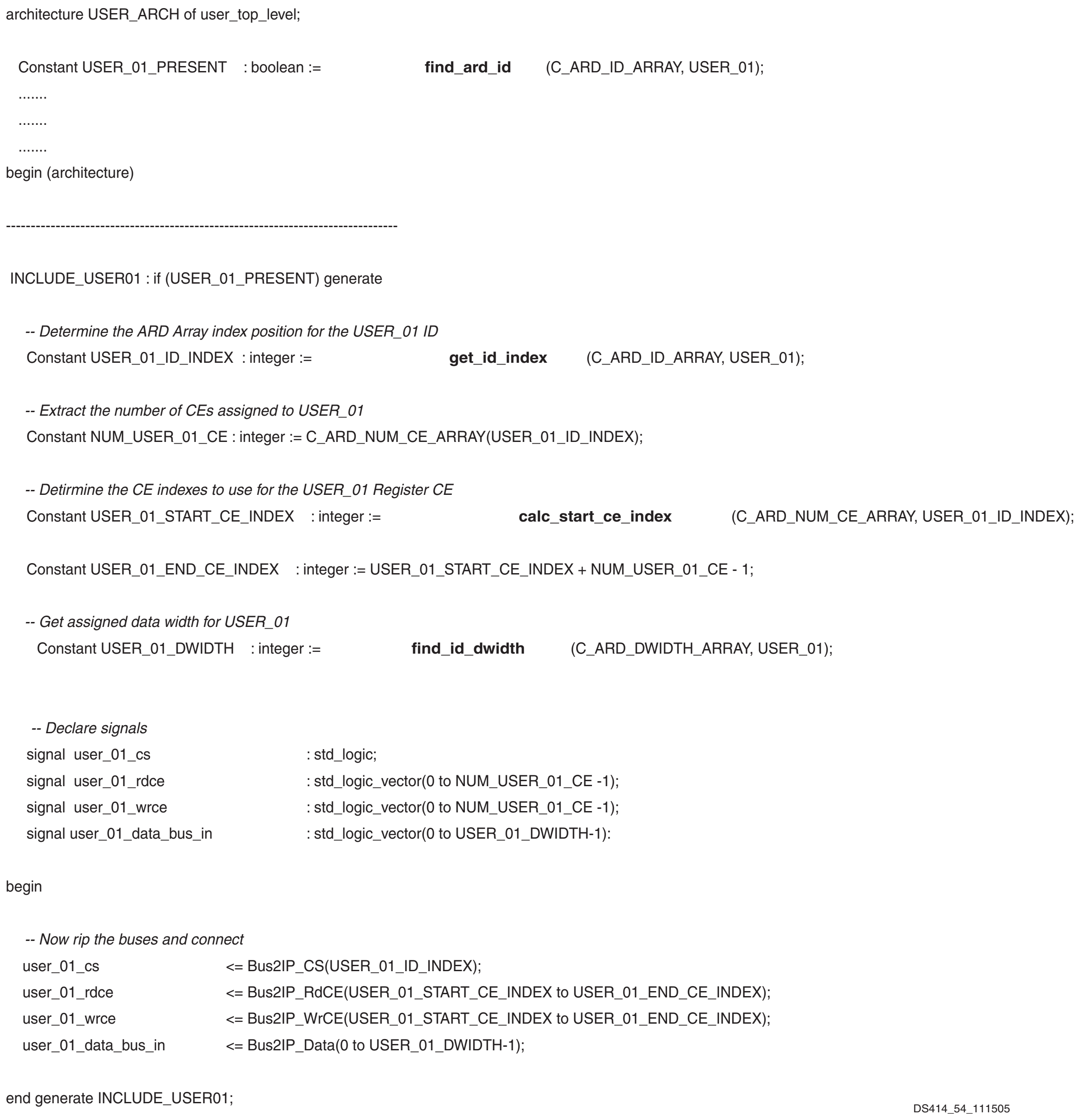
VHDL Function Name	Input Parameter Name	Input Parameter Type	Return Type	Description
find_ard_id			boolean	This function is used to determine if a particular entry exists in the 'id_array' parameter. The return value can then be used to perform conditional operations or build conditional logic structures using VHDL IF-Generates. Example: constant USER0_PRESENT : boolean := **find_ard_id**(C_ARD_ID_ARRAY, USER0);
	id_array	INTEGER_ ARRAY_TYPE		
	id	integer		

get_id_index			Integer	This function is used to get the array index value for an entry in the 'id_array' parameter. The return value can then be used to index into other ARD arrays to get associated parameters for the 'id' address space. The returned value can also be used as the index into the Bus2IP_CS bus for the 'id' address space. If the 'id' is not found in the 'id_array', a value of 10,000 is returned which will usually cause a simulation or build error and allow the problem to be found and fixed.
				Example:
				constant USER0_ARRAY_INDEX : integer := **get_id_index**(C_ARD_ID_ARRAY, USER0);
				constant USER0_CS_INDEX : integer := **get_id_index**(C_ARD_ID_ARRAY, USER0);

	id_array	INTEGER_ ARRAY_TYPE		
	id	integer		
get_id_index_iboe			Integer	This function is similar to get_id_index, except that it returns an "in bounds" index of zero (rather than 10,000) when the 'id' is not found in the 'id_array'. This in turn allows to avoid simulation or build errors in the infrequent cases where a 'get_id_index' function must be called even for an 'id' that is not used. When this happens it is advised that a separate guard based on the find_ard_id function be employed to avoid build errors. Example: constant USER0_INDEX : integer := **get_id_index_iboe**(C_ARD_ID_ARRAY, USER0);
	id_array	INTEGER_ ARRAY_ TYPE		
	id	integer		

Table 45: **IPIF Support VHDL Functions. *(Contd)***

VHDL Function Name	Input Parameter Name	Input Parameter Type	Return Type	Description
calc_num_ce			Integer	This function is used to get the total number of signals that make up each of the Bus2IP_CE, the Bus2IP_RdCE, and the Bus2IP_WrCE buses (they are all the same size and order). The information is derived from the ’ce_num_array’ parameter. Example: constant CE_BUS_SIZE : integer := **calc_num_ce**(C_ARD_NUM_CE_ARRAY) ;
	ce_num_array	INTEGER_ARRAY_TYPE		
calc_start_ce_index			Integer	This function is used to get the starting index of the CE or range of CEs to rip from the Bus2IP_CE, the Bus2IP_RdCE, and the Bus2IP_WrCE buses relating to the ’index’ value of the address space entry in the ARD Arrays. The information is derived from the ’ce_num_ce_array’ parameter and the returned value from the use of the **get_id_index** function. Example: constant USER0_START_CE_INDEX : integer := **calc_start_ce_index**(C_ARD_NUM_CE_ARRAY, USER0_INDEX);
	ce_num_array	INTEGER_ARRAY_TYPE		
	index	integer		
find_id_dwidth			Integer	This function is used to extract the entered data width entry from the ’dwidth_array’ parameter that corresponds to the ’id’ value. If the ’id’ value does not exist in the ’id_array’, the ’default’ value is returned. Example: constant USER0_DATA_WIDTH : integer := **find_id_dwidth**(C_ARD_ID_ARRAY, C_ARD_DWIDTH_ARRAY, USER0, 64);
	id_array	INTEGER_ARRAY_TYPE		
	dwidth_array	INTEGER_ARRAY_TYPE		
	id	Integer		
	default	Integer		



```
architecture USER_ARCH of user_top_level;

  Constant USER_01_PRESENT    : boolean :=            find_ard_id       (C_ARD_ID_ARRAY, USER_01);
  .......
  .......
  .......
begin (architecture)

--------------------------------------------------------------------------------

 INCLUDE_USER01 : if (USER_01_PRESENT) generate

    -- Determine the ARD Array index position for the USER_01 ID
    Constant USER_01_ID_INDEX  : integer :=              get_id_index        (C_ARD_ID_ARRAY, USER_01);

    -- Extract the number of CEs assigned to USER_01
    Constant NUM_USER_01_CE : integer := C_ARD_NUM_CE_ARRAY(USER_01_ID_INDEX);

    -- Detirmine the CE indexes to use for the USER_01 Register CE
    Constant USER_01_START_CE_INDEX    : integer :=             calc_start_ce_index          (C_ARD_NUM_CE_ARRAY, USER_01_ID_INDEX);

    Constant USER_01_END_CE_INDEX    : integer := USER_01_START_CE_INDEX + NUM_USER_01_CE - 1;

    -- Get assigned data width for USER_01
      Constant USER_01_DWIDTH    : integer :=           find_id_dwidth        (C_ARD_DWIDTH_ARRAY, USER_01);


     -- Declare signals
    signal  user_01_cs                      : std_logic;
    signal  user_01_rdce                    : std_logic_vector(0 to NUM_USER_01_CE -1);
    signal  user_01_wrce                    : std_logic_vector(0 to NUM_USER_01_CE -1);
    signal user_01_data_bus_in              : std_logic_vector(0 to USER_01_DWIDTH-1):

begin

    -- Now rip the buses and connect
   user_01_cs                 <= Bus2IP_CS(USER_01_ID_INDEX);
   user_01_rdce               <= Bus2IP_RdCE(USER_01_START_CE_INDEX to USER_01_END_CE_INDEX);
   user_01_wrce               <= Bus2IP_WrCE(USER_01_START_CE_INDEX to USER_01_END_CE_INDEX);
   user_01_data_bus_in        <= Bus2IP_Data(0 to USER_01_DWIDTH-1);

end generate INCLUDE_USER01;
```

DS414_54_111505

Figure 54: **Bus Ripping Example.**

Understanding Byte Steering

The OPB IPIF incorporates a Service for automatic data steering to support those Target functions that have data widths less than the OPB Bus data width. The Byte Steering works in both read and write directions and it's operation is dynamically configured as each defined address space for the IPIF is accessed. The operation is set by the access type (Read or Write) and via the address space data width information provided by the C_ARD_DWIDTH_ARRAY parameter. The Byte Steering design requires that target functions must connect to the Bus2IP_Data, IP2Bus_Data, and Bus2IP_BE interfaces in ascending Byte Lane order. The Byte Steering operation during read and write operations with targets of supported data widths (8, 16, and 32) is shown in Table 46 through Table 51.

Table 46: **Write to 8-Bit Target by a 32-Bit OPB IPIF**

Access Type (1)	**Bus Write Data Bytes**				**Bus BE**				**Bus2IP_Data**				**Bus2IP_BE**			
Byte Write	BL 0	BL 1	BL 2	BL 3	BE 0	BE 1	BE 2	BE 3	BL 0	BL 1	BL 2	BL 3	BE 0	BE 1	BE 2	BE 3
	D0	D1	D2	D3	1	0	0	0	D0	U	U	U	1	0	0	0
	D0	D1	D2	D3	0	1	0	0	D1	U	U	U	1	0	0	0
	D0	D1	D2	D3	0	0	1	0	D2	U	U	U	1	0	0	0
	D0	D1	D2	D3	0	0	0	1	D3	U	U	U	1	0	0	0

Notes:

1. Half-word and Word accesses are not valid with an 8-bit wide target. Only byte accesses are valid.

Table 47: **Write to 16-Bit Target by a 32-Bit OPB IPIF**

Access Type (1)	**Bus Write Data Bytes**				**Bus BE**				**Bus2IP_Data**				**Bus2IP_BE**			
Byte Write	BL 0	BL 1	BL 2	BL 3	BE 0	BE 1	BE 2	BE 3	BL 0	BL 1	BL 2	BL 3	BE 0	BE 1	BE 2	BE 3
	D0	D1	D2	D3	1	0	0	0	D0	U	U	U	1	0	0	0
	D0	D1	D2	D3	0	1	0	0	U	D1	U	U	0	1	0	0
	D0	D1	D2	D3	0	0	1	0	D2	U	U	U	1	0	0	0
	D0	D1	D2	D3	0	0	0	1	U	D3	U	U	0	1	0	0
Half-Word Write	**D0**	**D1**	**D2**	**D3**	**1**	**1**	**0**	**0**	D0	D1	U	U	1	1	0	0
	D0	**D1**	**D2**	**D3**	**0**	**0**	**1**	**1**	D2	D3	U	U	1	1	0	0

Notes:

1. Word accesses are not valid with a 16-bit wide target. Only byte and half-word accesses are valid.

Table 48: **Write to 32-Bit Target by a 32-Bit OPB IPIF**

Access Type (1)	Bus Write Data Bytes				Bus BE				Bus2IP_Data				Bus2IP_BE			
	BL 0	BL 1	BL 2	BL 3	BE 0	BE 1	BE 2	BE 3	BL 0	BL 1	BL 2	BL 3	BE 0	BE 1	BE 2	BE 3
Byte Write	D0	D1	D2	D3	1	0	0	0	D0	U	U	U	1	0	0	0
	D0	D1	D2	D3	0	1	0	0	U	D1	U	U	0	1	0	0
	D0	D1	D2	D3	0	0	1	0	U	U	D2	U	0	0	1	0
	D0	D1	D2	D3	0	0	0	1	U	U	U	D3	0	0	0	1
Half-Word Write	D0	D1	D2	D3	1	1	0	0	D0	D1	U	U	1	1	0	0
	D0	D1	D2	D3	0	0	1	1	U	U	D2	D3	0	0	1	1
Word Write	D0	D1	D2	D3	1	1	1	1	D0	D1	D2	D3	1	1	1	1

Notes:

1. All accesses, byte, half-word, and word accesses, are valid.

Table 49: **Read from an 8-Bit Target by a 32-Bit OPB IPIF**

Access Type (1)	Bus Read Data Bytes				Bus BE				IP2Bus_Data				Bus2IP_BE			
	BL 0	BL 1	BL 2	BL 3	BE 0	BE 1	BE 2	BE 3	BL 0	BL 1	BL 2	BL 3	BE 0	BE 1	BE 2	BE 3
Byte Read	D0	U	U	U	1	0	0	0	D0	U	U	U	1	0	0	0
	U	D0	U	U	0	1	0	0	D0	U	U	U	1	0	0	0
	U	U	D0	U	0	0	1	0	D0	U	U	U	1	0	0	0
	U	U	U	D0	0	0	0	1	D0	U	U	U	1	0	0	0

Notes:

1. Half-word and Word accesses are not valid with an 8-bit wide target. Only byte accesses are valid.

Table 50: **Read from a 16-Bit Target by a 32-Bit OPB IPIF**

Access Type (1)	Bus Read Data Bytes				Bus BE				IP2Bus_Data				Bus2IP_BE			
	BL 0	BL 1	BL 2	BL 3	BE 0	BE 1	BE 2	BE 3	BL 0	BL 1	BL 2	BL 3	BE 0	BE 1	BE 2	BE 3
Byte Read	D0	U	U	U	1	0	0	0	D0	U	U	U	1	0	0	0
	U	D1	U	U	0	1	0	0	U	D1	U	U	0	1	0	0
	U	U	D0	U	0	0	1	0	D0	U	U	U	1	0	0	0
	U	U	U	D1	0	0	0	1	U	D1	U	U	0	1	0	0
Half-Word Read	D0	D1	U	U	1	1	0	0	D0	D1	U	U	1	1	0	0
	U	U	D0	D1	0	0	1	1	D0	D1	U	U	1	1	0	0

Notes:

1. Word accesses are not valid with a 16-bit wide target. Only byte and half-word accesses are valid.

Table 51: **Read from 32-Bit Target by a 32-Bit OPB IPIF**

Access Type (1)	Bus Read Data Bytes				Bus BE				IP2Bus_Data				Bus2IP_BE			
	BL 0	BL 1	BL 2	BL 3	BE 0	BE 1	BE 2	BE 3	BL 0	BL 1	BL 2	BL 3	BE 0	BE 1	BE 2	BE 3
Byte Read	D0	U	U	U	1	0	0	0	D0	U	U	U	1	0	0	0
	U	D1	U	U	0	1	0	0	U	D1	U	U	0	1	0	0
	U	U	D2	U	0	0	1	0	U	U	D2	U	0	0	1	0
	U	U	U	D3	0	0	0	1	U	U	U	D3	0	0	0	1
Half-Word Read	D0	D1	U	U	1	1	0	0	D0	D1	U	U	1	1	0	0
	U	U	D2	D3	0	0	1	1	U	U	D2	D3	0	0	1	1
Word Read	D0	D1	D2	D3	1	1	1	1	D0	D1	D2	D3	1	1	1	1

Notes:

1. All accesses, byte, half-word, and word accesses are valid.

FPGA Design Application Hints

Single Entry in Unconstrained Array Parameters

As has been discussed previously, the OPB IPIF parameterization employes generics that are defined as unconstrained arrays, i.e. arrays whose size is left unbound at declaration and fixed later by the User. This is the underlying VHDL mechanism that allows OPB IPIF services and vector ports such as Bus2IP_CE to grow or shrink to the size required by the application. Generally, the size of the unconstrained array and its element values are fixed simultaneously by the User by assigning to the array a *constant aggregate*. The aggregate is simply a list of values enclosed in parentheses and separated by commas.

The list can take either of two forms, *positional association*, in which the values at indices in the array are populated by the aggregate elements as they appear, left to right, or *named association*, in which each value populates an index to which it is explicitly assigned by being proceeded by "*INDX* => ". Thus, the following positional and named aggregates are identical: (4, 3, 9) and (0 => 4, 1 => 3, 2 =>9).

The reader should be aware that for aggregates with a single element, positional association is unfortunately not allowed. The reason for this is that otherwise the syntax would be ambiguous with a parenthesized expression. Being aware of this VHDL restriction might save the User some aggravation should this usage case come up. The following example shows both the incorrect and the correct way to associate a single element to an unconstrained array:

C_ARD_ID_ARRAY => (USER_00); -- *VHDL* ***positional association*** *not allowed because it*

-- would be ambiguous with a parenthesized expression

C_ARD_ID_ARRAY => (0 => USER_00); -- *Use VHDL* ***named association, instead***

Specification Exceptions

The OPB IPIF supports only the 32-bit operational mode of the OPB. The IPIF's OPB interface does not support dynamic bus sizing using the transfer-size request and acknowledgement signals. Also, outputs to the OPB are qualified by device selection and driven to zero when the device is not selected, so the OPB's separate enable signals are not needed.

OPB & IPIC Signaling Exceptions

This OPB IPIF version does not use the signal set as indicated by grey cell shading in Table 9, "OPB IPIF I/O Signals," on page 21.

Reference Documents

The following documents contain supplemental information that might be of interest.

- IBM CoreConnect™ *64-Bit On-Chip Peripheral Bus, Architectural Specification* (v2.1).
- Xilinx LogiCORE *DS415 On-Chip Peripheral Bus IP Interface Packet FIFO* Product Specification.
- Xilinx LogiCORE *DS-413 OPB IPIF Interrupt* Product Specification.
- Xilinx LogiCORE *DS-416 Direct Memory Access and Scatter Gather* Product Specification.

Revision History

Date	Version	Revision
07/06/04	1.0	Initial Xilinx release
10/5/04	1.1	Updated trademarks and supported device families listing. Changed description of Bus2IP_Burst.
4/6/05	1.2	Updated trademarks and supported device listing for EDK 7.1.1 SP1.
11/9/05	1.3	Added section *Agents that Activate the Slave Interface*. Other minor changes.
11/17/05	1.4	Converted to new DS template; updated figures to Xilinx graphic standards; reformatted tables; updated cross references; reviewed trademark usage.
12/2/05	1.5	Added Spartan-3E to supported device family listing.
11/3/06	1.6	Added Virtex-4 to supported device family listing.