

## Introduction

The OPB IPIF is a continuation of the Xilinx family of IBM CoreConnect™ compatible LogiCORE™ products. It provides a bidirectional interface between a user IP core and the OPB 32-bit bus standard. The Xilinx OPB IPIF is available for use with various FPGA device families that support embedded hard or soft processors.

## 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.
  - Sequential-Address transfer support.
  - 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	
Version of Core	opb_ipif	v3.01c
Resources Used		
	Min	Max
Slices	27	544
LUTs	9	845
FFs	36	441
Block RAMs	0	4
Provided with Core		
Documentation	Product Specification	
Design File Formats	VHDL	
Constraints File	None	
Verification	EDK Simulation Support	
Instantiation Template	Wizard Support	
Reference Designs	N/A	
Design Tool Requirements		
Xilinx Implementation Tools	Xilinx EDK 6.2.2	
Verification	ModelSim SE 5.7b or later	
Simulation	ModelSim SE 5.7b or later	

© 2005 Xilinx, Inc. All rights reserved. All Xilinx trademarks, registered trademarks, patents, and further disclaimers are as listed at <http://www.xilinx.com/legal.htm>. All other trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

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.

## OPB IPIF Overview

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 in most aspects between IPIF modules in various versions for the OPB and the PLB for the features supported. This allows IP blocks using the IPIC to be adapted for either the OPB or the PLB for the features supported. 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 10 on page 18**.

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).

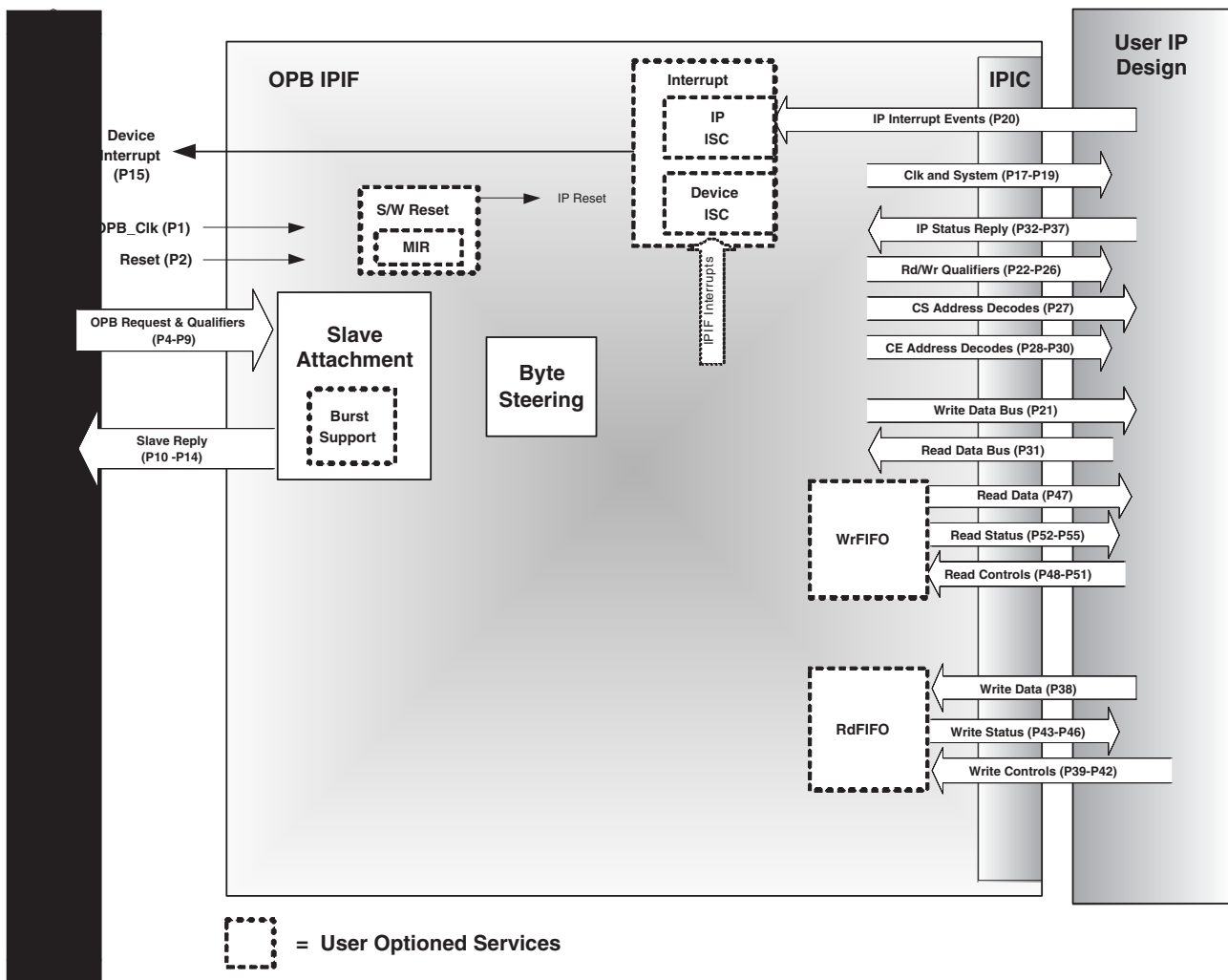


Figure 1: OPB IPIF Block Diagram

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 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 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.

## 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 the table.

*Table 1: OPB IPIF Design Parameters*

Grouping / Number	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 4 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 4 for description.	User must set values.	SLV64_ARRAY_TYPE(1)
	G3	Array of data widths for each target address space	C_ARD_DWIDTH_ARRAY	See "OPB IPIF Parameter Detailed Descriptions" on page 4 for description.	User must set values.	INTEGER_ARRAY_TYPE (1)
	G4	Array of the desired number of chip enables for each address space	C_ARD_NUM_CE_ARRAY	See "OPB IPIF Parameter Detailed Descriptions" on page 4 for description.	User must set values.	INTEGER_ARRAY_TYPE (1)
	G5	Array of unique properties for each address space specified	C_ARD_DEPENDENT_PROPS_ARRAY	See "OPB IPIF Parameter Detailed Descriptions" on page 4 for description.	User must set values.	DEPENDENT_PROPS_ARRAY_TYPE (1)

Table 1: OPB IPIF Design Parameters (Continued)

Grouping / Number	Feature / Description	Parameter Name	Allowable Values	Default Value	VHDL Type	
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	Specification of pipeline stages to be included in the IPIF	C_PIPELINE_MODEL	4, 5 or 7	User must set value	integer
	G10	Include or Exclude address counter useful to some burst transfers	C_INCLUDE_ADDR_CNTR	0 = Exclude (overriden when C_INCLUDE_WR_BUF = 1) 1 = Include	0	integer
	G11	Include or exclude write buffer that decouples OPB and IPIF write transactions.	C_INCLUDE_WR_BUF	0 = Exclude 1 = Include (also, force inclusion of the address counter)	0	integer
IP Interrupt Signal Specification	G12	Specification of the Number and type of interrupts from the IP.	C_IP_INTR_MODE_ARRAY	See "OPB IPIF Parameter Detailed Descriptions" on page 4 for description.	User must set.	INTEGER_ARRAY_TYPE (1)
OPB I/O Specification	G13	Width of the OPB Address Bus	C_OPB_AWIDTH	32	32	integer
	G14	Width of the OPB Data Bus	C_OPB_DWIDTH	32	32	integer
FPGA Family Type	G15	Xilinx FPGA Family	C_FAMILY	virtex2, virtex2p, virtex, spartan3	virtex2	string

**Notes:**

1. This Parameter VHDL type is a custom type defined in the ipif\_pkg.vhd.

## 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. Further, beyond the properties that are strictly needed for correct address-decoding, other properties can profitably be associated with different ranges of addresses. The solution for these requirements is a system of a variable number of *address ranges*, each with its own set of properties. We also use the term *address space* to refer to this notion of an address range with associated properties.

The user supplies an *Address Range Definition* for each address space to be included in the system. Each Address Range Definition actually has its properties spread across five parallel arrays. In other words, the properties are grouped by type,

with the properties of a given type for all Address Range Definitions placed in the same array. To accommodate the requirement for a variable number of address spaces, each of the five Address Range Definition arrays—recognizable by the common "C\_ARD" prefix—is a VHDL unconstrained array generic.

The concepts are developed more fully in what follows. 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 those defined address spaces during active OPB Bus transaction requests. Because 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 proc\_common\_v2\_00\_a 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 its 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 value USER\_00 smallest integer value beyond the block of values reserved for existing and future IPIF services.

The second category of identifier is the User Optional category. Starting with USER\_00, 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 with names descriptive of the corresponding address space. These user-defined identifiers can be associated with any value greater than or equal to USER\_00. 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 := USER\_00;
- Constant Status\_Regs : integer := USER\_00+1;
- Constant Data\_Buffer : integer := USER\_00+2;

Once declared, these identifiers can be entered into the C\_ARD\_ID\_ARRAY to provide a more descriptive identifier for user address spaces.

Table 2: Xilinx Predefined Address Space Identifiers

Category	Identifier Name	Identifier VHDL Type	Identifier Value	Identifier Description
IPIF Mandatory	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_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.
	IPIF_RDFIFO_REG	Integer	7	The presence of this identifier in the C_ARD_ID_ARRAY specifies the inclusion of the IPIF RdfFIFO register Service.
	IPIF_RDFIFO_DATA	Integer	8	The presence of this identifier in the C_ARD_ID_ARRAY specifies the inclusion of the IPIF RdfFIFO Service.
	Reserved IPIF	Integer	9 - 99	These identifier values are reserved for IPIF services that may be added in the future.
User Optional	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)
  IPIF_RST, -- IPIF Reset/MIR service
  IPIF_WRFIFO_REG, -- IPIF WRFIFO (two ARD entries for each FIFO)
  IPIF_WRFIFO_DATA,
  IPIF_RDFIFO_REG, -- IPIF WDFIFO (two ARD entries)
  IPIF_RDFIFO_DATA
);

```

In this example of a populated C\_ARD\_ID\_ARRAY entry, two User Address Ranges are defined and four IPIF services are being included. The four IPIF services are Interrupt, Reset/MIR, WRFIFO and RDFIFO. (Note that there are two Address Ranges per FIFO, one for the registers and one accessing the FIFO data.) This corresponds to eight 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 range of addresses for an address space is entered in this array. Each address range is a contiguous block of addresses as viewed from the host microprocessor's total addressable space. Its 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_1000_0040", -- IPIF Reset base address
  X"0000_0000_1000_0043", -- IPIF Reset high address
  X"0000_0000_1000_0080", -- WRFIFO Registers base address
  X"0000_0000_1000_00BF", -- WRFIFO Data base address
  X"0000_0000_1000_00C0", -- RDFIFO Registers base address
  X"0000_0000_1000_00FF", -- RDFIFO Data base address
);

```

In this example, there are seven address pairs entered into the C\_ARD\_ADDR\_RANGE\_ARRAY VHDL Generic. This corresponds to the seven address spaces being defined. Each pair is comprised of a starting address and an ending address. Values are right justified and are byte address relative.

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 space will be correctly decoded in the IPIF. First, the user must decide the required address space 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 range 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 (0x40)
IPIF_RST	4 (0x04)
IPIF_WRFIFO_REG	8 (0x08)
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 (0x08)
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_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)
  32, -- IPIF Reset/MIR service (always 32 bits)
  32, -- WRFIFO Reg
  32, -- WRFIFO Data
  32, -- RDFIFO Reg
  32  -- RDFIFO Data
);

```



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 OPB.

Figure 4: Data Width Array Example.

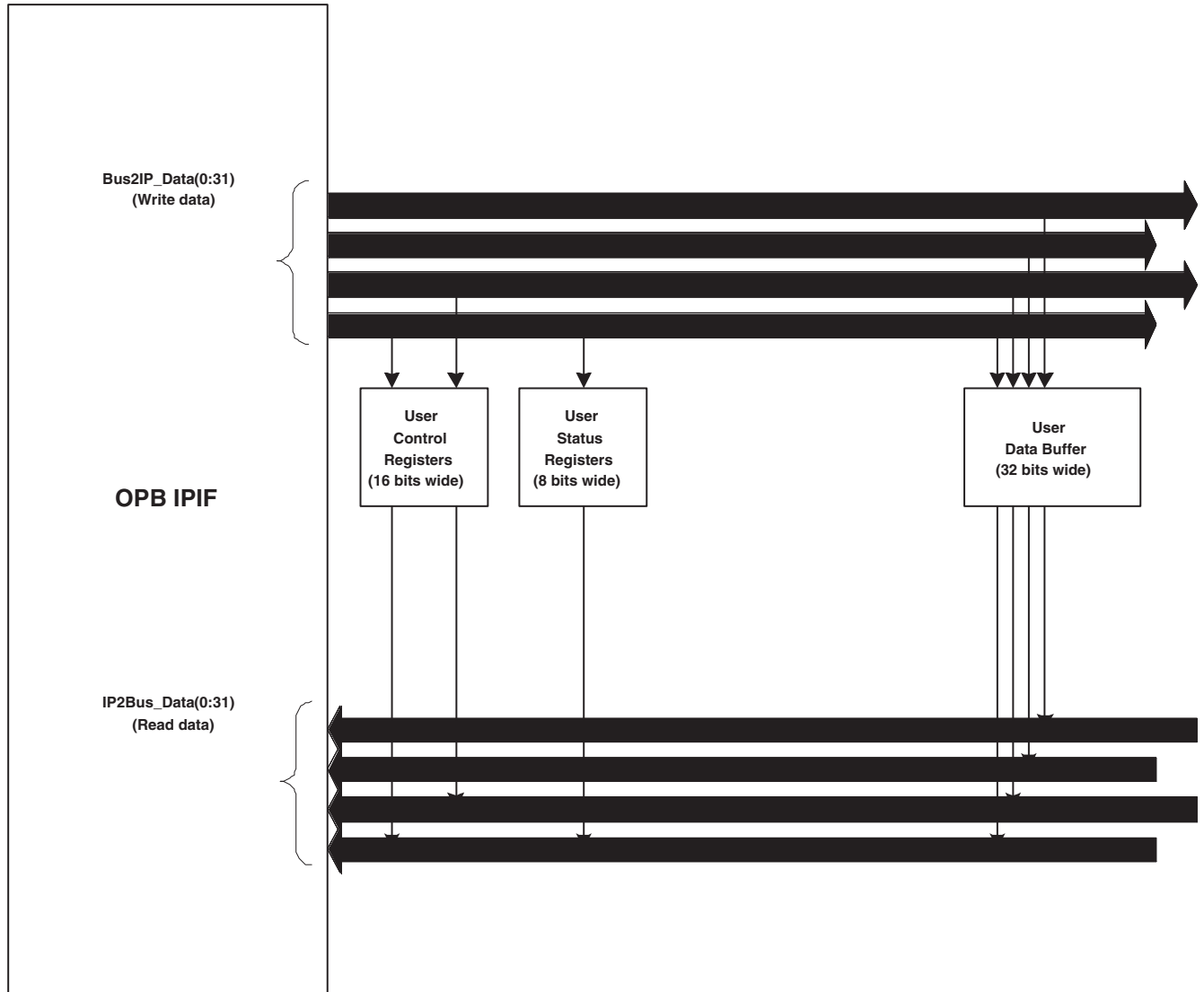


Figure 5: Example Byte Lane Connections of User Functions

**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 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 / 4) 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_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, -- IPIF Reset/MIR service (always 1 CE)
2, -- WRFIFO Reg
1, -- WRFIFO Data
2, -- RDFIFO Reg
1, -- RDFIFO Data
);
    
```

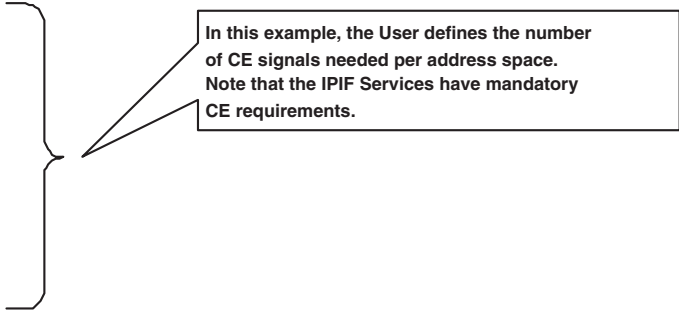


Figure 6: Chip Enable Specification Example

**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 space. That is, it is used to further define *properties* that are *dependent* on the type of address space. The number of dependent properties varies from address space to address space and may be zero. Whether or not there are dependent properties, an entry in C\_ARD\_DEPENDENT\_PROPS\_ARRAY is required for each address space 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 space, 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 spaces.

In the version of the OPB IPIF presented in this document, dependent-property parameters are defined for the Read and Write FIFOs and for the Interrupt Source Controller. See Table 6 for an enumeration of these parameters. Each FIFO type actually has two ARD IDs, but the dependent properties are attached to only one of these, to the IPIF\_WFIFO\_DATA address-space ID for the write FIFO or to the IPIF\_RFIFO\_DATA address-space ID for the read FIFO.

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<sup>1</sup>. 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 in **Figure 7**, where the "OTHERS =>" association is used to set all dependent properties to zero for address spaces that do have associated Dependent Properties and to default any unmentioned Dependent Properties.

```

C_ARD_DEPENDENT_PROPS_ARRAY: INTEGER_ARRAY_TYPE :=
  -- Properties dependent upon the type of address range
  (
    0 => (EXCLUDE_DEV_ISC => 0,           -- IPIF_IRPT
          INCLUDE_DEV_PENCODER => 1,
          others => 0),
    1 => (others => 0),                   -- USER_00
    2 => (others => 0),                   -- USER_01
    3 => (others => 0),                   -- IPIF_RST
    4 => (others => 0),                   -- IPIF_WRFIFO_REG
    5 => (FIFO_CAPACITY_BITS => 16384,    -- IPIF_WRFIFO_DATA
          WR_WIDTH_BITS => 32,
          RD_WIDTH_BITS => 32,
          EXCLUDE_VACANCY => 1,
          others => 0),
    6 => (others => 0),                   -- IPIF_RDFIFO_REG
    7 => (FIFO_CAPACITY_BITS => 16384,    -- IPIF_RDFIFO_DATA
          WR_WIDTH_BITS => 32,
          RD_WIDTH_BITS => 32,
          EXCLUDE_VACANCY => 1,
          others => 0)
  );

```

In this example, the User provides additional, parameterization of the Interrupt and FIFO services using the `C_ARD_DEPENDENT_PROPS_ARRAY`. The "others => 0" clauses provide default parameterization to all parameters not explicitly mentioned.

**Figure 7: Dependent Properties Specification Example**

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 example, each FIFO has, for example, an available parameter with symbolic name `EXCLUDE_PACKET_MODE`. However, because 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-space types<sup>2</sup> without changing the static generic interface of the IPIF.

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 `proc_common_v2_00_a` from the source code in `ipif_pkg.vhd`.
2. The expansion could also be with respect to adding parameters to an already defined address-space type.

Table 6: Packet FIFO Dependent Properties Parameters.

ARD_ID Type	FIFO Parameter	Predefined Index Alias	Description
IPIF_ RdFIFO_ DATA  or  IPIF_ WrFIFO_ DATA	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 the Packet Mode feature is not needed, this parameter allows the user to save FPGA resources by excluding it. 0 = Retain the Packet Mode feature. 1 = Exclude the Packet Mode Feature.
	Vacancy Calculation Exclusion/ Inclusion	EXCLUDE_VACANCY	If the Vacancy calculation feature is not needed, this parameter allows the user to save FPGA resources by excluding it 0 = Retain the Vacancy feature. 1 = Exclude the Vacancy Feature.
IPIF_INTR	Device Interrupt Source Controller Exclusion/ Inclusion	EXCLUDE_DEV_ISC	This parameter allows to exclude the bulk of the Device Interrupt Source Controller. 0 = The full Device ISC will be included 1 = Only the Global Interrupt Enable is present in the Device ISC and the only source of interrupts in the Device is the attached IP core.
	Device Interrupt ID Inclusion/ Exclusion	INCLUDE_DEV_PENCODER	This parameter allows to include the Device Interrupt ID register from the Device ISC, if present. 0 = Exclude the Device IID 1 = Include the Device IID and its supporting priority encoder

**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.

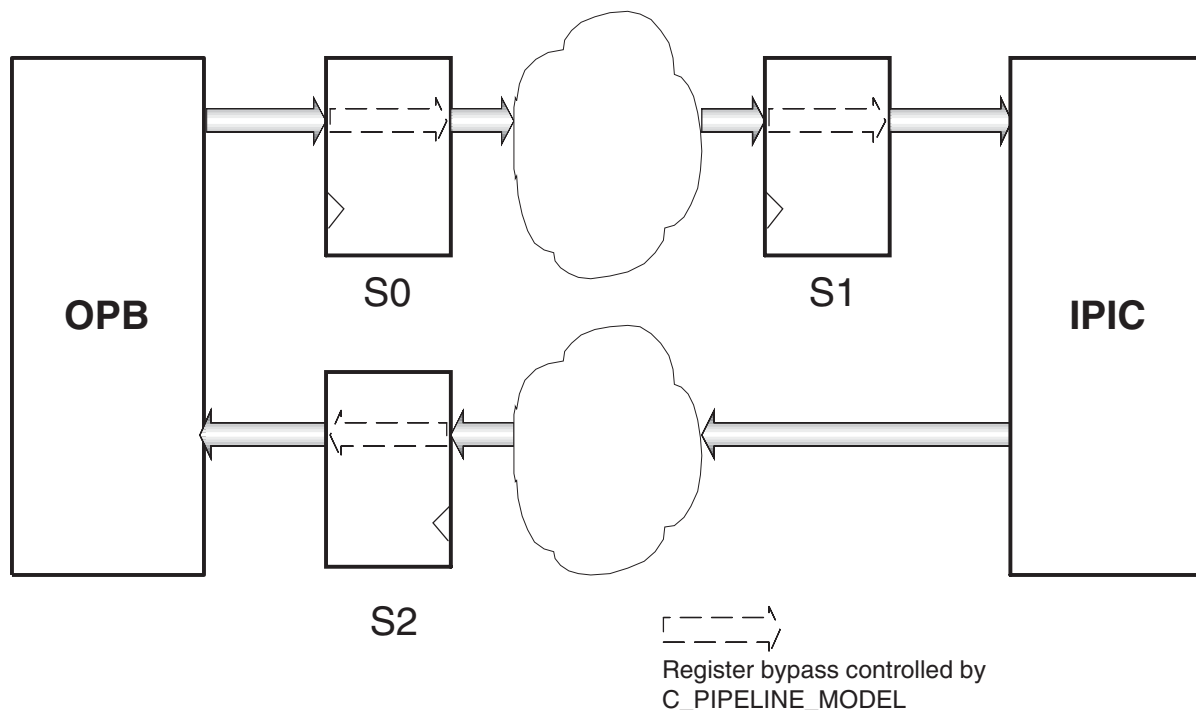
### ***C\_DEV\_BURST\_ENABLE***

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\_PIPELINE\_MODEL***

This integer parameter is used to control the trade off involving bus transaction latency, operating frequency, and resource usage by specifying which combination of the three IPIF pipeline registers will be used. The locations of the three optional pipeline registers, as shown in [Figure 8](#), are

- S0, at the point where OPB bus-logic signals enter the IPIF
- S1, at the point where IPIC signals leave the IPIF
- S2, at the point where OPB bus-logic signals leave the IPIF



*Figure 8: Optional pipeline stages in the OPB IPIF*

The least significant of the three bits needed to specify the `C_PIPELINE_MODEL` parameter controls whether S0 is included, the next most significant bit whether S1 is included and the most significant whether S2 is included, as shown with the following table.

Table 7: Supported Pipeline Models

C_PIPELINE_MO DEL <sup>1</sup>	Inclusion-Status of Pipe Stage			Round-Trip Latency of OPB Transaction (clocks)
	S2	S1	S0	
4	included	bypassed	bypassed	2
5	included	bypassed	included	3
7	included	included	included	4

1. Pipeline models 0, 1, 2, 3, and 6 are not supported.

The inclusion of pipelining has the potential to increase system operating frequency if the system critical delay path includes the inbound bus logic (S0), the IP logic (S1) or the outbound bus logic (S2). This increase in frequency comes at the cost of increased FPGA routing and register resources and increased latency of OPB round-trip transactions as measured in clock periods.

### **C\_INCLUDE\_ADDR\_CNTR**

This integer parameter has a range of 0 to 1. A value of 1 will result in the inclusion of an address counting capability on the Bus2IP\_Addr signal. The counter will advance the address in response to assertions of the IP2Bus\_AddrAck signal. This makes it possible for the IP to obtain the next sequential address before it is made available from the OPB, which in turn allows for achieving a data transfer per clock in the presence of pipelining.

A value of 0 will exclude the address counting capability, unless C\_INCLUDE\_WR\_BUF=1. Including the write buffer will automatically include the address counting capability.

### **C\_INCLUDE\_WR\_BUF**

This integer parameter has a range of 0 to 1. A value of 1 will result in the inclusion of a write-buffer FIFO stage between the IPIF and the IPIC for write transactions. The FIFO serves to decouple completion of OPB write transactions from the presentation of the transactions to the IPIC, which allows for reduced latency and increased efficiency from the standpoint of the bus when IP must throttle write transactions.

Inclusion of the write buffer forces automatic inclusion of the address counter, regardless of the value of C\_INCLUDE\_ADDR\_CNTR.

A value of 0 will exclude the write buffer.

### **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 8. 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.

The number of IP interrupts that can be specified by the user is bounded by the number of bits in data bus width of the OPB.

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 8: Xilinx Predefined Interrupt Capture Mode Identifiers

Category	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\_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\_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 9: 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-II E
OPB IPIF Supported Families	spartan3	String	"spartan3"	Spartan-3
	virtex	String	"virtex"	Virtex
	virtexe	String	"virtexe"	Virtex-E
	virtex2	String	"virtex2"	Virtex-II
	virtex2p	String	"virtex2p"	Virtex-II Pro
	qrvirtex2	String	"qrvirtex2"	Virtex-II QR
	qvirtex2	String	"qvirtex2"	Virtex-II Q

### 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 10](#). The interfaces referenced in this table are shown in [Figure 1](#) in the OPB IPIF block diagram.

Table 10: OPB IPIF I/O Signals

Grouping		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.
	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	SIn_DBus(0:C_OPB_DWIDTH-1)	OPB	O	See table note 1.
	P11	SIn_xferAck	OPB	O	See table note 1.
	P12	SIn_errAck	OPB	O	See table note 1.
	P13	SIn_retry	OPB	O	See table note 1.
	P14	SIn_toutSup	OPB	O	See table note 1.
Interrupt Controller  Device Interrupt Output	P15	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	P16	IP2Bus_Clk <sup>(2)</sup>	User IP	I	Input clock from the IP. This is currently not used by any IPIF services.
	P17	Bus2IP_Clk	User IP	O	IPIC clock pass-through from OPB_Clk.
	P18	Bus2IP_Reset	User IP	O	Active high reset for use by the user IP.
	P19	Bus2IP_Freeze	User IP	O	Active high signal requesting an operational freeze of the user IP.
IPIC  User Interrupt Inputs	P20	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.

Table 10: OPB IPIF I/O Signals (Continued)

Grouping		Signal Name	Interface	I/O	Description
IPIC  IP Slave Interface Request and Qualifier Signals	P21	Bus2IP_Data(0:C_IPIF_DWIDTH-1)	User IP	O	Write data bus to the user IP. Write data is accepted by the IP by assertion of the IP2Bus_Ack signal at the rising edge of the Bus2IP_Clk (or unconditionally on the cycle presented for posted writes).
	P22	Bus2IP_Addr(0:C_OPB_AWIDTH-1)	User IP	O	Address bus indicating the desired address of the requested read or write operation.
	P23	Bus2IP_AddrValid	User IP	O	This signal indicates that a valid address is available on Bus2IP_Addr. For burst write transactions, this signal will remain asserted until the number of clocked IP2Bus_AddrAck events equals the number of beats in the transaction. (This property does not generally hold for burst read transactions because, for reads, Bus2IP_Addr may be ahead of the address on the bus so that a continuous read pipeline can be established.)
	P24	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.
	P25	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.
	P26	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 at the initiation of a burst transaction with the user IP and de-asserted at the completion of the second to last data beat of the burst data transfer.

Table 10: OPB IPIF I/O Signals (Continued)

Grouping		Signal Name	Interface	I/O	Description
IPIC IP Slave Interface Address Decode Service	P27	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.
	P28	Bus2IP_CE(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. Chip enables are asserted during active transaction requests with the target address space and in conjunction with the corresponding sub-address within the space.
	P29	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.
	P30	Bus2IP_WrCE(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 write transaction requests with the target address space and in conjunction with the corresponding sub-address within the space.

Table 10: OPB IPIF I/O Signals (Continued)

Grouping		Signal Name	Interface	I/O	Description
IPIC IP Slave Interface Reply Signals	P31	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_Ack signal and the rising edge of the Bus2IP_Clk.
	P32	IP2Bus_AddrAck	User IP	I	Active high signal that advances the IPIF Address counter during multiple data beat transfers.
	P33	IP2Bus_Ack	User IP	I	Active high data acknowledgement For a write transaction, data on the Bus2IP_Data bus is deemed accepted by the user IP at the rising edge of the Bus2IP_Clk whenever IP2Bus_Ack is asserted and an IPIC write transaction is active. For a read transaction, data on the IP2Bus_Data bus is deemed accepted by the user IP at the rising edge of the Bus2IP_Clk whenever IP2Bus_Ack is asserted and an IPIC read transaction is active.
	P34	IP2Bus_Retry	User IP	I	Active high signal indicating the user IP is requesting a retry of an active operation.
	P35	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_Ack.
	P36	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.
	P37	IP2Bus_PostedWrInh(0:C_ARD_ID_ARRAY 'length - 1)	User IP	I	Active high signals, one per defined address space, requesting full handshake transfer protocol for write transactions to the user IP. Signal IP2Bus_postedWrInh(i) will enable or disable posted writes if and only if i is the active address space, which is indicated by assertion of Bus2IP_CS(i).

Table 10: OPB IPIF I/O Signals (Continued)

Grouping		Signal Name	Interface	I/O	Description
IPIC  Read Packet FIFO Signals	P38	IP2RFIFO_Data(0: See note (4))	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_Ack is high, and a rising edge of the Bus2IP_Clk occurs. (5)
	P39	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. (5)
	P40	IP2RFIFO_WrMark	User IP	I	Active high signal commanding the RdFIFO to perform a "Mark" operation. (5)
	P41	IP2RFIFO_WrRelease	User IP	I	Active high signal commanding the RdFIFO to perform a "Release" operation. (5)
	P42	IP2RFIFO_WrRestore	User IP	I	Active high signal commanding the RdFIFO to perform a "Restore" operation. (5)
	P43	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. (5)
	P44	RFIFO2IP_AlmostFull	User IP	O	Active high signal indicating that the RdFIFO can accept only one more data write. (5)
	P45	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. (5)
	P46	RFIFO2IP_Vacancy(0: See note (6))	User IP	O	Status bus indicating the available locations for writing in the RdFIFO. (5)

Table 10: OPB IPIF I/O Signals (Continued)

Grouping		Signal Name	Interface	I/O	Description
Write Packet FIFO Signals	P47	WFIFO2IP_Data(0: See note <sup>(4)</sup> )	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. <sup>(5)</sup>
	P48	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. <sup>(5)</sup>
	P49	IP2WFIFO_RdMark	User IP	I	Active high signal commanding the WrFIFO to perform a "Mark" operation. <sup>(5)</sup>
	P50	IP2WFIFO_RdRelease	User IP	I	Active high signal commanding the WrFIFO to perform a "Release" operation. <sup>(5)</sup>
	P51	IP2WFIFO_RdRestore	User IP	I	Active high signal commanding the WrFIFO to perform a "Restore" operation. <sup>(5)</sup>
	P52	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. <sup>(5)</sup>
	P53	WFIFO2IP_AlmostEmpty	User IP	O	Active high signal indicating that the WrFIFO can provide only one more data read. <sup>(5)</sup>
	P54	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. <sup>(5)</sup>
	P55	WFIFO2IP_Occupancy(0: See Note <sup>(6)</sup> )	User IP	O	Status bus indicating the available locations for reading in the WrFIFO. <sup>(5)</sup>

**Notes:**

1. This signal's function and timing is defined in the IBM<sup>®</sup> **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. 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.
5. See "[Using the IPIF FIFO Service](#)" on page 49.
6. The high index of the RFIFO2IP\_Vacancy or WFIFO2IP\_Occupancy bus is determined by the number of bits required to support the chosen FIFO depth. The FIFO Depth is set through the parameterization of the FIFO by means of the **C\_ARD\_DEPENDENT\_PROPS\_ARRAY**.

## Parameter - Port Dependencies

The OPB IPIF parameterization has effects on many of its I/O port sizes. These are indicated in the port definitions presented in Table 10. There are cases where IPIF ports are rendered unused based on parameter settings. These dependencies are summarized in Table 11. Unused IPIF Input Ports need to be tied to logic '0'.

Table 11: Parameter/Port Dependencies

		Name or Group Description	Depends on Parameter	Relationship Description
OPB IPIF Port	P20	IP2Bus_IntrEvent	<b>G1</b>	If the ID 'IPIF_INTR' is not present in the <b>C_ARD_ID_ARRAY</b> parameter, then input IP2Bus_IntrEvent(0) is connected directly to output port IP2INTC_Irpt
	P38-P46	RdFIFO Interface Ports	<b>G1</b>	These ports are used only if the IDs 'IPIF_RDFIFO_REG' and 'IPIF_RDFIFO_DATA' are present in the <b>C_ARD_ID_ARRAY</b> parameter.
	P40-P42	RdFIFO Interface Packet Mode Control Ports	<b>G1, G5</b>	These ports are used only if the IDs 'IPIF_RDFIFO_REG' and 'IPIF_RDFIFO_DATA' are present in the <b>C_ARD_ID_ARRAY</b> parameter and Packet Mode Features are enabled for the RdFIFO via the setting in <b>C_ARD_DEPENDENT_PROPS_ARRAY</b> .
	P47-P55	WrFIFO Interface Ports	<b>G1</b>	These ports are used only if the IDs 'IPIF_WRFIFO_REG' and 'IPIF_WRFIFO_DATA' are present in the <b>C_ARD_ID_ARRAY</b> parameter.
	P49-P51	WrFIFO Interface Packet Mode Control Ports	<b>G1, G5</b>	These ports are used only if the IDs 'IPIF_WRFIFO_REG' and 'IPIF_WRFIFO_DATA' are present in the <b>C_ARD_ID_ARRAY</b> parameter and Packet Mode Features are enabled for the WrFIFO via the setting in <b>C_ARD_DEPENDENT_PROPS_ARRAY</b> .

## 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 12: OPB IPIF Services Register Summary

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

**Notes:**

- Toggle each bit position to which a "1" is written.
- 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, [Figure 9](#), 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 13](#). 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**, **WFDL**, and **RFDL**) are pass through. Once asserted, they are 'held' by the source of the interrupt (IP ISR, 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.

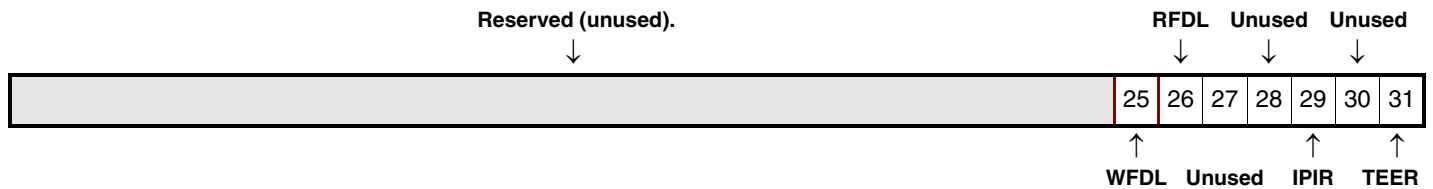


Figure 9: Device Interrupt Status Register

Table 13: Device Interrupt Status Register Description

Bit(s)	Name	Core Access	Reset Value	Description
31	<b>TERR</b>	Read/TOW <sup>(1)</sup>	'0'	<b>Transaction Error.</b> 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. <ul style="list-style-type: none"> <li>'0' = No transaction error detected.</li> <li>'1' = Transaction Error detected.</li> </ul>
30	<b>Unused</b>	Read	'0'	<ul style="list-style-type: none"> <li><b>Unused.</b> Reserved.</li> </ul>
29	<b>IPIR</b>	Read	'0'	<b>IP Interrupt Request.</b> 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. <ul style="list-style-type: none"> <li>'0' = No enabled interrupt is captured</li> <li>'1' = IP interrupt is captured and enabled.</li> </ul>
28	<b>Unused</b>	Read	'0'	<ul style="list-style-type: none"> <li><b>Unused.</b> Reserved.</li> </ul>
27	<b>Unused</b>	Read	'0'	<ul style="list-style-type: none"> <li><b>Unused.</b> Reserved.</li> </ul>
26	<b>RFDL</b>	Read	'0'	<b>Read FIFO Deadlock Interrupt Request.</b> This interrupt indicates that the Read FIFO is asserting it's Deadlock interrupt request. <sup>(2)</sup> <ul style="list-style-type: none"> <li>'0' = No Read FIFO Deadlock Interrupt.</li> <li>'1' = Read FIFO Deadlock interrupt asserted.</li> </ul>

Table 13: Device Interrupt Status Register Description (Continued)

Bit(s)	Name	Core Access	Reset Value	Description
25	WFDL	Read	'0'	<b>Write FIFO Deadlock Interrupt Request.</b> This interrupt indicates that the Write FIFO is asserting its Deadlock interrupt request. (2) <ul style="list-style-type: none"> <li>'0' = No Write FIFO Deadlock Interrupt.</li> <li>'1' = Write FIFO Deadlock interrupt asserted.</li> </ul>
0-24	Unused	Read	zeroes	<b>Unused</b> 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, Figure 10, 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.

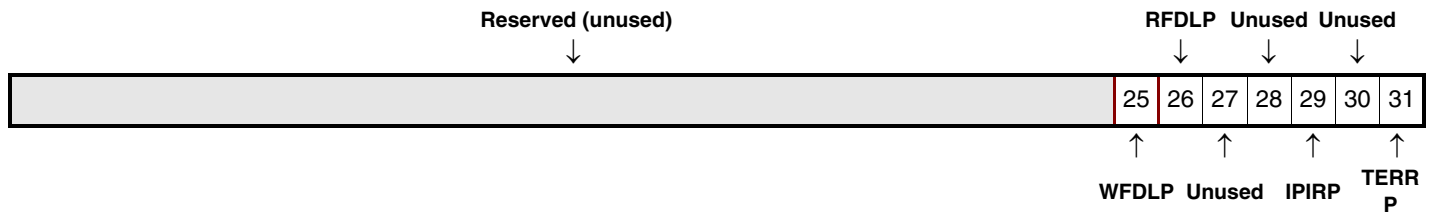


Figure 10: Device Interrupt Pending Register

Table 14: Device Interrupt Pending Register Description

Bit(s)	Name	Core Access	Reset Value	Description
31	TERRP	Read	'0'	<b>Transaction Error Pending.</b> This bit is the logical 'AND' of the TERR bit in the DISR and the corresponding bit in the DIER <ul style="list-style-type: none"> <li>'0' = No Transaction Error pending.</li> <li>'1' = Transaction Error captured and enabled.</li> </ul>
30	Unused	Read	'0'	<b>Unused.</b> Reserved.
29	IPIRP	Read	'0'	<b>IP Interrupt Request Pending.</b> This bit is the logical 'AND' of the IPIR bit in the DISR and the corresponding bit in the DIER. <ul style="list-style-type: none"> <li>'0' = No IP interrupt pending</li> <li>'1' = IP interrupt is pending.</li> </ul>
28	Unused	Read	'0'	<b>Unused.</b> Reserved.
27	Unused	Read	'0'	<b>Unused.</b> Reserved.

Table 14: Device Interrupt Pending Register Description (Continued)

Bit(s)	Name	Core Access	Reset Value	Description
26	RFDLP	Read	'0'	<b>Read FIFO Deadlock Interrupt Request Pending.</b> This bit is the logical 'AND' of the RFDL bit in the DISR and the corresponding bit in the DIER. <ul style="list-style-type: none"> <li>'0' = No Read FIFO Deadlock Interrupt pending.</li> <li>'1' = Read FIFO Deadlock interrupt pending.</li> </ul>
25	WFDLP	Read	'0'	<b>Write FIFO Deadlock Interrupt Request Pending.</b> This bit is the logical 'AND' of the WFDL bit in the DISR and the corresponding bit in the DIER. <ul style="list-style-type: none"> <li>'0' = No Write FIFO Deadlock Interrupt pending.</li> <li>'1' = Write FIFO Deadlock interrupt pending.</li> </ul>
0-24	Unused	Read	zeroes	<b>Unused</b> bit positions. Reserved.

### Device Interrupt Enable Register (DIER)

The Device Interrupt Enable Register, [Figure 11](#), determines which interrupt sources in the Device Interrupt Status Register are allowed to generate interrupts to the system.

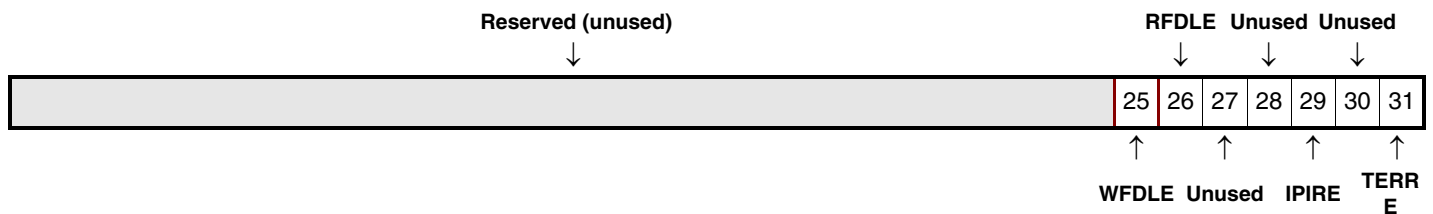


Figure 11: Device Interrupt Enable Register

Table 15: Device Interrupt Enable Register Description

Bit(s)	Name	Core Access	Reset Value	Description
31	TERRE	Read/Write	'0'	<b>Transaction Error Enable.</b> This bit is the interrupt enable for the TERR bit in the DISR . <ul style="list-style-type: none"> <li>'0' = Mask Interrupt.</li> <li>'1' = Enable Interrupt.</li> </ul>
30	Unused	Read	'0'	• <b>Unused.</b> Reserved.
29	IPIRE	Read/Write	'0'	<b>IP Interrupt Request Enable.</b> This bit is the interrupt enable for the IPIR bit in the DISR . <ul style="list-style-type: none"> <li>'0' = Mask Interrupt.</li> <li>'1' = Enable Interrupt.</li> </ul>
28	Unused	Read	'0'	• <b>Unused.</b> Reserved.
27	Unused	Read	'0'	• <b>Unused.</b> Reserved.

Table 15: Device Interrupt Enable Register Description (Continued)

Bit(s)	Name	Core Access	Reset Value	Description
26	RFDLE	Read/Write	'0'	<b>Read FIFO Deadlock Interrupt Request Enable.</b> This bit is the interrupt enable for the RFDL bit in the DISR . <ul style="list-style-type: none"> <li>'0' = Mask Interrupt.</li> <li>'1' = Enable Interrupt.</li> </ul>
25	WFDLE	Read/Write	'0'	<b>Write FIFO Deadlock Interrupt Request Enable.</b> This bit is the interrupt enable for the WFDL bit in the DISR . <ul style="list-style-type: none"> <li>'0' = Mask Interrupt.</li> <li>'1' = Enable Interrupt.</li> </ul>
0-24	Unused	Read	zeroes	<b>Unused</b> bit positions. Reserved.

**Device Interrupt ID Register (DIID)**

The Device Interrupt ID Register, Figure 12, 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).



Figure 12: Device Interrupt ID Register

Table 16: 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. <ul style="list-style-type: none"> <li>0x80 - The DIPR has no pending interrupts.</li> <li>Otherwise - The ordinal ID of the highest-priority pending interrupt in the DIPR.</li> </ul>

**Device Global Interrupt Enable Register (DGIE)**

The Device Global Interrupt Enable Register, Figure 13, 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 IP2INTC\_Irpt (P15) 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.

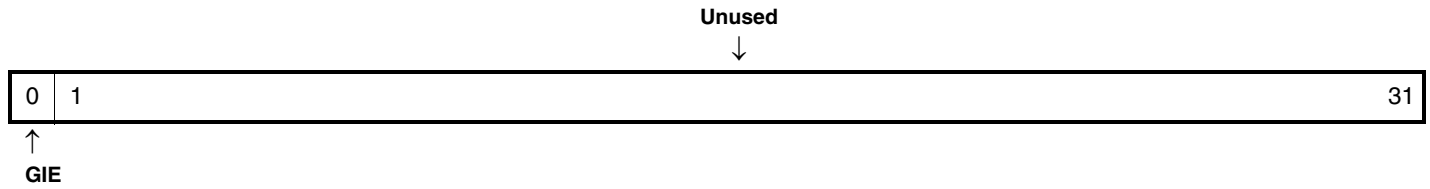


Figure 13: Device Global Interrupt Enable Register

Table 17: Device Global Interrupt Enable Register Description

Bit(s)	Name	Core Access	Reset Value	Description
0	GIE	Read/Write	0	Global Interrupt Enable. <ul style="list-style-type: none"> <li>0 - Interrupts disabled; no interrupt possible from this device.</li> <li>1 - Device Interrupt enabled.</li> </ul>
1-31		Read	zeros	Unused

**IP Interrupt Status Register (IPISR)**

The IP Interrupt Status Register, shown in Figure 14, 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 (P20) 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).

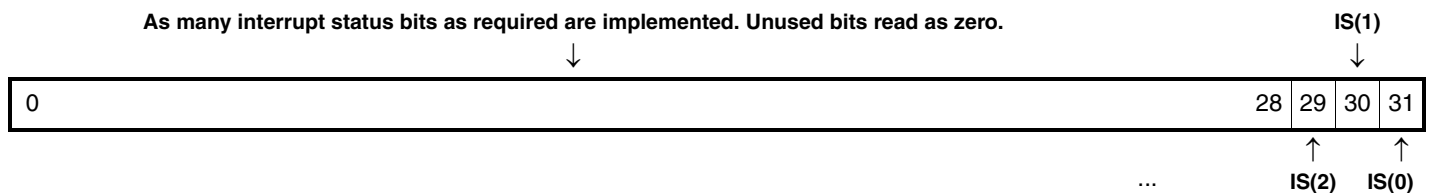


Figure 14: IP Interrupt Status Register

Table 18: IP Interrupt Status Register Description

Bit(s)	Name	Core Access	Reset Value	Description
0-31	IS(i)	Read/TOW <sup>(1)</sup>	zeroes	<b>Interrupt Status(i).</b> 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. <ul style="list-style-type: none"> <li>1 - active</li> <li>0 - not active</li> </ul>

**Notes:**

- 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, [Figure 15](#), has an enable bit for each defined bit of the IP Interrupt Status Register.

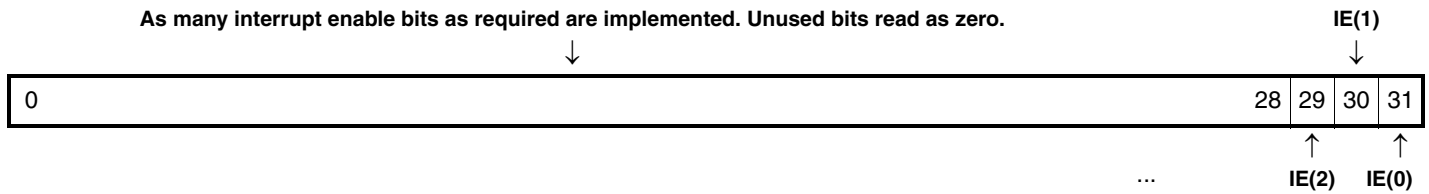


Figure 15: IP Interrupt Enable Register

Table 19: IP Interrupt Enable Register Description

Bit(s)	Name	Core Access	Reset Value	Description
0-31	IE(i)	Read/Write	zeroes	<b>Interrupt Enable(i).</b> <ul style="list-style-type: none"> <li>1 - enabled</li> <li>0 - disabled</li> </ul>

### IPIF Software Reset/MIR Service Registers

The section details the register complement 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, [Figure 16](#), is actually not a register but, rather, a write-only address, which, when written with a specific value, generates a reset for the device.

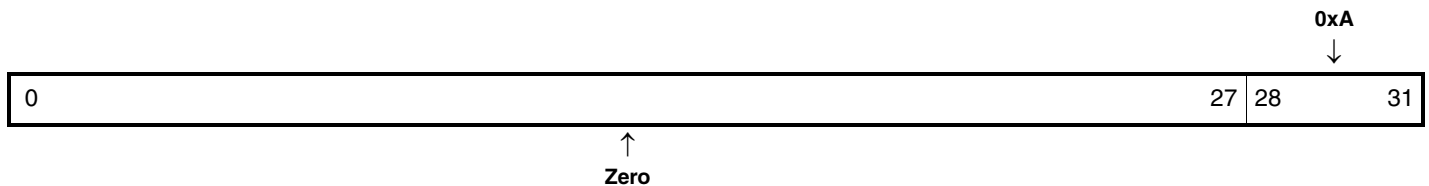


Figure 16: IPIF Software Reset Register

Table 20: IPIF Software Reset Register Description

Bit(s)	Name	Core Access	Reset Value	Description
0-31	Reset Key	Write	n/a	<b>Reset write value.</b> <ul style="list-style-type: none"> <li>'0x0000000A' - Generate a device reset (the implementation is allowed to check just the four least-significant bits for the value 0xA).</li> <li>others - No effect.</li> </ul>

#### IPIF Module Identification Register

The Module Identification Register, [Figure 17](#), 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.

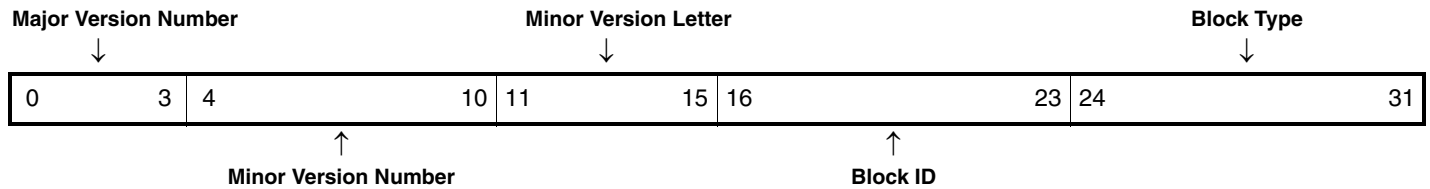


Figure 17: IPIF Module Identification Register

Table 21: 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 18). This reset re-initializes the RdFIFO logic. Previously stored data in the FIFO is lost.

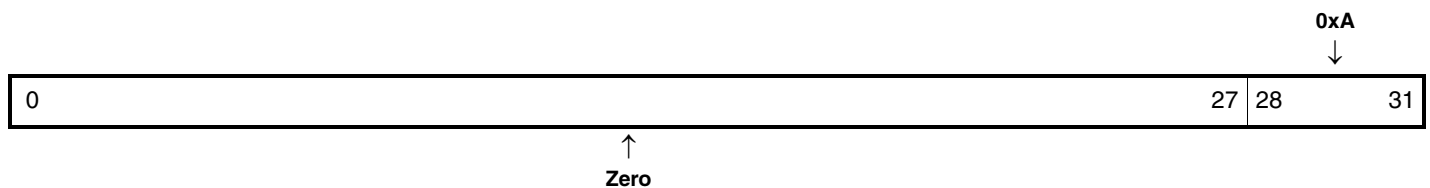


Figure 18: RdFIFO Reset Register

Table 22: RdFIFO Reset Register Description

Bit(s)	Name	Core Access	Reset Value	Description
0-31		Write	n/a	<b>Reset write value.</b> <ul style="list-style-type: none"> <li>'0x0000000A' - Generate a FIFO reset (the implementation is allowed to check just the four least-significant bits for the value 0xA).</li> <li>others - No effect.</li> </ul>

### RdFIFO Module Identification Register

The RdFIFO Module Identification Register, [Figure 19](#), 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.

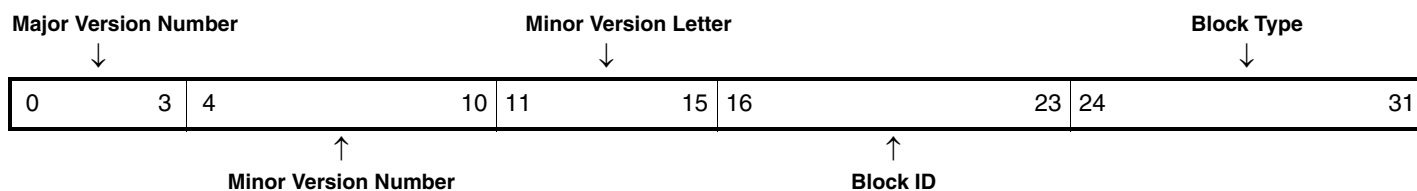


Figure 19: RdFIFO Module Identification Register

Table 23: 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, [Figure 20](#), is a read-only register that gives the occupancy status of the RdFIFO.

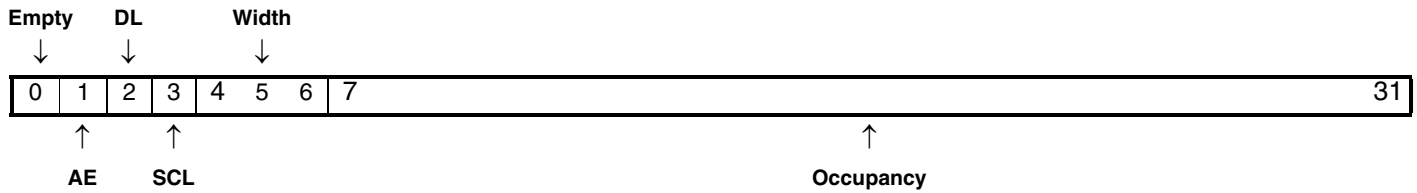


Figure 20: RdFIFO Status Register Modified Bit Layout

Table 24: RdFIFO Status Register Bit Definitions

Bit(s)	Name	Core Access	Reset Value	Description
7 to 31	<b>Occupancy</b>	Read	Zero <sup>(1)</sup>	RdFIFO Occupancy. <ul style="list-style-type: none"> <li>This is an unsigned value reflecting a current snapshot of the number of locations occupied with data in the RdFIFO memory core.</li> </ul>
4 to 6	<b>Width</b>	Read	FIFO Width <sup>(2)</sup>	<b>Encoded FIFO Data Port Width.</b> This field reflects the parametrized width of the FIFO data port. <ul style="list-style-type: none"> <li>'000' = 32 bits (legacy PFIFO default)</li> <li>'001' = 8 bits</li> <li>'010' = 16 bits</li> <li>'011' = 32 bits</li> <li>'100' = 64 bits</li> <li>'101' = 128 bits (not currently used)</li> <li>'110' = 256 bits (not currently used)</li> <li>'111' = 512 bits (not currently used)</li> </ul>
3	<b>SCL</b>	Read	See Note <sup>(3)</sup>	<b>Occupancy Scaling Enabled.</b> <ul style="list-style-type: none"> <li>'0' = The Occupancy value is not scaled.</li> <li>'1' = The Occupancy value is scaled to fit in the available Status Register space.</li> </ul>
2	<b>DL</b>	Read	'0'	<b>FIFO DeadLock Condition.</b> <ul style="list-style-type: none"> <li>'0' = The FIFO is not in DeadLock.</li> <li>'1' = The FIFO is in DeadLock (Simultaneously Full and Empty due to Packet Mode Operations).</li> </ul>
1	<b>AE</b>	Read	'0'	<b>FIFO Almost Empty Condition.</b> <ul style="list-style-type: none"> <li>'0' = If FIFO is not Empty, then at least two data values are available for reading.</li> <li>'1' = FIFO is Almost Empty (only one more data value is available for reading).</li> </ul>
0	<b>Empty</b>	Read	'1'	<b>FIFO Empty Condition.</b> <ul style="list-style-type: none"> <li>'0' = FIFO is not Empty.</li> <li>'1' = FIFO is Empty and cannot provide anymore data.</li> </ul>

**Notes:**

- The FIFO Occupancy value defaults to 0. The max value that can be set is the FIFO depth which is set via user parameterization.
- The FIFO data port width is set via user parameterization.
- 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, **Figure 21**, is a read-only register that is used to read data from the FIFO.

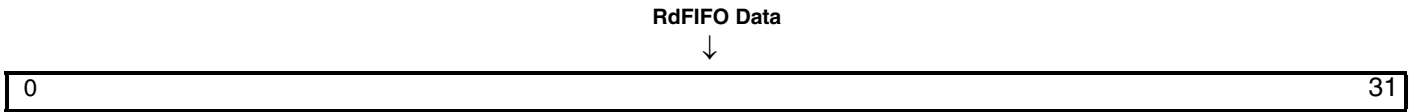


Figure 21: RdFIFO Data Register

Table 25: 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 22**).

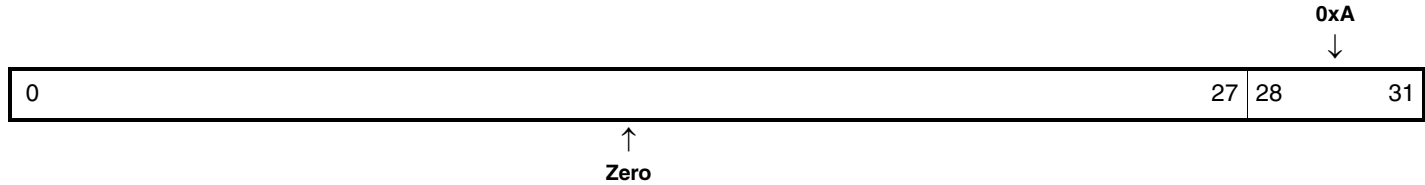


Figure 22: WrFIFO Reset Register

Table 26: WrFIFO Reset Register Description

Bit(s)	Name	Core Access	Reset Value	Description
0-31	Reset Key	Write	n/a	<p><b>Reset write value.</b></p> <ul style="list-style-type: none"> <li>'0x0000000A' - Generate a FIFO reset (the implementation is allowed to check just the four least-significant bits for the value 0xA).</li> <li>others - No effect.</li> </ul>

**WrFIFO Module Identification Register**

The WrFIFO Module Identification Register, **Figure 23**, 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.

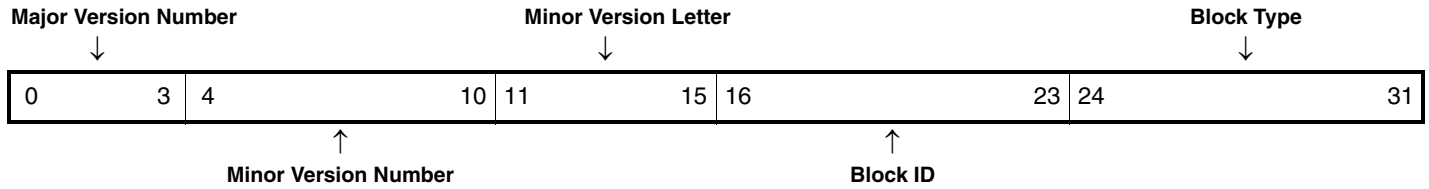


Figure 23: WRFIFO Module Identification Register

Table 27: 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.
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, [Figure 24](#), is a read-only register that gives the vacancy status of the WRFIFO.

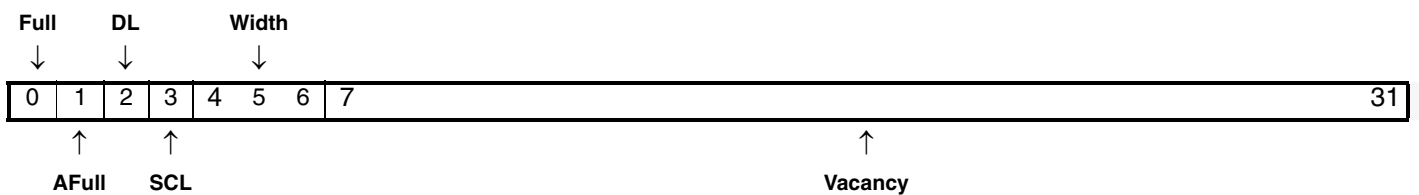


Figure 24: WRFIFO Status Register Modified Bit Layout

Table 28: WrFIFO Status Register Bit Definitions

Bit(s)	Name	Core Access	Reset Value	Description
7 to 31	<b>Vacancy</b>	Read	FIFO Depth <sup>(1)</sup>	WrFIFO Vacancy. <ul style="list-style-type: none"> <li>This is an unsigned value reflecting a current snapshot of the number of locations available for data storage in the WrFIFO memory core.</li> </ul>
4 to 6	<b>Width</b>	Read	FIFO Width <sup>(2)</sup>	<b>Encoded FIFO Data Port Width.</b> <ul style="list-style-type: none"> <li>'000' = 32 bits (legacy PFIFO default)</li> <li>'001' = 8 bits</li> <li>'010' = 16 bits</li> <li>'011' = 32 bits</li> <li>'100' = 64 bits</li> <li>'101' = 128 bits (not currently used)</li> <li>'110' = 256 bits (not currently used)</li> <li>'111' = 512 bits (not currently used)</li> </ul>
3	<b>SCL</b>	Read	See Note <sup>(3)</sup>	<b>Vacancy Scaling Enabled.</b> <ul style="list-style-type: none"> <li>'0' = The Vacancy value is not scaled.</li> <li>'1' = The Vacancy value is scaled to fit in the available Status Register space.</li> </ul>
2	<b>DL</b>	Read	'0'	<b>FIFO DeadLock Condition.</b> <ul style="list-style-type: none"> <li>'0' = The FIFO is not in DeadLock.</li> <li>'1' = The FIFO is in DeadLock (Simultaneously Full and Empty due to Packet Mode Operations).</li> </ul>
1	<b>AF</b>	Read	'0'	<b>FIFO Almost Full Condition.</b> <ul style="list-style-type: none"> <li>'0' = If FIFO is not Full, then at least two locations are available for writing.</li> <li>'1' = FIFO is Almost Full (only one more location available for writing).</li> </ul>
0	<b>Full</b>	Read	'0'	<b>FIFO Full Condition.</b> <ul style="list-style-type: none"> <li>'0' = FIFO is not Full.</li> <li>'1' = FIFO is Full and cannot accept anymore data.</li> </ul>

**Notes:**

- The FIFO Vacancy value defaults to the depth of the FIFO. The FIFO depth is set via user parameterization.
- The FIFO data port width is set via user parameterization.
- 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 25](#), is a write-only register that is used to write data to the FIFO.

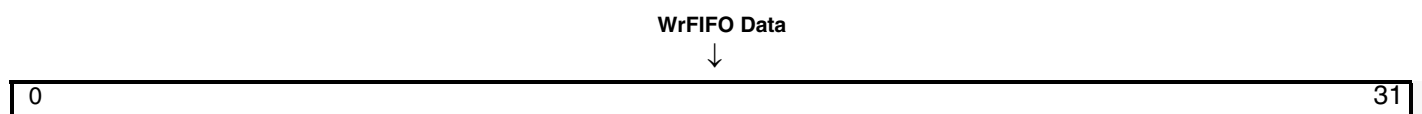


Figure 25: WrFIFO Data Register

Table 29: WrFIFO Data Register Description

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

## 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 decodes for all of the individual addresses of interest within a given address space. These decodes, called '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<sup>1</sup>. An application that uses the CE decodes is sometimes referred to as a *register model*.

Figure 26 shows examples of read transactions as seen from a register model and Figure 27 shows similar write transactions.

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 IP is able to respond to the transaction, but not within 8 cycles, the first cycle being the one on which a Bus2IP\_CS signal (or, similarly, a 'CE', 'RdCE' or 'WrCE' signal) 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 Bus2IP\_Ack, 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. Bus2IP\_Error must be a valid indicator of presence or absence of an error during the cycle in which an acknowledgement is asserted and otherwise is a "don't care".

For write transactions, there is an alternative to the acknowledged behavior. This will be described later on [page 40](#).

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.

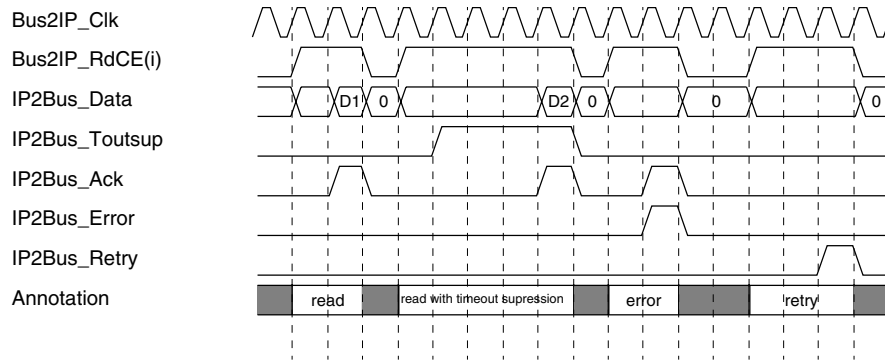
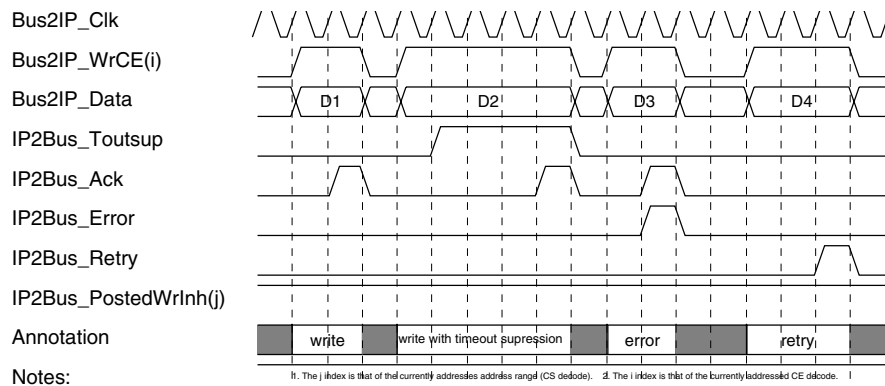


Figure 26: Slave read transactions, register model



Notes:

1. The j index is that of the currently addressed address range (CS decode). 2. The i index is that of the currently addressed CE decode.

Figure 27: Slave write 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 28 and Figure 29 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 in reality 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.

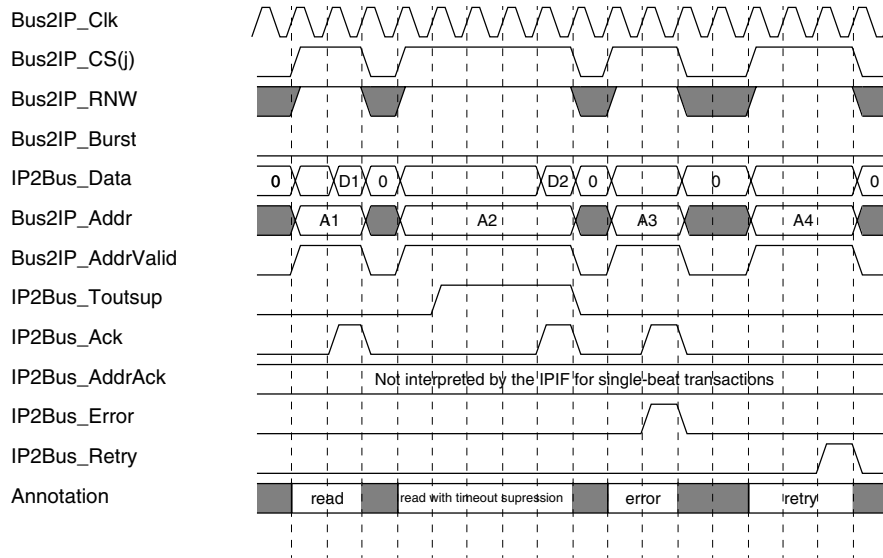


Figure 28: Slave read transactions, memory model

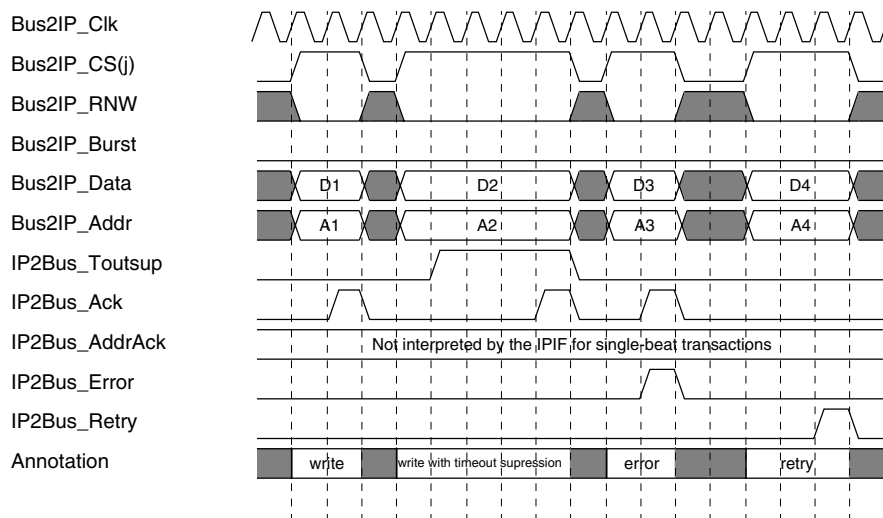


Figure 29: Slave write transactions, memory model

### Posted Write Transactions

An alternative kind of write behavior, *posted write*, is also supported for slave transactions. Under posted write behavior, the IPIF unconditionally acknowledges the write transaction to the OPB on the earliest clock cycle that is consistent with the IPIF pipeline model. The Data, address, address decodes and other qualifier signals are forwarded through any stages included in the pipeline model and presented for one cycle at the IPIC. The IP core takes the data on the cycle presented and does not have the obligation or the opportunity to acknowledge, or respond with error or retry

The posted write is a more efficient operation than the *acknowledged write*. There is no need to drive reply signals and there is one less cycle of latency at the OPB if the IPIF includes pipeline stage S1 to the IPIC.

The IP core has control over whether writes are posted or acknowledged through use of the IP2Bus\_PostedWrInh signal (PostedWrInh = "posted write inhibit"). IP2Bus\_PostedWrInh is an array signal with width equal to the number of address spaces set up through user parameterization. Thus, each address space can independently control how IPIC write transactions will be handled. The IP2Bus\_PostedWrInh(i) signal is tied high to force all writes to address space *i* to be acknowledged, tied low to force all writes to be posted, or may be dynamically switched between the two options. Figure 30 shows examples of IPIC posted write transactions.

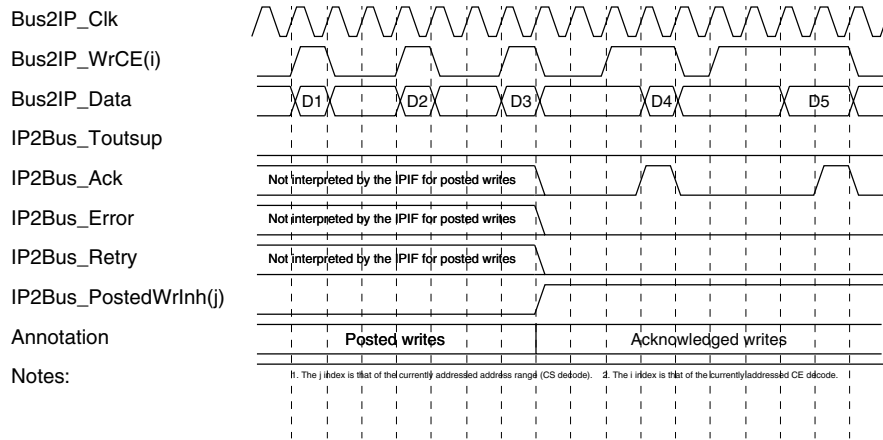


Figure 30: Posted write transactions

The user needs take into account the pipeline model that is being used when dynamically switching between accepting posted writes and inhibiting posted writes. The IPIF's acknowledgment of the mode change depends on the pipeline model. See Table 30.

Table 30: Dynamic postwrinh Change Latency

Pipeline Model	Clocks Before Mode Change	Example Figure
4	1	Figure 31
5	2	Figure 32
7	3	Figure 33

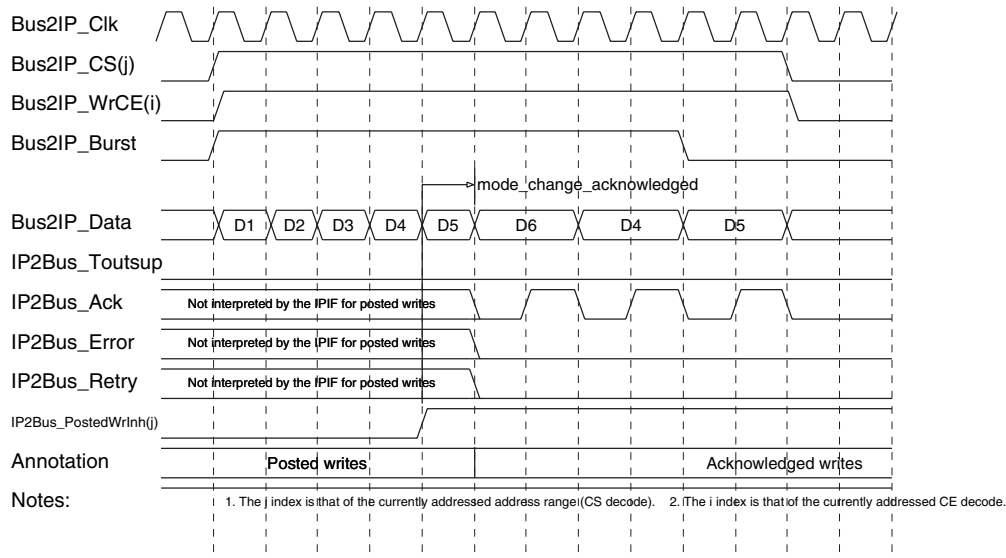


Figure 31: Dynamic postedwrinh During Burst (Pipeline Model 4)

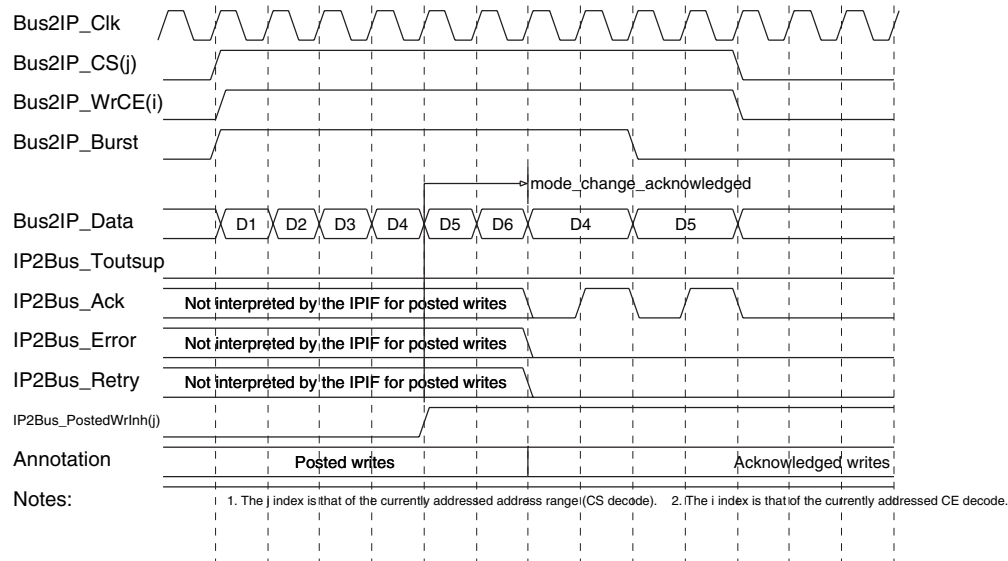


Figure 32: Dynamic postedwrinh During Burst (Pipeline Model 5)

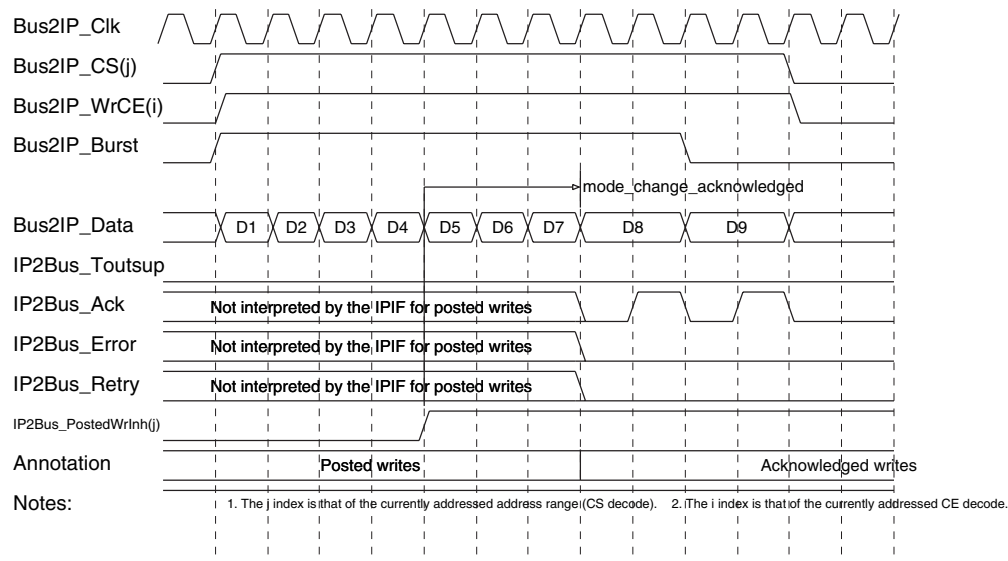


Figure 33: Dynamic postedwrinh During Burst (Pipeline Model 7)

### Slave Burst Transactions

Burst transactions group multiple, related data transfers (or data beats) and have the aim of allowing data to be moved each clock cycle. Additional support by the IPIF is available to the user IP to help achieve burst behavior. This additional support takes the form of three signals (Bus2IP\_Burst, Bus2IP\_AddrValid and IP2Bus\_AddrAck) and two optional IPIF services, an Address Counter and a mechanism to buffer IPIF write transactions. Burst transactions and the use of the burst-support features is the topic of this section.

Bursts can be characterized as *determinant* or *indeterminant*. Determinant bursts have a fixed number of data beats, known when the burst starts. Indeterminant bursts are of variable size; the number of data beats is not known until the burst ends. Determinant bursts tend to be easier to process because of the predictable end point.

The OPB supports what is effectively one, indeterminate burst mode. This mode, called the *sequential address* mode, is embodied in an OPB signal called OPB\_seqAddr. This signal serves to qualify the next data beat following the current beat. If OPB\_seqAddr is asserted, the next data beat is guaranteed to be at the successor address of the current beat and in the same direction, read or write<sup>1</sup>. This does not by itself lead to a data-beat-per-clock behavior but knowing that the next address is sequential allows the slave to optimize the transaction in such a way that it can immediately acknowledge each beat.

**Read Burst**

The fundamental problem encountered with read burst is that, if there is any pipelining in the IPIF or IP core, the core must anticipate that further read data beats will follow and *read-ahead*. This, in turn, means

1. That a look-ahead address generation may be necessary and
2. That when an indeterminate burst finishes, there will have been more reads performed than are actually requested and consumed by the OPB.

To help solve the first issue, an address counter may be optioned into the IPIF<sup>2</sup>. The second issue only needs to be solved if the extra reads leave a state in which a re-read at the same address will give a different result (*destructive read*, or sometimes called *non-idempotent read*). Regular memories do not have a destructive-read characteristic but FIFOs, for example, do and must be able to recover from or "back out of" the extra reads if they support bursts.

A read burst transaction is shown in **Figure 34**. It is assumed that the look-ahead address counter has been optioned in.

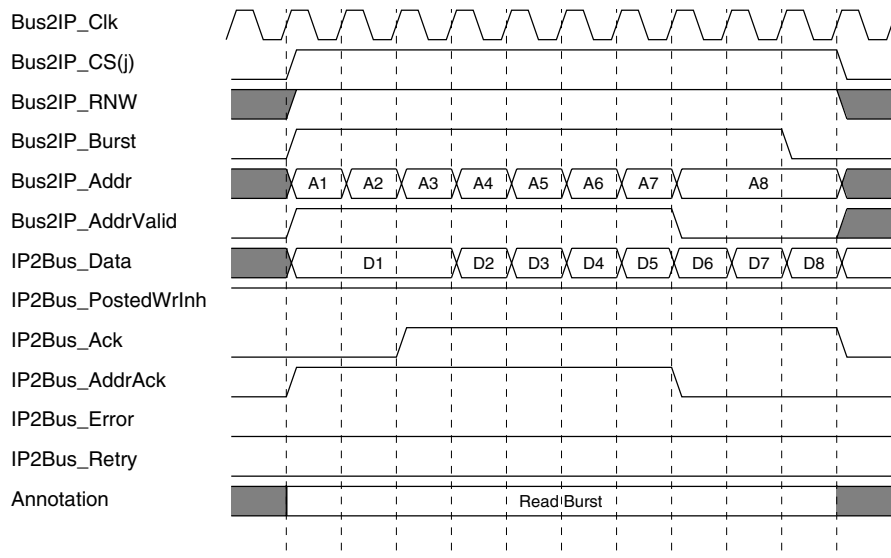


Figure 34: Read burst transaction

When Bus2IP\_Burst and Bus2IP\_AddrValid are asserted, the address space selected by Bus2IP\_CS(j) knows that the initial address is available and that it can be incremented by asserting IP2Bus\_AddrAck for a cycle. The IP core performs as many reads as are necessary to fill a data pipeline driven to the IP2Bus\_Data signal. With each read, it advances the address counter. On the cycle that data becomes available at the head of the pipeline (at signal IP2Bus\_Data), the IP core may assert both IP2Bus\_Ack—acknowledging the data—and IP2Bus\_AddrAck. This will uniformly advance the pipeline at the head and the tail. Such advancement may, and typically will, continue until the burst terminates with negation of Bus2IP\_CS. Because the burst is indeterminate, Bus2IP\_Burst and Bus2IP\_AddrValid will continue to assert right to the end of the burst transaction and the pipeline will contain reads that will not be requested by the OPB<sup>3</sup>.

1. The Xilinx usage of this signal is slightly tighter than the IBM CoreConnect definition. For IBM, it is highly recommended that the signal be negated for the last beat of the transaction. Under Xilinx usage, this is required.
2. The counter is optional even when bursts are supported because some burst targets such as FIFOs do not need an incrementing address (*keyhole burst*).

Note that the IP core has the ability to exercise flow control by deciding not to advance the pipeline. This accommodates a device that can achieve burst performance in most cases but may encounter conditions of absence of data availability that "throttle" the burst.

### Write Burst

In the absence of the need to exercise flow control, write bursts take the form of a sequence of posted writes. First, the IPIF pre-acknowledges enough OPB write transactions to fill any pipeline stages implied by the pipeline model. Then, the IPIF feeds these data beats to the IP core one per clock, at the same time back-filling with OPB acknowledges to keep the pipeline filled. The IP core is obligated to accept the posted-write data beats on the cycle in which they are presented. The protocol is illustrated in **Figure 35**.

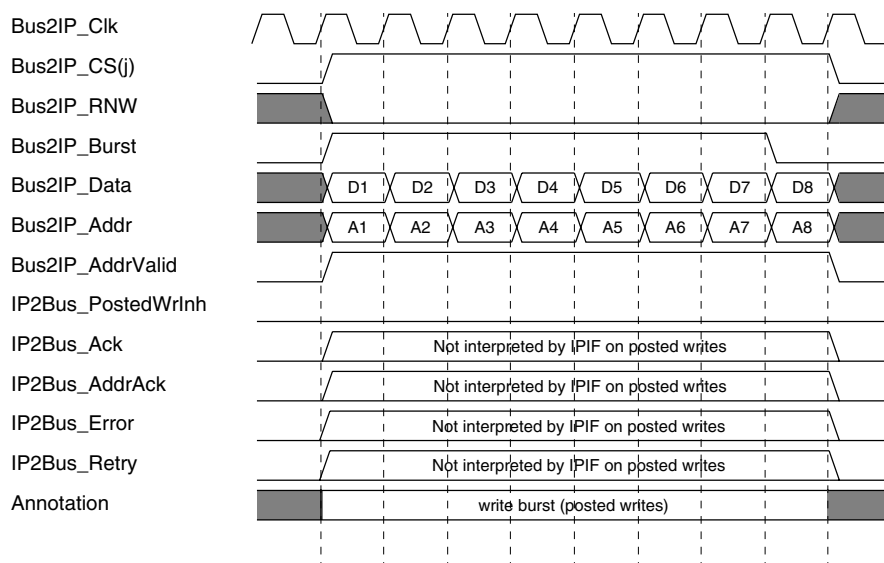


Figure 35: Posted-write burst transaction

### Write Buffer

Posted write IPIC transactions do not allow for a slave IP core to delay the start of an IPIC burst or to throttle a burst in progress. Some IP burst applications require this kind of flexibility, however. When this is the case, additional write-burst support is needed. This support can take the form of extra logic in the IP core—such as a FIFO, accepting posted writes from the IPIF but allowing them to be consumed at the application’s pace on the back end.

The OPB IPIF described in this document also has such a buffering capability available as an optional *service*. If the `C_INCLUDE_WR_BUF` parameter (see [page 4](#)) is set to 1, all write transactions are sent through the write buffer. The output of the write buffer conforms to the IPIC slave interface, and takes its place. Thus, the IP core sees the same IPIC protocol with the write buffer in place<sup>1</sup> as without it. There will, however, be an increase in FPGA resource usage and in OPB-to-IPIC latency for write transactions, if the write buffer is used.

Note also that look-ahead address generation can be used in conjunction with the write buffer, and that the address counter is always included whenever the write buffer is included, independent of the value of the `C_INCLUDE_ADDR_CNTR` parameter.

**Figure 36** shows how the write buffer allows a combination of beat-per-clock bursting and flow control by using throttled acknowledgements.

3. Note that if the OPB did have a determinant burst mode—which it doesn’t—then `Bus2IP_AddrValid` would negate as soon as the last address was presented and `Bus2IP_Burst` would negate for the duration of the last data beat. Further, if the IP core used the negation of `Bus2IP_AddrValid` as an indication to stop pre-reading, there would be no extra reads.

1. This includes honoring the meaning of the `IP2Bus_PostedWrInh` signals. Any address space, *i*, that wishes to throttle write bursts through the write buffer will need to hold `IP2Bus_PostedWrInh(i)` true.

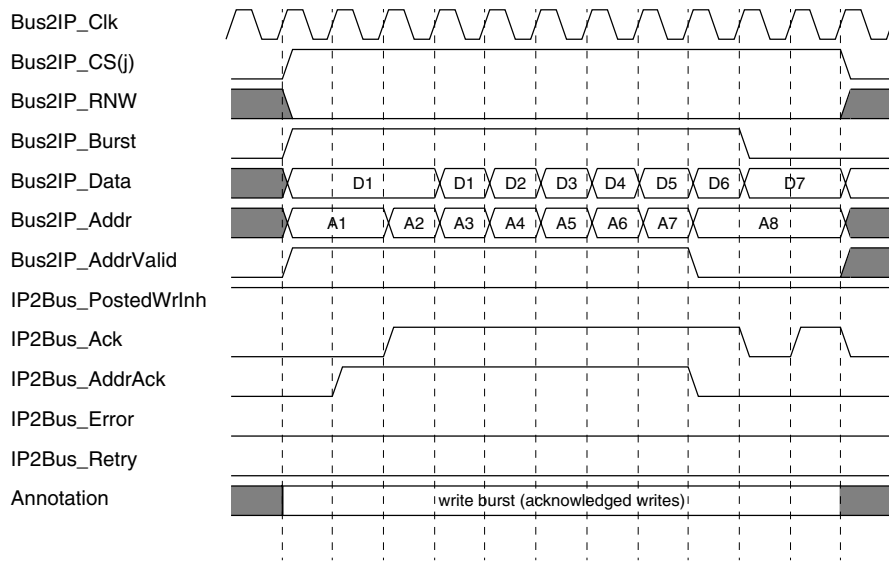


Figure 36: Write burst with acknowledged writes, as enabled by the Write Buffer.

Table 31 summarizes some methods that can be used to allow the throttling of write bursts.

Table 31: Methods of Implementing Write-Burst Throttling

Method	How	Properties
IPIF Write Buffer	Set C_INCLUDE_WR_BUF = 1	<ul style="list-style-type: none"> <li>Preserves IPIC protocol at buffer back end</li> <li>Increases IPIF usage of FPGA resources</li> <li>Increases write latency through the IPIF</li> </ul>
Custom buffering solution in the IP core	Client implemented	<ul style="list-style-type: none"> <li>Allows a custom interface/protocol at the buffer back end. May enable optimization of                             <ul style="list-style-type: none"> <li>FPGA resource usage</li> <li>latency</li> <li>"fit" to application</li> </ul> </li> </ul>
Dynamic Posted Write Inhibit	Client alternates mode between posted and acknowledged writes by switching IP2Bus_PostedWrInh	<ul style="list-style-type: none"> <li>No additional FPGA resource usage</li> <li>Reduced OPB utilization when throttling</li> <li>More latency than buffer methods when throttling</li> <li>Pipeline-model dependent latency between switching of Bus2IP_PostedWrInh and mode change</li> </ul>

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

*Table 32: 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
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 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 37](#) shows the OPB IPIF interrupt structure for an example user device that is maximally populated with interrupts.

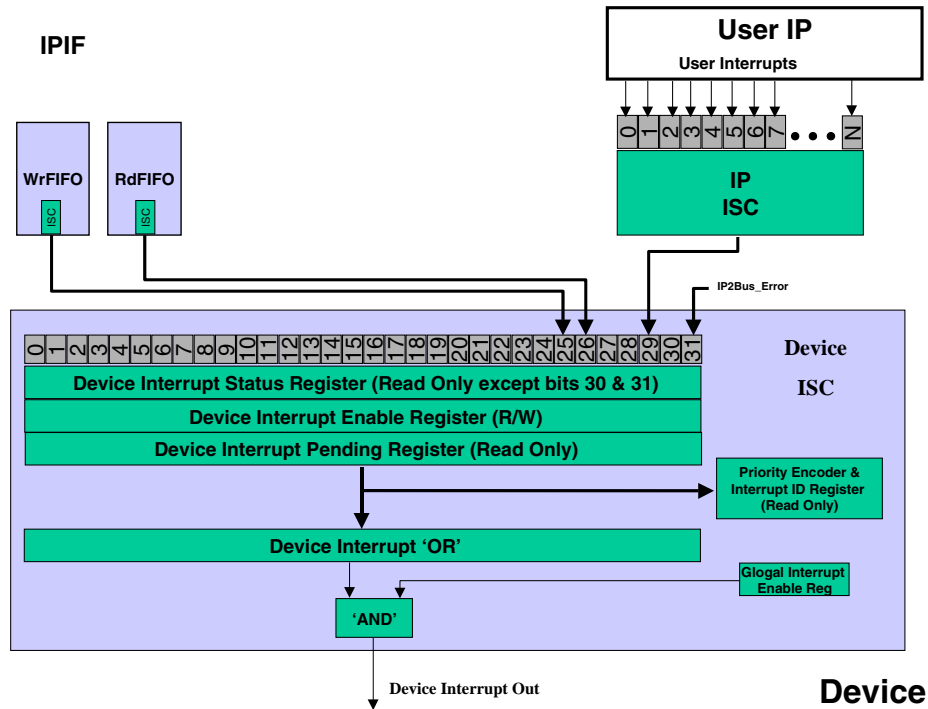


Figure 37: Example User Device Interrupt Hierarchy.

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 three 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 IP2INTC\_Irpt output port. Note that it is the highest level of interrupt hierarchy for the Device. Its 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.

The Packet FIFO interrupts are only included 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 no FIFOs are included and the Transaction Error interrupt is not needed, 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. This option, which reduces hardware cost and software accesses, is shown in [Figure 38](#).
- For IPIF configurations where there are no FIFOs 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 IP2INTC\_Irpt output signal. This implementation is generated when the ARD Arrays have no entry for the IPIF\_INTR identifier. This is graphically shown in [Figure 39](#).

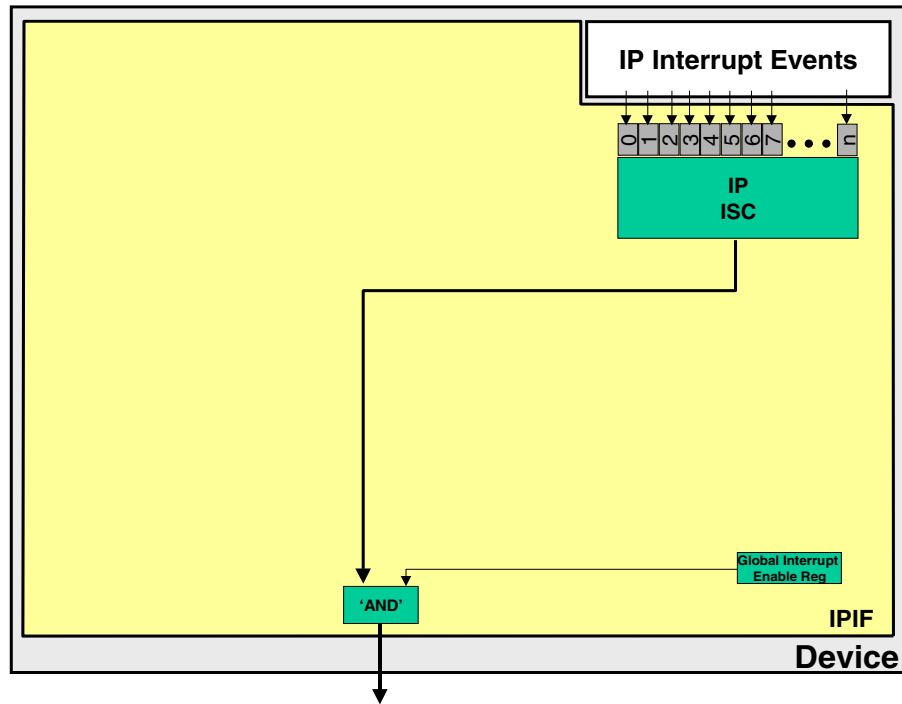


Figure 38: OPB IPIF Interrupt Service Structure with Device ISC Removed.

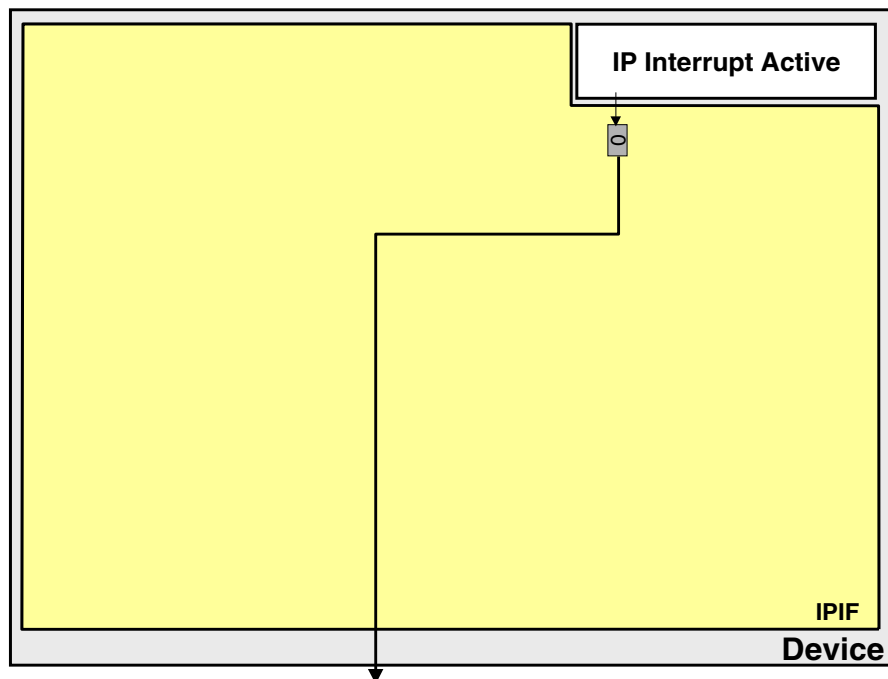


Figure 39: OPB IPIF Interrupt Structure with Interrupt Service Removed.

## 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 and write FIFO services are made available through the read or write FIFO interfaces of the IPIC. 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 effect on the clock edge on which they are asserted and are not acknowledged.

The figures, [Figure 40](#) through [Figure 43](#), give examples of read and write FIFO operation. These examples are for FIFOs with a capacity of 512 items.

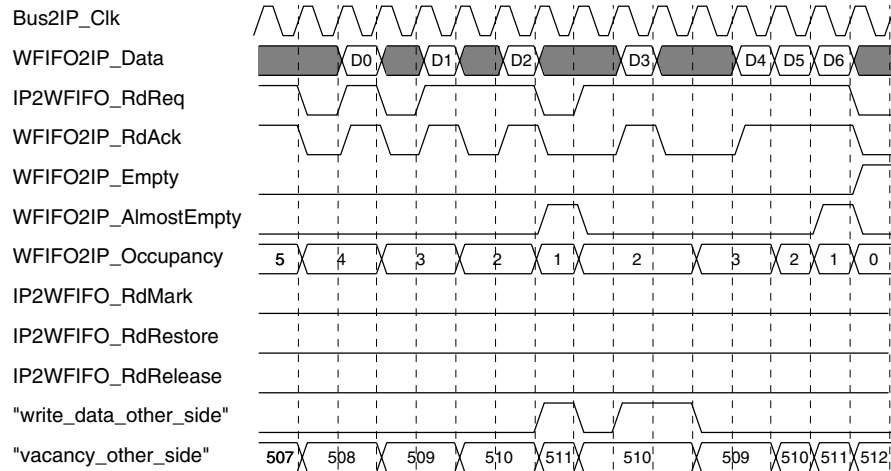


Figure 40: Reading the Write FIFO

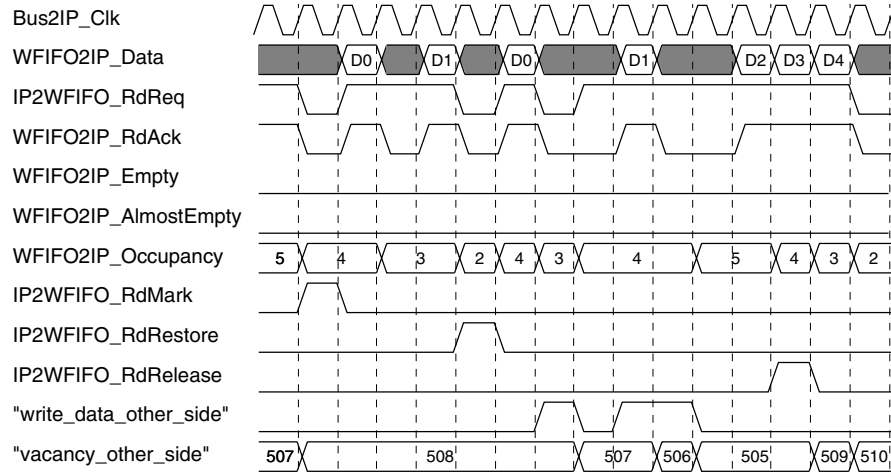


Figure 41: Reading the Write FIFO, including uncommitted reads using mark, restore and release

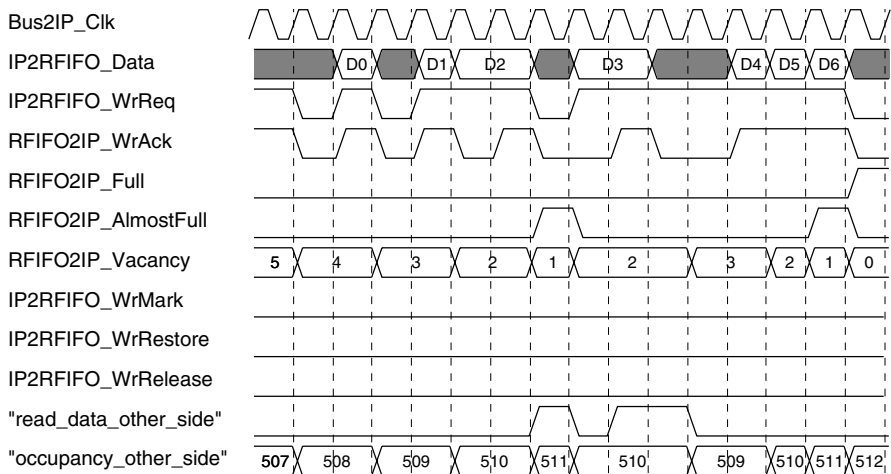


Figure 42: Writing the Read FIFO

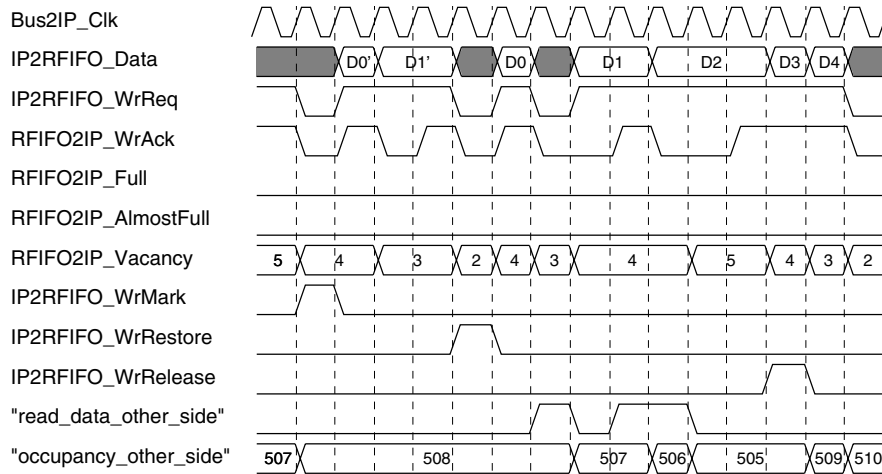


Figure 43: Writing the Read FIFO, including uncommitted writes using mark, restore and release

## 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 44. The timing characteristics of these signals are shown in the timing diagrams of the section titled "Using the IP Slave Interface" on page 38. 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 or Bus2IP\_RNW.

#### 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 range 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 that range. 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 a Chip Enable that is activated by an address, A, is also activated by all addresses in the address space that are displaced from A by N, where N is the greatest power of two that is less than or equal to the product of the number of Chip Enables requested and the width in bytes of the

address space. Operationally, this means that only as many low-order address bits are decoded to form CEs as are needed to make the Chip Enables unique. If the address space implies more bits than this, aliasing results.

In some cases where even small gains in FPGA resource usage are desired, the user might wish to purposely specify a larger than needed (aliased) address range to reduce the number of address bits that need to be decoded.

#### ***Read Chip Enable Bus (Bus2IP\_RdCE(0:y))***

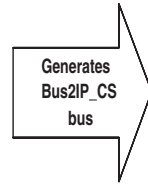
The Bus2IP\_RdCE bus is the same as the Bus2IP\_CE bus except that the Bus2IP\_RdCE signals are only asserted if the requested transaction is a read.

#### ***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 :=

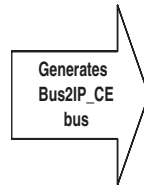
```
-- Memory space identifiers
(
  IPIF_IRPT  -- IPIF Interrupt service
  USER_00,  -- User Control Register Bank (4 registers x 16 bits wide)
  USER_01,  -- User Status Register Bank (16 registers x 8 bits wide)
  Data_Buffer, -- User Data Buffer (BRAM 512 x 64 bits wide)
  IPIF_RST   -- IPIF Reset/MIR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



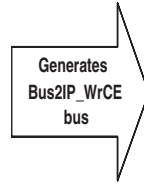
- Bus2IP\_CS(0) IPIF Interrupt Service Chip Select
- Bus2IP\_CS(1) USER00 Control Register Bank Chip Select
- Bus2IP\_CS(2) USER01 Status Register Bank Chip Select
- Bus2IP\_CS(3) User Data Buffer Chip Select
- Bus2IP\_CS(4) IPIF\_RST Service Chip Select

**C\_ARD\_NUM\_CE\_ARRAY** : INTEGER\_ARRAY\_TYPE :=

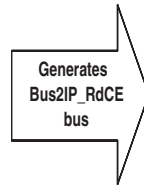
```
-- Memory space Chip Enable definition (in bits)
(
  16, -- IPIF Interrupt service (always 16 CEs)
  4,  -- User Control Register Bank (4 registers = 4 CEs)
  16, -- User Status Register Bank (16 registers 16 CEs)
  1,  -- User Data Buffer (BRAM 512 x 32 bits wide)
  1  -- IPIF Reset/MIR service (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



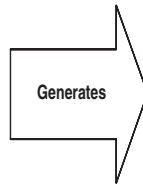
- Bus2IP\_WrCE(0:15) IPIF Interrupt Service WrCE (Registers 0 to 15)
- Bus2IP\_WrCE(16:19) USER00 WrCE (Registers 0 to 3)
- Bus2IP\_WrCE(20:35) USER01 WrCE (Registers 0 to 15)
- Bus2IP\_WrCE(36) User Data Buffer WrCE
- Bus2IP\_WrCE(37) IPIF Reset/MIR Service Reg 0 WrCE



- 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)
  32, -- User Data Buffer (BRAM 512 x 32 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 44: 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 param-

eterization adds or removes IPIF services and user IP functions (which changes the size and ordering of the CS and CE buses). **Table 33** 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\ipilib\pcores\proc_common_v2_00_a\hdl\vhdl\ipif_pkg.vhd`.

The following declarations in the user's VHDL source makes the functions visible:

```
library proc_common_v2_00_a;
```

```
use proc_common_v2_00_a.ipif_pkg.<name of function used>; -- or ".all" to make all functions visible
```

An example of how these functions are used is shown in **Figure 45**.

**Table 33: 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:  <pre>constant USER0_PRESENT : boolean := <b>find_ard_id</b>(C_ARD_ID_ARRAY, USER0);</pre>
	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:  <pre>constant USER0_ARRAY_INDEX : integer := <b>get_id_index</b>(C_ARD_ID_ARRAY, USER0);</pre>  <pre>constant USER0_CS_INDEX : integer := <b>get_id_index</b>(C_ARD_ID_ARRAY, USER0);</pre>
	id_array	INTEGER_ARRAY_TYPE		
	id	integer		

Table 33: IPIF Support VHDL Functions. (Continued)

VHDL Function Name	Input Parameter Name	Input Parameter Type	Return Type	Description
get_id_index_ibo			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 accommodates cases where a 'get_id_index' function must be called even for an 'id' that is not used. (A separate guard based on the find_ard_id function can be used to detect errors.)  Example: constant USER0_INDEX : integer := <b>get_id_index_ibo</b> (C_ARD_ID_ARRAY, USER0);
	id_array	INTEGER_ARRAY_TYPE		
	id	integer		
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 := <b>calc_num_ce</b> (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_array' parameter and the returned value from the use of the <b>get_id_index</b> function.  Example: constant USER0_START_CE_INDEX : integer := <b>calc_start_ce_index</b> (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 := <b>find_id_dwidth</b> (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);

  -- Determine 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;

```

Figure 45: 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 its 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 34 through Table 39.

Table 34: Write to 8-Bit Target by a 32-Bit OPB IPIF

Access Type (1)	Bus Write Data Bytes				Bus BE				Bus2IP_Data				Bus2IP_BE			
	B L 0	B L 1	B L 2	B L 3	B E 0	B E 1	B E 2	B E 3	B L 0	B L 1	B L 2	B L 3	B E 0	B E 1	B E 2	B E 3
Byte Write	D 0	D 1	D 2	D 3	1	0	0	0	D 0	U	U	U	1	0	0	0
	D 0	D 1	D 2	D 3	0	1	0	0	D 1	U	U	U	1	0	0	0
	D 0	D 1	D 2	D 3	0	0	1	0	D 2	U	U	U	1	0	0	0
	D 0	D 1	D 2	D 3	0	0	0	1	D 3	U	U	U	1	0	0	0
	D 0	D 1	D 2	D 3	0	0	0	0		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 35: Write to 16-Bit Target by a 32-Bit OPB IPIF

Access Type (1)	Bus Write Data Bytes				Bus BE				Bus2IP_Data				Bus2IP_BE			
	B L 0	B L 1	B L 2	B L 3	B E 0	B E 1	B E 2	B E 3	B L 0	B L 1	B L 2	B L 3	B E 0	B E 1	B E 2	B E 3
Byte Write	D 0	D 1	D 2	D 3	1	0	0	0	D 0	U	U	U	1	0	0	0
	D 0	D 1	D 2	D 3	0	1	0	0		D 1	U	U	0	1	0	0
	D 0	D 1	D 2	D 3	0	0	1	0	D 2	U	U	U	1	0	0	0
	D 0	D 1	D 2	D 3	0	0	0	1		D 3	U	U	0	1	0	0
	D 0	D 1	D 2	D 3	0	0	0	0		U	U	U	1	0	0	0
Half-Word Write	D 0	D 1	D 2	D 3	1	1	0	0	D 0	D 1	U	U	1	1	0	0
	D 0	D 1	D 2	D 3	0	0	1	1	D 2	D 3	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 36: Write to 32-Bit Target by a 32-Bit OPB IPIF

Access Type (1)	Bus Write Data Bytes				Bus BE				Bus2IP_ Data				Bus2IP_ BE			
	B L 0	B L 1	B L 2	B L 3	B E 0	B E 1	B E 2	B E 3	B L 0	B L 1	B L 2	B L 3	B E 0	B E 1	B E 2	B E 3
Byte Write	D 0	D 1	D 2	D 3	1	0	0	0	D 0	U	U	U	1	0	0	0
	D 0	D 1	D 2	D 3	0	1	0	0		D 1	U	U	0	1	0	0
	D 0	D 1	D 2	D 3	0	0	1	0		U	D 2	U	0	0	1	0
	D 0	D 1	D 2	D 3	0	0	0	1		U	U	D 3	0	0	0	1
	D 0	D 1	D 2	D 3	1	1	0	0	D 0	D 1	U	U	1	1	0	0
Half-Word Write	D 0	D 1	D 2	D 3	0	0	1	1		U	D 2	D 3	0	0	1	1
	D 0	D 1	D 2	D 3	1	1	1	1	D 0	D 1	D 2	D 3	1	1	1	1
Word Write	D 0	D 1	D 2	D 3	1	1	1	1	D 0	D 1	D 2	D 3	1	1	1	1

Notes:

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

Table 37: 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			
	B L 0	B L 1	B L 2	B L 3	B E 0	B E 1	B E 2	B E 3	B L 0	B L 1	B L 2	B L 3	B E 0	B E 1	B E 2	B E 3
Byte Read	D 0	U	U	U	1	0	0	0	D 0	U	U	U	1	0	0	0
	U	D 0	U	U	0	1	0	0	D 0	U	U	U	1	0	0	0
	U	U	D 0	U	0	0	1	0	D 0	U	U	U	1	0	0	0
	U	U	U	D 0	0	0	0	1	D 0	U	U	U	1	0	0	0
	D 0	U	U	U	1	0	0	0	D 0	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 38: 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			
	B L 0	B L 1	B L 2	B L 3	B E 0	B E 1	B E 2	B E 3	B L 0	B L 1	B L 2	B L 3	B E 0	B E 1	B E 2	B E 3
Byte Read	D 0	U	U	U	1	0	0	0	D 0	U	U	U	1	0	0	0
	U	D 1	U	U	0	1	0	0		D 1	U	U	0	1	0	0
	U	U	D 0	U	0	0	1	0	D 0	U	U	U	1	0	0	0
	U	U	U	D 1	0	0	0	1		D 1	U	U	0	1	0	0
Half-Word Read	D 0	D 1	U	U	1	1	0	0	D 0	D 1	U	U	1	1	0	0
	U	U	D 0	D 1	0	0	1	1	D 0	D 1	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 39: Read from 32-Bit Target by a 32-Bit OPB IPIF

Access Type (1)	Bus Read Data Bytes				Bus BE				IP2Bus_ Data				Bus2IP_ BE			
	B L 0	B L 1	B L 2	B L 3	B E 0	B E 1	B E 2	B E 3	B L 0	B L 1	B L 2	B L 3	B E 0	B E 1	B E 2	B E 3
Byte Read	D 0	U	U	U	1	0	0	0	D 0	U	U	U	1	0	0	0
	U	D 1	U	U	0	1	0	0		D 1	U	U	0	1	0	0
	U	U	D 2	U	0	0	1	0		U	D 2	U	0	0	1	0
	U	U	U	D 3	0	0	0	1		U	U	D 3	0	0	0	1
Half-Word Read	D 0	D 1	U	U	1	1	0	0	D 0	D 1	U	U	1	1	0	0
	U	U	D 2	D 3	0	0	1	1		U	D 2	D 3	0	0	1	1
Word Read	D 0	D 1	D 2	D 3	1	1	1	1	D 0	D 1	D 2	D 3	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 employs 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 array ports (such as, for example, port 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 preceded 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
```

## Design Implementation

### Target Technology

See [Table 9 on page 17](#) for a list of supported Xilinx FPGA families.

### Device Utilization and Performance Benchmarks

The PLB IPIF benchmarks are shown in [Table 40](#) for a Virtex-II -7 FPGA.

Table 40: OPB IPIF FPGA Performance and Resource Utilization Benchmarks (Virtex-II Pro -7)

Parameter Values	Device Resources				f <sub>MAX</sub>
	Pipeline Model	Slices	Slice Flip-Flops	4-input LUTs	f <sub>MAX_REG</sub> <sup>(2)</sup>
No IPIF services, User IP with 1 CE	4	27	36	9	287.3 MHz
	5	67	112	13	344.6 MHz
	7	103	182	14	344.4 MHz
Write Buffer, Address Counter, User IP with 1 CE, Burst Support	4	130	104	178	147.6 MHz
	5	171	181	186	195.6 MHz
	7	178	185	191	186.2 MHz
2 interrupts, Reset, MIR, User IP with 8 CEs	4	88	57	88	106.3 MHz
	5	127	117	127	151.5 MHz
	7	143	178	143	152.2 MHz
2 interrupts, Reset, MIR, User IP with 8 CEs, Burst Support, Read and Write FIFO	4	378	255	613	103.1 MHz
	5	404	346	561	141.8 MHz
	7	425	430	611	138.7 MHz

**Table 40: OPB IPIF FPGA Performance and Resource Utilization Benchmarks (Virtex-II Pro -7) (Continued)**

2 Interrupts, Reset, MIR, User IP with 8 CEs, Burst Support, Read and Write FIFO, Address Counter, Write Buffer	4	485	373	796	102.8 MHz
	5	544	441	845	125.9 MHz
	7	526	442	816	110.4 MHz

**Notes:**

1.

## 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 separate enable signals of the OPB 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 10, "OPB IPIF I/O Signals," on page 18.](#)

## 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.

## Revision History

Date	Version	Revision
2/4/05	1.0	Initial Xilinx release
4/27/05	1.1	Fixed problem with missing figures.
9/7/05	1.2	Fixed other issues with missing figures
12/2/05	1.3	Added Spartan-3E to supported device listing.