User Core Templates Reference Guide

April 2003

XILINX®

User Core Templates Reference Guide

The following table shows the revision history for this document.

	Version	Revision
11/19/02	1.0	Initial release.
1/17/03	1.1	Updated for EDK SP3
4/1/03	1.2	Updated for EDK 3.2 SP1

XILINX®

Chapter 1

Adding User Cores to Your Embedded System

Overview

There are several types of user logic associated with FPGA designs that incorporate an embedded processor subsystem. User logic may be unrelated to the embedded processor subsystem, it may have a weak coupling to the processor, or it may be included as an integral part of the processor subsystem. There are also several configurations that can be used to connect user cores and user logic to an embedded subsystem. Some of these configurations are shown below in Figure 1-1. This document deals primarily with the creation and use of a user core that is intended to be included as part of the embedded processor subsystem. This configuration is shown as System 3 in Figure 1-1.

Definitions of the terms used in the document are:

- FPGA - the entire design to be loaded into the FPGA, consisting of an embedded processor subsystem (created by the platform generation tools) and other logic (created by the user).
- Embedded processor subsystem - a design described in an MHS (Microprocessor Hardware Specification) file and generated with the platform generation tools. This typically consists of one or more processors, bus peripherals, bus arbiters, bridges, support logic (such as reset circuitry), and user cores.
- User core - a core designed to attach to an embedded processor bus, such as OPB or PLB. From the viewpoint of the platform generation tools, the user core looks just like the Xilinx-supplied embedded system cores.
- User logic - to simplify the process of attaching a user core to a CoreConnect bus, the user core can make use of a portable, predesigned bus interface (called the IP Interface, IPIF) that takes care of the bus interface signals, bus protocol, and other interface issues. The IPIF presents an interface to the user logic called the IP InterConnect (IPIC). User logic is logic that has been designed with an IPIC interface to make use of the IPIF bus attachment and other services. User logic that is designed with an IPIC has the advantage that it is portable and can be easily reused on different processor buses by changing the IPIF to which it is attached.
- User core template - the user core template simplifies the task of attaching the IPIF to user logic. The user core template is a VHDL file that instantiates the IPIF and provides most of the VHDL code required to create a user core. The template provides a place to instantiate the user logic, which can be VHDL or a black box created from verilog, schematic, etc. There are a total of nine user core templates that address specific bus attachment needs. They will be described in greater detail below.
- Other logic - logic that is not included as part of the embedded processor subsystem. It may have some connection to the embedded subsystem, but it typically does not have a bus interface such as an IPIC and it is not considered a bus peripheral, although a bus peripheral may provide an interface from the embedded system to the other logic.

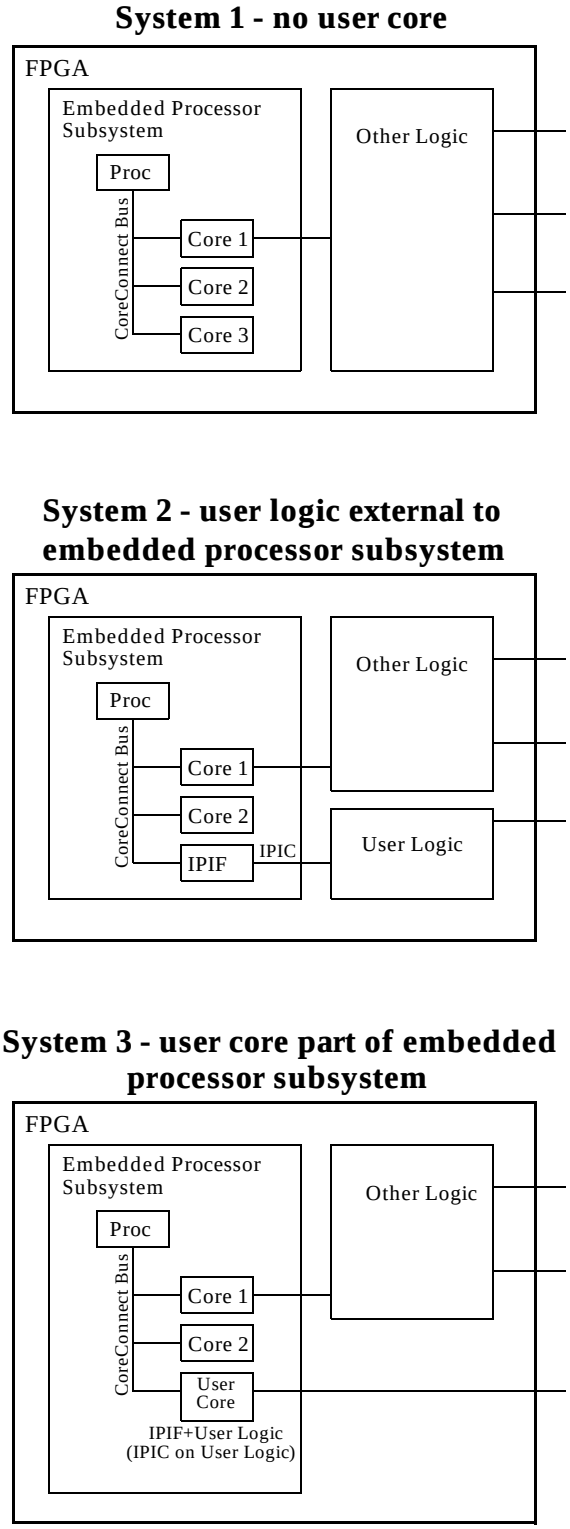

Figure 1-1: **Three Embedded System Configurations**

OPB User Core Templates

The user core templates provide a convenient way to get your user logic attached to the OPB bus. These templates are provided as VHDL source and provide a framework into which user logic can be instantiated. Each user core template contains an instantiation of an IPIF (IP Interface). The IPIF instantiated in each template is customized so that only the features and services required by the template are used. For example, the user core template that provides a simple OPB slave interface uses only the slave signal set and bus attachment logic required for a slave interface.

The user core templates are provided as starting points for building custom OPB peripherals. Each template has some degree of parameterization as well, so the best way to use the templates is to find a template that is closest to the features and services required by your core and customize from there.

Templates are named according to the features that are provided by the template. For OPB slaves, there are four templates that provide varying degrees of services for the user. OPB slave templates are denoted by the ssp<*n*> (Slave Services Package *n*) suffix, where in general the higher number indicates inclusion of more services. Table 1-1 below shows the OPB slave templates available and the services provided by each.

Table 1-1: **OPB User Core Templates (Slave)**

	OPB Slave Attach-ment	Address Decode	Module Identifica-tion Register	Reset Register	Device Interrupt Controller	R/W FIFOs	R/W Packet FIFOs
opb_core_ssp0	●	●	●	●			
opb_core_ssp1	●	●	●	●	●		
opb_core_ssp2	●	●	●	●	●	●	
opb_core_ssp3	●	●	●	●	●		●

For OPB masters, there are five templates that provide varying degrees of services for the user. OPB master templates are denoted by the msp<*n*> (Master Services Package *n*) suffix, where in general the higher number indicates inclusion of more services. Table 1-2 below shows the OPB master templates available and the services provided by each.

Table 1-2: **OPB User Core Templates (Master)**

	OPB Slave Attach-ment	Address Decode	Module Identif-cation Register	Reset Register	Device Interrupt Control-ler	R/W FIFOs	R/W Packet FIFOs	OPB Master Attach-ment	Simple DMA	DMA with scatter/ gather
opb_core_msp0								●		
opb_core_msp1	●	●	●	●				●		
opb_core_msp2	●	●	●	●	●			●		
opb_core_msp3	●	●	●	●	●	●		●	●	
opb_core_msp4	●	●	●	●	●		●	●		●

Creating a User Core

The process of creating a user core from the OPB user core templates is shown in Figure 1-2. First, an appropriate template is selected based on the level of services required. Then, the user logic is inserted into the template and the template file names and entities

are renamed if desired. Finally, the template MPD and PAO files are modified as necessary and the user core is instantiated in the system's MHS file. For definitions of the MPD, PAO, and MHS file formats, see the *Embedded System Tools Guide* document (est_guide.pdf) in the $EDK/doc directory. Some important points about this process are:

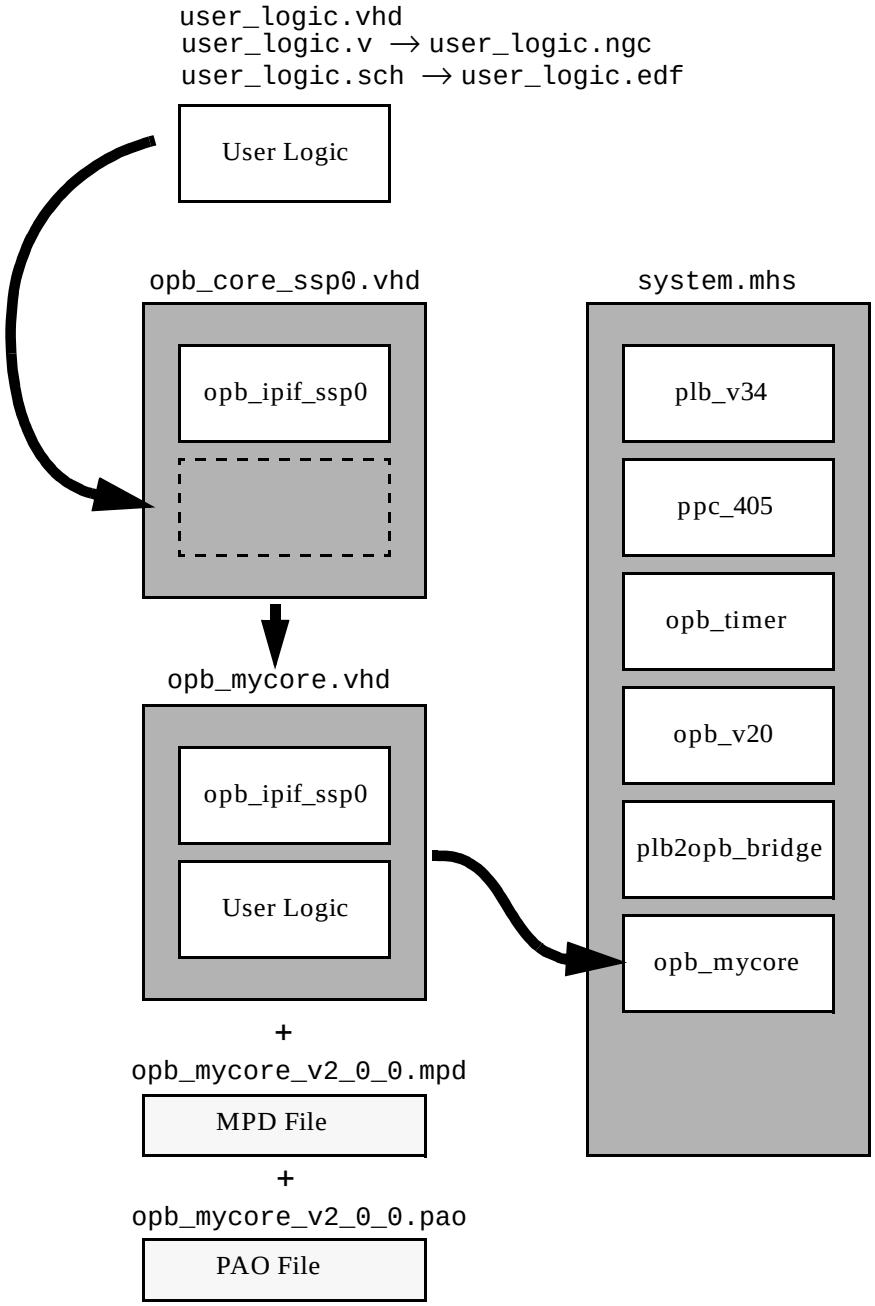

Figure 1-2: **OPB User Core Process**

- The User Logic can either be a VHDL entity or any type of black box (fixed netlist) recognized by the Xilinx implementation tools. Black boxes are either NGC files created by XST (Xilinx Synthesis Technology) or EDIF files created by a variety of sources.
- Each template directory contains a VHDL source template, an MPD template, and a

PAO template. These files are located in the following directories (the template for Slave Services Package 0 is used for this example):
- opb_core_ssp0.vhd is in:
$EDK/hw/user_core_templates/myip/opb_core_ssp0_v1_00_a/hdl/vhdl/
- opb_core_ssp0_v2_0_0.mpd is in:
$EDK/hw/user_core_templates/myip/opb_core_ssp0_v1_00_a/data/
- opb_core_ssp0_v2_0_0.pao is in:
$EDK/hw/user_core_templates/myip/opb_core_ssp0_v1_00_a/data/

- Note that the version number in the MPD and PAO file names are the Platform Specification File (PSF) format version, not the core version number. The current PSF format version is v2.0.0.
- Lines in the templates that can be customized by the user (to change the peripheral name, for example) contain the comment --USER--. Searching for --USER-- in the VHDL, MPD, and PAO will guide the user to all lines that may require modification.
- For detailed instructions on how to use a particular template, refer to the chapter on that template.

IPIC Interface (v3.00a)

To effectively use the OPB user core templates, the user logic should be designed with an IP Interconnect (IPIC). The IPIC is a simple set of signals that connect the user logic to the IPIF logic. The IPIC is common to all Xilinx IPIFs, so user logic that is designed with an IPIC can be easily ported to a different bus simply by using the appropriate IPIF (and user core template). IPIFs currently exist for the two Xilinx-supported CoreConnect buses, OPB and PLB. The IPIC is designed so that only the subset required in a particular application can be used. For simple slaves, only the slave interface signals are used, while for simple masters, only the master signals are used. This simplifies use of the IPIC and reduces the complexity of interfacing to simple peripherals.

IPIC Signal Set

The signal set for slave and master peripherals is shown below in.

Table 2: **IPIC I/O Signals**

Signal Name	Range	I/O	Description	Page
Bus2IP_Addr	0:C_<bus>_AWIDTH-1	I	Address to User Logic	
Bus2IP_BE	0:C_<bus>_DWIDTH / 8-1	I	Byte enables to User Logic	
Bus2IP_Burst	none	I	Burst-mode qualifier to User Logic	
Bus2IP_Clk	none	I	IPIC clock. Identical to the <bus> clock	
Bus2IP_CE	0:C_NUM_CE-1	I	“chip” enable to User Logic	
Bus2IP_CS	0:C_NUM_CS-1	I	“chip” select to User Logic	
Bus2IP_Data	0:C_<bus>_DWIDTH-1	I	Data to User Logic	
Bus2IP_Freeze	none	I	Tells the User Logic to freeze	
Bus2IP_RdCE	0:C_NUM_CE-1	I	Read enables to User Logic	
Bus2IP_Reset	none	I	Signal to reset the User Logic	
Bus2IP_RNW	none	I	Read / Not Write Signal to User Logic	
Bus2IP_WrCE	0:C_NUM_CE-1	I	Write enables to User Logic	
IP2Bus_Ack	none	O	Acknowledgement from User Logic	
IP2Bus_Data	0:C_<bus>_DWIDTH-1	O	Data from IP	
IP2Bus_Error	none	O	Error response	

Table 2: **IPIC I/O Signals *(Continued)***

Signal Name	Range	I/O	Description	Page
IP2Bus_Intr	0:C_IP_INTR_NUM-1	O	Interrupt event signals from User Logic	
IP2Bus_PostedWrInh	none	O	Posted write inhibit from User Logic	
IP2Bus_Retry	none	O	Retry response from User Logic	
IP2Bus_ToutSup	none	O	Timeout suppress from User Logic	
Bus2IP_MstError	none	I	Master Error from IPIF	
Bus2IP_MstLastAck	none	I	Master Last Acknowledge from IPIF	
Bus2IP_MstAck	none	I	Master Acknowledge from IPIF	
Bus2IP_MstRetry	none	I	Master Retry from IPIF	
Bus2IP_MstTimeOut	none	I	Master Timeout from IPIF	
IP2Bus_Addr	0:C_<bus>_AWIDTH-1	O	<bus> address for the master transaction	
IP2Bus_Clk	none	O	Possible future signal to allow for dual-clock-domain (asynchronous) FIFOs	
IP2Bus_MstBE	0:C_<bus>_DWIDTH / 8-1	O	Byte-enables qualifiers from User Logic	
IP2Bus_MstBurst	none	O	Burst qualifier from User Logic	
IP2Bus_MstBusLock	none	O	Bus-lock qualifier from User Logic	
IP2Bus_MstNum	0:3	O	Burst size indicator from User Logic	
IP2Bus_MstReq	none	O	Master request from User Logic	
IP2Bus_MstRNW	none	O	Read / Not Write from User Logic	
IP2IP_Addr	0:C_<bus>_AWIDTH-1	O	Local device address for the master transaction	

Slave Interface Signal Descriptions

These signals listed below are primarily associated with the slave interface of the IPIC; however, some of the signals are shared with the master interface and IPIFs that contain a master (such as DMA) may make use of the slave signals to complete local transactions. These transactions are described in the chapters on the user templates that provide master services in the IPIF.

Bus2IP_Addr

This is the address bus from the IPIF to the user logic. This bus is the same width as the host bus address bus. The Bus2IP_Addr bus can be used for additional address decoding or as input to addressable memory devices.

Bus2IP_BE

The Bus2IP_BE is a bus of Byte Enable qualifiers from the IPIF to the user logic. A bit in the Bus2IP_BE set to ‘1’ indicates that the associated byte lane contains valid data. For example, if Bus2IP_BE = 0011, this indicates that byte lanes 2 and 3 contain valid data.

Bus2IP_Burst

The Bus2IP_Burst signal from the IPIF to the user logic indicates that the current transaction is a burst transaction.

Bus2IP_Clk

This is the clock input to the user logic. All IPIC signals are synchronous to this clock. It is identical to the <bus>_Clk signal that is an input to the user core. In an OPB core,

Bus2IP_Clk is the same as OPB_Clk, and in a PLB core, it is the same as PLB_Clk. No additional buffering is provided on the clock; it is passed through as is.

Bus2IP_CE

Bus2IP_CE is a bus of "chip" enables from the IPIF to the user core. In the general case, there can be an arbitrary number of CE signals for each Bus2IP_CS signal, but the number of CE signals is fixed in some of the user core templates. The assertion of a bit in Bus2IP_CE indicates that a point address associated with the CE has been decoded. For example, the Bus2IP_CS signal indicates a decode of a block of addresses, and the Bus2IP_CE signal indicates a decode of a particular register or address within the block of addresses.

Bus2IP_CS

Bus2IP_CS is a bus of "chip" select signals from the IPIF to the user core. It indicates a decode within a block of addresses defined by a base address and a high address. In the simplest templates, there is only one Bus2IP_CS signal.

Bus2IP_Data

This is the data bus from the IPIF to the user logic; it is used for both master and slave transactions. It is the same width as the host bus data bus.

Bus2IP_Freeze

The Bus2IP_Freeze signal is an input to the user logic that indicates a freeze has been requested. A freeze is typically used in a debugging situation in which the core should gracefully stop it's internal operations but remain active on the bus. An example is a watchdog timer which should be stopped when a software breakpoint is reached so that spurious system resets are not generated. It is up to the user core to define the action caused by the Bus2IP_Freeze input.

Bus2IP_RdCE

The Bus2IP_RdCE bus is an input to the user logic. It is Bus2IP_CE qualified by a read transaction.

Bus2IP_Reset

Signal to reset the User Logic; asserts whenever the <bus>_Rst signal does and, if the Reset block is included, whenever there is a software-programmed reset.

Bus2IP_RNW

Bus2IP_RNW is an input to the user logic that indicates the transaction type (read or write). Bus2IP_RNW = 1 indicates a read transaction and Bus2IP_RNW = 0 indicates a write transaction. It is valid whenever at least one Bus2IP_CS is active.

Bus2IP_WrCE

The Bus2IP_WrCE bus is an input to the user logic. It is Bus2IP_CE qualified by a write transaction.

IP2Bus_Ack

IP2Bus_Ack is the read / write acknowledgement from the user logic to the IPIF. For writes, it indicates the data has been taken by the user logic. For reads, it indicates that valid data is available. For immediate acknowledgement (such as for a register read / write), this signal can be tied to '1'. Wait states can be inserted in the transaction by delaying the assertion of IP2Bus_Ack. If the IP2Bus_Ack will be delayed more than 8 clocks, then the IP2Bus_ToutSup (timeout suppress) signal must also be asserted to prevent a timeout on the host bus.

IP2Bus_Data

This is the data bus from the user logic to the IPIF; it is used for both master and slave transactions. It is the same width as the host bus data bus.

IP2Bus_Error

This signal from the user logic to the IPIF indicates an error has occurred during the current transaction. It is valid when IP2Bus_Ack is asserted.

IP2Bus_Intr

The IP2Bus_Intr bus is an output from the user logic to the IPIF that consists of interrupt event signals to be detected and latched inside the IPIF.

IP2Bus_PostedWrInh

This signal from the user logic to the IPIF indicates that posted writes should be inhibited. Normally burst write operations are treated as posted writes to improve performance, but assertion of the IP2Bus_PostedWrInh signal indicates that all writes should be treated as single-beat write transactions.

IP2Bus_Retry

IP2Bus_Retry is a response from the user logic to the IPIF that indicates the currently requested transaction cannot be completed at this time and that the requesting master should retry the operation. If the IP2Bus_Retry signal will be delayed more than 8 clocks, then the IP2Bus_ToutSup (timeout suppress) signal must also be asserted to prevent a timeout on the host bus.

IP2Bus_ToutSup

The IP2Bus_ToutSup must be asserted by the user logic whenever its acknowledgement or retry response will take longer than 8 clock cycles.

Master Interface Signal Descriptions

These signals listed below are primarily associated with the master interface of the IPIC; however, some of the master signals are shared with the slave interface and IPIFs that contain a master (such as DMA) may make use if the slave signals to complete local transactions. These transactions are described in the chapters on the user templates that provide master services in the IPIF.

Bus2IP_MstError

The Bus2IP_MstError signal from the IPIF to the user logic indicates whether the transfer has an error. The signal is valid during the cycle that Bus2IP_MstLastAck is active. Note: a burst transaction reporting an error may have terminated prematurely.

Bus2IP_MstLastAck

Bus2IP_MstLastAck is a one-cycle acknowledgement of a master transaction from the IPIF to the user logic. A transaction may consist of multiple transfers (burst transaction); Bus2IP_MstLastAck will always accompany the last Bus2IP_MstAck for the transaction.

Bus2IP_MstAck

Bus2IP_MstAck is a one-cycle acknowledgement of a master transfer from the IPIF to the user logic. For writes it indicates that the IPIF has accepted the current data and is ready for the next data; for reads it indicates that valid data is present on the Bus2IP_Data bus.

Bus2IP_MstRetry

Bus2IP_MstRetry is a one-cycle alternative completion signal to Bus2IP_MstLastAck. It indicates that the requested transaction could not be performed but may succeed if retried; if IP2Bus_MstReq remains asserted in the following cycle, the IPIF will retry the transaction and may reuse any state that it has built up in support of the transaction (the user logic must leave addresses and transaction qualification signals unchanged). If otherwise the request signal is deasserted on the following cycle and the transaction is considered abandoned from the point of view of the IPIF.

Bus2IP_MstTimeOut

Bus2IP_MstTimeOut (from the IPIF to the user logic) is a one-cycle alternative completion signal to Bus2IP_MstLastAck. It indicates that the requested transaction could not be performed within the timeout interval associated with the host bus.

IP2Bus_Addr

IP2Bus_Addr is an output from the user logic to the IPIF. It is the address bus for the current master transaction. It is valid when IP2Bus_Req is active.

IP2Bus_Clk

Possible future signal from the user logic to the IPIF to allow for dual-clock-domain (asynchronous) FIFOs. Not currently used.

IP2Bus_MstBE

IP2Bus_MstBE is a bus of Byte Enables qualifiers from the user logic to the IPIF for a master transaction. A bit in the IP2Bus_MstBE set to '1' indicates that the associated byte lane contains valid data. For example, if IP2Bus_MstBE = 0011, this indicates that byte lanes 2 and 3 contain valid data.

IP2Bus_MstBurst

The IP2Bus_MstBurst qualifier from the user logic to the IPIC indicates the master transaction is a burst operation.

IP2Bus_MstBusLock

The IP2Bus_MstBusLock qualifier from the user logic to the IPIC indicates the master is requesting that the host bus be locked until IP2Bus_MstBusLock is deasserted. The assertion of IP2Bus_MstBusLock must accompany a master request, and can be deasserted at any time.

IP2Bus_MstNum

The IP2Bus_MstNum bus indicates the burst length for burst transfers. The number of transfers for the burst is IP2Bus_MstNum+1, so that a value of 0000 indicates a burst length of one, and a value of 1111 indicates a burst length of 16. Bursts may be from 1 to 16 words, halfwords, or bytes.

IP2Bus_MstReq

This signal from the user logic to the IPIF indicates that the user logic is requesting a master transaction. This request signal must remain asserted until acknowledged by Bus2IP_MstLastAck, Bus2IP_Retry, or Bus2IP_TimeOut.

IP2Bus_MstRNW

IP2Bus_MstRNW is an input to the IPIF from the user logic that indicates the transaction type (read or write). IP2Bus_MstRNW = 1 indicates a read transaction and

IP2Bus_MstRNW = 0 indicates a write transaction. It is valid when IP2Bus_MstReq is active.

IP2IP_Addr

The IP2IP_Addr signal is an output from the user logic that indicates the local device address for the master transaction. This address will be the source for a master write transaction and the sink for a master read transaction. This is used only in bus peripherals that are both master and slave and the master requires access to the slave devices to perform master operations. An example is a master that must read from a local memory and then write that data to the host bus. In this case IP2IP_Addr is used to address the local memory that provides the data for the write.

Example IPIC transactions

The timing diagrams shown below are intended to illustrate example IPIC transactions to clarify the timing relationships between signals. These timing diagrams do not completely define the behaviour of all signals and are intended to supplement the textual descriptions given above.

Slave Read – Single Beat

The timing diagram shows two typical IPIC read transactions, each started with the assertion of Bus2IP_CS and terminated with assertion of the single-cyle IP2Bus_Ack signal.

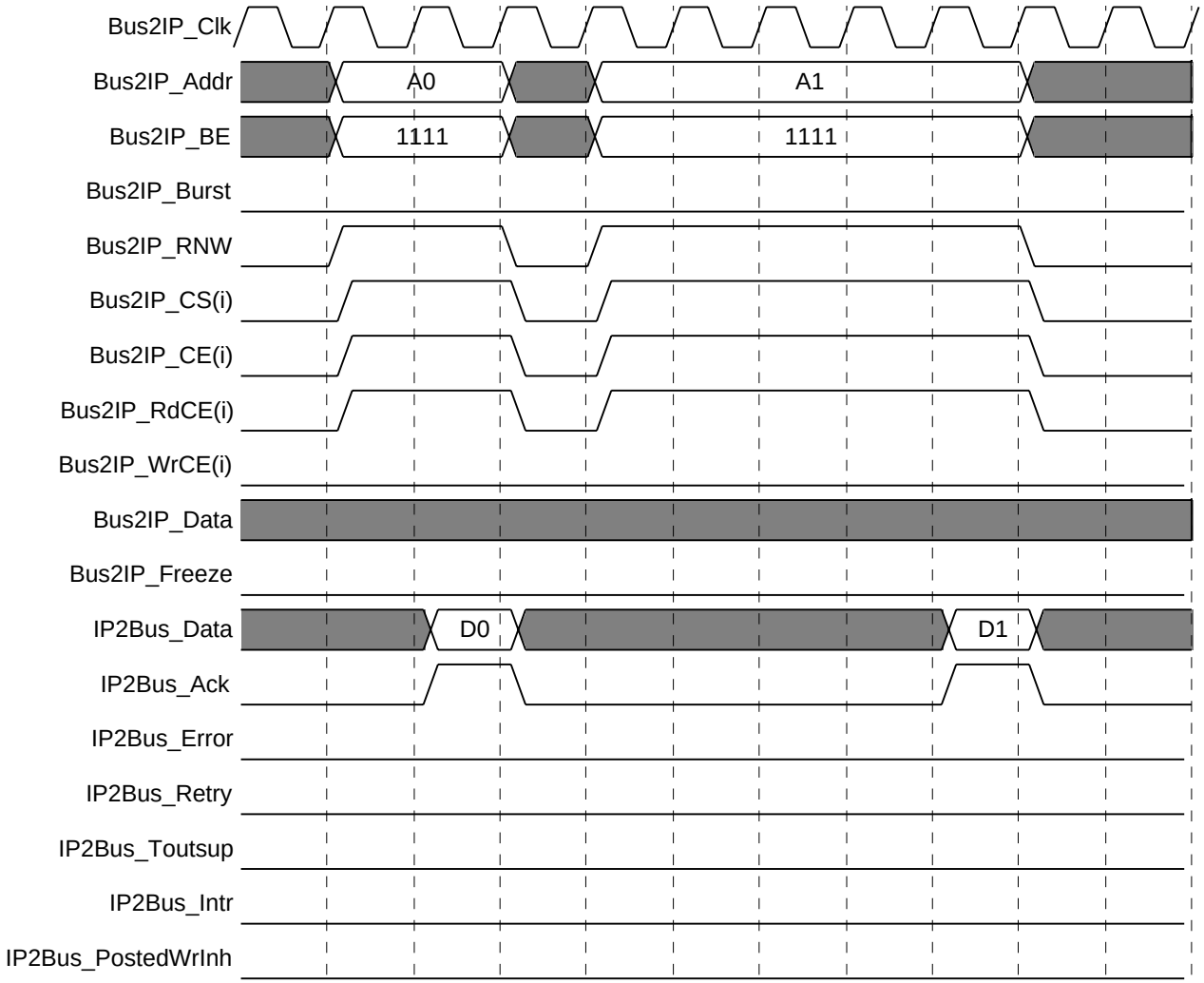

Figure 1-3: **Slave Read – Single Beat**

Slave Write – Single Beat

This example shows two typical single-beat write transactions, each starting with assertion of Bus2IP_CS and ending with assertion of the the single-cycle IP2Bus_Ack signal. Note that two Bus2IP_CE signals are shown, indicating A0 and A1 are in the same Bus2IP_CS space but are accessing different chip enables (Bus2IP_CE). This could represent writing to two sequential registers within the same device. The empty cycle between the two transactions is not required but is typical due to pipelining of the acknowledge signal to the host bus.

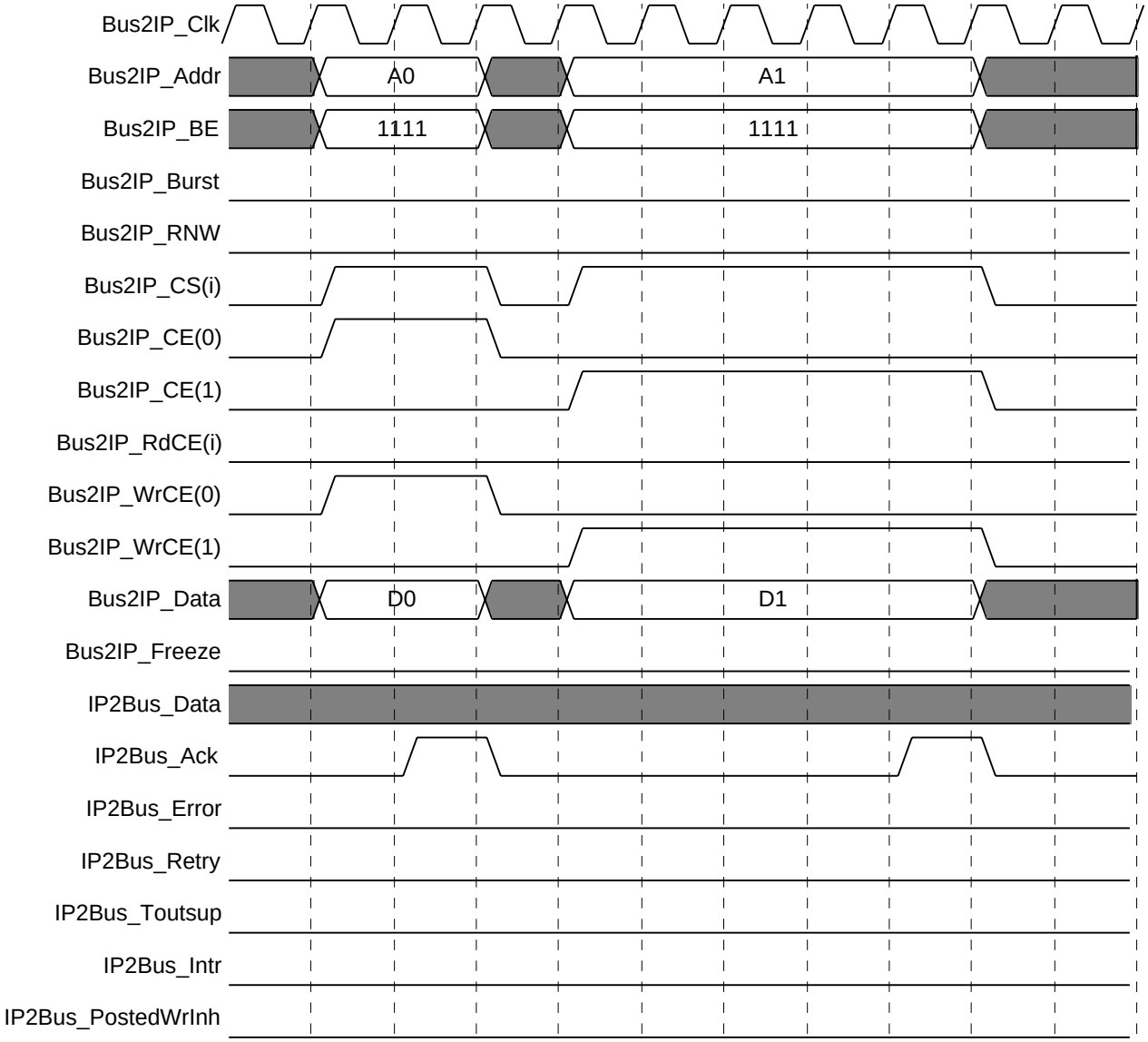

Figure 1-4: **Slave Write – Single Beat**

Slave Read – Single Beat Back to Back

Single-cycle, back to back reads are possible if supported by the host bus, the IP, and the pipelining within the IPIF is turned off (this is an advanced feature of the IPIF and is not currently accessible from the user core template; it will be available in future user core templates). The operation of the interface is the same: one data and acknowledge for each assertion of Bus2IP_CS.

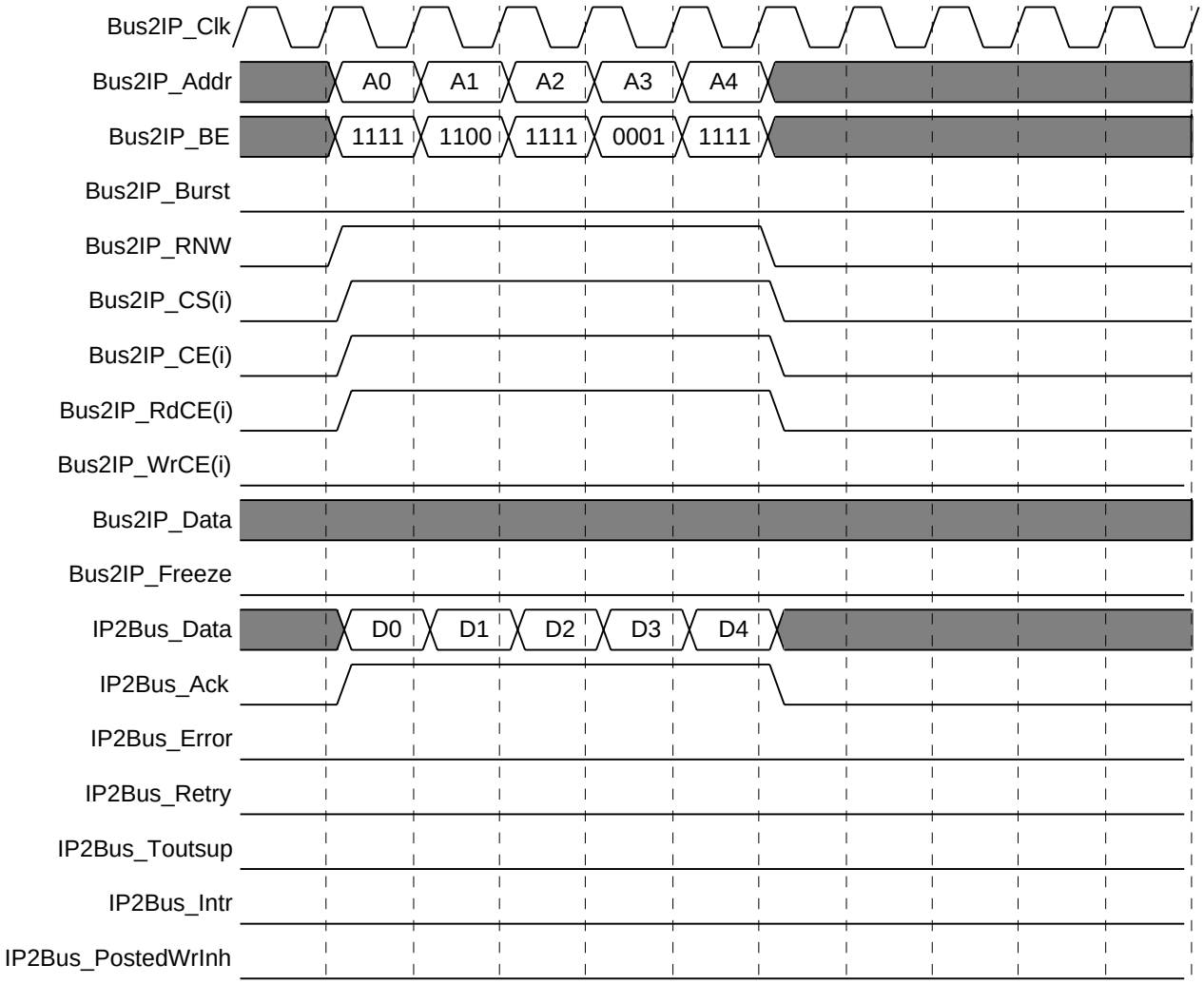

Figure 1-5: **Slave Read – Single Beat Back to Back**

Slave Write – Single Beat Back to Back

Single-cycle, back to back writes are possible if supported by the host bus, the IP, and the pipelining within the IPIF is turned off (this is an advanced feature of the IPIF and is not currently accessible from the user core template; it will be available in future user core templates). The operation of the interface is the same: one data and acknowledge for each assertion of Bus2IP_CS.

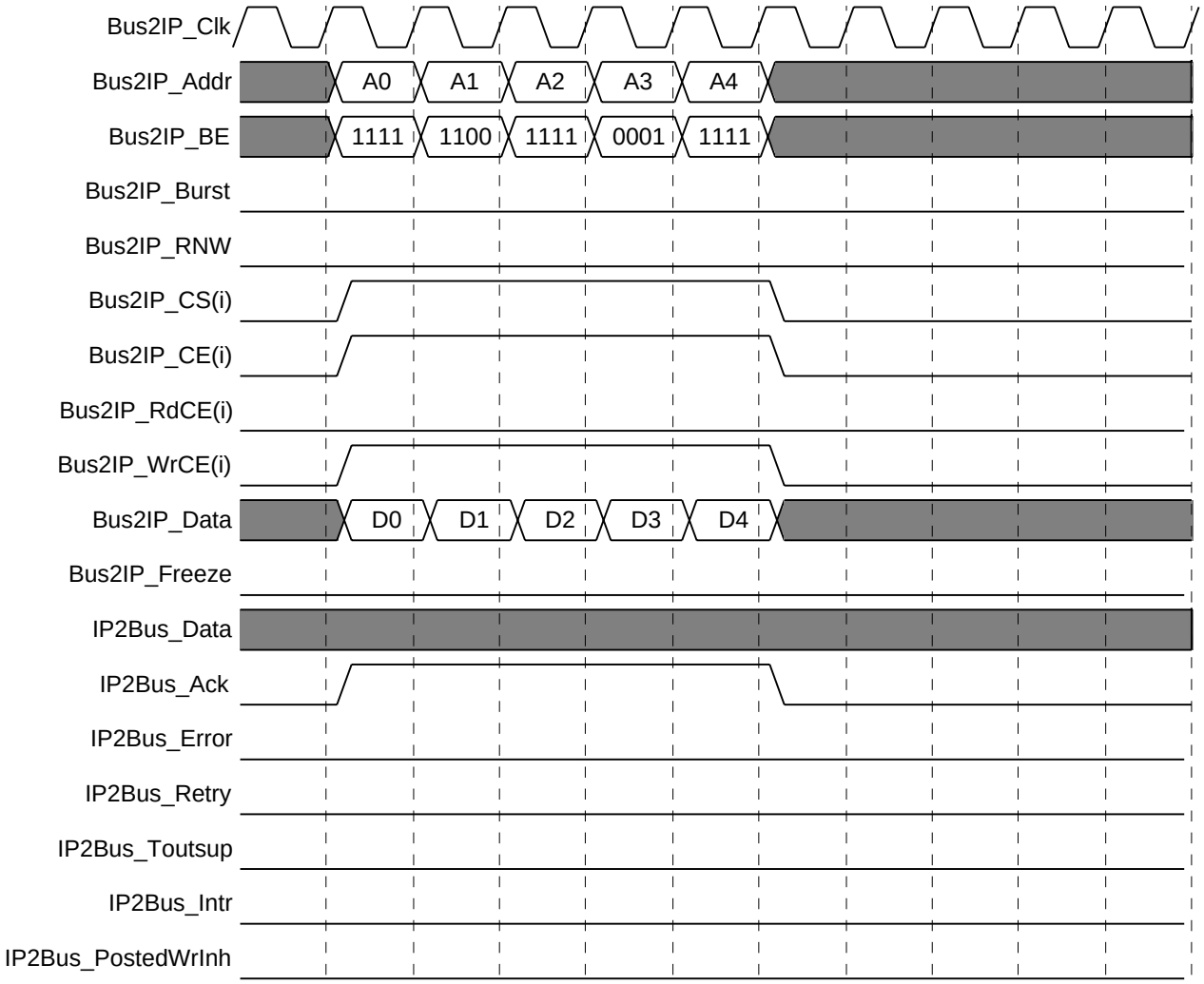

Figure 1-6: **Slave Write – Single Beat Back to Back**

Slave Read – Single Beat with Retry

Retry is an alternate transaction completion that may be used in place of IP2Bus_Ack. Assertion of IP2Bus_Retry indicates to the bus master that the transaction could not be completed but will succeed at some time in the future if retried. Retry transactions are identical to normally completed transactions except no data is returned and IP2Bus_Retry is asserted in place of IP2Bus_Ack.

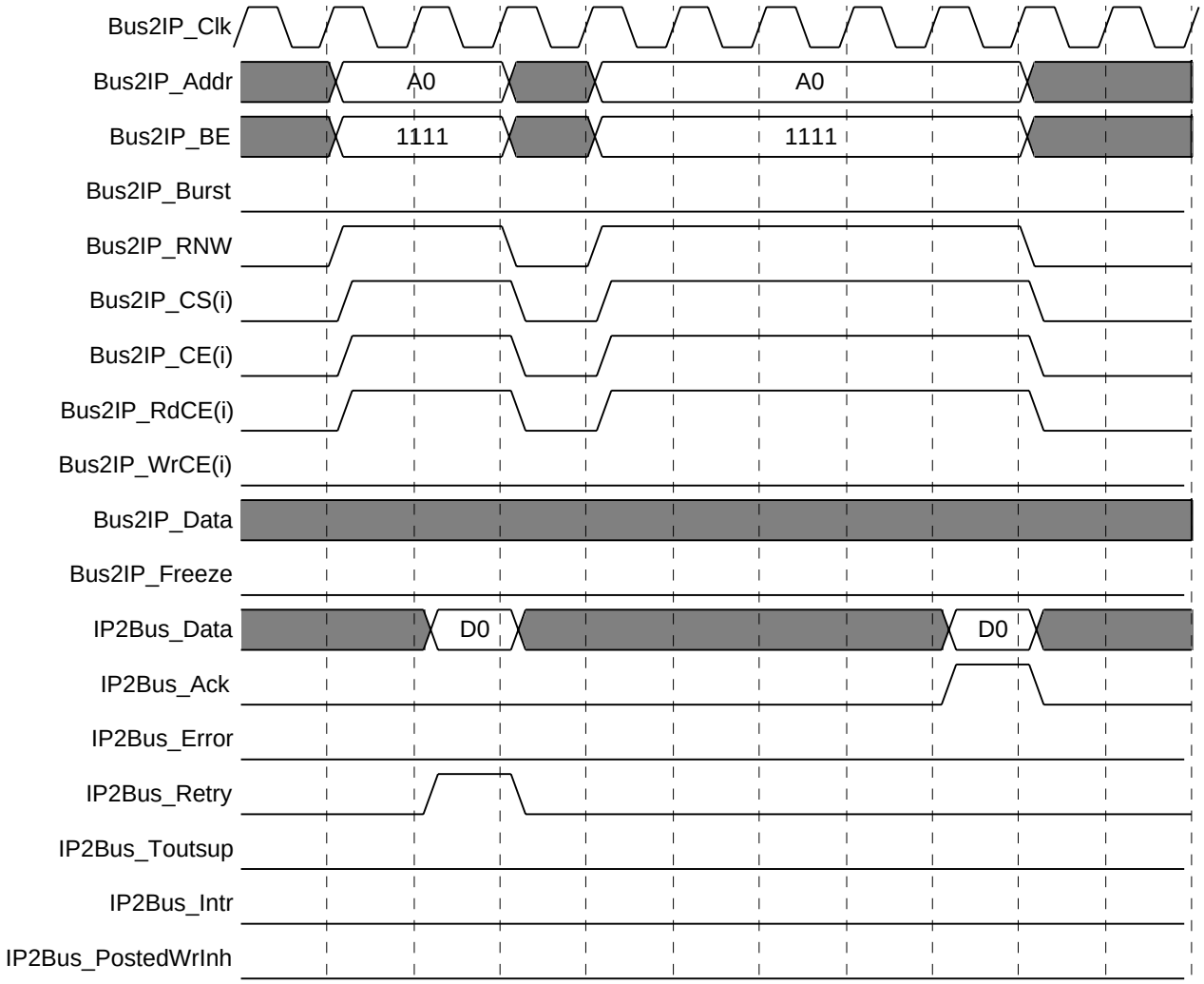

Figure 1-7: **Slave Read – Single Beat with Retry**

Slave Write – Single Beat with Retry

Retry is an alternate transaction completion that may be used in place of IP2Bus_Ack. Assertion of IP2Bus_Retry indicates to the bus master that the transaction could not be completed but will succeed at some time in the future if retried. Retry transactions are identical to normally completed transactions except no data is written and IP2Bus_Retry is asserted in place of IP2Bus_Ack.

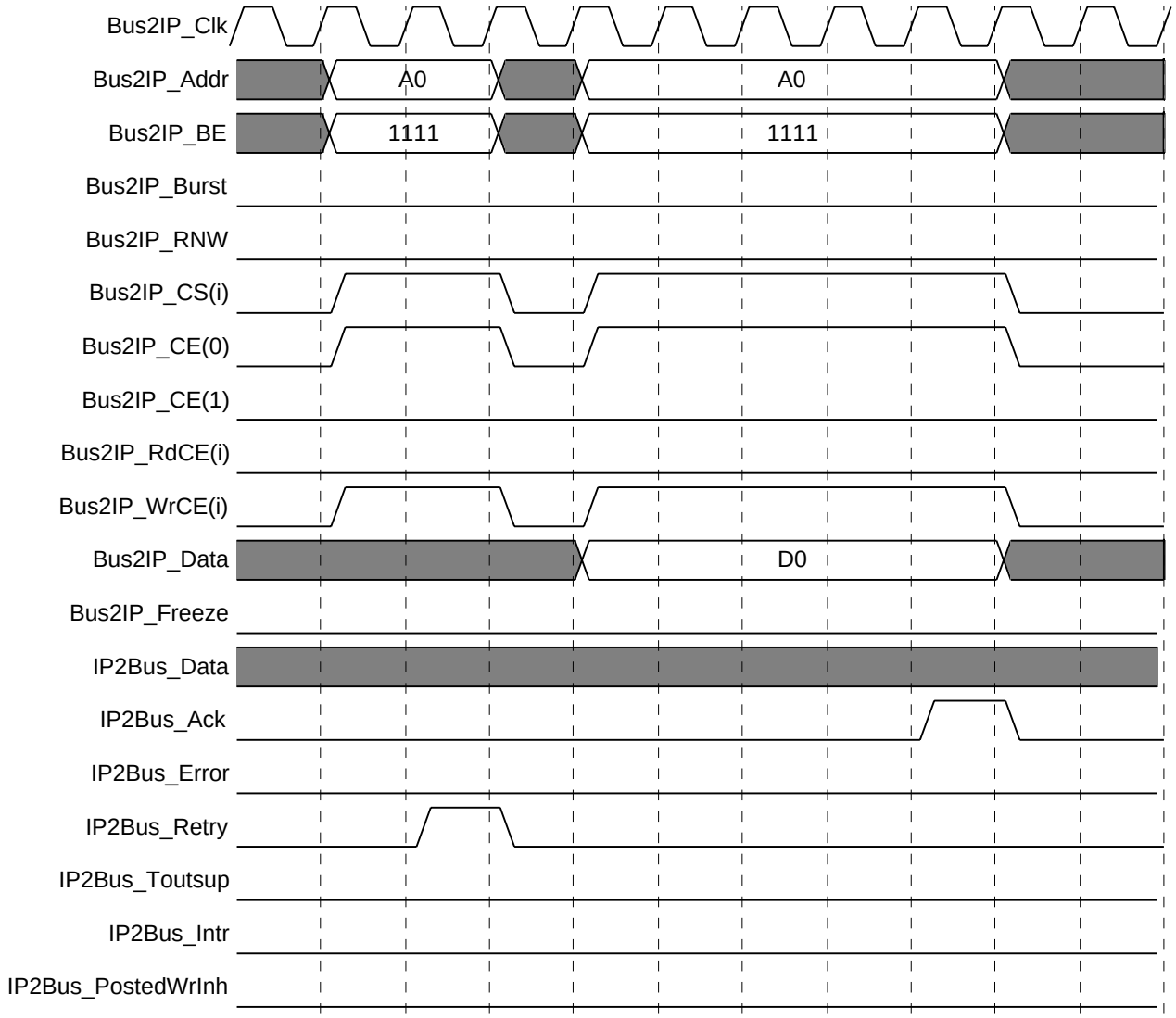

Figure 1-8: **Slave Write – Single Beat with Retry**

Slave Read – Single Beat with Timeout Suppress

IP2Bus_Toutsup must be asserted if the assertion of IP2Bus_Ack will be delayed by more than 8 clocks from Bus2IP_CS. If more than 8 clocks elapse from assertion of Bus2IP_CS without assertion of either IP2Bus_Ack or IP2Bus_Toutsup, a timeout error may occur on the host bus.

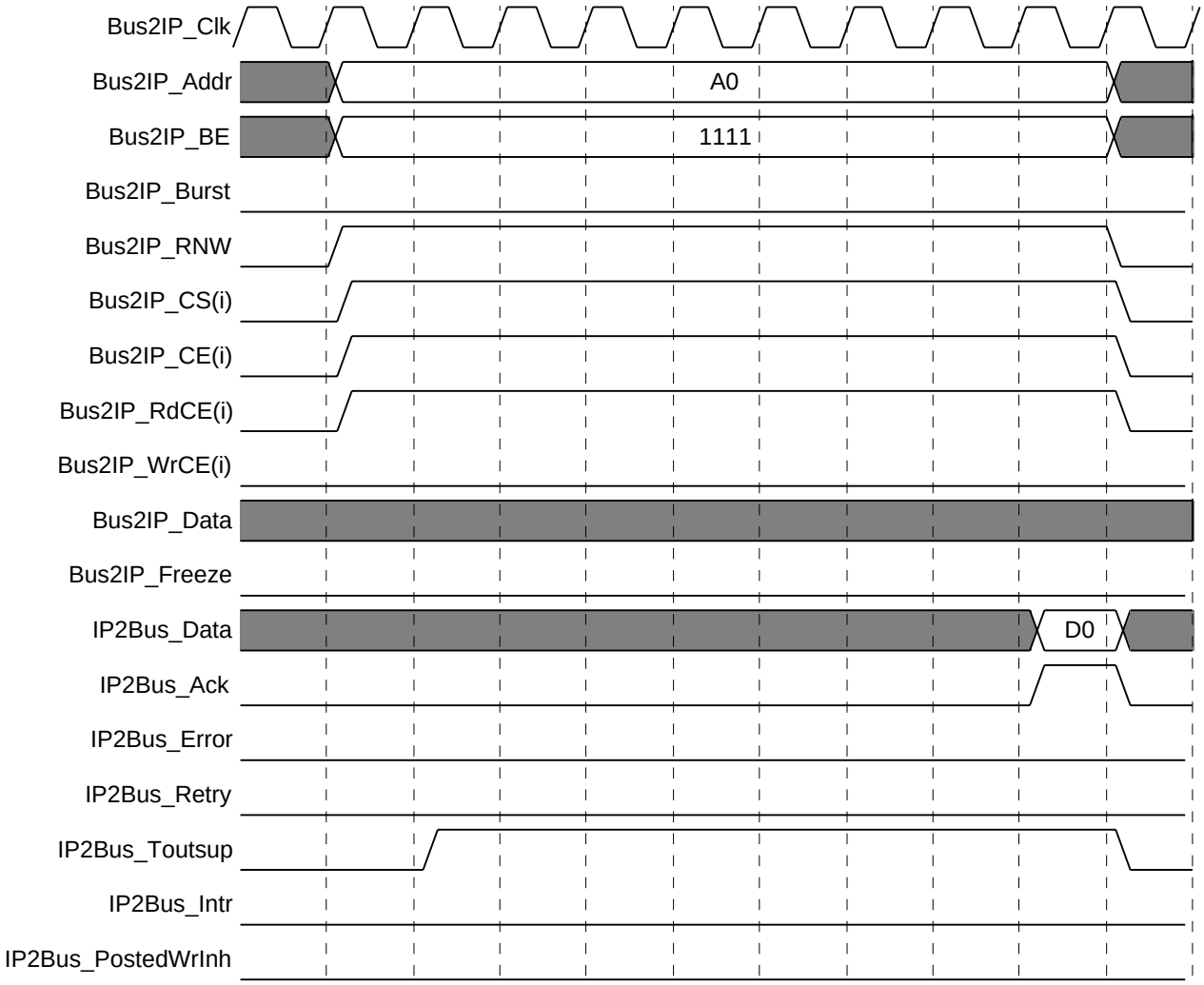

Figure 1-9: **Slave Read – Single Beat with Timeout Suppress**

Slave Write – Single Beat with Timeout Suppress

IP2Bus_Toutsup must be asserted if the assertion of IP2Bus_Ack will be delayed by more than 8 clocks from Bus2IP_CS. If more than 8 clocks elapse from assertion of Bus2IP_CS without assertion of either IP2Bus_Ack or IP2Bus_Toutsup, a timeout error may occur on the host bus.

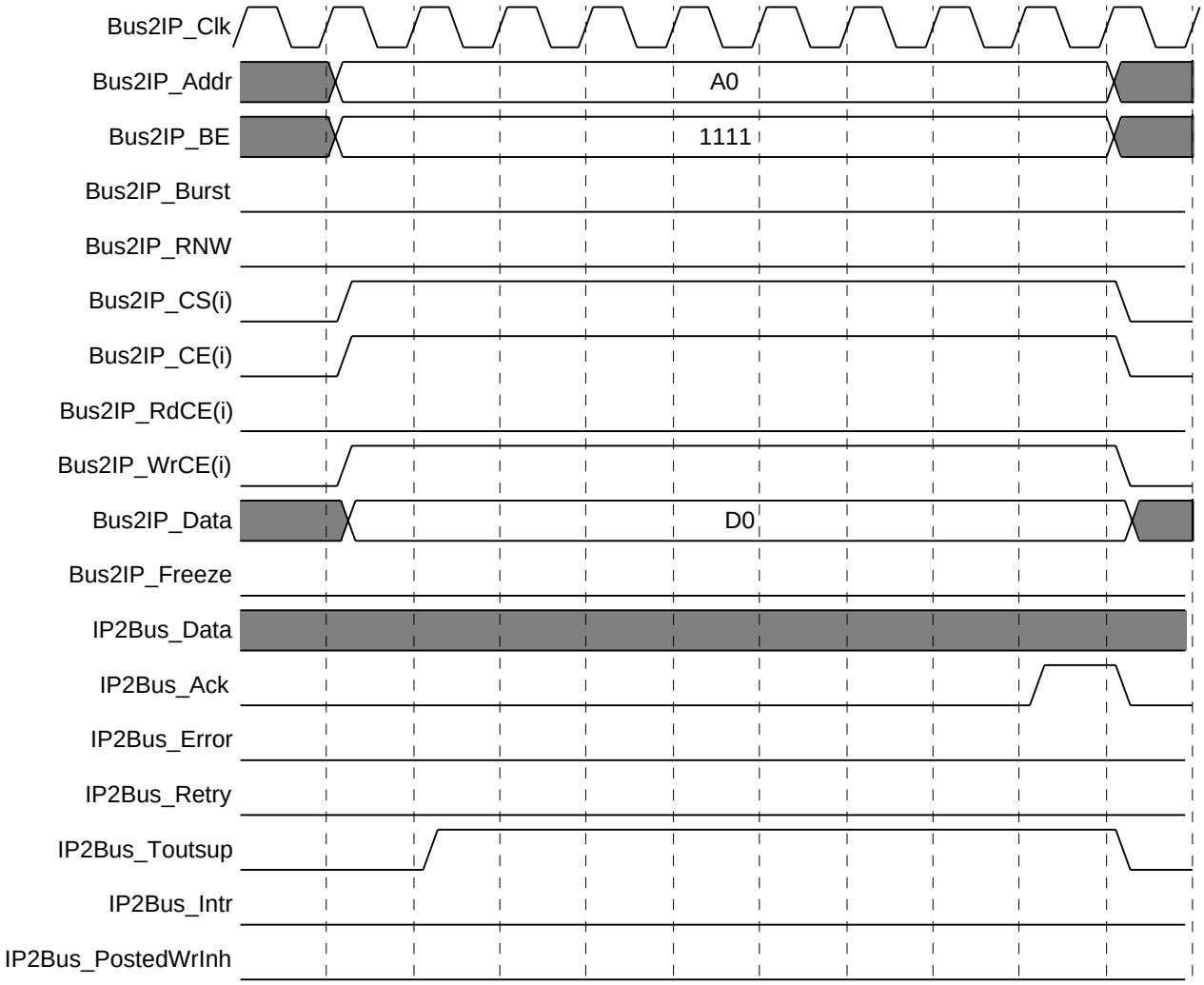

Figure 1-10: **Slave Write – Single Beat with Timeout Suppress**

Slave Read – Single Beat with Error

The IP2Bus_Error signal is a qualifier for IP2Bus_Ack (not an alternate completion) and indicates that an error occurred during the transaction.

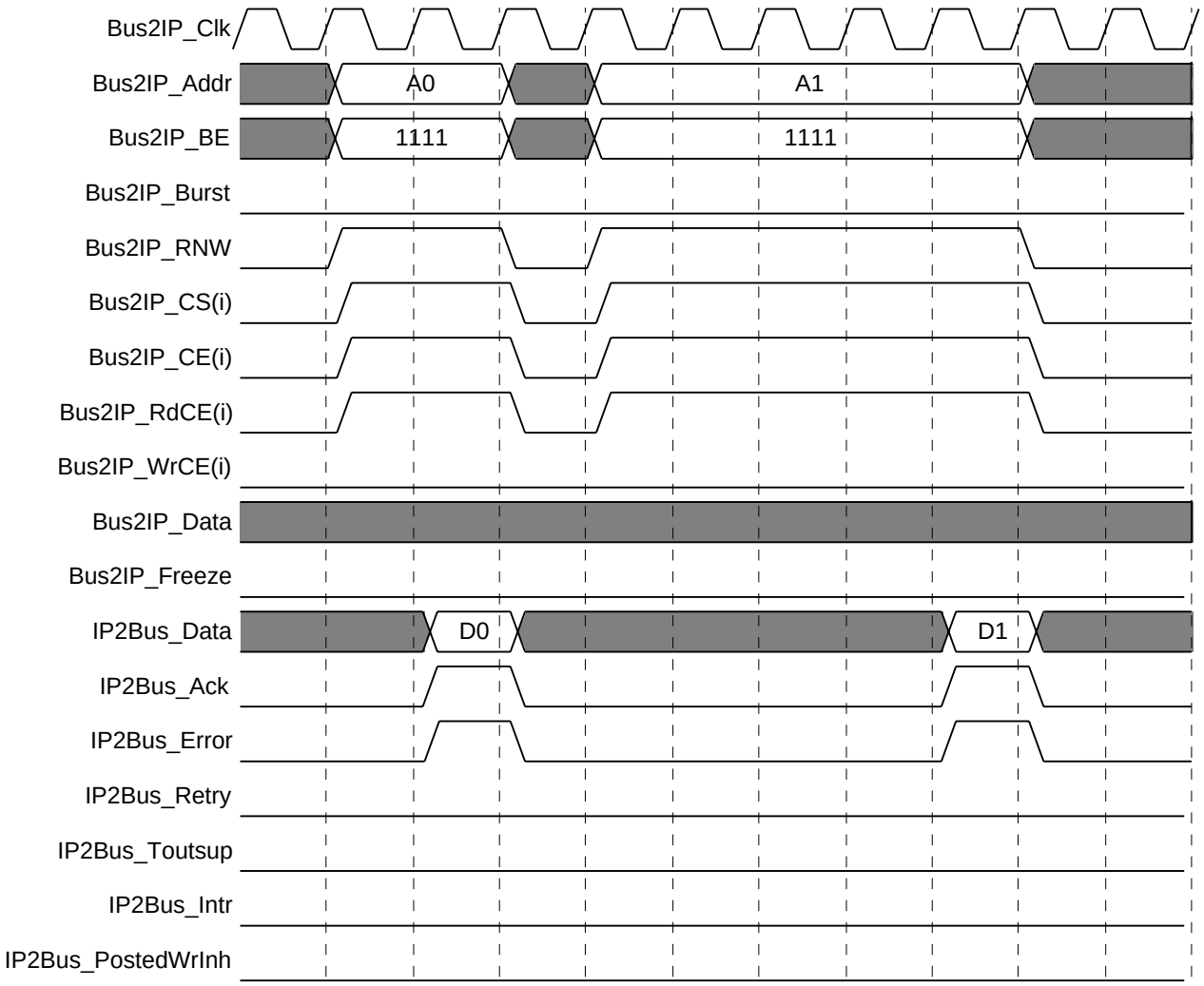

Figure 1-11: **Slave Read – Single Beat with Error**

Slave Write – Single Beat with Error

The IP2Bus_Error signal is a qualifier for IP2Bus_Ack (not an alternate completion) and indicates that an error occurred during the transaction.

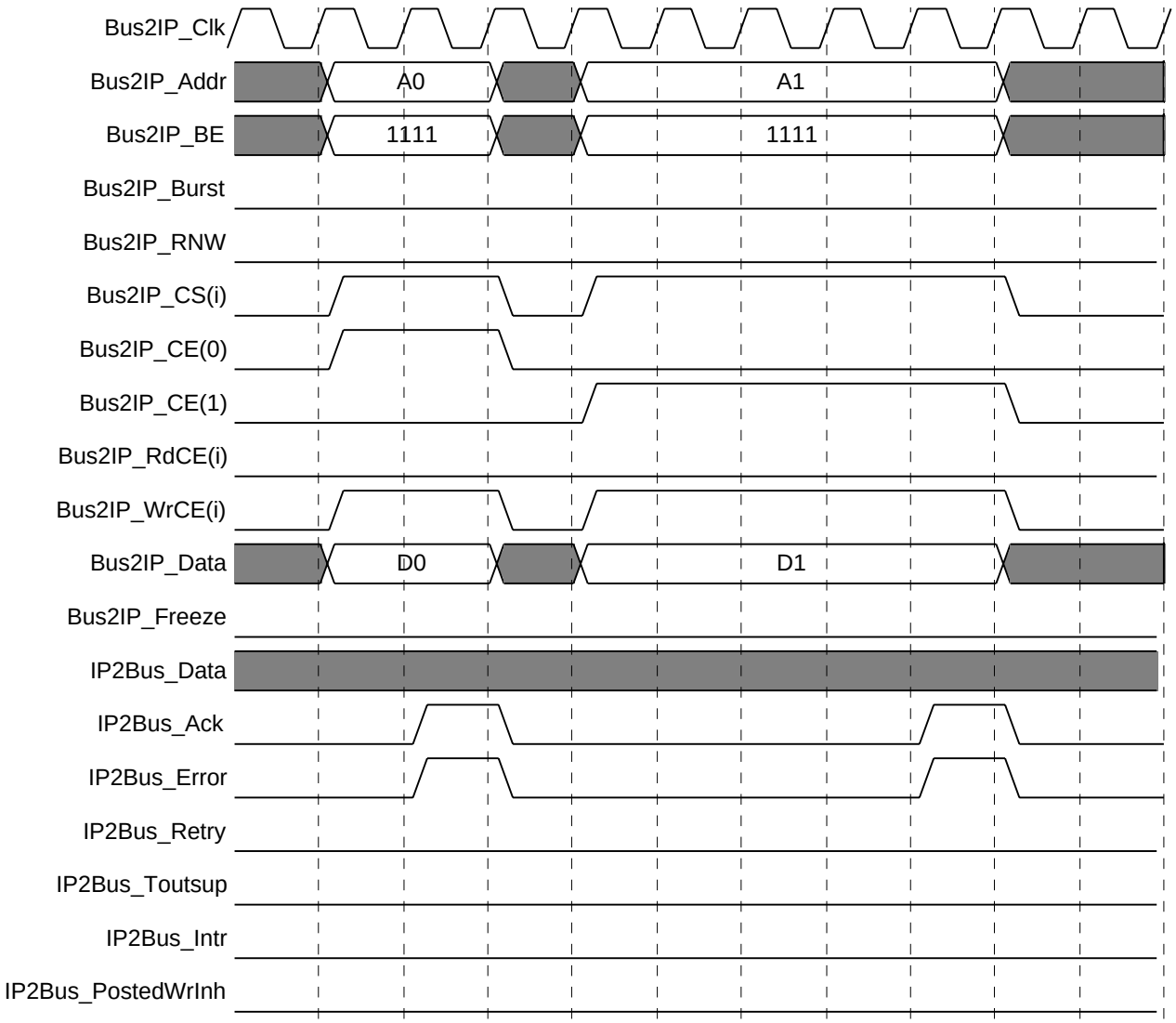

Figure 1-12: **Slave Write – Single Beat with Error**

Slave Read – Burst Operation

In burst operation, assertion of Bus2IP_Burst indicates that a burst is in progress and the addresses follow sequential order. Bus2IP_BE must be consistent throughout the burst and indicate sequential accesses. For example, a burst of words must have a constant Bus2IP_BE of 1111, while a burst of bytes must sequence as 1000, 0100, 0010, 0001, 1000, etc. Slaves may throttle the burst by negating IP2Bus_Ack during the burst, but the host bus master is not allowed to throttle the burst. A burst may be any length and is terminated by deassertion of the Bus2IP_Burst signal.

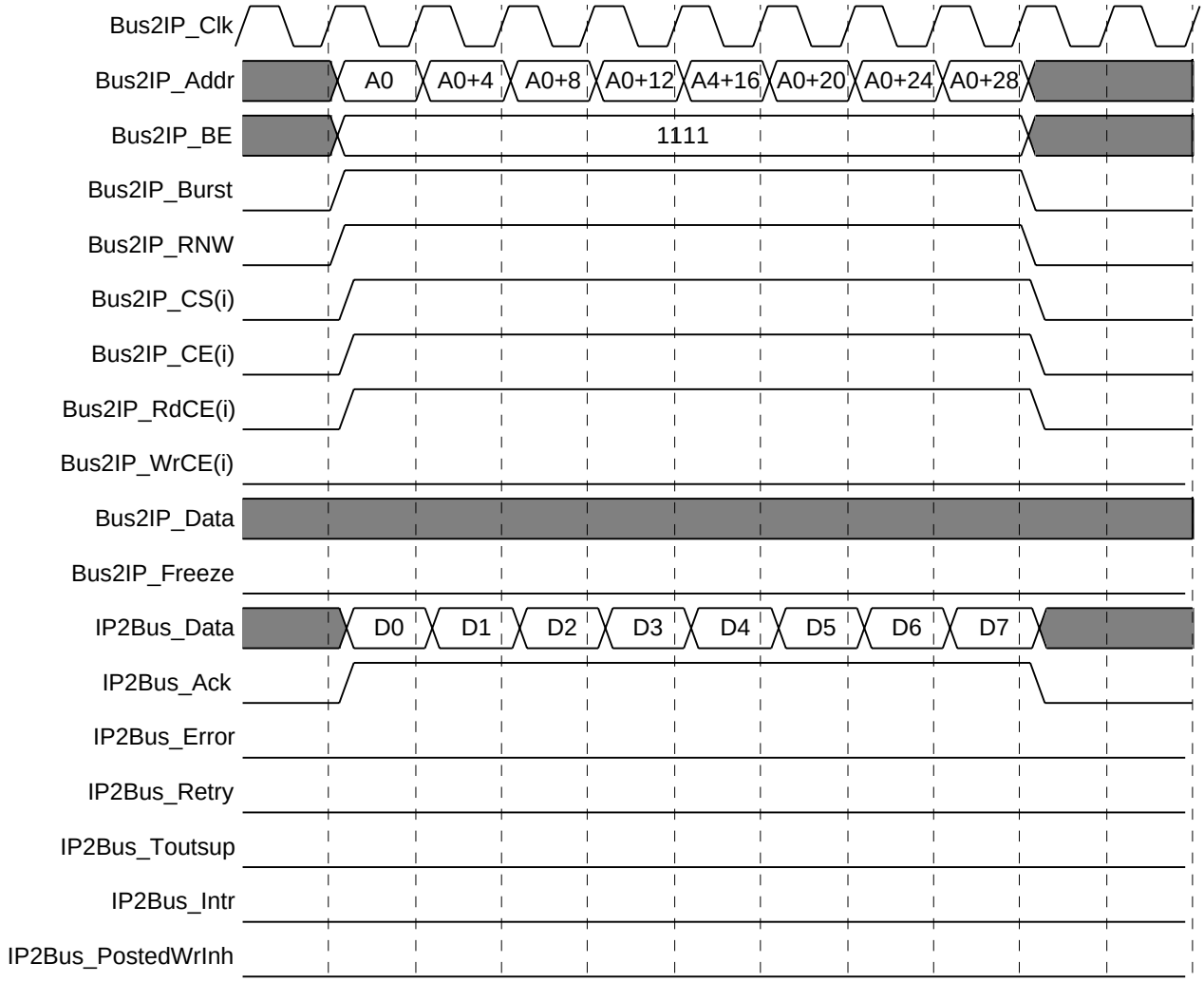

Figure 1-13: **Slave Read – Burst Operation**

Slave Write – Burst Operation

In burst operation, assertion of Bus2IP_Burst indicates that a burst is in progress and the addresses follow sequential order. Bus2IP_BE must be consistent throughout the burst and indicate sequential accesses. For example, a burst of words must have a constant Bus2IP_BE of 1111, while a burst of bytes must sequence as 1000, 0100, 0010, 0001, 1000, etc. Slaves may throttle the burst by negating IP2Bus_Ack during the burst, but the host bus master is not allowed to throttle the burst. A burst may be any length and is terminated by deassertion of the Bus2IP_Burst signal.

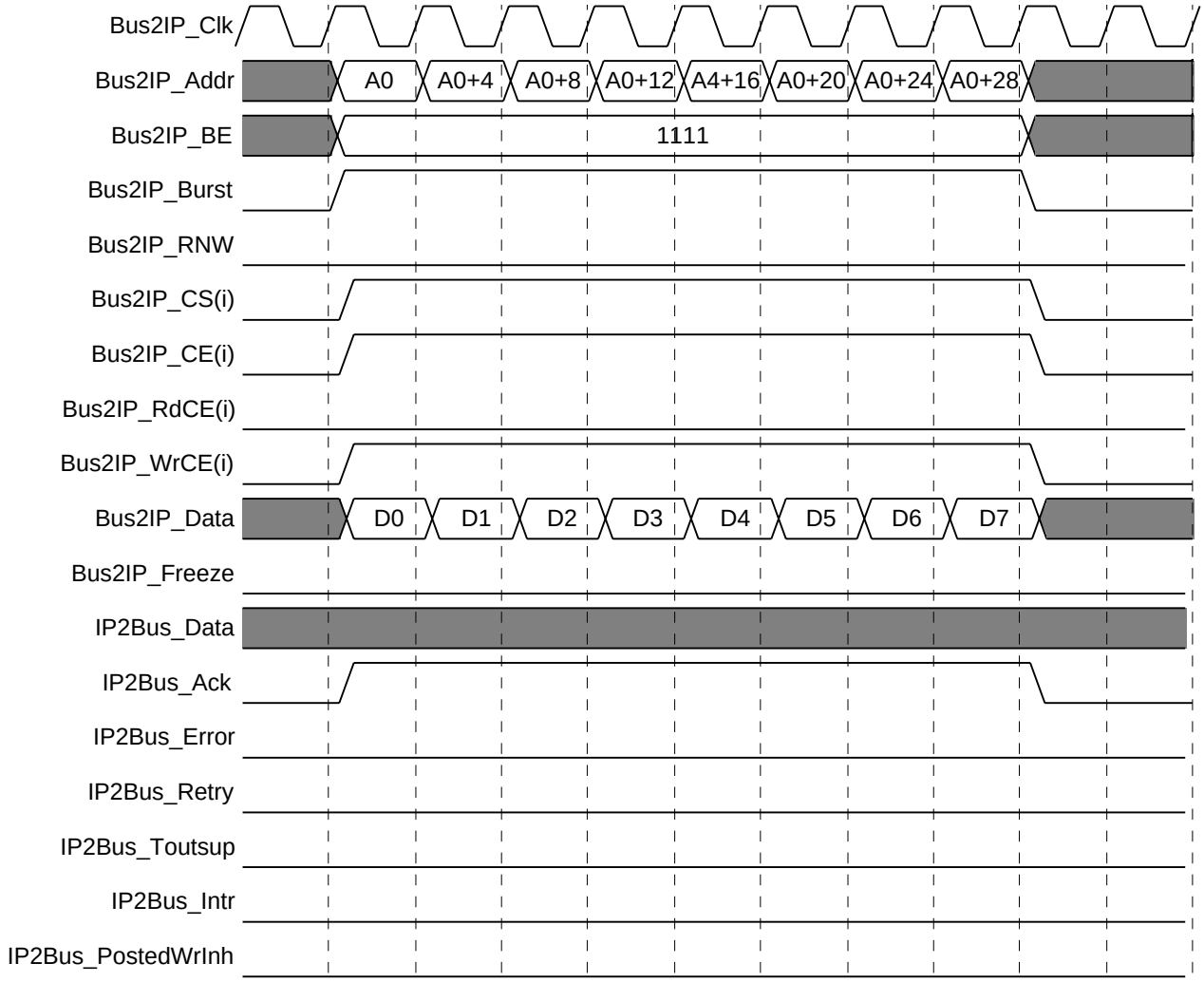

Figure 1-14: **Slave Write – Burst Operations**

Master Read – Single Beat

A single-beat master read is initiated with assertion of IP2Bus_MstReq and terminated with assertion of Bus2IP_MstAck and Bus2IP_MstLastAck. Bus2IP_MstLastAck is used to indicate the last acknowledge of a transfer and hence must be asserted concurrently with Bus2IP_MstAck for single-beat transfers; it is asserted only on the last data transfer of a burst transfer.

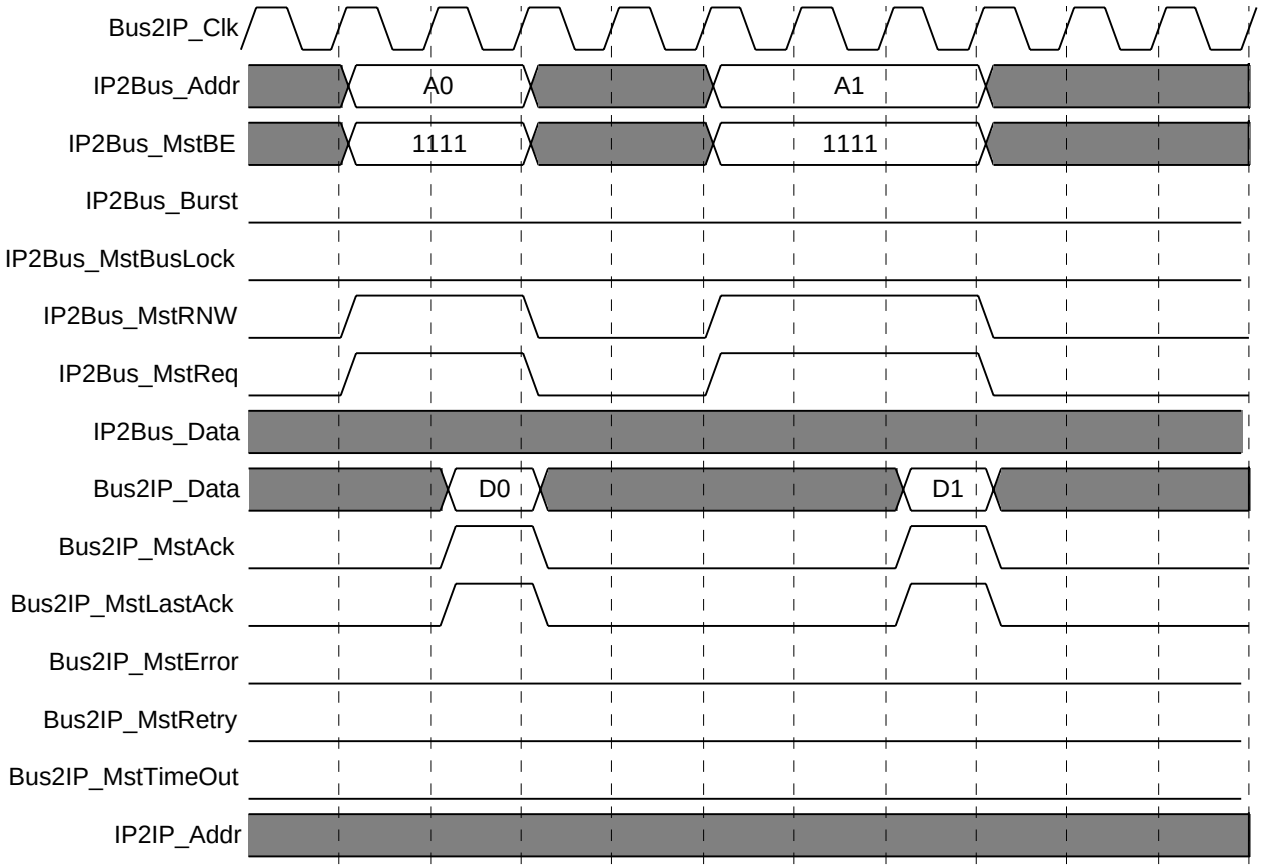

Figure 1-15: **Master Read – Single Beat**

Master Write – Single Beat

A single-beat master write is initiated with assertion of IP2Bus_MstReq and terminated with assertion of Bus2IP_MstAck and Bus2IP_MstLastAck. Bus2IP_MstLastAck is used to indicate the last acknowledge of a transfer and hence must be asserted concurrently with Bus2IP_MstAck for single-beat transfers; it is asserted only on the last data transfer of a burst transfer.

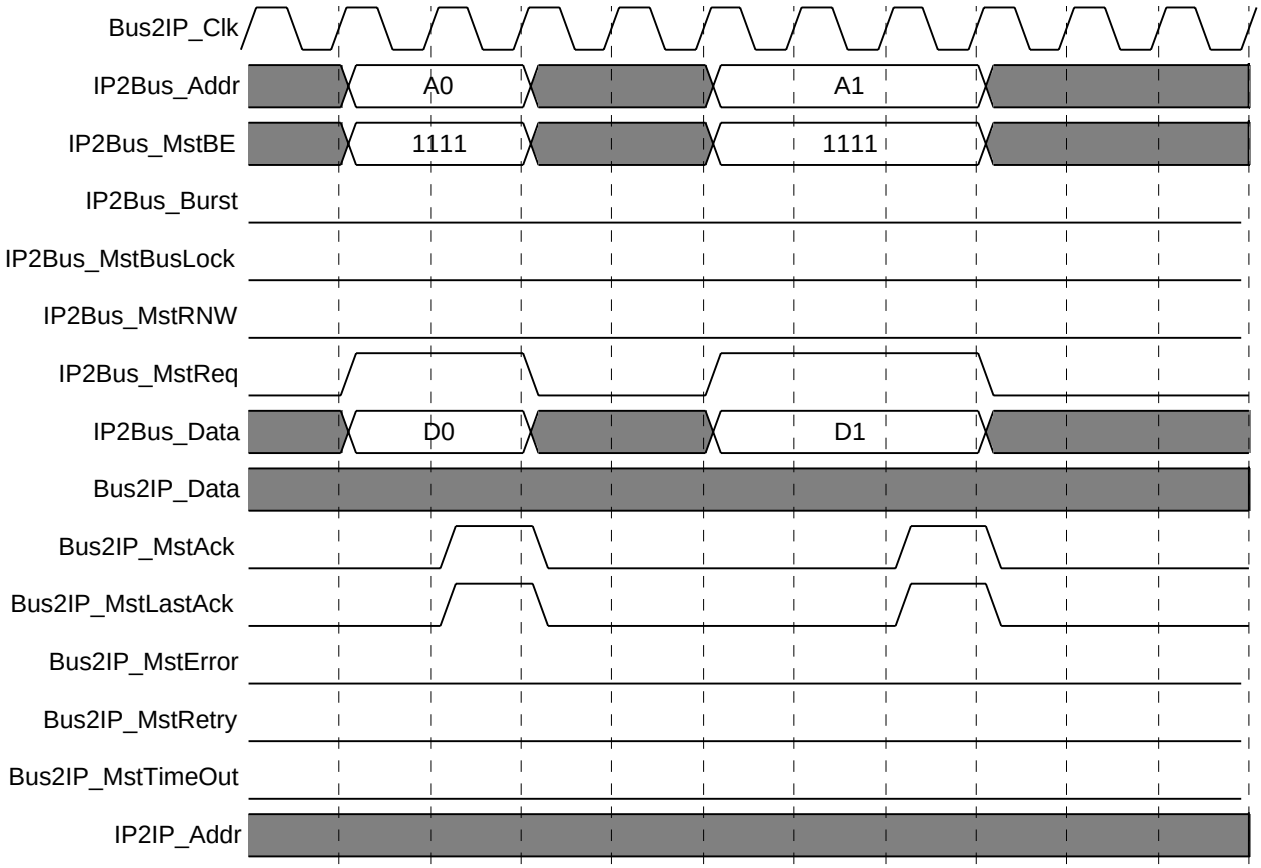

Figure 1-16: **Master Write – Single Beat**

Master Read – Single Beat Back to Back

This example illustrates single-cycle completion of single-beat master read transactions. The unused cycle between transfers is not required but is typical due to pipeline delays in the master logic.

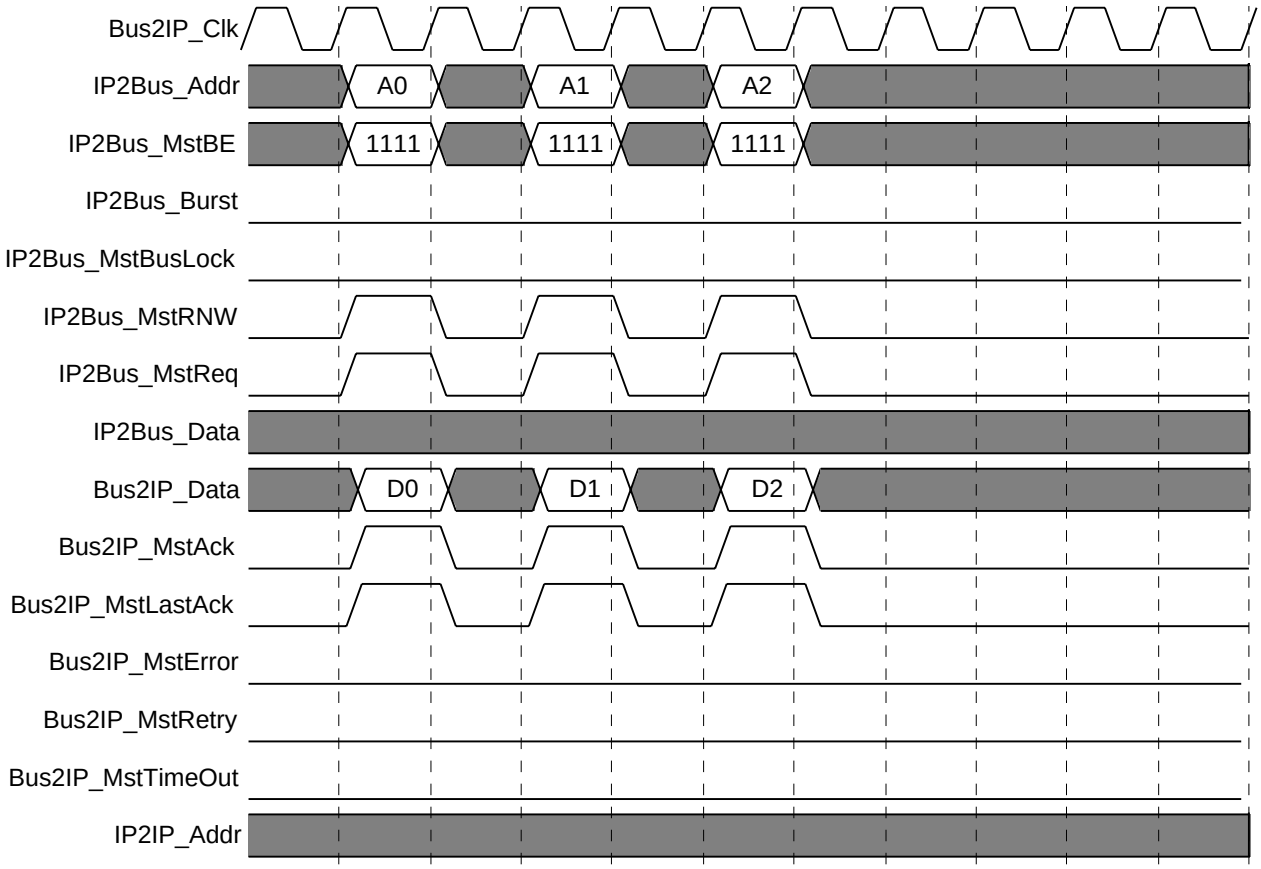

Figure 1-17: **Master Read – Single Beat Back to Back**

Master Write – Single Beat Back to Back

This example illustrates single-cycle completion of single-beat master write transactions. The unused cycle between transfers is not required but is typical due to pipeline delays in the master logic.

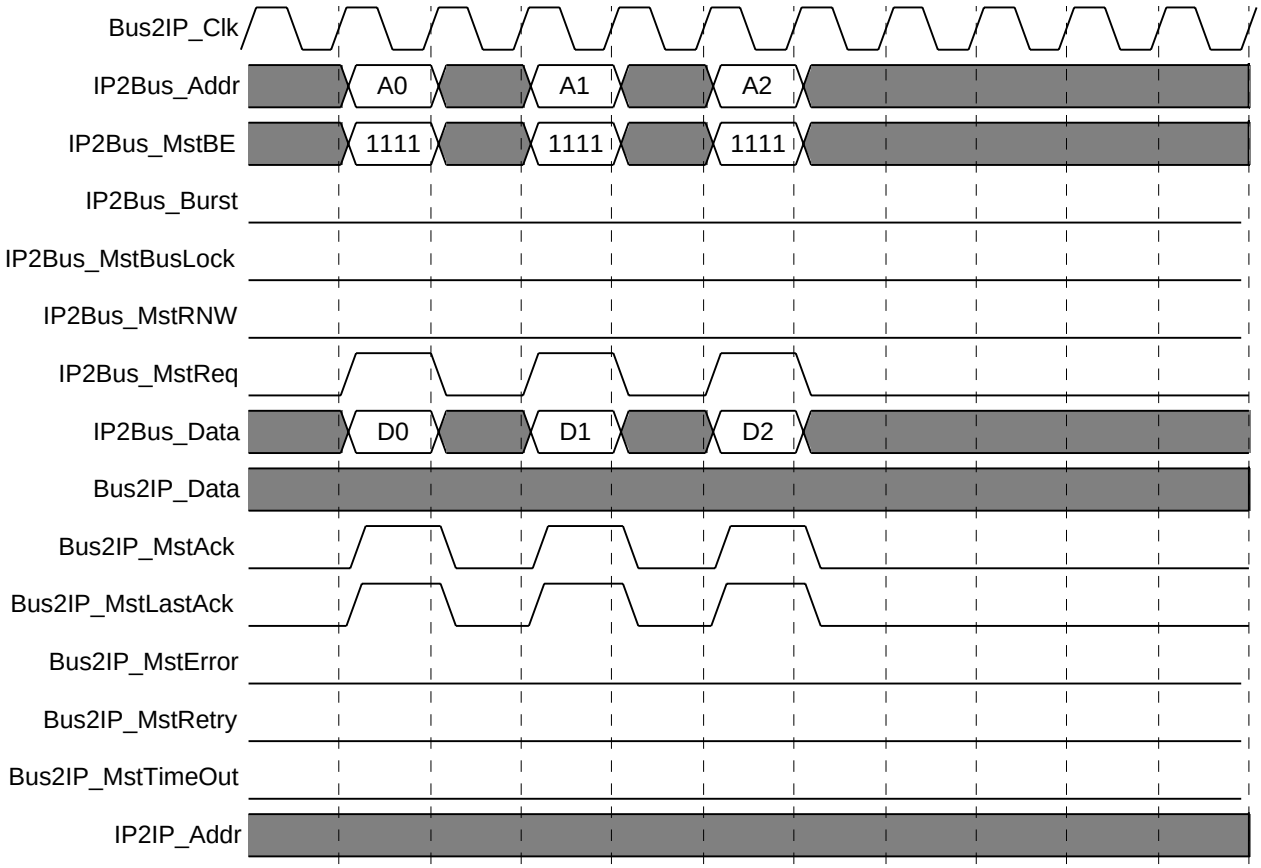

Figure 1-18: **Master Write – Single Beat Back to Back**

Master Read – Burst Operation

During master burst operation, the master must provide sequential addresses on IP2Bus_Addr, with the address increment determined by the transaction size. For example, the address must increment by 4 for fullword bursts, 2 for halfword bursts and 1 for byte bursts. The IP2Bus_MstBE must be consistent with the address presented on IP2Bus_Addr. A burst is indicated by the assertion of IP2Bus_Burst, and the length of the burst is defined by the IP2Bus_MstNum bus. The burst length is IP2Bus_MstNum+1. Each transfer is terminated with assertion of Bus2IP_MstAck, and the last transfer of the burst must be terminated by Bus2IP_MstLstAck.

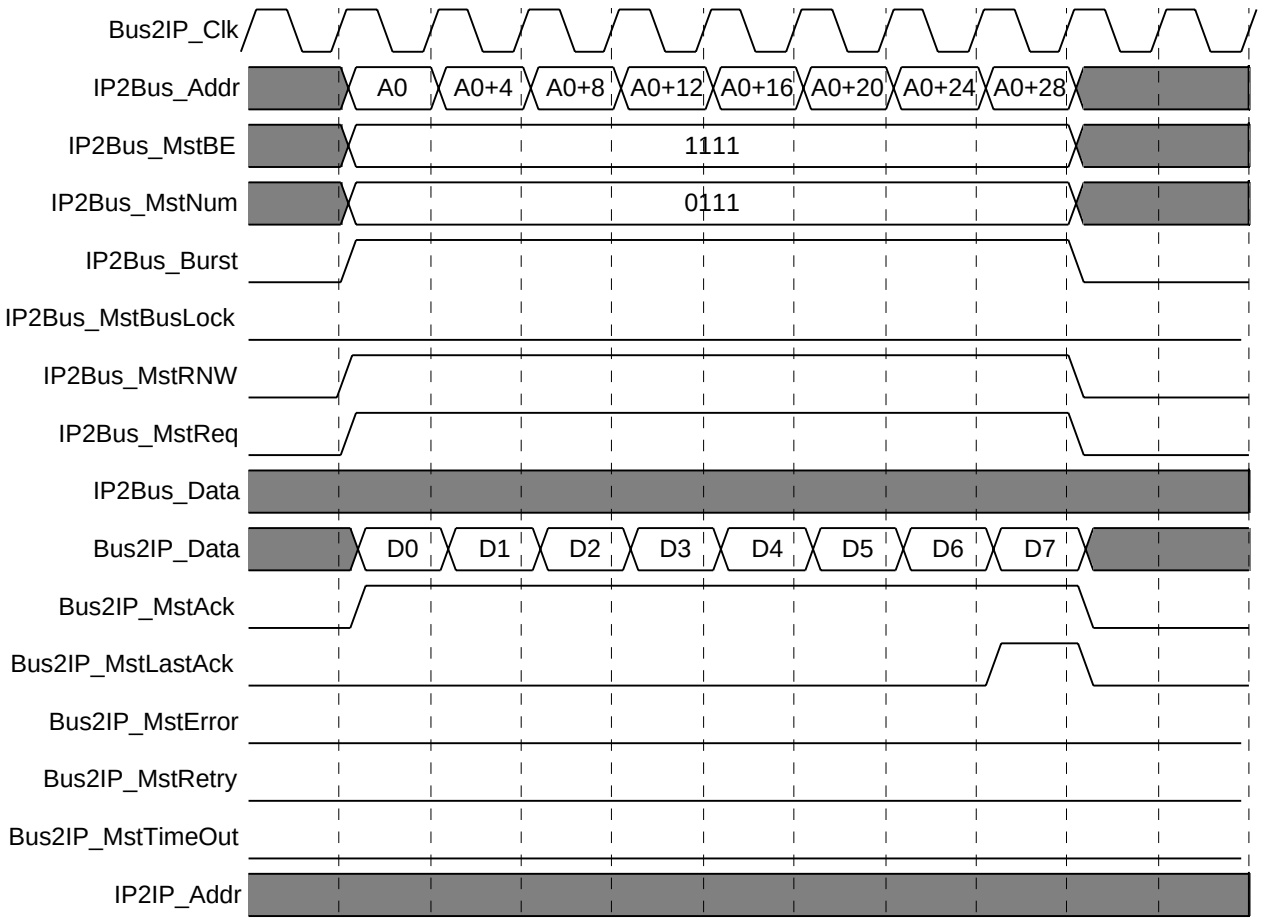

Figure 1-19: **Master Read – Burst Operation**

Master Write – Burst Operation

During master burst operation, the master must provide sequential addresses on IP2Bus_Addr, with the address increment determined by the transaction size. For example, the address must increment by 4 for fullword bursts, 2 for halfword bursts and 1 for byte bursts. The IP2Bus_MstBE must be consistent with the address presented on IP2Bus_Addr. A burst is indicated by the assertion of IP2Bus_Burst, and the length of the burst is defined by the IP2Bus_MstNum bus. The burst length is IP2Bus_MstNum+1. Each transfer is terminated with assertion of Bus2IP_MstAck, and the last transfer of the burst must be terminated by Bus2IP_MstLstAck.

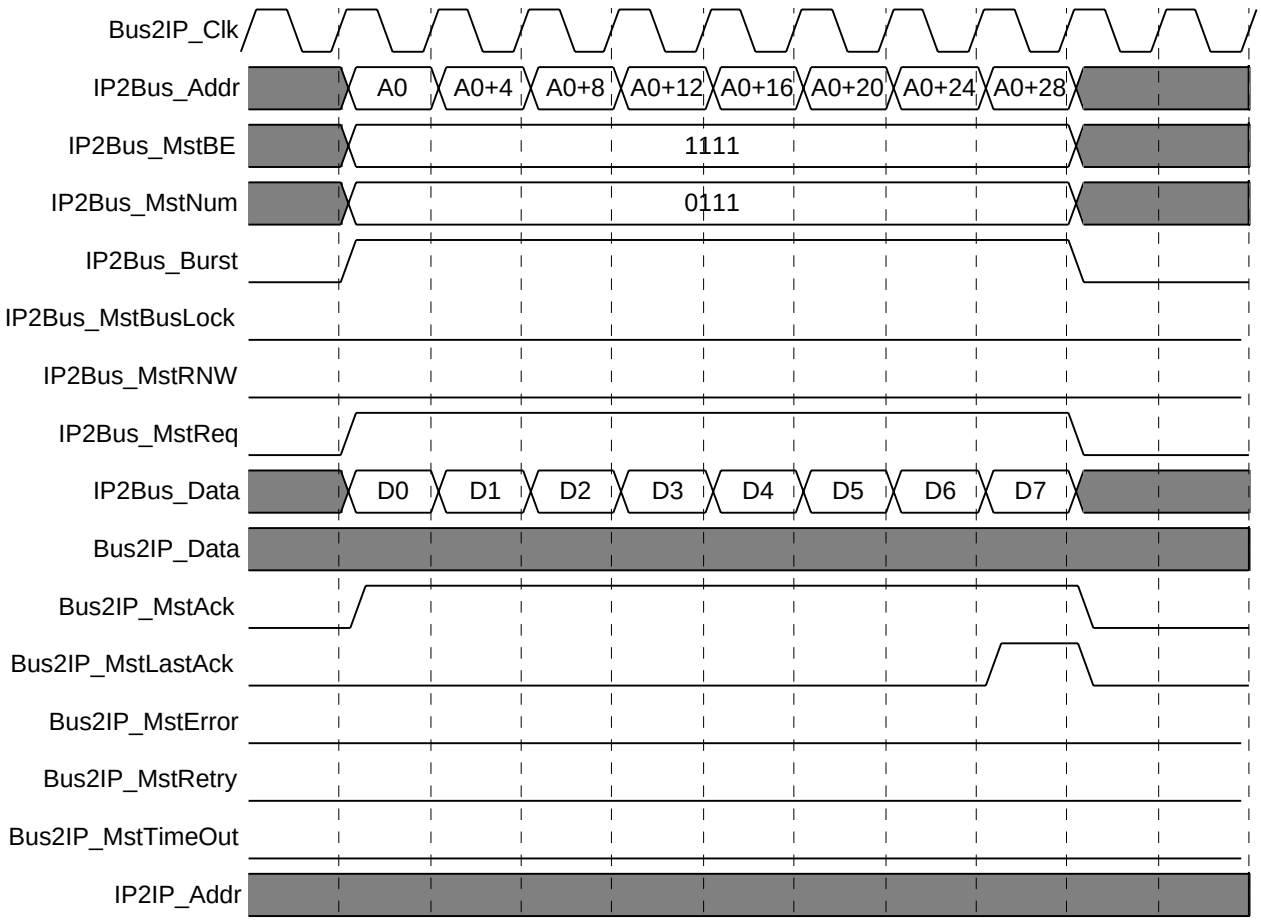

Figure 1-20: **Master Write – Burst Operation**

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
11/15/02	1.0	Initial Xilinx version for EDK3.1 SP2
4/1/03	1.1	Added IPIC timing diagrams

User Core Template for OPB (Slave Services Package 0)

DS458 (v1.1) Jan. 17, 2003 **Product Specification**

Summary

This document provides the specification for a VHDL template that can be used to create a simple OPB slave. The design template also includes an MPD file template and a PAO file template that can be modified as needed for the user core. The set of features and services provided in this template is called the Slave Services Package 0 (SSP0). It can be used to simplify creation of user cores that require only an OPB slave attachment.

opb_core_ssp0	v1.00b

Introduction

The SSP0 OPB user core template consists of the following files:

- A VHDL file, opb_core_ssp0.vhd, that provides the core's entity and architectural sections, instantiates an IP Interface (IPIF) for connection to the host bus, and provides an example instantiation of user logic.
- An MPD (Microprocessor Peripheral Definition) file that provides all of the required MPD contents except for the additional ports required by the user logic.
- A PAO (Peripheral Analyze Order) file that provides all of the required PAO entries except for the files required by the user logic.
- An example user logic design which consists of a quad LED brightness control based on four OPB-controlled Pulse Width Modulation (PWM) blocks.

The services provided by the SSP0 OPB user core template are:

- Slave-only attachment to the OPB bus.
- Support for burst transfers.
- Support for delayed bus acknowledge controlled by user logic.
- Module Identification Register (MIR) for software identification of the user core.
- Reset Register for generating a core reset via software.
- Address decode for one address range (any size) and one MIR/Reset register.
- Connection to user logic with a simplified set of IPIC (IP Interconnect) signals.

Block Diagram

The block diagram of the SSP0 user core template is shown below in Figure 1. The VHDL template supplies two components: a component for the user logic and a component for an IPIF (IP Interface). The IPIF used in the template is a simplified IPIF that uses only the services and ports required by the capabilities provided with the template. The opb_core_ssp0.vhd file uses a similarly named IPIF called opb_ipif_ssp0.vhd. It is important to understand that the IPIF used in the SSP0 user core template is not a full IPIF but only provides the parts of the full IPIF

required for a simple slave. This simplifies the use of the IPIF, reduces the complexity of interfacing to it, and insures that FPGA resources are used efficiently.

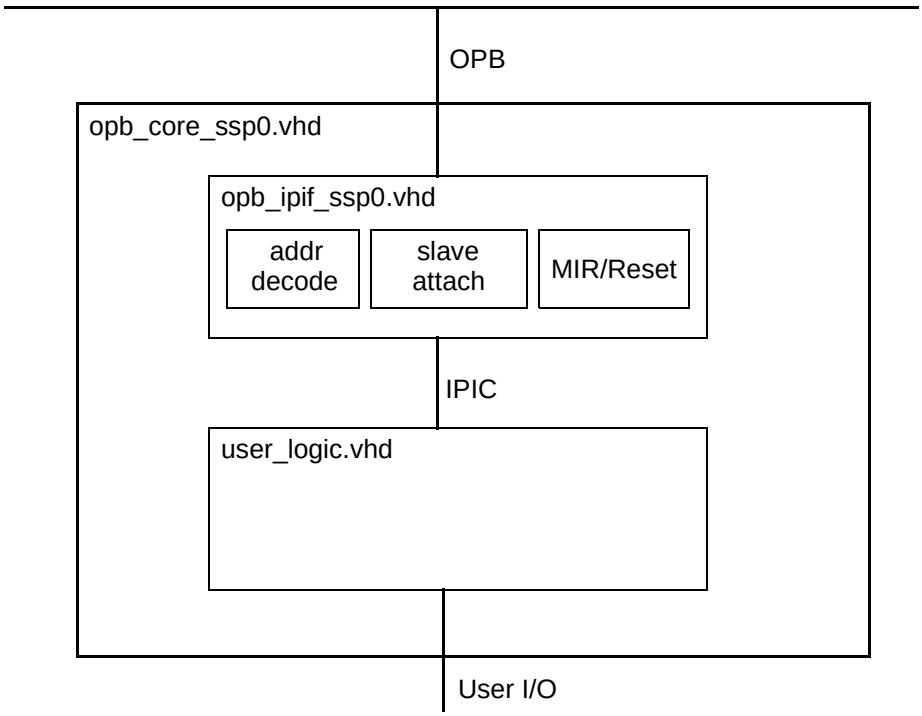

Figure 1: **SSP0 User Core Template Block Diagram**

Base Address Specification

The SSP0 user core template provides two regions of memory that must be specified. The C_BASEADDR/C_HIGHADDR pair specifies the address range for the user logic portion of the design. The other address range that appears in the template, C_MIR_BASEADDR/ C_MIR_HIGHADDR, specifies the address range for the Software Reset Register and Module Identification Register (MIR) within the IPIF. The C_MIR_BASEADDR and C_MIR_HIGHADDR must be set to a valid address range because the Software Reset Register is always present in the IPIF. The MIR may or may not be included, depending on the value of the C_MIR_INCLUDE generic. Even if a MIR is not included, the Software Reset Register uses the C_MIR_BASEADDR and C_MIR_HIGHADDR generics. These generics are typically set to define a small address range, such as 256 bytes. The address range size must be a power of 2.

Using the Template

The basic steps for using the OPB core template are as follows. A detailed example follows this listing.

- Create a directory under the "myip" directory in your project directory with the name and version of the user core (example: opb_mycore_v1_00_a).
- Create a user logic design that uses the IPIC signal set described below.
- Rename the opb_core_ssp0.vhd template file to the same name as the user core (example: rename opb_core_ssp0.vhd to opb_mycore.vhd)
- In the renamed template file, edit the entity name and library statements as required for the user core. In the template file, lines that may require editing are marked with a --USER-- comment.
- Remove the example user logic component declaration and instantiation and replace them with the user logic component declaration and instantiation.

- Add the ports required by the user logic to the entity declaration.
- In the "data" directory, change the name of the .mpd and .pao files to be the same as the user core name (example: rename opb_core_ssp0_v2_0_0.mpd to opb_mycore_v2_0_0.mpd and rename opb_core_ssp0_v2_0_0.pao to opb_mycore_v2_0_0.pao). Note that the _v2_0_0 suffix is required to identify the PSF format version. Use the same version suffix as the supplied template.
- In the renamed MPD file, edit the core name and add any user ports. Lines that may require editing are identified by the # --USER-- comment.
- In the renamed PAO file, add the source files required by the user core. The library name for the user core is the user core name plus the version suffix. Lines that may require editing are identified by the # --USER-- comment.

IPIC Signal Set

The following IPIC signals are used in the SSP0 OPB user core template. These signals are required to be present as ports on the user logic or they must be handled appropriately in the opb_core_ssp0 template. All signals are active high. See the chapter "Adding User Cores to Your Embedded System" for more details on the IPIC signals.

```
signal Bus2IP_Addr       : std_logic_vector(0 to C_OPB_AWIDTH-1);
signal Bus2IP_BE         : std_logic_vector(0 to C_OPB_DWIDTH/8-1);
signal Bus2IP_Burst      : std_logic;
signal Bus2IP_Clk        : std_logic;
signal Bus2IP_CS         : std_logic;
signal Bus2IP_Data       : std_logic_vector(0 to C_OPB_DWIDTH-1);
signal Bus2IP_RdCE       : std_logic;
signal Bus2IP_Reset      : std_logic;
signal Bus2IP_RNW        : std_logic;
signal Bus2IP_WrCE       : std_logic;
signal IP2Bus_Ack        : std_logic;
signal IP2Bus_Clk        : std_logic;
signal IP2Bus_Data       : std_logic_vector(0 to C_OPB_DWIDTH-1);
signal IP2Bus_Error      : std_logic;
signal IP2Bus_PostedWrInh : std_logic;
signal IP2Bus_Retry       : std_logic;
signal IP2Bus_ToutSup     : std_logic;
```

Example: Creating an OPB User Core

This example goes through the entire process of creating an OPB user core from the template. The example creates a core called "opb_mycore" and uses it in an MHS file. This example assumes you have a project directory called "system_example" in which your MHS, MSS, and other system files are placed, and that your EDK install directory is `C:/EDK`. This example also assumes that you have your user logic in a directory called `C:/user_logic` and that the user logic consists of two files, `mycore_user_logic.vhd` and `my_subblock.vhd`.

Step 1. In the `system_example` directory, create a directory called `myip` if it doesn't already exist:

```
$ mkdir myip
```

Step 2. Copy the OPB user core template from the install area to the myip directory that you just created (this assumes you are creating version v1.00a of your user core; change the destination name as required for a different user core version):

```
$ cp -r c:/EDK/hw/user_core_templates/opb_core_ssp0_v1_00_b
myip/opb_mycore_v1_00_a
```

Step 3. Go into the myip/opb_mycore_v1_00_a/data directory and rename the .mpd and .pao files to opb_mycore:

```
$ cd myip/opb_mycore_v1_00_a/data
$ mv opb_core_ssp0_v2_0_0.mpd opb_mycore_v2_0_0.mpd
$ mv opb_core_ssp0_v2_0_0.pao opb_mycore_v2_0_0.pao
```

Step 4. Go into the VHDL source directory and rename the opb_core_ssp0.vhd file to opb_mycore.vhd:

```
$ cd ../hdl/vhdl
$ mv opb_core_ssp0.vhd opb_mycore.vhd
```

Step 5. Copy your user logic files into the VHDL directory where opb_mycore.vhd resides.

```
$ cp c:/user_logic/*.vhd .
```

Step 6. Edit the `opb_mycore.vhd` file, modifying the comment line at the top, library name if needed, entity declaration, and architecture statement. These lines are all marked by the --USER-- comment. What you are accomplishing in this step is to replace all occurrences of opb_core_ssp0 with opb_mycore. When you are finished, the beginning of the file should look like this:

```
-- $Id: opb_mycore.vhd,v 1.1 2002/11/08 22:56:56 tise Exp $
--USER-- Add comment with name of this source file
-- opb_mycore.vhd

library ieee;
use ieee.std_logic_1164.all;

library opb_ipif_ssp0_v1_00_a;
use opb_ipif_ssp0_v1_00_a.all;

--  Add user library if this source file uses
--  entities from the user library
library opb_mycore_v1_00_a; --USER-- library name
use opb_mycore_v1_00_a.all; --USER-- use statement

------------------------------------------------------------------------------
-- entity
------------------------------------------------------------------------------

entity OPB_Mycore is  --USER-- change entity name
  generic
  (
    C_BASEADDR         : std_logic_vector(0 to 31) := X"FFFFFFFF";
    C_HIGHADDR         : std_logic_vector(0 to 31) := X"00000000";
    C_MIR_BASEADDR     : std_logic_vector(0 to 31) := X"FFFFFFFF";
    C_MIR_HIGHADDR     : std_logic_vector(0 to 31) := X"00000000";
    C_USER_ID_CODE     : integer                   := 3;
    C_INCLUDE_MIR      : integer                   := 0;
    C_OPB_AWIDTH       : integer                   := 32;
    C_OPB_DWIDTH       : integer                   := 32;
    C_FAMILY           : string   := "virtex2" -- not used
  );
  port
  (
    --Required OPB bus ports, do not add to or delete
    OPB_ABus     : in  std_logic_vector(0 to C_OPB_AWIDTH-1);
    OPB_BE       : in  std_logic_vector(0 to C_OPB_DWIDTH/8-1);
    OPB_Clk      : in  std_logic;
    OPB_DBus     : in  std_logic_vector(0 to C_OPB_DWIDTH-1);
    OPB_RNW      : in  std_logic;
    OPB_Rst      : in  std_logic;
    OPB_select   : in  std_logic;
    OPB_seqAddr  : in  std_logic;
    Sln_DBus     : out std_logic_vector(0 to C_OPB_DWIDTH-1);
    Sln_errAck   : out std_logic;
```

```
    Sln_retry    : out std_logic;
    Sln_toutSup  : out std_logic;
    Sln_xferAck  : out std_logic;

    --USER-- add user I/Os to port list
    LED          : out std_logic_vector(0 to 3)

  );
end entity OPB_Mycore; --USER-- change entity name

------------------------------------------------------------------------------
-- architecture
------------------------------------------------------------------------------
architecture imp of OPB_Mycore is --USER-- change entity name
```

Step 7. Edit the entity ports to include the user I/Os:

```
entity OPB_Mycore is  --USER-- change entity name
  generic
  (
    C_BASEADDR         : std_logic_vector(0 to 31) := X"FFFFFFFF";
    C_HIGHADDR         : std_logic_vector(0 to 31) := X"00000000";
    C_MIR_BASEADDR     : std_logic_vector(0 to 31) := X"FFFFFFFF";
    C_MIR_HIGHADDR     : std_logic_vector(0 to 31) := X"00000000";
    C_USER_ID_CODE     : integer                   := 3;
    C_INCLUDE_MIR      : integer                   := 0;
    C_OPB_AWIDTH       : integer                   := 32;
    C_OPB_DWIDTH       : integer                   := 32;
    C_FAMILY           : string    := "virtex2" -- not used
  );
  port
  (
    --Required OPB bus ports, do not add to or delete
    OPB_ABus     : in  std_logic_vector(0 to C_OPB_AWIDTH-1);
    OPB_BE       : in  std_logic_vector(0 to C_OPB_DWIDTH/8-1);
    OPB_Clk      : in  std_logic;
    OPB_DBus     : in  std_logic_vector(0 to C_OPB_DWIDTH-1);
    OPB_RNW      : in  std_logic;
    OPB_Rst      : in  std_logic;
    OPB_select   : in  std_logic;
    OPB_seqAddr  : in  std_logic;
    Sln_DBus     : out std_logic_vector(0 to C_OPB_DWIDTH-1);
    Sln_errAck   : out std_logic;
    Sln_retry    : out std_logic;
    Sln_toutSup  : out std_logic;
    Sln_xferAck  : out std_logic;

    --USER-- add user I/Os to port list
    -- LED          : out std_logic_vector(0 to 3);

    MyInPort        : in  std_logic_vector(0 to 3);
    MyOutPort       : out std_logic_vector(0 to 3)
  );
end entity OPB_Mycore; --USER-- change entity name
```

Step 8. Edit the user logic component declaration and instantiation:

```
------------------------------------------------------------------------------
-- architecture
------------------------------------------------------------------------------
```

```
architecture imp of OPB_Mycore is --USER-- change entity name

  --USER-- component declaration for user core logic. Change the
  --component declaration below to the declaration of the user's
  --logic.

  component mycore_user_logic is
    port
    (
      Bus2IP_Addr  : in  std_logic_vector(0 to 31);
      Bus2IP_Clk   : in  std_logic;
      Bus2IP_CS    : in  std_logic;
      Bus2IP_Data  : in  std_logic_vector(0 to 31);
      Bus2IP_RdCE  : in  std_logic;
      Bus2IP_Reset : in  std_logic;
      Bus2IP_WrCE  : in  std_logic;
      IP2Bus_Data  : out std_logic_vector(0 to 31);
      MyInPort     : in  std_logic_vector(0 to 3);
      MyOutPort    : out std_logic_vector(0 to 3)
    );
  end component mycore_user_logic;

  --OPB IPIF lite declaration -- do not change
  component opb_ipif_ssp0 is
    generic
    (
      C_BASEADDR     : std_logic_vector(0 to 31) := X"FFFFFFFF";
      C_HIGHADDR     : std_logic_vector(0 to 31) := X"00000000";
      C_MIR_BASEADDR : std_logic_vector(0 to 31) := X"FFFFFFFF";
      C_MIR_HIGHADDR : std_logic_vector(0 to 31) := X"00000000";
      C_USER_ID_CODE : INTEGER                   := 1;
      C_INCLUDE_MIR  : INTEGER                   := 0;
      C_OPB_AWIDTH   : INTEGER                   := 32;
      C_OPB_DWIDTH   : INTEGER                   := 32;
      C_FAMILY       : string                    := "virtex2"
    );
    port
    (
      OPB_ABus       : in  std_logic_vector(0 to C_OPB_AWIDTH-1);
      OPB_BE         : in  std_logic_vector(0 to C_OPB_DWIDTH/8-1);
      OPB_Clk        : in  std_logic;
      OPB_DBus       : in  std_logic_vector(0 to C_OPB_DWIDTH-1);
      OPB_RNW        : in  std_logic;
      OPB_Rst        : in  std_logic;
      OPB_select     : in  std_logic;
      OPB_seqAddr    : in  std_logic;
      Sln_DBus       : out std_logic_vector(0 to C_OPB_DWIDTH-1);
      Sln_errAck     : out std_logic;
      Sln_retry      : out std_logic;
      Sln_toutSup    : out std_logic;
      Sln_xferAck    : out std_logic;
      Bus2IP_Addr    : out std_logic_vector(0 to C_OPB_AWIDTH-1);
      Bus2IP_BE      : out std_logic_vector(0 to C_OPB_DWIDTH/8-1);
      Bus2IP_Burst   : out std_logic;
      Bus2IP_Clk     : out std_logic;
      Bus2IP_CS      : out std_logic;
      Bus2IP_Data    : out std_logic_vector(0 to C_OPB_DWIDTH-1);
      Bus2IP_RdCE    : out std_logic;
      Bus2IP_Reset   : out std_logic;
      Bus2IP_RNW     : out std_logic;
      Bus2IP_WrCE    : out std_logic;
```

```
      IP2Bus_Ack     : in  std_logic;
      IP2Bus_Clk     : in  std_logic;
      IP2Bus_Data    : in  std_logic_vector(0 to C_OPB_DWIDTH-1);
      IP2Bus_Error       : in  std_logic;
      IP2Bus_PostedWrInh : in  std_logic;
      IP2Bus_Retry       : in  std_logic;
      IP2Bus_ToutSup     : in  std_logic
    );
  end component opb_ipif_ssp0;

  --IP Interconnect (IPIC) signal list --do not delete
  signal Bus2IP_Addr   : std_logic_vector(0 to C_OPB_AWIDTH-1);
  signal Bus2IP_BE     : std_logic_vector(0 to C_OPB_DWIDTH/8-1);
  signal Bus2IP_Burst  : std_logic;
  signal Bus2IP_Clk    : std_logic;
  signal Bus2IP_CS     : std_logic;
  signal Bus2IP_Data   : std_logic_vector(0 to C_OPB_DWIDTH-1);
  signal Bus2IP_RdCE   : std_logic;
  signal Bus2IP_Reset  : std_logic;
  signal Bus2IP_RNW    : std_logic;
  signal Bus2IP_WrCE   : std_logic;
  signal IP2Bus_Ack    : std_logic;
  signal IP2Bus_Clk    : std_logic;
  signal IP2Bus_Data   : std_logic_vector(0 to C_OPB_DWIDTH-1);
  signal IP2Bus_Error       : std_logic;
  signal IP2Bus_PostedWrInh : std_logic;
  signal IP2Bus_Retry       : std_logic;
  signal IP2Bus_ToutSup     : std_logic;

---------------------------------------------------------------------------
begin
---------------------------------------------------------------------------

  --OPB IPIF lite instantiation --do not delete
  OPB_IPIF_SSP0_I : opb_ipif_ssp0
    generic map
    (
      C_BASEADDR         => C_BASEADDR,
      C_HIGHADDR         => C_HIGHADDR,
      C_MIR_BASEADDR     => C_MIR_BASEADDR,
      C_MIR_HIGHADDR     => C_MIR_HIGHADDR,
      C_USER_ID_CODE     => C_USER_ID_CODE,
      C_INCLUDE_MIR      => C_INCLUDE_MIR,
      C_OPB_AWIDTH       => C_OPB_AWIDTH,
      C_OPB_DWIDTH       => C_OPB_DWIDTH,
      C_FAMILY           => C_FAMILY
    )
    port map
    (
      OPB_ABus           => OPB_ABus,
      OPB_BE             => OPB_BE,
      OPB_Clk            => OPB_Clk,
      OPB_DBus           => OPB_DBus,
      OPB_RNW            => OPB_RNW,
      OPB_Rst            => OPB_Rst,
      OPB_select         => OPB_select,
      OPB_seqAddr        => OPB_seqAddr,
      Sln_DBus           => Sln_DBus,
      Sln_errAck         => Sln_errAck,
      Sln_retry          => Sln_retry,
      Sln_toutSup        => Sln_toutSup,
      Sln_xferAck        => Sln_xferAck,
```

```
      Bus2IP_Addr        => Bus2IP_Addr,
      Bus2IP_BE          => Bus2IP_BE,
      Bus2IP_Burst       => Bus2IP_Burst,
      Bus2IP_Clk         => Bus2IP_Clk,
      Bus2IP_CS          => Bus2IP_CS,
      Bus2IP_Data        => Bus2IP_Data,
      Bus2IP_RdCE        => Bus2IP_RdCE,
      Bus2IP_Reset       => Bus2IP_Reset,
      Bus2IP_RNW         => Bus2IP_RNW,
      Bus2IP_WrCE        => Bus2IP_WrCE,
      IP2Bus_Ack         => IP2Bus_Ack,
      IP2Bus_Clk         => IP2Bus_Clk,
      IP2Bus_Data        => IP2Bus_Data,
      IP2Bus_Error       => IP2Bus_Error,
      IP2Bus_PostedWrInh => IP2Bus_PostedWrInh,
      IP2Bus_Retry       => IP2Bus_Retry,
      IP2Bus_ToutSup     => IP2Bus_ToutSup
    );

  --USER-- change the USER_LOGIC_I instantiation below to the
  --instantiation of the user logic.

  MYUSER_LOGIC_I : mycore_user_logic
    port map
    (
      Bus2IP_Addr  => Bus2IP_Addr,
      Bus2IP_Clk   => Bus2IP_Clk,
      Bus2IP_CS    => Bus2IP_CS,
      Bus2IP_Data  => Bus2IP_Data,
      Bus2IP_Reset => Bus2IP_Reset,
      Bus2IP_RdCE  => Bus2IP_RdCE,
      Bus2IP_WrCE  => Bus2IP_WrCE,
      IP2Bus_Data  => IP2Bus_Data,
      MyInPort     => MyInPort,
      MyOutPort    => MyOutPort
    );

end architecture imp;

--USER-- The following signals must be driven by the user_logic or by the
--       following assignments.

 IP2Bus_Ack        <= '1';  -- no wait states, immediate Ack. Drive
                            -- from user_logic to add wait states
 IP2Bus_Retry    <= '0'; -- no retry
 IP2Bus_Error    <= '0'; -- no error
 IP2Bus_ToutSup    <=  '0'; -- no timeout suppress. If IP2Bus_Ack is
                            -- delayed more than 15 clocks, drive
                               -- IP2Bus_ToutSup to '1' to avoid
                            -- arbiter timeout on OPB.
 IP2Bus_PostedWrInh <= '0'; -- do not inhibit posted write
```

Step 9. The next step is to edit the .mpd and .pao template files. In the .mpd file you just need to change the IP name and add the user logic ports. In the .pao file, you need to add the files you have added to the user core that will be compiled when the platform generation tools are run. You can again search for --USER-- to find the lines that may require editing. Remember that the .pao file defines the Peripheral Analyze Order, so the files must be put in the correct order for VHDL compilation (subdesigns first followed by the top level file).

```
##############################################################################
##
## Microprocessor Peripheral Definition
##
##############################################################################

BEGIN opb_mycore, IPTYPE = PERIPHERAL, EDIF=TRUE # --USER-- change core name

BUS_INTERFACE BUS = SOPB, BUS_STD = OPB, BUS_TYPE = SLAVE

## Generics for VHDL or Parameters for Verilog
PARAMETER c_baseaddr     = 0xFFFFFFFF, DT = std_logic_vector, MIN_SIZE = 0xFF
PARAMETER c_highaddr     = 0x00000000, DT = std_logic_vector
PARAMETER c_mir_baseaddr = 0xFFFFFFFF, DT = std_logic_vector, MIN_SIZE = 0xFF
PARAMETER c_mir_highaddr = 0x00000000, DT = std_logic_vector
PARAMETER c_user_id_code = 3,          DT = integer
PARAMETER c_include_mir  = 0,          DT = integer
PARAMETER c_opb_awidth   = 32,         DT = integer
PARAMETER c_opb_dwidth   = 32,         DT = integer
PARAMETER c_family       = virtex2,    DT = string
# --USER-- Add user core parameters

## Ports
PORT opb_abus    = OPB_ABus,    DIR = IN, VEC = [0:(c_opb_awidth-1)],     BUS = SOPB
PORT opb_be      = OPB_BE,      DIR = IN, VEC = [0:((c_opb_dwidth/8)-1)], BUS = SOPB
PORT opb_clk     = "",          DIR = IN,                                 BUS = SOPB
PORT opb_dbus    = OPB_DBus,    DIR = IN, VEC = [0:(c_opb_dwidth-1)],     BUS = SOPB
PORT opb_rnw     = OPB_RNW,     DIR = IN,                                 BUS = SOPB
PORT opb_rst     = OPB_Rst,     DIR = IN,                                 BUS = SOPB
PORT opb_select  = OPB_select,  DIR = IN,                                 BUS = SOPB
PORT opb_seqaddr = OPB_seqAddr, DIR = IN,                                 BUS = SOPB
PORT sln_dbus    = Sl_DBus,     DIR = OUT, VEC = [0:(c_opb_dwidth-1)],    BUS = SOPB
PORT sln_errack  = Sl_errAck,   DIR = OUT,                                BUS = SOPB
PORT sln_retry   = Sl_retry,    DIR = OUT,                                BUS = SOPB
PORT sln_toutsup = Sl_toutSup,  DIR = OUT,                                BUS = SOPB
PORT sln_xferack = Sl_xferAck,  DIR = OUT,                                BUS = SOPB

# --USER-- change to user core ports

PORT MyInPort      = "",   DIR = IN,  VEC = [0:3]
PORT MyOutPort     = "",   DIR = OUT, VEC = [0:3]

END

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

##############################################################################
#
# opb_core_ssp0 pao file
#
##############################################################################

lib proc_common_v1_00_b    proc_common_pkg
lib proc_common_v1_00_b    pselect
lib proc_common_v1_00_b    or_muxcy
lib ipif_common_v1_00_a    ipif_pkg
lib ipif_common_v1_00_a    ipif_steer
lib opb_bus_attach_v1_00_a reset_mir
lib opb_bus_attach_v1_00_a opb_bus_attach
lib opb_ipif_ssp0_v1_00_a  opb_ipif_ssp0

# --USER-- add all user core source files and change the following source to
#          your top level core name and library

lib opb_mycore_v1_00_a my_subblock
lib opb_mycore_v1_00_a mycore_user_logic
lib opb_mycore_v1_00_a opb_mycore
```

Step 10. The final step is to get your core into the system's MHS file. This can be done by editing the MHS file with a text editor or by using the Project → Add/Edit Cores... (dialog) menu selection or the Project → Add Core... (text) menu selection.

The following table shows the revision history for this document.

Date	Version	Revision
11/19/02	1.0	Initial Xilinx release.
1/17/03	1.1	Added base address specification section.

User Core Template for OPB (Master Services Package 0)

DS459 (v1.1) Jan. 17, 2003 **Product Specification**

Summary

This document provides the specification for a VHDL template that can be used to create a simple OPB master. The design template also includes an MPD file template and a PAO file template that can be modified as needed for the user core. The set of features and services provided in this template is called the Master Services Package 0 (MSP0). It can be used to simplify creation of user cores that require only an OPB master attachment.

opb_core_msp0	v1.00b

Introduction

The MSP0 OPB user core template consists of the following files:

- A VHDL file, opb_core_msp0.vhd, that provides the core's entity and architectural sections, instantiates an IP Interface (IPIF) for connection to the host bus, and provides an example instantiation of user logic.
- An MPD (Microprocessor Peripheral Definition) file that provides all of the required MPD contents except for the additional ports required by the user logic.
- A PAO (Peripheral Analyze Order) file that provides all of the required PAO entries except for the files required by the user logic.
- An example user logic design which consists of a simple peripheral that fills memory with an incrementing data pattern.

The services provided by the MSP0 OPB user core template are:

- Master-only attachment to the OPB bus.
- Support for single-beat transfers (no burst support).
- Support for delayed data acknowledge controlled by the host bus.
- Connection to user logic with a simplified set of IPIC (IP Interconnect) signals.

Block Diagram

The block diagram of the MSP0 user core template is shown below in Figure 1. The VHDL template supplies two components: a component for the user logic and a component for an IPIF (IP Interface). The IPIF used in the template is a simplified IPIF that uses only the services and ports required by the capabilities provided with the template. The opb_core_msp0.vhd file uses a similarly named IPIF called opb_ipif_msp0.vhd. It is important to understand that the IPIF used in the MSP0 user core template is not a full IPIF but only provides the parts of the full

IPIF required for a simple master. This simplifies the use of the IPIF, reduces the complexity of interfacing to it, and insures that FPGA resources are used efficiently.

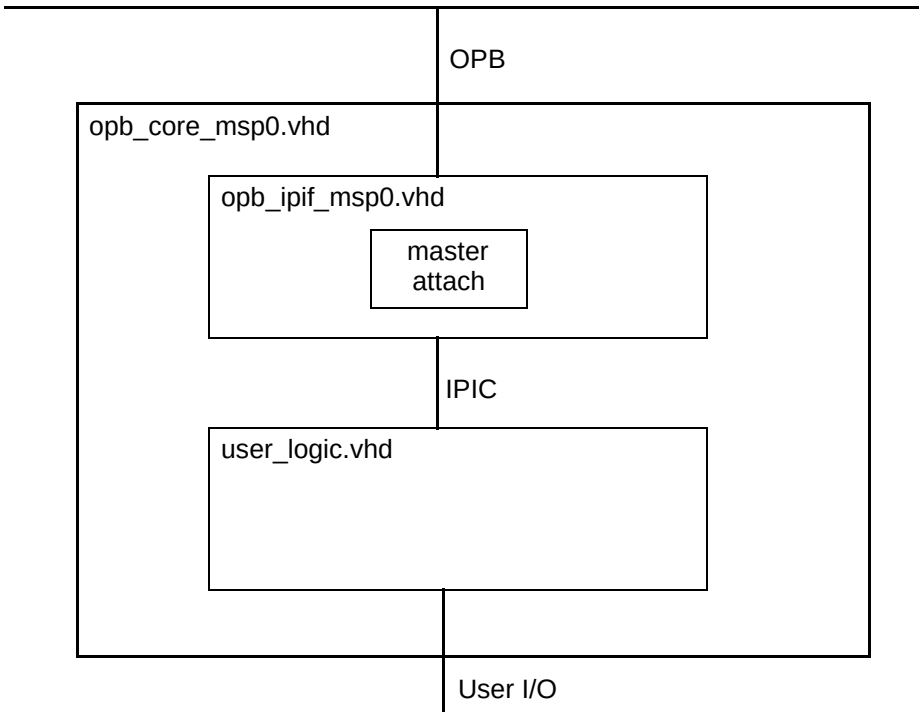

Figure 1: **MSP0 User Core Template Block Diagram**

Using the Template

The basic steps for using the OPB core template are as follows. A detailed example follows this listing.

- Create a directory under the "myip" directory in your project directory with the name and version of the user core (example: opb_mycore_v1_00_a).
- Create a user logic design that uses the IPIC signal set described below.
- Rename the opb_core_msp0.vhd template file to the same name as the user core (example: rename opb_core_msp0.vhd to opb_mycore.vhd)
- In the renamed template file, edit the entity name and library statements as required for the user core. In the template file, lines that may require editing are marked with a --USER-- comment.
- Remove the example user logic component declaration and instantiation and replace them with the user logic component declaration and instantiation.
- Add the ports required by the user logic to the entity declaration.
- In the "data" directory, change the name of the .mpd and .pao files to be the same as the user core name (example: rename opb_core_msp0_v2_0_0.mpd to opb_mycore_v2_0_0.mpd and rename opb_core_msp0_v2_0_0.pao to opb_mycore_v2_0_0.pao). Note that the _v2_0_0 suffix is required to identify the PSF format version. Use the same version suffix as the supplied template.
- In the renamed MPD file, edit the core name and add any user ports. Lines that may require editing are identified by the # --USER-- comment.
- In the renamed PAO file, add the source files required by the user core. The library name for the user core is the user core name plus the version suffix. Lines that may require editing are identified by the # --USER-- comment.

IPIC Signal Set

The following IPIC signals are used in the MSP0 OPB user core template. These signals are required to be present as ports on the user logic or they must be handled appropriately in the opb_core_msp0 template. All signals are active high. See the chapter "Adding User Cores to Your Embedded System" for more details on the IPIC signals.

```
signal Bus2IP_Clk        : std_logic;
signal Bus2IP_Data    : std_logic_vector(0 to C_OPB_DWIDTH-1);
signal Bus2IP_Freeze     : std_logic;
signal Bus2IP_IPMstTrans : std_logic;
signal Bus2IP_MstAck     : std_logic;
signal Bus2IP_MstRetry   : std_logic;
signal Bus2IP_MstError   : std_logic;
signal Bus2IP_MstTimeOut : std_logic;
signal Bus2IP_MstLastAck : std_logic;
signal Bus2IP_Reset      : std_logic;
signal IP2Bus_Addr    : std_logic_vector(0 to C_OPB_AWIDTH - 1);
signal IP2Bus_Data    : std_logic_vector(0 to C_OPB_DWIDTH - 1);
signal IP2Bus_MstReq     : std_logic := '0';
signal IP2Bus_MstRNW     : std_logic := '0';
signal IP2Bus_MstBurst   : std_logic := '0';
signal IP2Bus_MstBusLock : std_logic := '0';
```

Example: Creating an OPB User Core

This example goes through the entire process of creating an OPB user core from the template. The example creates a core called "opb_mycore" and uses it in an MHS file. This example assumes you have a project directory called "system_example" in which your MHS, MSS, and other system files are placed, and that your EDK install directory is `C:/EDK`. This example also assumes that you have your user logic in a directory called `C:/user_logic` and that the user logic consists of two files, `mycore_user_logic.vhd` and `my_subblock.vhd`.

Step 1. In the `system_example` directory, create a directory called `myip` if it doesn't already exist:

```
$ mkdir myip
```

Step 2. Copy the OPB user core template from the install area to the myip directory that you just created (this assumes you are creating version v1.00a of your user core; change the destination name as required for a different user core version):

```
$ cp -r c:/EDK/hw/user_core_templates/opb_core_msp0_v1_00_b
myip/opb_mycore_v1_00_a
```

Step 3. Go into the myip/opb_mycore_v1_00_a/data directory and rename the .mpd and .pao files to opb_mycore:

```
$ cd myip/opb_mycore_v1_00_a/data
$ mv opb_core_msp0_v2_0_0.mpd opb_mycore_v2_0_0.mpd
$ mv opb_core_msp0_v2_0_0.pao opb_mycore_v2_0_0.pao
```

Step 4. Go into the VHDL source directory and rename the opb_core_msp0.vhd file to opb_mycore.vhd:

```
$ cd ../hdl/vhdl
$ mv opb_core_msp0.vhd opb_mycore.vhd
```

Step 5. Copy your user logic files into the VHDL directory where opb_mycore.vhd resides.

```
$ cp c:/user_logic/*.vhd .
```

Step 6. Edit the `opb_mycore.vhd` file, modifying the comment line at the top, library name if needed, entity declaration, and architecture statement. These lines are all marked by the --USER-- comment. What you are accomplishing in this step is to replace all occurrences of opb_core_msp0 with opb_mycore. When you are finished, the beginning of the file should look like this:

```
-- $Id: opb_mycore.vhd,v 1.1 2002/11/08 22:56:56 tise Exp $
--USER-- Add comment with name of this source file
-- opb_mycore.vhd

library ieee;
use ieee.std_logic_1164.all;

library opb_ipif_msp0_v1_00_a;
use opb_ipif_msp0_v1_00_a.all;

--  Add user library if this source file uses
--  entities from the user library
library opb_mycore_v1_00_a; --USER-- library name
use opb_mycore_v1_00_a.all; --USER-- use statement

------------------------------------------------------------------------------
-- entity
------------------------------------------------------------------------------

entity OPB_Mycore is  --USER-- change entity name
  generic
  (
    C_OPB_AWIDTH         : INTEGER := 32;
    C_OPB_DWIDTH         : INTEGER := 32;
    C_DEV_BURST_ENABLE   : INTEGER := 0;
    C_DEV_MAX_BURST_SIZE : INTEGER := 64;
    C_FAMILY             : string  := "virtex2"  );
  port
  (
    --Required OPB bus ports, do not add to or delete
    Mn_ABus      : out std_logic_vector(0 to C_OPB_DWIDTH - 1 );
    Mn_DBus      : out std_logic_vector(0 to C_OPB_DWIDTH - 1 );
    Mn_request   : out std_logic;
    Mn_busLock   : out std_logic;
    Mn_select    : out std_logic;
    Mn_RNW       : out std_logic;
    Mn_BE        : out std_logic_vector(0 to C_OPB_DWIDTH/8 - 1 );
    Mn_seqAddr   : out std_logic;
    OPB_Clk      : in  std_logic := '0';
    OPB_Rst      : in  std_logic := '0';
    OPB_DBus     : in  std_logic_vector(0 to C_OPB_DWIDTH - 1 )
                         := (others => '0');
    OPB_MGrant   : in  std_logic := '0';
    OPB_xferAck  : in  std_logic := '0';
    OPB_errAck   : in  std_logic := '0';
    OPB_retry    : in  std_logic := '0';
    OPB_timeout  : in  std_logic := '0';

    -- Other control signals
    Freeze           : in  std_logic := '0';

    --USER-- add user I/Os to port list
    LED              : out std_logic_vector(0 to 3)

  );
end entity OPB_Mycore; --USER-- change entity name

------------------------------------------------------------------------------
-- architecture
------------------------------------------------------------------------------
architecture imp of OPB_Mycore is --USER-- change entity name
```

Step 7. Edit the entity ports to include the user I/Os:

```
entity OPB_Mycore is  --USER-- change entity name
  generic
  (
    C_OPB_AWIDTH         : INTEGER := 32;
    C_OPB_DWIDTH         : INTEGER := 32;
    C_DEV_BURST_ENABLE   : INTEGER := 0;
    C_DEV_MAX_BURST_SIZE : INTEGER := 64;
    C_FAMILY             : string  := "virtex2"  );
  port
  (
    --Required OPB bus ports, do not add to or delete
    Mn_ABus      : out std_logic_vector(0 to C_OPB_DWIDTH - 1 );
    Mn_DBus      : out std_logic_vector(0 to C_OPB_DWIDTH - 1 );
    Mn_request   : out std_logic;
    Mn_busLock   : out std_logic;
    Mn_select    : out std_logic;
    Mn_RNW       : out std_logic;
    Mn_BE        : out std_logic_vector(0 to C_OPB_DWIDTH/8 - 1 );
    Mn_seqAddr   : out std_logic;
    OPB_Clk      : in  std_logic := '0';
    OPB_Rst      : in  std_logic := '0';
    OPB_DBus     : in  std_logic_vector(0 to C_OPB_DWIDTH - 1 )
                       := (others => '0');
    OPB_MGrant   : in  std_logic := '0';
    OPB_xferAck  : in  std_logic := '0';
    OPB_errAck   : in  std_logic := '0';
    OPB_retry    : in  std_logic := '0';
    OPB_timeout  : in  std_logic := '0';

    -- Other control signals
    Freeze           : in  std_logic := '0';

    --USER-- add user I/Os to port list
    --LED            : out std_logic_vector(0 to 3);

    MyInPort         : in  std_logic_vector(0 to 3);
    MyOutPort        : out std_logic_vector(0 to 3)
  );
end entity OPB_Mycore; --USER-- change entity name
```

Step 8. Edit the user logic component declaration and instantiation:

```
------------------------------------------------------------------------------
-- architecture
------------------------------------------------------------------------------

architecture imp of OPB_Mycore is --USER-- change entity name

  --USER-- component declaration for user core logic. Change the
  --component declaration below to the declaration of the user's
  --logic.

  component mycore_user_logic is
    generic
    (
      C_MEM_FILL_ADDR_START : std_logic_vector(0 to 31)
                              := X"FFFFC000";
      C_MEM_FILL_ADDR_END   : std_logic_vector(0 to 31)
                              := X"FFFFFFFF";
      C_REPEAT              : INTEGER := 1;
      C_OPB_AWIDTH          : INTEGER := 32;
      C_OPB_DWIDTH          : INTEGER := 32;
```

```
    C_DEV_BURST_ENABLE    : INTEGER := 0;
    C_DEV_MAX_BURST_SIZE  : INTEGER := 64;
    C_FAMILY              : string  := "virtex2"
  );
  port
  (
    Bus2IP_Clk        : in  std_logic;
    Bus2IP_Data       : in  std_logic_vector(0 to
                                          C_OPB_DWIDTH-1);
    Bus2IP_Freeze     : in  std_logic;
    Bus2IP_IPMstTrans : in  std_logic;
    Bus2IP_MstAck     : in  std_logic;
    Bus2IP_MstRetry   : in  std_logic;
    Bus2IP_MstError   : in  std_logic;
    Bus2IP_MstTimeOut : in  std_logic;
    Bus2IP_MstLastAck : in  std_logic;
    Bus2IP_Reset      : in  std_logic;
    IP2Bus_Addr       : out std_logic_vector(0 to
                        C_OPB_AWIDTH - 1 ) := (others => '0');
    IP2Bus_Data       : out std_logic_vector(0 to
                        C_OPB_DWIDTH -1) := (others => '0');
    IP2Bus_MstBE      : out std_logic_vector(0 to
                        C_OPB_DWIDTH/8 - 1 ) := (others => '0');
    IP2Bus_MstReq     : out std_logic := '0';
    IP2Bus_MstRNW     : out std_logic := '0';
    IP2Bus_MstBurst   : out std_logic := '0';
    IP2Bus_MstBusLock : out std_logic := '0';
    MyInPort          : in  std_logic_vector(0 to 3);
    MyOutPort         : out std_logic_vector(0 to 3)
  );
end component mycore_user_logic;

--OPB_IPIF_MSP0 declaration -- do not change
component opb_ipif_msp0 is
  generic
  (
    C_OPB_AWIDTH         : INTEGER := 32;
    C_OPB_DWIDTH         : INTEGER := 32;
    C_DEV_BURST_ENABLE   : INTEGER := 0;
    C_DEV_MAX_BURST_SIZE : INTEGER := 64;
    C_FAMILY             : string  := "virtex2"
  );
  port
  (
    Mn_ABus     : out std_logic_vector(0 to C_OPB_DWIDTH - 1 );
    Mn_DBus     : out std_logic_vector(0 to C_OPB_DWIDTH - 1 );
    Mn_request  : out std_logic;
    Mn_busLock  : out std_logic;
    Mn_select   : out std_logic;
    Mn_RNW      : out std_logic;
    Mn_BE       : out std_logic_vector(0 to C_OPB_DWIDTH/8 - 1 );
    Mn_seqAddr  : out std_logic;
    OPB_Clk     : in  std_logic := '0';
    OPB_Rst     : in  std_logic := '0';
    OPB_DBus    : in  std_logic_vector(0 to
                      C_OPB_DWIDTH - 1 ) := (others => '0');
    OPB_MGrant  : in  std_logic := '0';
    OPB_xferAck : in  std_logic := '0';
    OPB_errAck  : in  std_logic := '0';
    OPB_retry   : in  std_logic := '0';
    OPB_timeout : in  std_logic := '0';
    Bus2IP_Clk  : out std_logic;
```

```
      Bus2IP_Data       : out std_logic_vector(0 to C_OPB_DWIDTH-1);
      Bus2IP_Freeze     : out std_logic;
      Bus2IP_IPMstTrans : out std_logic;
      Bus2IP_MstAck     : out std_logic;
      Bus2IP_MstRetry   : out std_logic;
      Bus2IP_MstError   : out std_logic;
      Bus2IP_MstTimeOut : out std_logic;
      Bus2IP_MstLastAck : out std_logic;
      Bus2IP_Reset      : out std_logic;
      IP2Bus_Addr       : in  std_logic_vector(0 to
                                C_OPB_AWIDTH-1) := (others => '0');
      IP2Bus_Data       : in  std_logic_vector(0 to
                                C_OPB_DWIDTH-1) := (others => '0');
      IP2Bus_MstBE      : in  std_logic_vector(0 to
                                C_OPB_DWIDTH/8-1) := (others => '0');
      IP2Bus_MstReq     : in  std_logic := '0';
      IP2Bus_MstRNW     : in  std_logic := '0';
      IP2Bus_MstBurst   : in  std_logic := '0';
      IP2Bus_MstBusLock : in  std_logic := '0';
      Freeze            : in  std_logic := '0'
    );
  end component opb_ipif_msp0;

  --IP Interconnect (IPIC) signal list --do not delete
  signal Bus2IP_Clk        : std_logic;
  signal Bus2IP_Data       : std_logic_vector(0 to C_OPB_DWIDTH-1);
  signal Bus2IP_Freeze     : std_logic;
  signal Bus2IP_IPMstTrans : std_logic;
  signal Bus2IP_MstAck     : std_logic;
  signal Bus2IP_MstRetry   : std_logic;
  signal Bus2IP_MstError   : std_logic;
  signal Bus2IP_MstTimeOut : std_logic;
  signal Bus2IP_MstLastAck : std_logic;
  signal Bus2IP_Reset      : std_logic;
  signal IP2Bus_Addr       : std_logic_vector(0 to
                             C_OPB_AWIDTH - 1) := (others => '0');
  signal IP2Bus_Data       : std_logic_vector(0 to
                             C_OPB_DWIDTH - 1) := (others => '0');
  signal IP2Bus_MstBE      : std_logic_vector(0 to
                                  C_OPB_DWIDTH/8 - 1 ) := (others => '0');
  signal IP2Bus_MstReq     : std_logic := '0';
  signal IP2Bus_MstRNW     : std_logic := '0';
  signal IP2Bus_MstBurst   : std_logic := '0';
  signal IP2Bus_MstBusLock : std_logic := '0';

------------------------------------------------------------------------------
begin
------------------------------------------------------------------------------

  --OPB IPIF lite instantiation --do not delete
  OPB_IPIF_MSP0_I : opb_ipif_msp0
    generic map
    (
      C_OPB_AWIDTH         => C_OPB_AWIDTH,
      C_OPB_DWIDTH         => C_OPB_DWIDTH,
      C_DEV_BURST_ENABLE   => C_DEV_BURST_ENABLE,
      C_DEV_MAX_BURST_SIZE => C_DEV_MAX_BURST_SIZE,
      C_FAMILY             => C_FAMILY
    )
    port map
    (
      Mn_ABus           => Mn_ABus,
```

```
    Mn_DBus           => Mn_DBus,
    Mn_request        => Mn_request,
    Mn_busLock        => Mn_busLock,
    Mn_select         => Mn_select,
    Mn_RNW            => Mn_RNW,
    Mn_BE             => Mn_BE,
    Mn_seqAddr        => Mn_seqAddr,
    OPB_Clk           => OPB_Clk,
    OPB_Rst           => OPB_Rst,
    OPB_DBus          => OPB_DBus,
    OPB_MGrant        => OPB_MGrant,
    OPB_xferAck       => OPB_xferAck,
    OPB_errAck        => OPB_errAck,
    OPB_retry         => OPB_retry,
    OPB_timeout       => OPB_timeout,
    Bus2IP_Clk        => Bus2IP_Clk,
    Bus2IP_Data       => Bus2IP_Data,
    Bus2IP_Freeze     => Bus2IP_Freeze,
    Bus2IP_IPMstTrans => Bus2IP_IPMstTrans,
    Bus2IP_MstAck     => Bus2IP_MstAck,
    Bus2IP_MstRetry   => Bus2IP_MstRetry,
    Bus2IP_MstError   => Bus2IP_MstError,
    Bus2IP_MstTimeOut => Bus2IP_MstTimeOut,
    Bus2IP_MstLastAck => Bus2IP_MstLastAck,
    Bus2IP_Reset      => Bus2IP_Reset,
    IP2Bus_Addr       => IP2Bus_Addr,
    IP2Bus_Data       => IP2Bus_Data,
    IP2Bus_MstBE      => IP2Bus_MstBE,
    IP2Bus_MstReq     => IP2Bus_MstReq,
    IP2Bus_MstRNW     => IP2Bus_MstRNW,
    IP2Bus_MstBurst   => IP2Bus_MstBurst,
    IP2Bus_MstBusLock => IP2Bus_MstBusLock,
    Freeze            => Freeze
  );

--USER-- change the USER_LOGIC_MASTER_I instantiation below to the
--instantiation of the user logic.

USER_LOGIC_MASTER0_I : mycore_user_logic
  generic map
  (
    C_MEM_FILL_ADDR_START => X"10000000",
    C_MEM_FILL_ADDR_END   => X"1000FFFF",
    C_REPEAT              => 1,
    C_OPB_AWIDTH          => C_OPB_AWIDTH,
    C_OPB_DWIDTH          => C_OPB_DWIDTH,
    C_DEV_BURST_ENABLE    => C_DEV_BURST_ENABLE,
    C_DEV_MAX_BURST_SIZE  => C_DEV_MAX_BURST_SIZE,
    C_FAMILY              => C_FAMILY
  )
  port map
  (
    Bus2IP_Clk        => Bus2IP_Clk,
    Bus2IP_Data       => Bus2IP_Data,
    Bus2IP_Freeze     => Bus2IP_Freeze,
    Bus2IP_IPMstTrans => Bus2IP_IPMstTrans,
    Bus2IP_MstAck     => Bus2IP_MstAck,
    Bus2IP_MstRetry   => Bus2IP_MstRetry,
    Bus2IP_MstError   => Bus2IP_MstError,
    Bus2IP_MstTimeOut => Bus2IP_MstTimeOut,
    Bus2IP_MstLastAck => Bus2IP_MstLastAck,
    Bus2IP_Reset      => Bus2IP_Reset,
```

```
        IP2Bus_Addr       => IP2Bus_Addr,
        IP2Bus_Data       => IP2Bus_Data,
        IP2Bus_MstBE      => IP2Bus_MstBE,
        IP2Bus_MstReq     => IP2Bus_MstReq,
        IP2Bus_MstRNW     => IP2Bus_MstRNW,
        IP2Bus_MstBurst   => IP2Bus_MstBurst,
        IP2Bus_MstBusLock => IP2Bus_MstBusLock,
        MyInPort          => MyInPort,
        MyOutPort         => MyOutPort
      );

  end architecture imp;
```

Step 9. The next step is to edit the .mpd and .pao template files. In the .mpd file you just need to change the IP name and add the user logic ports. In the .pao file, you need to add the files you have added to the user core that will be compiled when the platform generation tools are run. You can again search for --USER-- to find the lines that may require editing. Remember that the .pao file defines the Peripheral Analyze Order, so the files must be put in the correct order for VHDL compilation (subdesigns first followed by the top level file).

```
##############################################################################
##
## Microprocessor Peripheral Definition
##
##############################################################################

BEGIN opb_mycore, IPTYPE = PERIPHERAL, IMP_NETLIST=TRUE #--USER--
BUS_INTERFACE BUS = MOPB, BUS_STD = OPB, BUS_TYPE = MASTER

## Generics for VHDL or Parameters for Verilog
PARAMETER c_opb_awidth         = 32,      DT = integer
PARAMETER c_opb_dwidth         = 32,      DT = integer
PARAMETER c_dev_burst_enable   = 0,       DT = integer
PARAMETER c_dev_max_burst_size = 64,      DT = integer
PARAMETER c_family             = virtex2, DT = string
#--USER-- Add User PARAMETERS here

## Port
PORT mn_abus = M_ABus,DIR=OUT,VEC=[0:(c_opb_dwidth-1)],BUS = MOPB
PORT mn_dbus = M_DBus,DIR=OUT,VEC=[0:(c_opb_dwidth-1)],BUS = MOPB
PORT mn_request = M_request, DIR = OUT,                BUS = MOPB
PORT mn_buslock = M_busLock, DIR = OUT,                BUS = MOPB
PORT mn_select  = M_select, DIR = OUT,                 BUS = MOPB
PORT mn_rnw      = M_RNW,   DIR = OUT,                 BUS = MOPB
PORT mn_be = M_BE,DIR = OUT,VEC=[0:((c_opb_dwidth/8)-1)],BUS = MOPB
PORT mn_seqaddr  = M_seqAddr,   DIR = OUT,             BUS = MOPB
PORT opb_clk     = "",          DIR = IN, SIGIS = CLK, BUS = MOPB
PORT opb_rst     = OPB_Rst,     DIR = IN,              BUS = MOPB
PORT opb_dbus = OPB_DBus,DIR = IN,VEC = [0:(c_opb_dwidth-1)],BUS = MOPB
PORT opb_mgrant  = OPB_MGrant,  DIR = IN,              BUS = MOPB
PORT opb_xferack = OPB_xferAck, DIR = IN,              BUS = MOPB
PORT opb_errack  = OPB_errAck,  DIR = IN,              BUS = MOPB
PORT opb_retry   = OPB_retry,   DIR = IN,              BUS = MOPB
PORT opb_timeout = OPB_timeout, DIR = IN,              BUS = MOPB
PORT freeze      = "",          DIR = IN

#--USER-- Add User PORTS here
PORT MyInPort    = "",          DIR = IN,  VEC = [0:3]
PORT MyOutPort   = "",          DIR = OUT, VEC = [0:3]
END
```

```
-------------------------------------------------

##############################################################
#
# opb_core_msp0 pao file
#
##############################################################

lib proc_common_v1_00_b   proc_common_pkg
lib proc_common_v1_00_b   srl_fifo
lib proc_common_v1_00_b   srl_fifo_rbu
lib proc_common_v1_00_b   pselect
lib proc_common_v1_00_b   ld_arith_reg
lib proc_common_v1_00_b   down_counter
lib proc_common_v1_00_b   inferred_lut4
lib proc_common_v1_00_b   or_muxcy
lib proc_common_v1_00_b   or_gate
lib proc_common_v1_00_b   family
lib ipif_common_v1_00_a   ipif_pkg
lib ipif_common_v1_00_a   interrupt_control
lib ipif_common_v1_00_a   ipif_steer
lib opb_ipif_v2_00_f      address_decoder
lib opb_ipif_v2_00_f      slave_attachment
lib opb_ipif_v2_00_f      pf_counter_bit
lib opb_ipif_v2_00_f      pf_counter
lib opb_ipif_v2_00_f      pf_counter_top
lib opb_ipif_v2_00_f      pf_occ_counter
lib opb_ipif_v2_00_f      pf_occ_counter_top
lib opb_ipif_v2_00_f      pf_adder_bit
lib opb_ipif_v2_00_f      pf_adder
lib opb_ipif_v2_00_f      pf_dpram_select
lib opb_ipif_v2_00_f      srl16_fifo
lib opb_ipif_v2_00_f      pf_dly1_mux
lib opb_ipif_v2_00_f      rdpfifo_dp_cntl
lib opb_ipif_v2_00_f      ipif_control_rd
lib opb_ipif_v2_00_f      rdpfifo_top
lib opb_ipif_v2_00_f      wrpfifo_dp_cntl
lib opb_ipif_v2_00_f      ipif_control_wr
lib opb_ipif_v2_00_f      wrpfifo_top
lib opb_ipif_v2_00_f      bus2ip_amux
lib opb_ipif_v2_00_f      ip2bus_dmux
lib opb_ipif_v2_00_f      ip2bus_dmux_blk
lib opb_ipif_v2_00_f      ip2bus_srmux
lib opb_ipif_v2_00_f      ip2bus_srmux_blk
lib opb_ipif_v2_00_f      ipif_reset
lib opb_ipif_v2_00_f      reset_control
lib opb_ipif_v2_00_f      dma_sg_pkg
lib opb_ipif_v2_00_f      dma_sg_cmp
lib opb_ipif_v2_00_f      ctrl_reg
lib opb_ipif_v2_00_f      dma_sg
lib opb_ipif_v2_00_f      dma_sg_sim
lib opb_ipif_v2_00_f      addr_load_and_incr
lib opb_ipif_v2_00_f      master_attachment
lib opb_ipif_v2_00_f      opb_ipif
lib opb_ipif_msp0_v1_00_a opb_ipif_msp0

# --USER-- add all user core source files and change the following source to
#          your top level core name and library

lib opb_mycore_v1_00_a my_subblock
lib opb_mycore_v1_00_a mycore_user_logic
lib opb_mycore_v1_00_a opb_mycore
```

Step 10. The final step is to get your core into the system's MHS file. This can be done by editing the MHS file with a text editor or by using the Project → Add/Edit Cores... (dialog) menu selection or the Project → Add Core... (text) menu selection.

The following table shows the revision history for this document.

Date	Version	Revision
11/19/02	1.0	Initial Xilinx release.
1/17/03	1.1	Updated to v1.00.b