

User Core Templates Reference Guide

August 2003



The Xilinx logo shown above is a registered trademark of Xilinx, Inc.

ASYL, FPGA Architect, FPGA Foundry, NeoCAD, NeoCAD EPIC, NeoCAD PRISM, NeoROUTE, Timing Wizard, TRACE, XACT, XILINX, XC2064, XC3090, XC4005, XC5210, and XC-DS501 are registered trademarks of Xilinx, Inc.



The shadow X shown above is a trademark of Xilinx, Inc.

All XC-prefix product designations, A.K.A Speed, Alliance Series, AllianceCORE, BITA, CLC, Configurable Logic Cell, CoolRunner, CORE Generator, CoreLINX, Dual Block, EZTag, FastCLK, FastCONNECT, FastFLASH, FastMap, Fast Zero Power, Foundation, HardWire, IRL, LCA, Logi-BLOX, Logic Cell, LogiCORE, LogicProfessor, MicroBlaze, MicroVia, MultiLINX, NanoBlaze, PicoBlaze, PLUSASM, PowerGuide, PowerMaze, QPro, RealPCI, RealPCI 64/66, SelectI/O, SelectRAM, SelectRAM+, Silicon Xpresso, Smartguide, Smart-IP, SmartSearch, Smartspec, SMART-Switch, Spartan, TrueMap, UIM, VectorMaze, VersaBlock, VersaRing, Virtex, WebFitter, WebLINX, WebPACK, XABEL, XACTstep, XACTstep Advanced, XACTstep Foundry, XACT-Floorplanner, XACT-Performance, XAM, XAPP, X-BLOX, X-BLOX plus, XChecker, XDM, XDS, XEPLD, Xilinx Foundation Series, XPP, XSI, and ZERO+ are trademarks of Xilinx, Inc. The Programmable Logic Company and The Programmable Gate Array Company are service marks of Xilinx, Inc.

All other trademarks are the property of their respective owners.

Xilinx, Inc. does not assume any liability arising out of the application or use of any product described or shown herein; nor does it convey any license under its patents, copyrights, or maskwork rights or any rights of others. Xilinx, Inc. reserves the right to make changes, at any time, in order to improve reliability, function or design and to supply the best product possible. Xilinx, Inc. will not assume responsibility for the use of any circuitry described herein other than circuitry entirely embodied in its products. Xilinx, Inc. devices and products are protected under one or more of the following U.S. Patents: 4,642,487; 4,695,740; 4,706,216; 4,713,557; 4,746,822; 4,750,155; 4,758,985; 4,820,937; 4,821,233; 4,835,418;

4,855,619; 4,855,669; 4,902,910; 4,940,909; 4,967,107; 5,012,135; 5,023,606; 5,028,821; 5,047,710; 5,068,603; 5,140,193; 5,148,390; 5,155,432; 5,166,858; 5,224,056; 5,243,238; 5,245,277; 5,267,187; 5,291,079; 5,295,090; 5,302,866; 5,319,252; 5,319,254; 5,321,704; 5,329,174; 5,329,181; 5,331,220; 5,331,226; 5,332,929; 5,337,255; 5,343,406; 5,349,248; 5,349,249; 5,349,250; 5,349,691; 5,357,153; 5,360,747; 5,361,229; 5,362,999; 5,365,125; 5,367,207; 5,386,154; 5,394,104; 5,399,924; 5,399,925; 5,410,189; 5,410,194; 5,414,377; 5,422,833; 5,426,378; 5,426,379; 5,430,687; 5,432,719; 5,448,181; 5,448,493; 5,450,021; 5,450,022; 5,453,706; 5,455,525; 5,466,117; 5,469,003; 5,475,253; 5,477,414; 5,481,206; 5,483,478; 5,486,707; 5,486,776; 5,488,316; 5,489,858; 5,489,866; 5,491,353; 5,495,196; 5,498,979; 5,498,989; 5,499,192; 5,500,608; 5,500,609; 5,502,000; 5,502,440; 5,504,439; 5,506,518; 5,506,523; 5,506,878; 5,513,124; 5,517,135; 5,521,835; 5,521,837; 5,523,963; 5,523,971; 5,524,097; 5,526,322; 5,528,169; 5,528,176; 5,530,378; 5,530,384; 5,546,018; 5,550,839; 5,550,843; 5,552,722; 5,553,001; 5,559,751; 5,561,367; 5,561,629; 5,561,631; 5,563,527; 5,563,528; 5,563,529; 5,563,827; 5,565,792; 5,566,123; 5,570,051; 5,574,634; 5,574,655; 5,578,946; 5,581,198; 5,581,199; 5,581,738; 5,583,450; 5,583,452; 5,592,105; 5,594,367; 5,598,424; 5,600,263; 5,600,264; 5,600,271; 5,600,597; 5,608,342; 5,610,536; 5,610,790; 5,610,829; 5,612,633; 5,617,021; 5,617,041; 5,617,327; 5,617,573; 5,623,387; 5,627,480; 5,629,637; 5,629,886; 5,631,577; 5,631,583; 5,635,851; 5,636,368; 5,640,106; 5,642,058; 5,646,545; 5,646,547; 5,646,564; 5,646,903; 5,648,732; 5,648,913; 5,650,672; 5,650,946; 5,652,904; 5,654,631; 5,656,950; 5,675,290; 5,659,484; 5,661,686; 5,661,685; 5,670,896; 5,670,897; 5,672,966; 5,673,198; 5,675,262; 5,675,270; 5,675,589; 5,677,638; 5,682,107; 5,689,133; 5,689,516; 5,691,907; 5,691,912; 5,694,047; 5,694,056; 5,724,276; 5,694,399; 5,696,454; 5,701,091; 5,701,441; 5,703,759; 5,705,932; 5,705,938; 5,708,597; 5,712,579; 5,715,197; 5,717,340; 5,719,506; 5,719,507; 5,724,276; 5,726,484; 5,726,584; 5,734,866; 5,734,868; 5,737,234; 5,737,235; 5,737,631; 5,742,178; 5,742,531; 5,744,974; 5,744,979; 5,744,995; 5,748,942; 5,748,979; 5,752,006; 5,752,035; 5,754,459; 5,758,192; 5,760,603; 5,760,604; 5,760,607; 5,761,483; 5,764,076; 5,764,534; 5,764,564; 5,768,179; 5,770,951; 5,773,993; 5,778,439; 5,781,756; 5,784,313; 5,784,577; 5,786,240; 5,787,007; 5,789,938; 5,790,479; 5,790,882; 5,795,068; 5,796,269; 5,798,656; 5,801,546; 5,801,547; 5,801,548; 5,811,985; 5,815,004; 5,815,016; 5,815,404; 5,815,405; 5,818,255; 5,818,730; 5,821,772; 5,821,774; 5,825,202; 5,825,662; 5,825,787; 5,828,230; 5,828,231; 5,828,236; 5,828,608; 5,831,448; 5,831,460; 5,831,845; 5,831,907; 5,835,402; 5,838,167; 5,838,901; 5,838,954; 5,841,296; 5,841,867; 5,844,422; 5,844,424; 5,844,829; 5,844,844; 5,847,577; 5,847,579; 5,847,580; 5,847,993; 5,852,323; 5,861,761; 5,862,082; 5,867,396; 5,870,309; 5,870,327; 5,870,586; 5,874,834; 5,875,111; 5,877,632; 5,877,979; 5,880,492; 5,880,598; 5,880,620; 5,883,525; 5,886,538; 5,889,411; 5,889,413; 5,889,701; 5,892,681; 5,892,961; 5,894,420; 5,896,047; 5,896,329; 5,898,319; 5,898,320; 5,898,602; 5,898,618; 5,898,893; 5,907,245; 5,907,248; 5,909,125; 5,909,453; 5,910,732; 5,912,937; 5,914,514; 5,914,616; 5,920,201; 5,920,202; 5,920,223; 5,923,185; 5,923,602; 5,923,614; 5,928,338; 5,931,962; 5,933,023; 5,933,025; 5,933,369; 5,936,415; 5,936,424; 5,939,930; 5,942,913; 5,944,813; 5,945,837; 5,946,478; 5,949,690; 5,949,712; 5,949,983; 5,949,987; 5,952,839; 5,952,846; 5,955,888; 5,956,748; 5,958,026; 5,959,821; 5,959,881; 5,959,885; 5,961,576; 5,962,881; 5,963,048; 5,963,050; 5,969,539; 5,969,543; 5,970,142; 5,970,372; 5,971,595; 5,973,506; 5,978,260; 5,986,958; 5,990,704; 5,991,523; 5,991,788; 5,991,880; 5,991,908; 5,995,419; 5,995,744; 5,995,988; 5,999,014; 5,999,025; 6,002,282; and 6,002,991; Re. 34,363, Re. 34,444, and Re. 34,808. Other U.S. and foreign patents pending. Xilinx, Inc. does not represent that devices shown or products described herein are free from patent infringement or from any other third party right. Xilinx, Inc. assumes no obligation to correct any errors contained herein or to advise any user of this text of any correction if such be made. Xilinx, Inc. will not assume any liability for the accuracy or correctness of any engineering or software support or assistance provided to a user.

Xilinx products are not intended for use in life support appliances, devices, or systems. Use of a Xilinx product in such applications without the written consent of the appropriate Xilinx officer is prohibited.

Copyright 1991-2003 Xilinx, Inc. All Rights Reserved.

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 |
| 8/22/03 | 1.3 | Updated for EDK 6.1 |

Adding User Cores to Your Embedded System

Overview

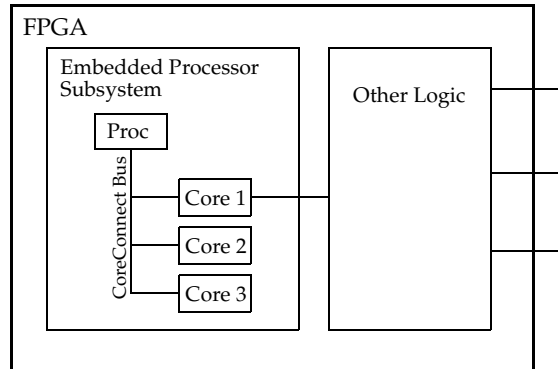
There are several types of user logic associated with FPGA designs that incorporate an embedded processor subsystem. User logic may be unrelated to the embedded processor subsystem, it may have a weak coupling to the processor, or it may be included as an integral part of the processor subsystem. There 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:

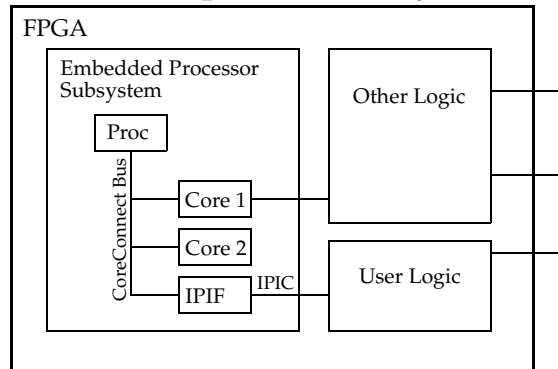
- FPGA - the entire design to be loaded into the FPGA, consisting of an embedded processor subsystem (created by the platform generation tools) and other logic (created by the user).
- Embedded processor subsystem - a design described in an MHS (Microprocessor Hardware Specification) file and generated 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 Reference Design - the User Core Reference Design simplifies the task of attaching the IPIF to user logic. The user core reference design is a VHDL file that instantiates the IPIF and provides most of the VHDL code required to create a user core. The reference design 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 reference designs 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.

System 1 - no user core



System 2 - user logic external to embedded processor subsystem



System 3 - user core part of embedded processor subsystem

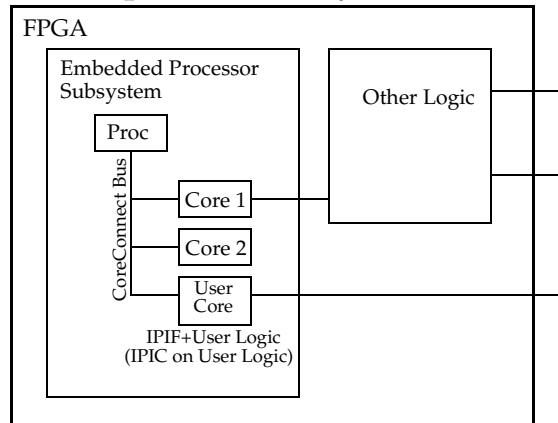


Figure 1-1: Three Embedded System Configurations

User Core Reference Designs

The user core reference designs provide a convenient way to get your user logic attached to the PLB or OPB bus. These reference designs are provided as VHDL source and provide a framework into which user logic can be instantiated. Each user core reference design contains an instantiation of an IPIF (IP Interface). The IPIF instantiated in each reference design is customized so that only the features and services required by the reference design are used. For example, the user core reference design 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 reference designs are provided as starting points for building custom PLB and OPB peripherals. Each reference design has some degree of parameterization as well, so the best way to use the reference designs is to find a reference design that is closest to the features and services required by your core and customize from there.

Reference Designs are named according to the features that are provided by the reference design. For PLB and OPB slaves, there are four reference designs that provide varying degrees of services for the user. PLB and OPB slave reference designs 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 PLB and OPB slave reference designs available and the services provided by each.

Table 1-1: PLB and OPB User Core Reference Designs (Slave)

| Reference Design | PLB/OPB Slave Attachment | Address Decode | Module Identification Register | Reset Register | Device Interrupt Controller | R/W FIFOs | R/W Packet FIFOs |
|---|--------------------------|----------------|--------------------------------|----------------|-----------------------------|-----------|------------------|
| plb_core_ssp0_ref opb_core_ssp0_ref | • | • | • | • | | | |
| plb_core_ssp1_ref opb_core_ssp1_ref | • | • | • | • | • | | |
| plb_core_ssp2_ref opb_core_ssp2_ref ⁽¹⁾ | • | • | • | • | • | • | |
| plb_core_ssp3_ref opb_core_ssp3_ref ⁽¹⁾ | • | • | • | • | • | | • |

Notes:

1. Available in a future release.

For PLB and OPB masters, there are five reference designs that provide varying degrees of services for the user. PLB and OPB master reference designs 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 PLB and OPB master reference designs available and the services provided by each.

Table 1-2: PLB and OPB User Core Reference Designs (Master)

| Reference Design | PLB/OPB Slave Attachment | Address Decode | Module Identification Register | Reset Register | Device Interrupt Controller | R/W FIFOs | R/W Packet FIFOs | OPB Master Attachment | Simple DMA | DMA with scatter/gather |
|--|--------------------------|----------------|--------------------------------|----------------|-----------------------------|-----------|------------------|-----------------------|------------|-------------------------|
| plb_core_msp0_ref ⁽¹⁾ opb_core_msp0_ref ⁽¹⁾ | | | | | | | | • | | |
| plb_core_msp1_ref ⁽¹⁾ opb_core_msp1_ref ⁽¹⁾ | • | • | • | • | | | | • | | |
| plb_core_msp2_ref ⁽¹⁾ opb_core_msp2_ref ⁽¹⁾ | • | • | • | • | • | | | • | | |
| plb_core_msp3_ref ⁽¹⁾ opb_core_msp3_ref ⁽¹⁾ | • | • | • | • | • | • | | • | • | |

Notes:

1. Available in a future release.

Creating a User Core

The process of creating a user core from the OPB user core reference designs is shown in Figure 1-2. This same process is followed for creating a user core from the PLB user core reference designs.

First, an appropriate reference design is selected based on the level of services required. Then, the user logic is inserted into the reference design and the reference design file names and entities are renamed if desired. Finally, the reference design 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.

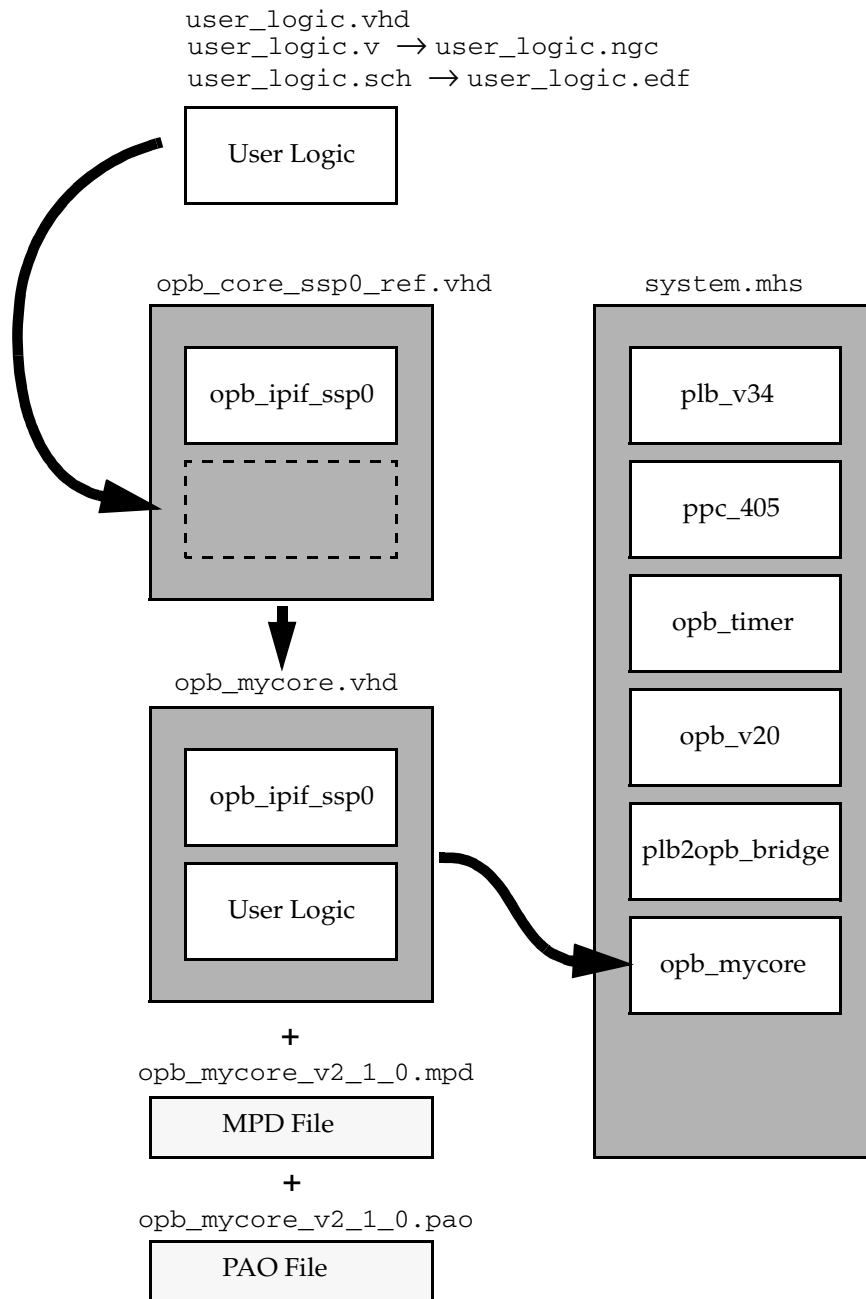


Figure 1-2: OPB User Core Process

Some important points about this process are:

- 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 reference design directory contains a VHDL source reference design, an MPD

template, and a PAO template. These files are located in the EDK installation area under the following directories :

- opb_core_ssp0.vhd is in:

```
$EDK/hw/XilinxReferenceDesigns/pcores/opb_core_ssp0_ref/<core_version>/hdl/vhdl
```

- opb_core_ssp0_v2_1_0.mpd is in:

```
$EDK/hw/XilinxReferenceDesigns/pcores/opb_core_ssp0_ref/<core_version>/data
```

- opb_core_ssp0_v2_1_0.pao is in:

```
$EDK/hw/XilinxReferenceDesigns/pcores/opb_core_ssp0_ref/<core_version>/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 v2.1.0.
- Lines in the reference designs 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 information on a particular reference design, refer to the documentation provided with that reference design in the EDK installation directory.

User Core Reference Systems

Reference System designs that incorporate the User Core Reference designs into a real system are provided in the EDK installation directory (the reference design for OPB Slave Services Package 0 is used for this example):

- `$EDK/hw/XilinxReferenceSystems/opb_ssp0`

These Reference System designs provide a EDK Xilinx Platform Studio project file (system.xps), a Microprocessor Hardware Specification file (system.mhs), and documentation of the reference system along with example software where applicable.

In addition to these files, an example UCF file is provided in the data directory of the project that targets a commercially available development board. This UCF file needs to be modified and renamed system.ucf for use in the user's application hardware.

Reference System designs are currently available for the following User Core Reference Designs:

- OPB Slave Services Package 0 & 1
- PLB Slave Services Package 0, 1, 2, & 3

Additional Reference System designs will be available in future releases.

PLB and OPB IPIC Interfaces

To effectively use the PLB and OPB user core reference designs, 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 majority of the IPIC signal set 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 reference design). 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.

OPB IPIC Signal Set

The IPIC signal set for OPB slave and master peripherals is shown below in [Table 2](#).

Table 2: OPB IPIC I/O Signals

| Signal Name | Range | I/O | Description |
|--------------------|----------------------|-----|--|
| 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 |
| 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 |

PLB IPIC Signal Set

The IPIC signal set for PLB slave and master peripherals is shown below in [Table 3](#).

Table 3: PLB IPIC I/O Signals

| Signal Name | Range | I/O | Description |
|--------------------|--------------------------------------|-----|--|
| 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 |
| IP2Bus_RdAck | none | O | Read transfer acknowledgement from User Logic |
| Bus2IP_RdCE | 0:C_NUM_CE-1 | I | Read enables to User Logic |
| Bus2IP_RdReq | none | I | Read request to User Logic |
| Bus2IP_Reset | none | I | Signal to reset the User Logic |
| Bus2IP_RNW | none | I | Read/Not Write Signal to User Logic |
| IP2Bus_WrAck | none | O | Write transfer acknowledgement from User Logic |
| Bus2IP_WrCE | 0:C_NUM_CE-1 | I | Write enables to User Logic |
| Bus2IP_WrReq | none | I | Read request to User Logic |
| IP2Bus_Data | 0:C_<bus>_DWIDTH-1 | O | Data from IP |
| IP2Bus_Error | none | O | Error response |
| IP2Bus_IntrEvent | 0: C_IP_INTR_MODE_ARRAYlength - 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_MstRdAck | none | I | Master Read Acknowledge from IPIF |
| Bus2IP_MstWrAck | none | I | Master Write 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_MstRdReq | none | O | Master read request from User Logic |

Table 3: PLB IPIC I/O Signals (Continued)

| Signal Name | Range | I/O | Description |
|---------------------|--|-----|---|
| IP2Bus_MstWrReq | none | O | Master write request from User Logic |
| IP2IP_Addr | 0:C_<bus>_AWIDTH-1 | O | Local device address for the master transaction |
| IP2RFIFO_WrReq | none | 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. ⁽¹⁾ |
| IP2RFIFO_Data | (0: C_RDFIFO_DWIDTH-1) | I | Write data from the User IP to the RdFIFO write port is transmitted on this bus. Data present on the bus is written when IP2RFIFO_WrReq is high, RFIFO2IP_WrAck is high, and a rising edge of the Bus2IP_Clk occurs. ⁽¹⁾ |
| IP2RFIFO_WrMark | none | I | Active high signal commanding the RdFIFO to perform a "Mark" operation. ⁽¹⁾ |
| IP2RFIFO_WrRelease | none | I | Active high signal commanding the RdFIFO to perform a "Release" operation. ⁽¹⁾ |
| IP2RFIFO_WrRestore | none | I | Active high signal commanding the RdFIFO to perform a "Restore" operation. ⁽¹⁾ |
| RFIFO2IP_WrAck | none | O | Active high signal indicating that the data write request will complete at the next rising edge of the Bus2IP_Clk signal. ⁽¹⁾ |
| RFIFO2IP_AlmostFull | none | O | Active high signal indicating that the RdFIFO can accept only one more data write. ⁽¹⁾ |
| RFIFO2IP_Full | none | 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. ⁽¹⁾ |
| RFIFO2IP_Vacancy | (0: log ₂ (C_RDFIFO_DEPTH)-1) | O | Status bus indicating the available locations for writing in the RdFIFO. ⁽¹⁾ |
| IP2WFIFO_RdReq | none | 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. ⁽¹⁾ |
| IP2WFIFO_RdMark | none | I | Active high signal commanding the WrFIFO to perform a "Mark" operation. ⁽¹⁾ |
| IP2WFIFO_RdRelease | none | I | Active high signal commanding the WrFIFO to perform a "Release" operation. ⁽¹⁾ |

Table 3: PLB IPIC I/O Signals (Continued)

| Signal Name | Range | I/O | Description |
|----------------------|--|-----|--|
| IP2WFIFO_RdRestore | none | I | Active high signal commanding the WrFIFO to perform a "Restore" operation. ⁽¹⁾ |
| WFIFO2IP_Data | (0: C_WRFIFO_DWIDTH-1) | 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. ⁽¹⁾ |
| WFIFO2IP_RdAck | none | 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. ⁽¹⁾ |
| WFIFO2IP_AlmostEmpty | none | O | Active high signal indicating that the WrFIFO can provide only one more data read. ⁽¹⁾ |
| WFIFO2IP_Empty | none | 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. ⁽¹⁾ |
| WFIFO2IP_Occupancy | (0: log ₂ (C_WRFIFO_DEPTH)-1) | O | Status bus indicating the available locations for reading in the WrFIFO. ⁽¹⁾ |

Notes:

- The behavior of the Read Packet FIFO and the Write Packet FIFO is discussed in the Product Specification **DS415 On-Chip Peripheral Bus IP Interface Packet FIFO**.

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. The behavior of the Read Packet FIFO and the Write Packet FIFO is discussed in the Product Specification **DS415 On-Chip Peripheral Bus IP Interface Packet FIFO** therefore, the signals associated with these services are not described here.

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 reference designs. 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 reference designs, 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 its 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_RdReq (PLB)

The Bus2IP_RdReq signal is an input to the user logic indicating that the requested transaction is a read transfer.

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.

Bus2IP_WrReq (PLB)

The Bus2IP_WrReq signal is an input to the user logic indicating that the requested transaction is a write transfer.

IP2Bus_Ack (OPB), IP2Bus_RdAck, IP2Bus_WrAck (PLB)

These signals provide 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 the acknowledgement. If the IP2Bus_Ack for OPB cores 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 (OPB), IP2Bus_IntrEvent (PLB)

The IP2Bus_Intr/IP2Bus_IntrEvent 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 of the slave signals to complete local transactions. These transactions are described in the chapters on the user reference designs 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 (OPB), Bus2IP_MstRdAck, Bus2IP_MstWrAck (PLB)

This 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 (OPB Only)

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 (OPB Only)

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(OPB Only)

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.

IP2Bus_MstRdReq, IP2Bus_MstWrReq(PLB Only)

IP2Bus_MstRdReq and IP2Bus_MstWrReq are inputs to the IPIF from the user logic that indicates that the user logic is requesting a master transaction (read or write). This request signal must remain asserted until acknowledged by Bus2IP_MstLastAck, Bus2IP_Retry, or Bus2IP_TimeOut.

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. For simplicity, the OPB IPIC signal set is used in these timing diagrams. The timing relationships of the PLB IPIC signals are for the most part identical. If there are differences, these differences are noted in the text description of the timing diagram. Please refer to the OPB and or PLB IPIF documentation for complete details.

The behavior of the Read Packet FIFO and the Write Packet FIFO is discussed in the Product Specification **DS415 On-Chip Peripheral Bus IP Interface Packet FIFO** therefore, the timing diagrams associated with these services are not repeated here.

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-cycle IP2Bus_Ack signal.

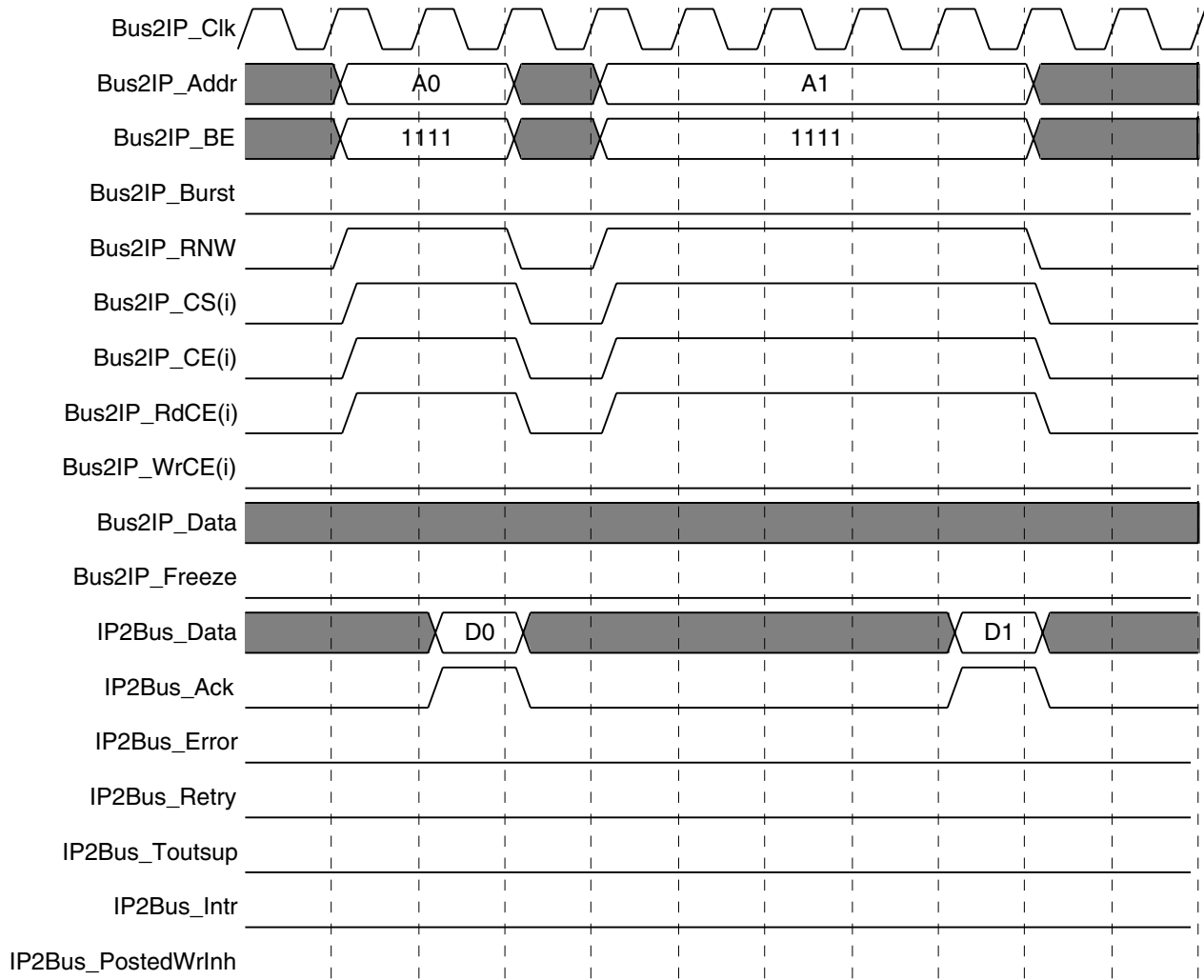


Figure 1-3: Slave Read – Single Beat

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 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.

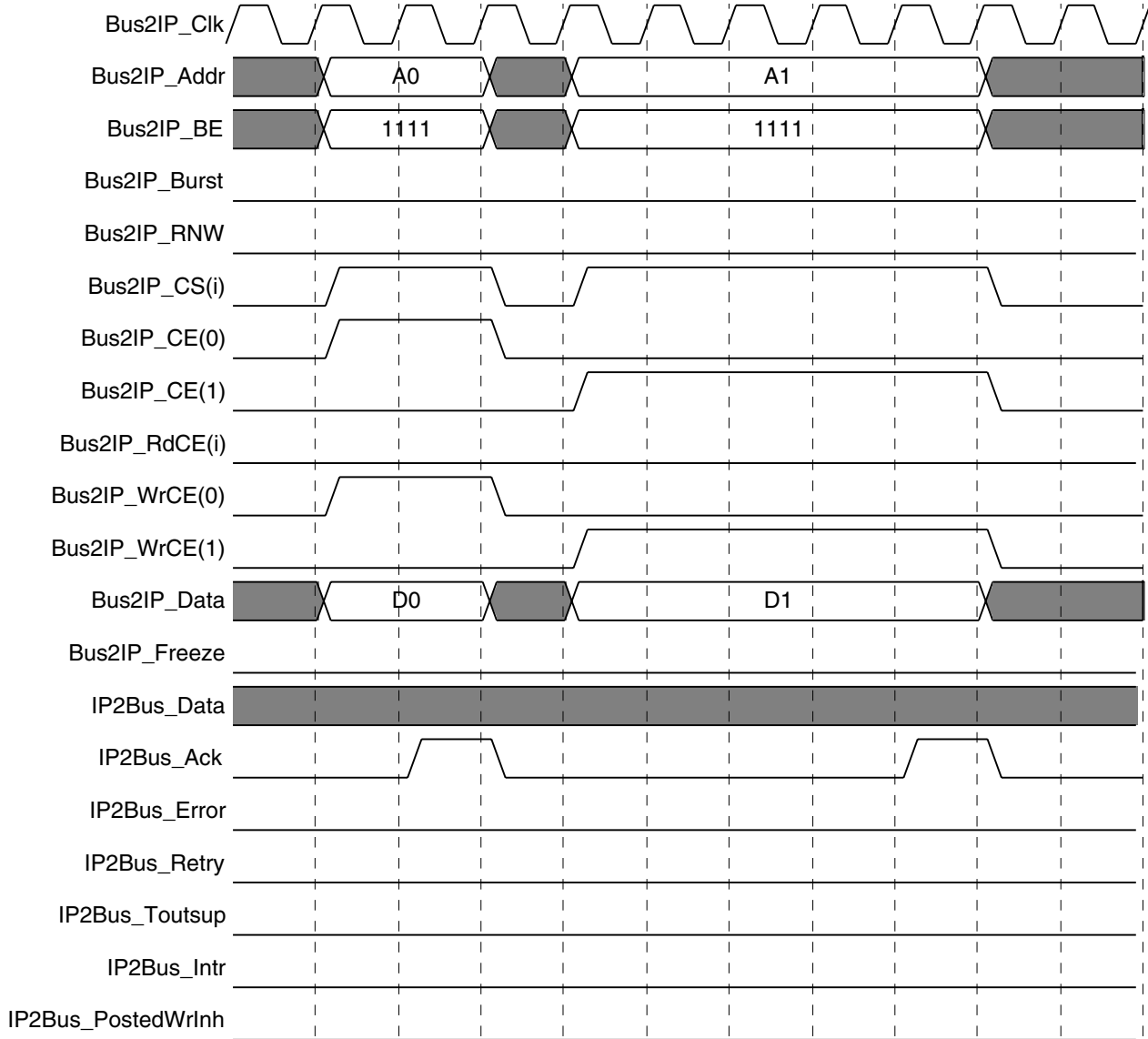


Figure 1-4: Slave Write – Single Beat

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 reference design; it will be available in future user core reference designs). The operation of the interface is the same: one data and acknowledge for each assertion of Bus2IP_CS.

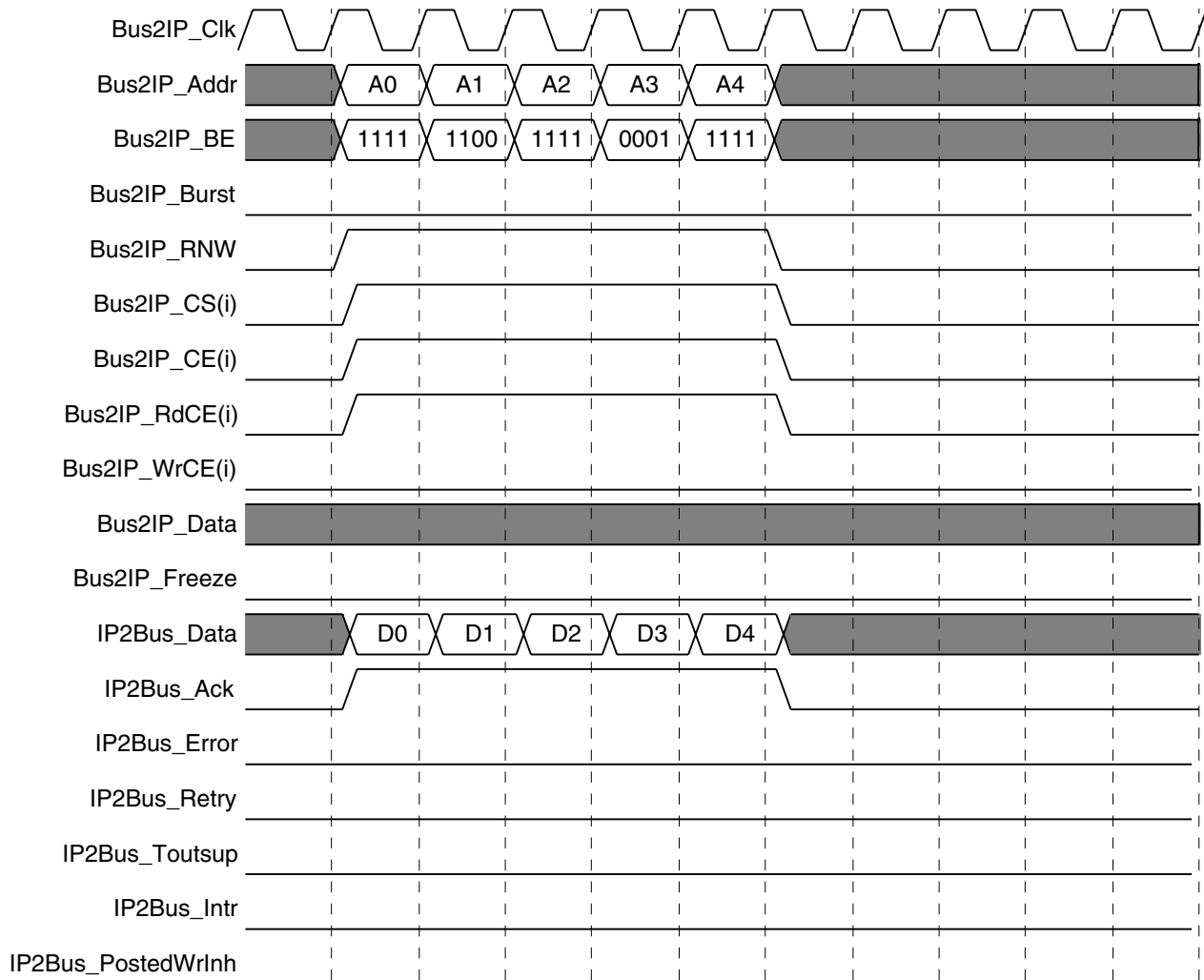


Figure 1-5: Slave Read – Single Beat Back to Back

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 reference design; it will be available in future user core reference designs). The operation of the interface is the same: one data and acknowledge for each assertion of Bus2IP_CS.

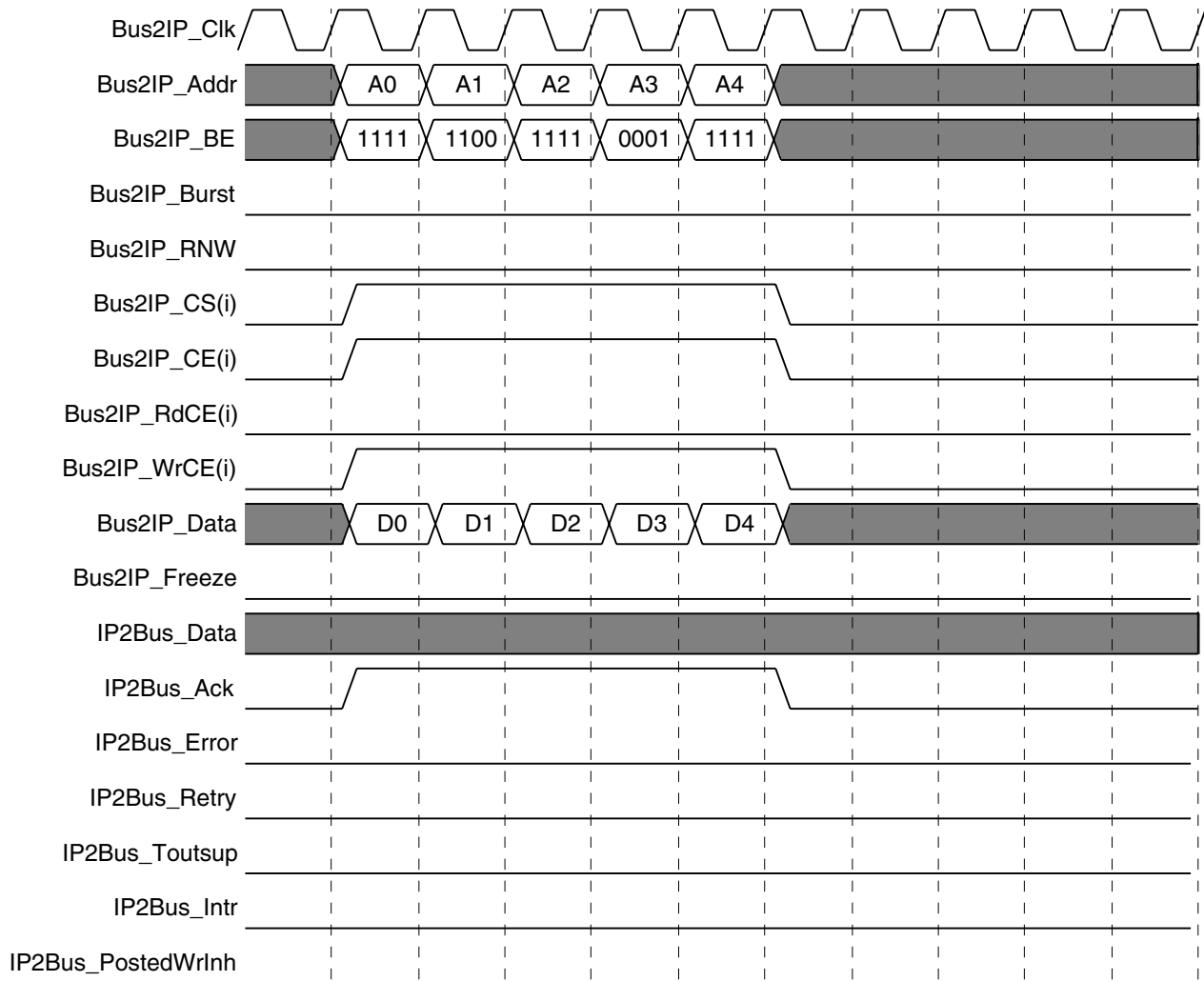


Figure 1-6: Slave Write – Single Beat Back to Back

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.

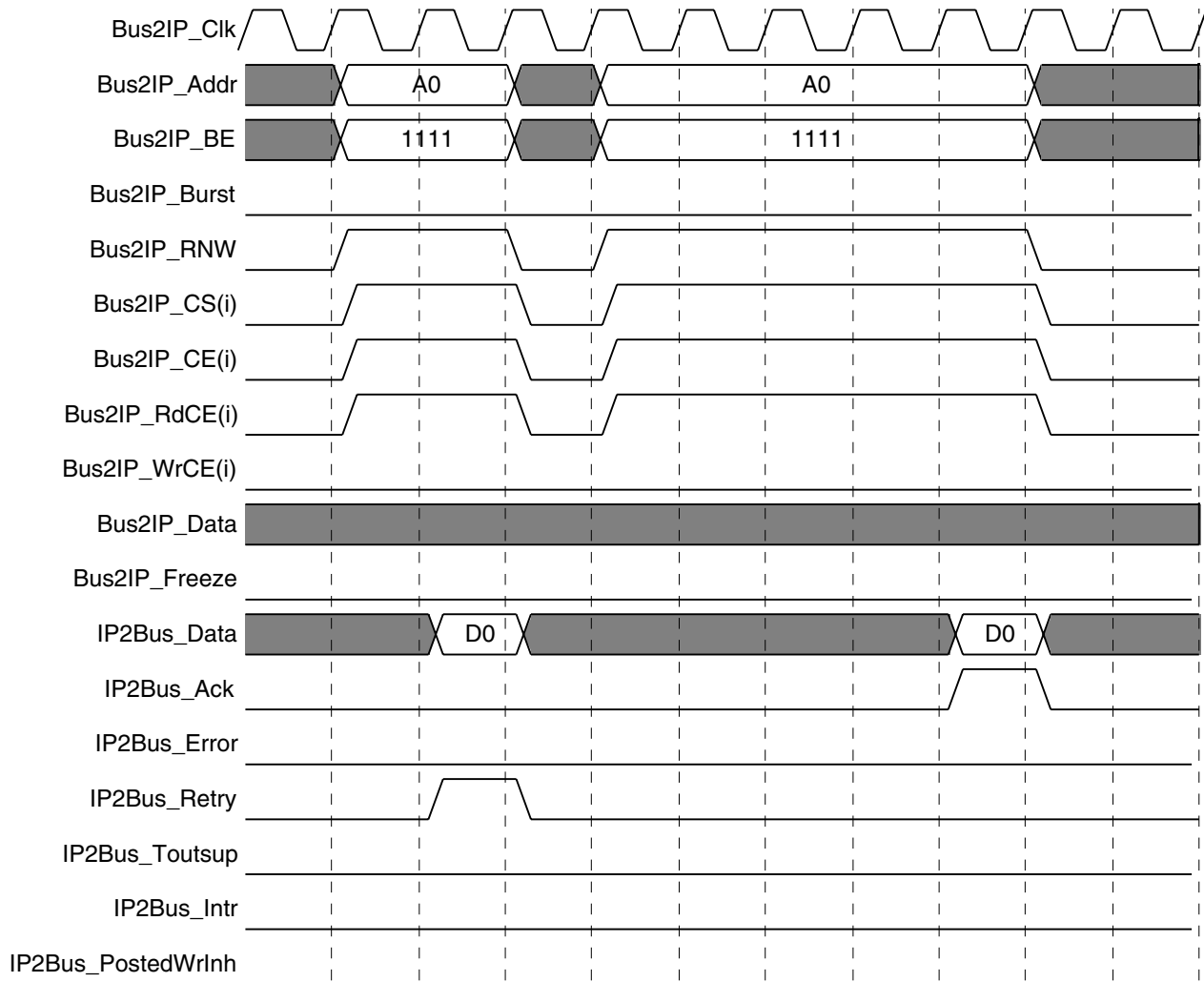


Figure 1-7: Slave Read – Single Beat with Retry

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.

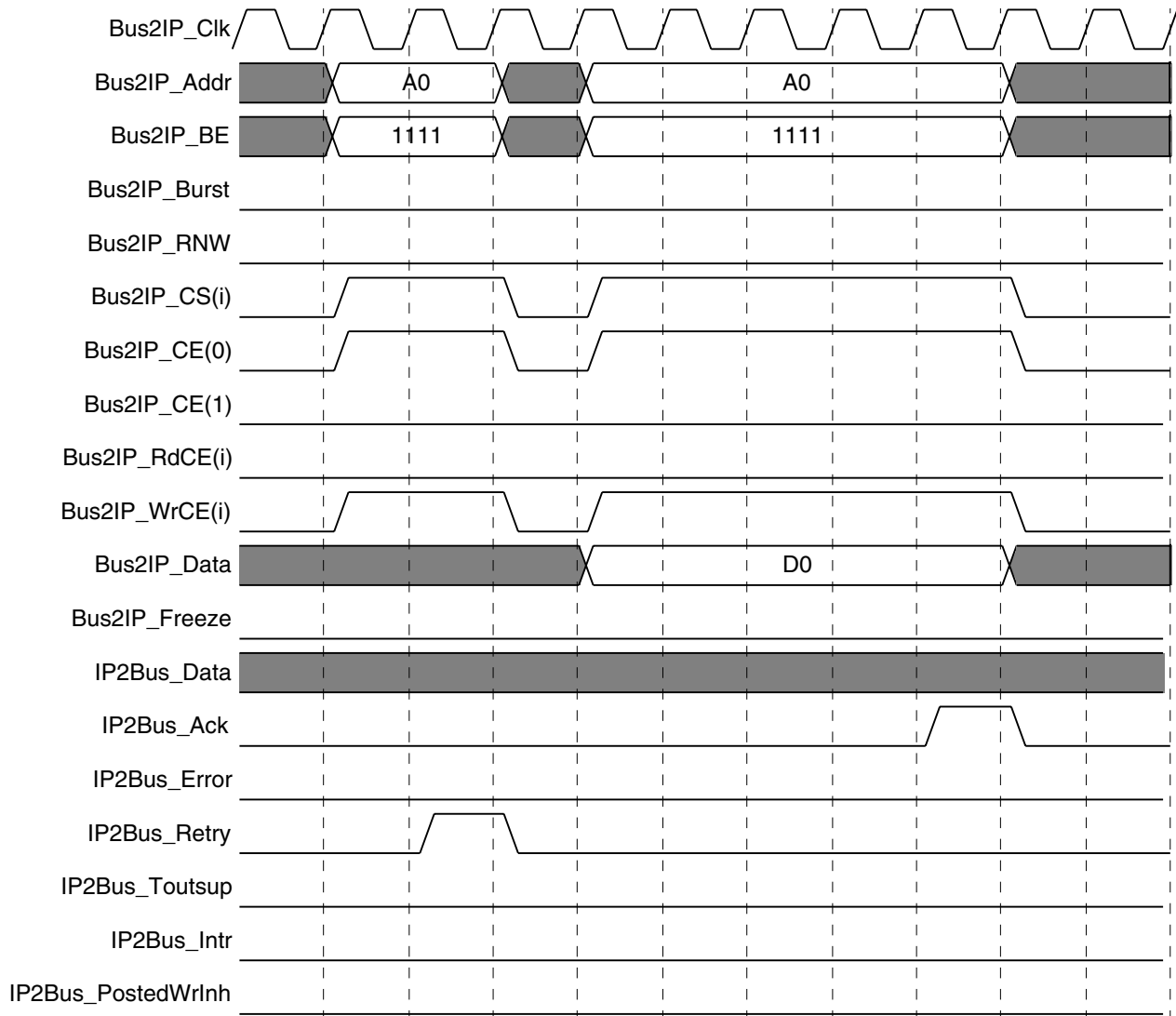


Figure 1-8: Slave Write – Single Beat with Retry

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.

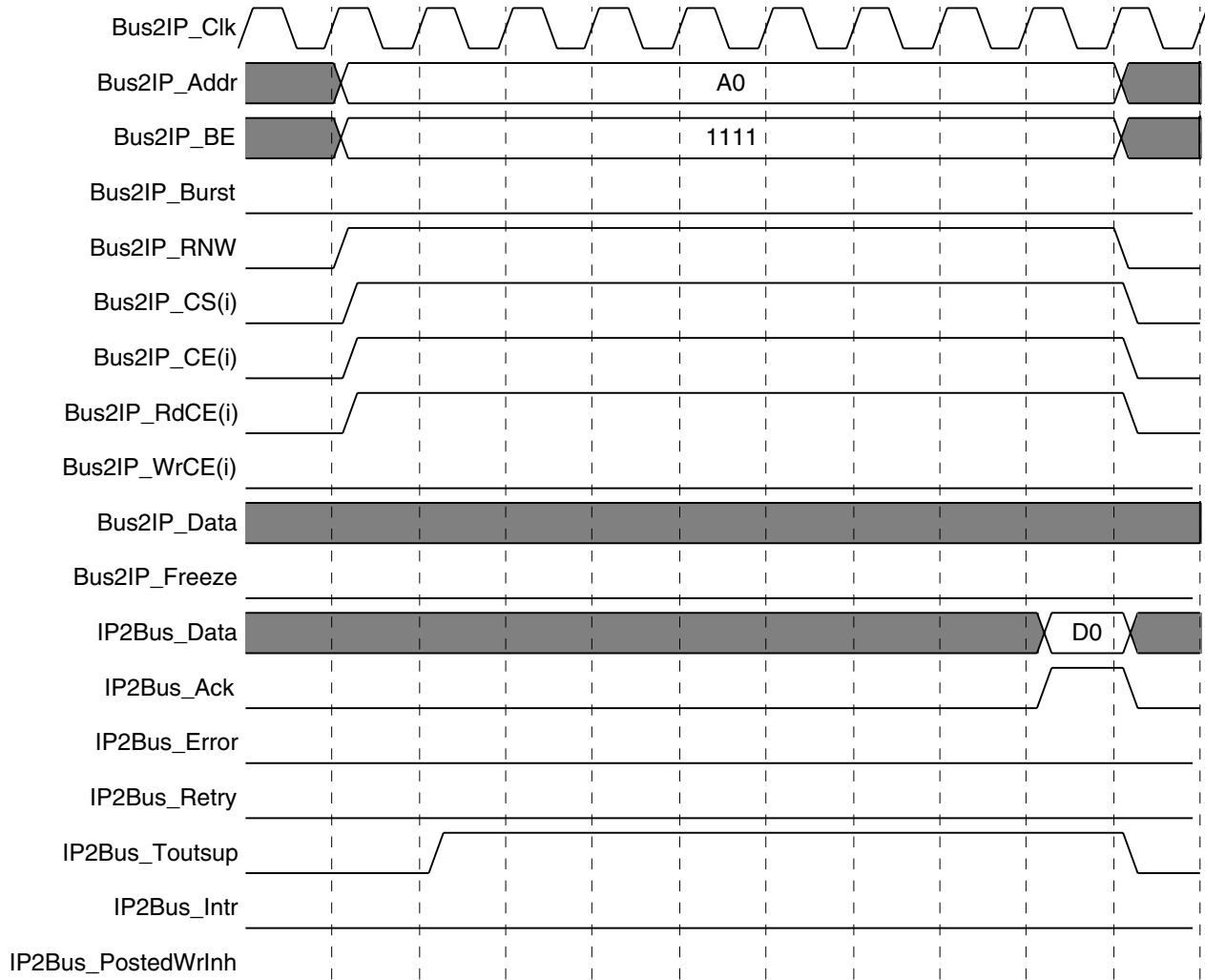


Figure 1-9: Slave Read – Single Beat with Timeout Suppress

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.

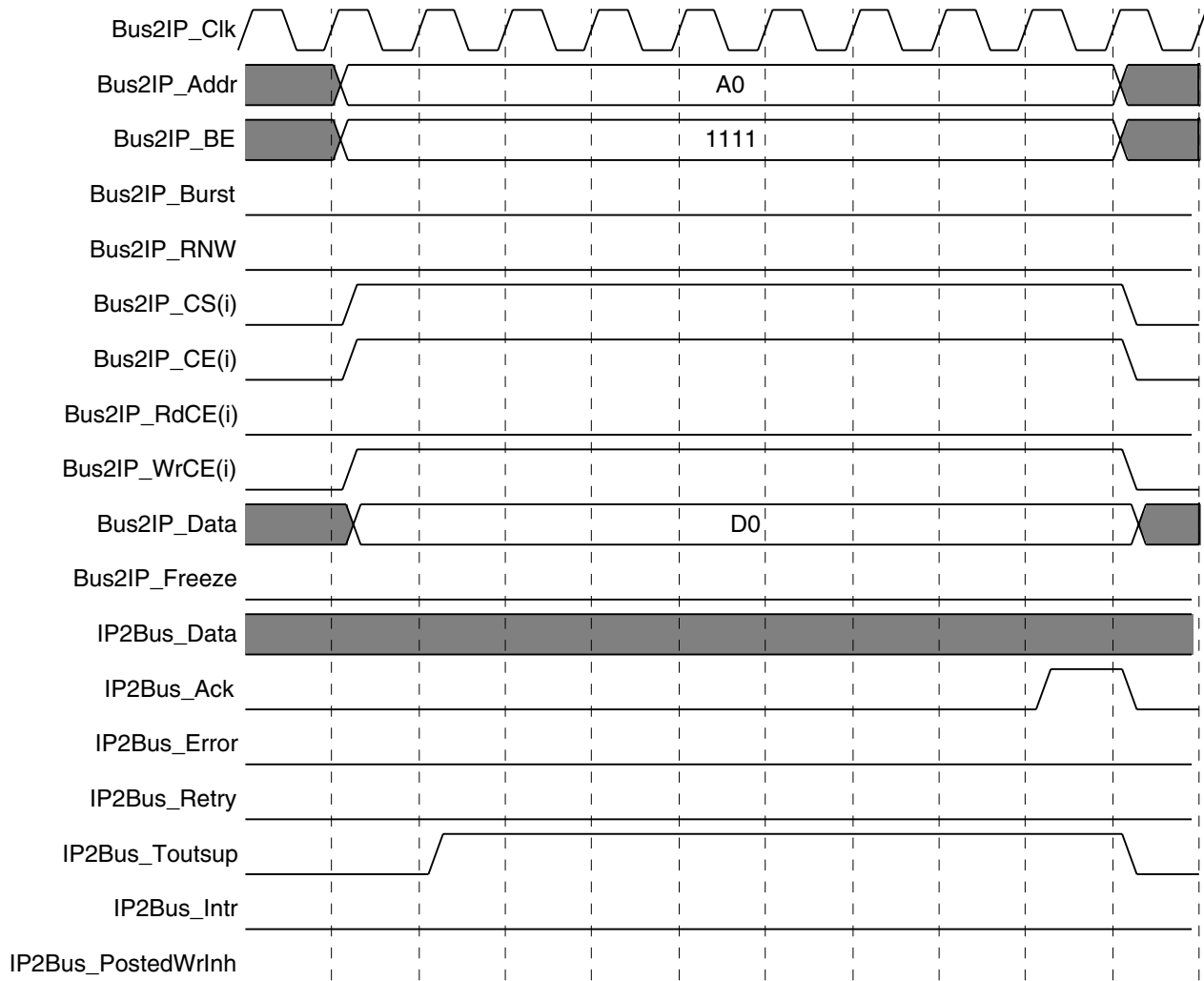


Figure 1-10: Slave Write – Single Beat with Timeout Suppress

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.

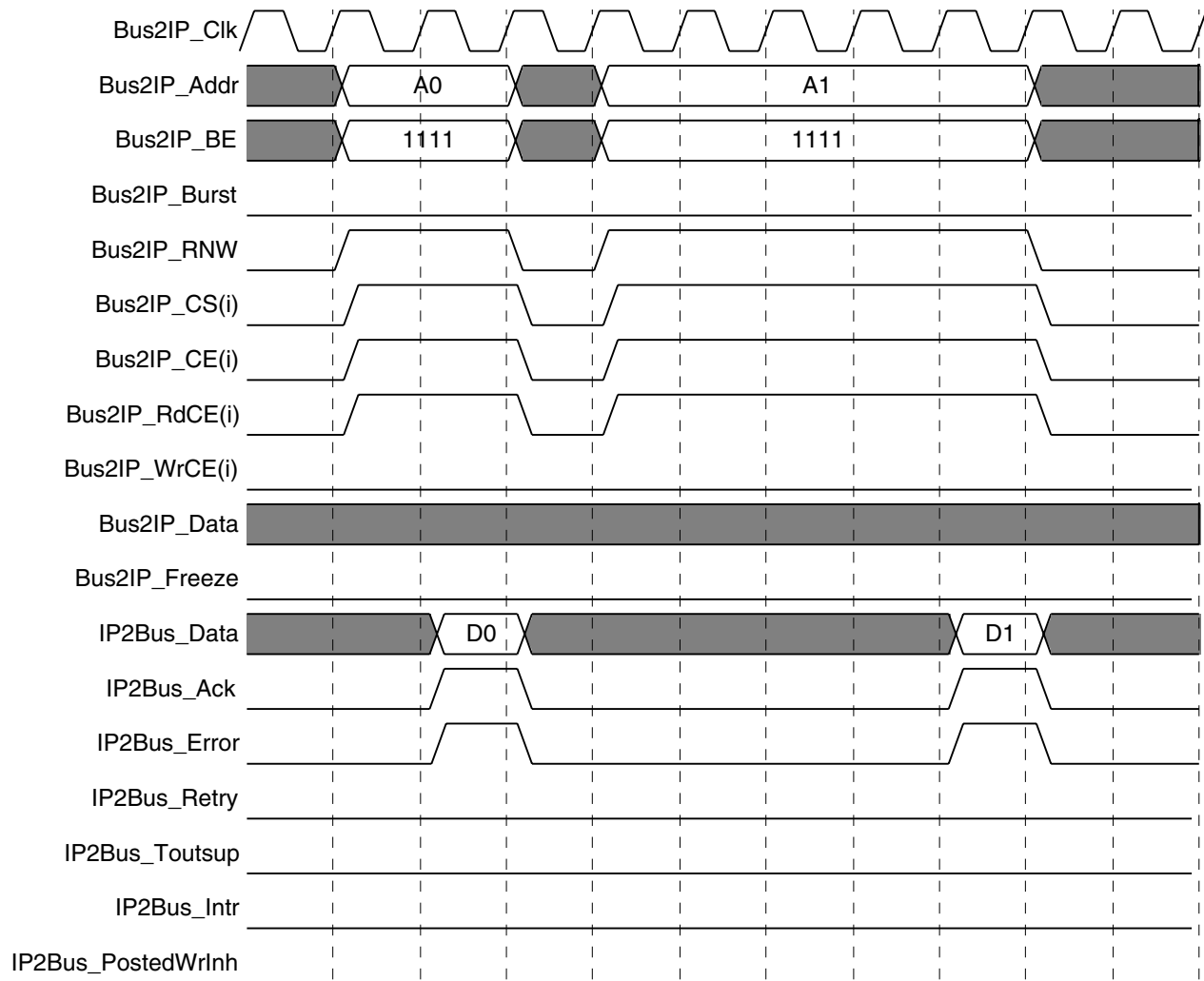


Figure 1-11: Slave Read – Single Beat with Error

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.

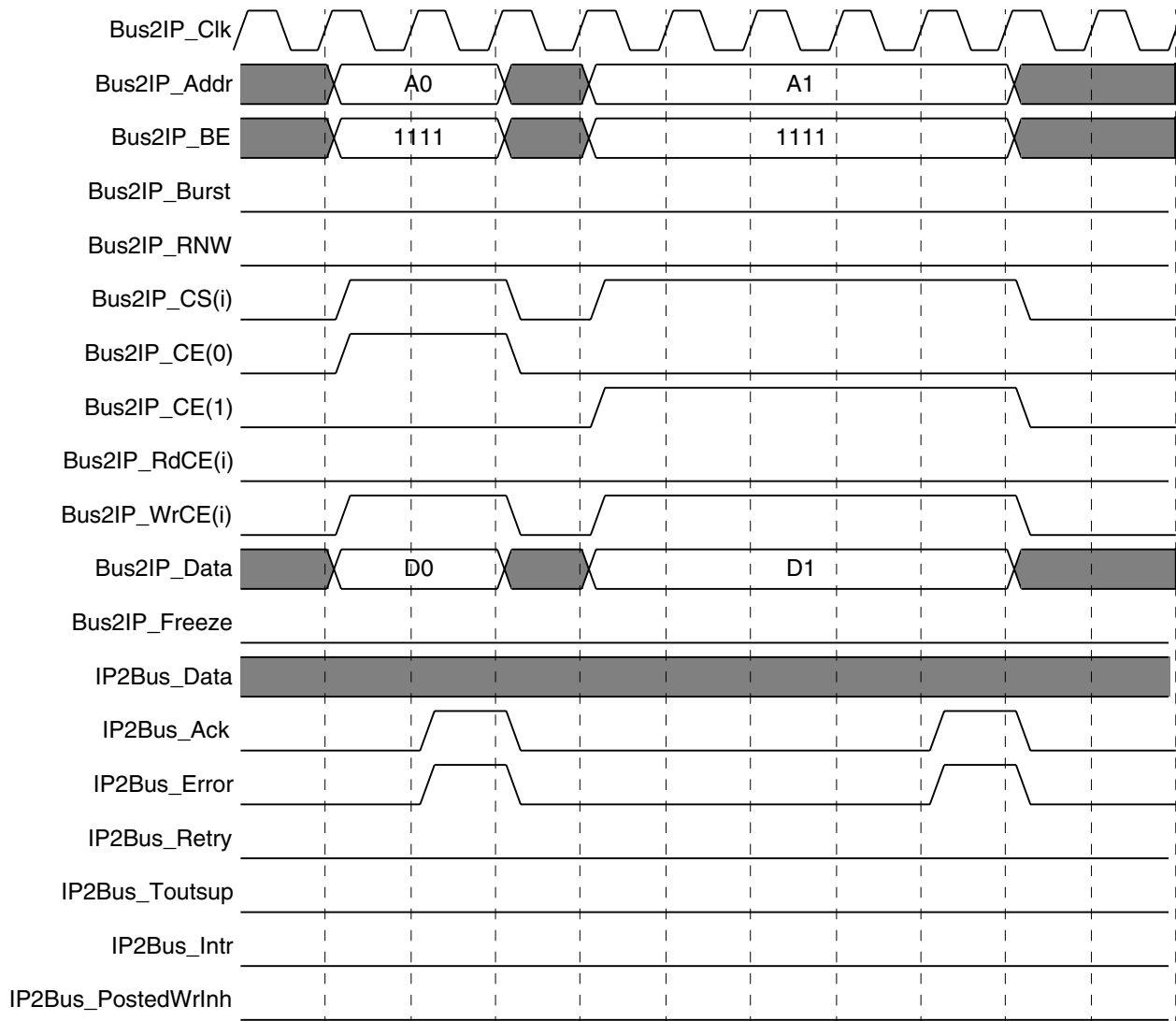


Figure 1-12: Slave Write – Single Beat with Error

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.

Note: The PLB Bus2IP_Burst signal will negate one clock cycle before the OPB Bus2IP_Burst signal shown below. It negates coincident with the last address of the burst (A0_28).

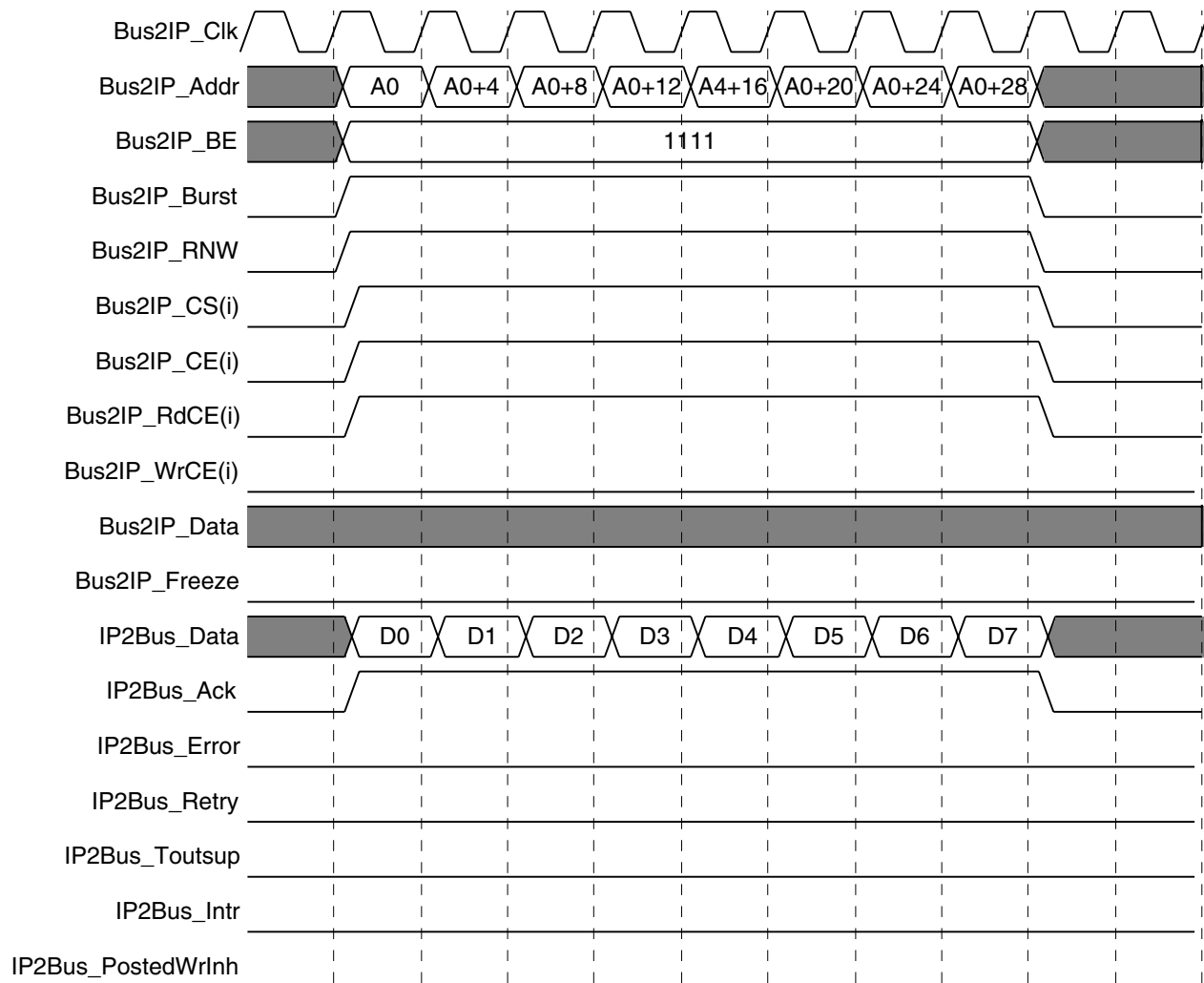


Figure 1-13: Slave Read – Burst Operation

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.

Note: The PLB Bus2IP_Burst signal will negate one clock cycle before the OPB Bus2IP_Burst signal shown below. It negates coincident with the last address of the burst (A0_28).

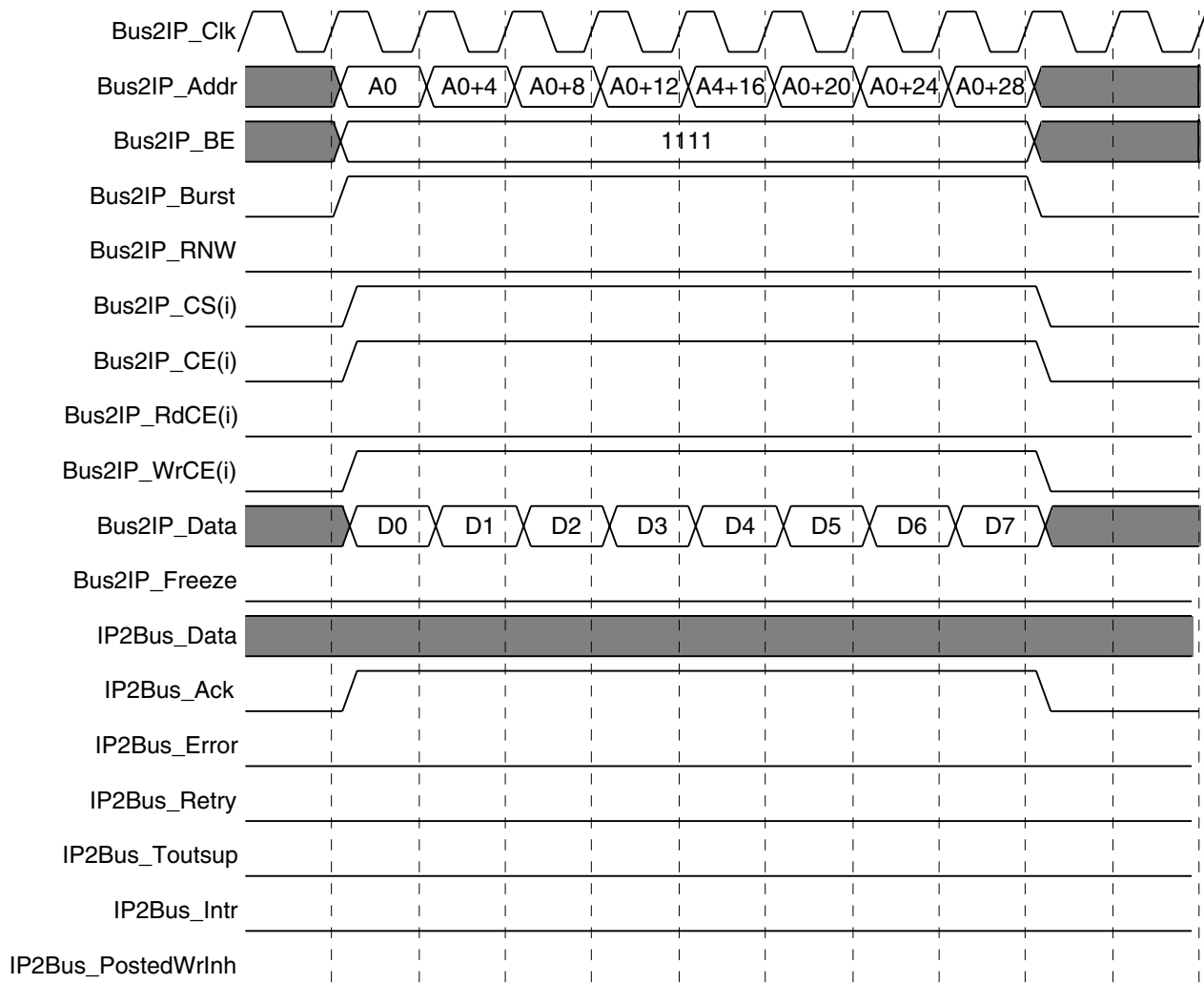


Figure 1-14: Slave Write – Burst Operations

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.

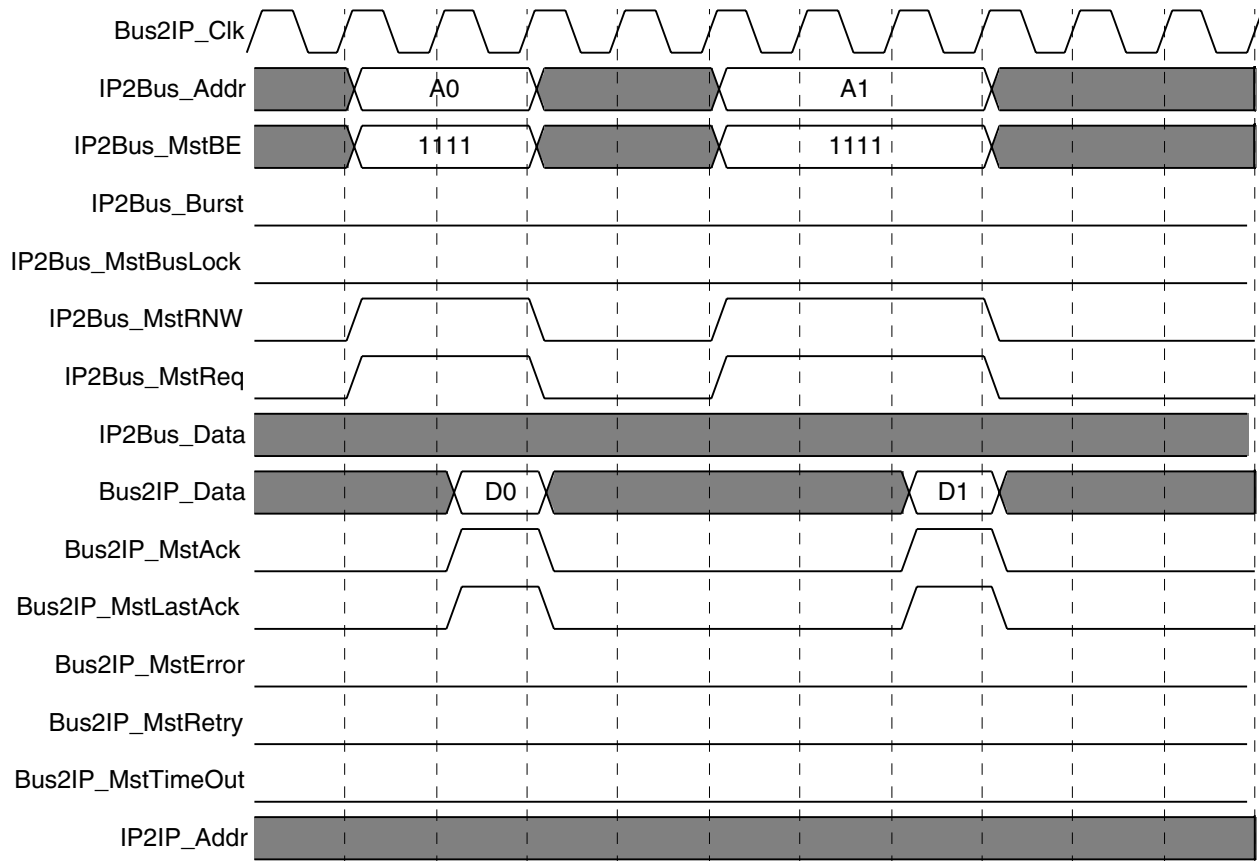


Figure 1-15: Master Read – Single Beat

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.

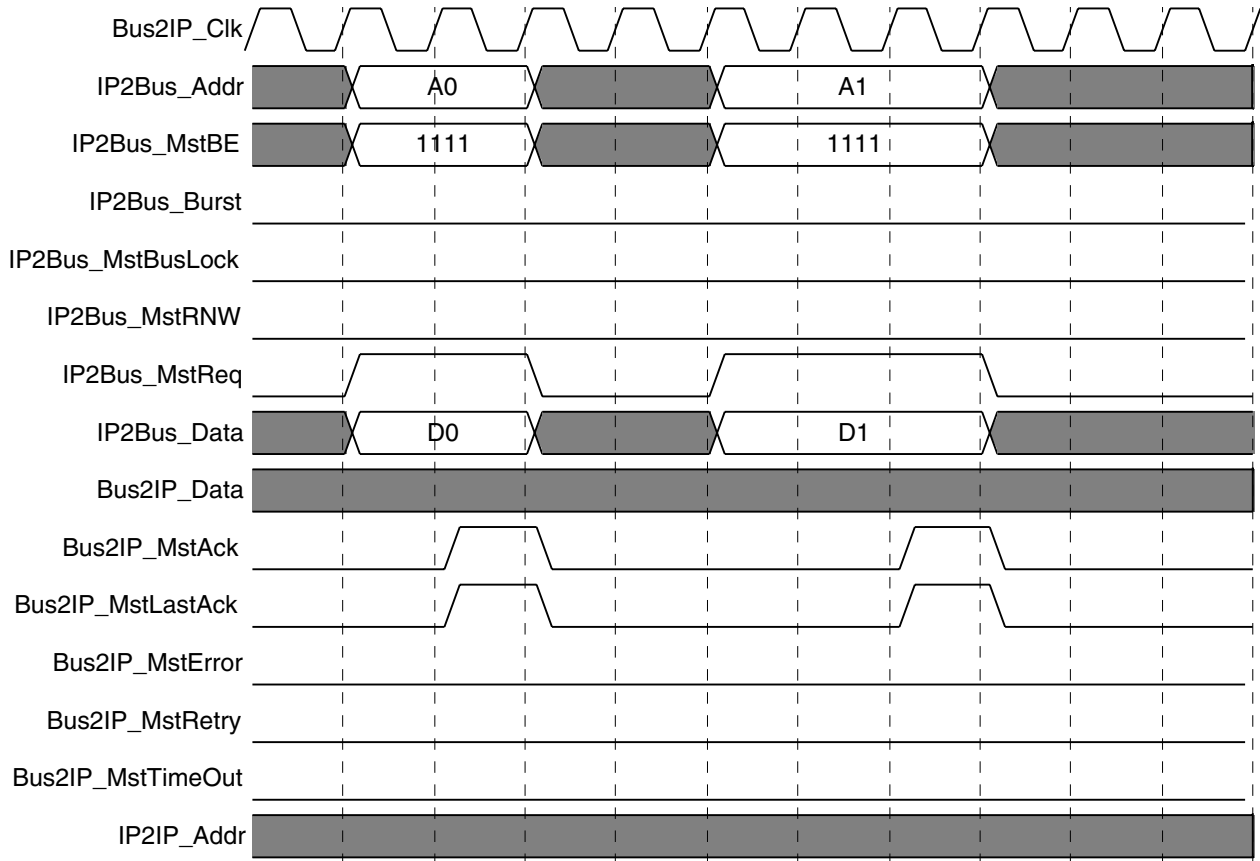


Figure 1-16: Master Write – Single Beat

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.

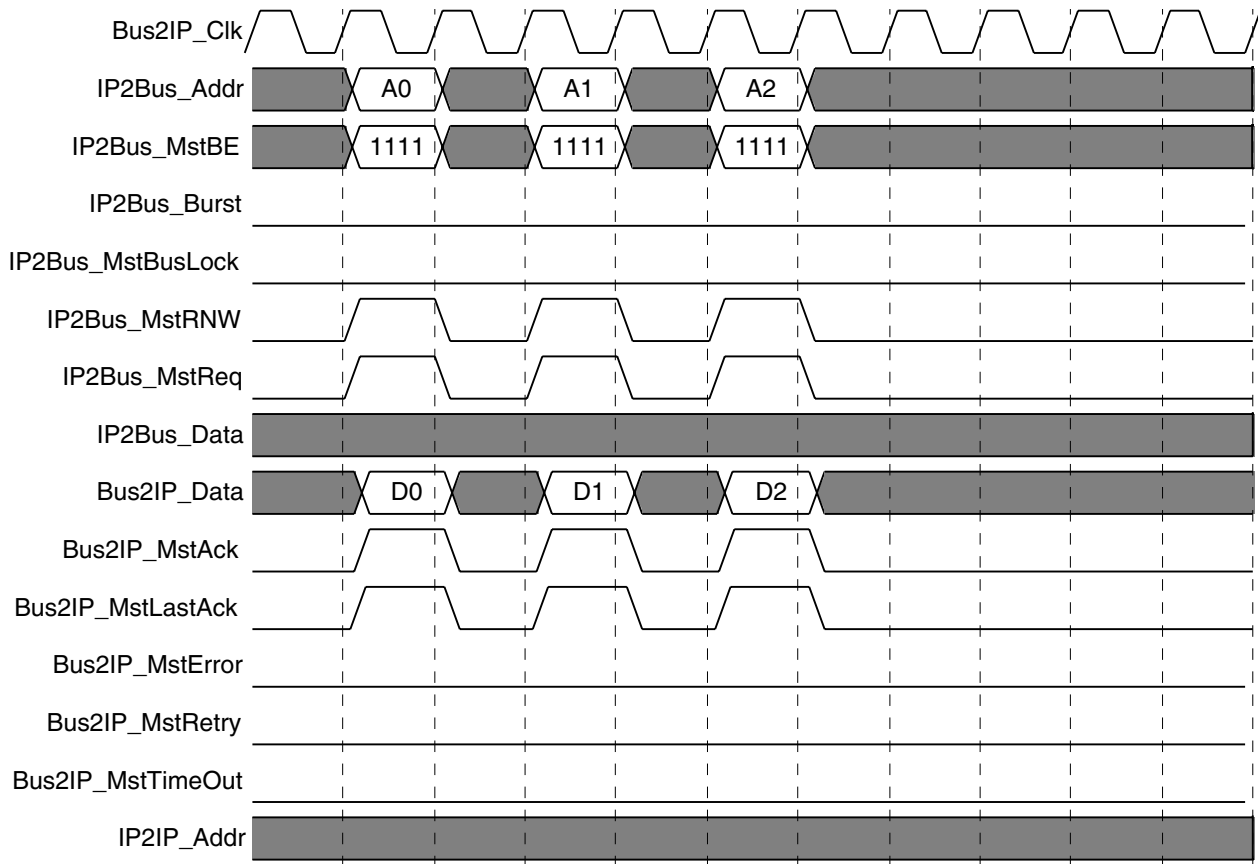


Figure 1-17: Master Read – Single Beat Back to Back

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.

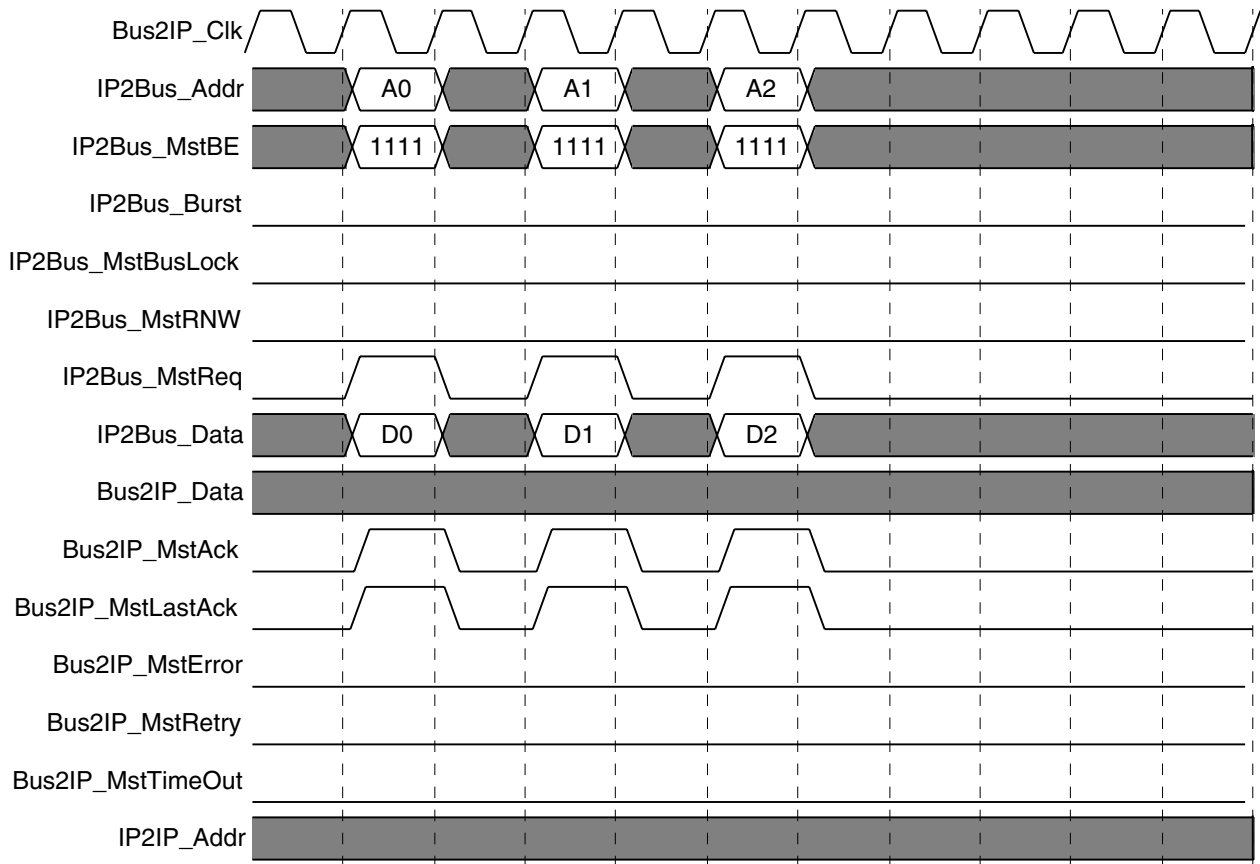


Figure 1-18: Master Write – Single Beat Back to Back

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.

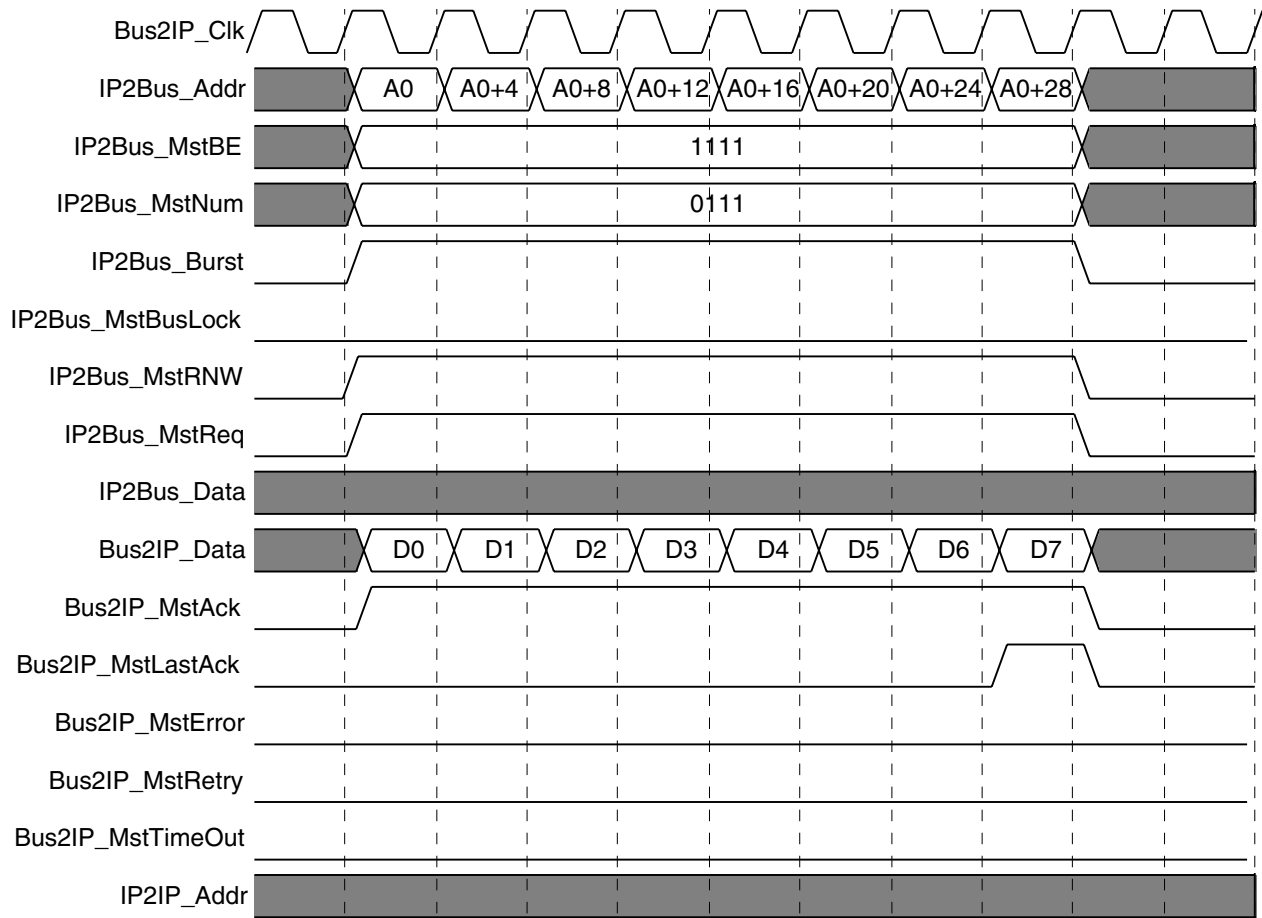


Figure 1-19: Master Read – Burst Operation

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.

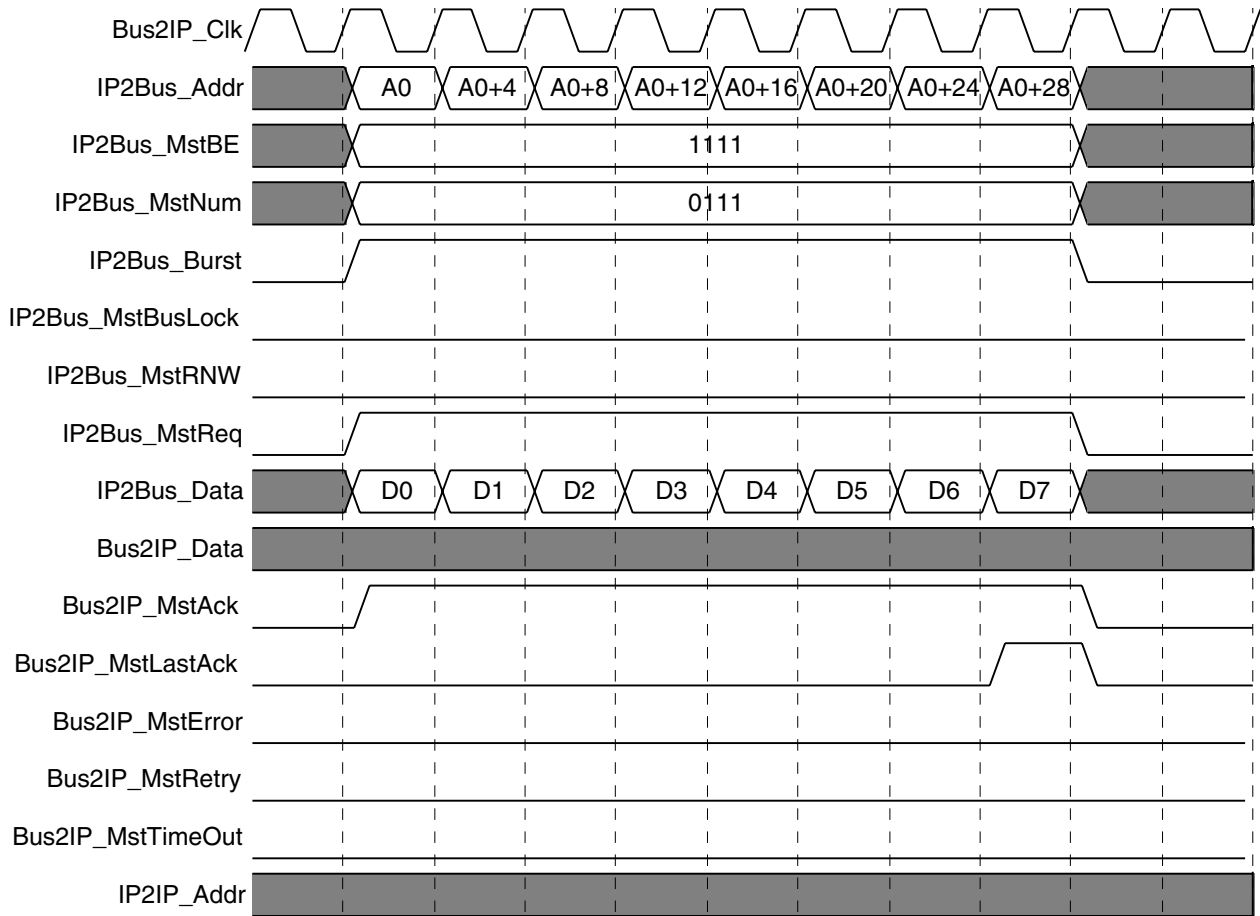


Figure 1-20: Master Write – Burst Operation

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 |
| 8/22/03 | 1.2 | Updated for PLB reference designs and EDK 6.1 |

