SYNOPSYS®

DesignWare® DW_axi_x2p

Databook

DW_axi_x2p – Product Code

2.04a
March 2020

Copyright Notice and Proprietary Information

Destination Control Statement

All technical data contained in this publication is subject to the export control laws of the United States of America. Disclosure to nationals of other countries contrary to United States law is prohibited. It is the reader's responsibility to determine the applicable regulations and to comply with them.

Disclaimer

SYNOPSYS, INC., AND ITS LICENSORS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Trademarks

Synopsys and certain Synopsys product names are trademarks of Synopsys, as set forth at https://www.synopsys.com/company/legal/trademarks-brands.html

All other product or company names may be trademarks of their respective owners.

Free and Open-Source Software Licensing Notices

If applicable, Free and Open-Source Software (FOSS) licensing notices are available in the product installation.

Third-Party Links

Any links to third-party websites included in this document are for your convenience only. Synopsys does not endorse and is not responsible for such websites and their practices, including privacy practices, availability, and content.

Synopsys, Inc.
www.synopsys.com

Contents

Revision History

This section tracks the significant documentation changes that occur from release-to-release and during a release from version 1.00c onward.

Date	Release	Description
2.04a	March 2020	■ Revision version change for 2020.03a release ■ Updated synthesis results in "Performance" on page 82 ■ Updated "Verification" on page 77 ■ Updated "BCM Library Components" on page 87 ■ Updated "Synchronization Depth" on page 34 ■ Rename Clock Adaption section to "Clocks and Resets" on page 34 ■ Rename Synchronizer chapter to "Basic Core Module (BCM) Library" on page 87 ■ "Signal Descriptions" on page 53, "Parameter Descriptions" on page 43, "Internal Parameter Descriptions" on page 75 auto-extracted from the RTL
2.03a	February 2018	■ Revision version change for 2018.02a release ■ Updated synthesis results in "Performance" on page 82 ■ Removed Chapter 2 Building and Verifying a Component or Subsystem from the Databook and added the contents in the newly created User Guide ■ "Signal Descriptions" on page 53, "Parameter Descriptions" on page 43, "Internal Parameter Descriptions" on page 75 auto-extracted from the RTL with change bars
2.02a	March 2016	■ Revision version change for 2016.03a release ■ Added "Running SpyGlass® Lint and SpyGlass® CDC" section ■ Added "Running Spyglass on Generated Code with coreAssembler" section ■ "Signal Descriptions" on page 53 and "Parameter Descriptions" on page 43 auto-extracted from the RTL ■ Added chapter "Internal Parameter Descriptions" on page 75 ■ Added Appendix A, "Basic Core Module (BCM) Library" ■ Updated area and power numbers in "Performance" on page 82

Date	Release	Description
2.01a	October 2014	■ Version change for 2014.10a release. ■ Updated "Integration Considerations" chapter. ■ Added the new X2P_ALLOW_SPARSE_TRANSFER parameter ■ Added the new section AXI-to-APB Sparse Transfers. ■ Made minor updates in APB Master Controller (X2PS) and AXI-to-APB Sparse Transfers sections to include the new X2P_ALLOW_SPARSE_TRANSFER parameter.
2.00a	June 2013	Added information about new and modified signals and parameters to support the AMBA 4 AXI and ACE-Lite specification.
1.08c	May 2013	Made a minor correction in the description of the X2P_APB_ADDR_WIDTH parameter. Corrected the penable signal in Figure 2-3 and Figure 2-8. Updated the template.
1.07b	Oct 2012	Added the product code on the cover and in Table 1-1.
1.07b	Oct 2011	Version change for 2011.10a release.
1.07a	Jun 2011	Updated system diagram in Figure 1-1; enhanced "Related Documents" section in Preface.
1.07a	Jan 2011	Version change for 2020.12a release.
1.06a	Nov 2010	Removed X2P_AXI_START_ADDR and X2P_AXI_END_ADDR parameters because they are now not visible; added X2P_START_PADDR_32_S*n* and X2P_END_PADDR_32_S*n*
1.05a	Sep 2010	Removed bullet at beginning of Chapter 1 saying that DW_axi_x2p supports APB master; corrected names of include files and vcs command used for simulation
1.04a	Apr 2010	Version change for 2010.04a release.
1.02b	Dec 2009	Updated databook to new template for consistency with other IIP/VIP/PHY databooks.
1.02b	May 2009	Removed references to QuickStarts, as they are no longer supported.
1.02b	Mar 2009	Reworded condition under which X2PS checks if the current APB word address is located within any of the mapped address pairs.
1.02a	Oct 2008	Version change for 2008.10a release.
1.01b	Jun 2008	Version change for 2008.06a release.
1.01a	Oct 2007	Version change for 1.01a release.
1.00c	Jun 2007	Version change for 2007.06a release.

Preface

This databook provides information about the DesignWare Advanced eXtensible Interface AXI-to-APB bridge (DW_axi_x2p). This component conforms to the AMBA 3 AXI and AMBA 4 AXI specifications defined in the *AMBA AXI and ACE Protocol Specification* from ARM.

The information in this databook includes a functional description, signal and parameter descriptions, as well as information on how you can configure, create RTL for, simulate, and synthesize the component using Synopsys coreConsultant. This manual also provides an overview of the component testbench, a description of the tests that are run to verify the coreKit, and synthesis information for the coreKit.

Organization

The chapters of this databook are organized as follows:

- Chapter 1, "Product Overview" provides a system overview, a component block diagram, basic features, and an overview of the verification environment.
- Chapter 2, "Functional Description" describes the functional operation of the DW_axi_x2p.
- Chapter 3, "Parameter Descriptions" identifies the configurable parameters supported by the DW_axi_x2p.
- Chapter 4, "Signal Descriptions" provides a list and description of the DW_axi_x2p signals.
- Chapter 5, "Internal Parameter Descriptions" provides a list of internal parameter descriptions that might be indirectly referenced in expressions in the Signals and Parameters chapters.
- Chapter 6, "Verification" provides information on verifying the configured DW_axi_x2p.
- Chapter 7, "Integration Considerations" includes information you need to integrate the configured DW_axi_x2p into your design.
- Appendix A, "Basic Core Module (BCM) Library" documents the synchronizer methods (blocks of synchronizer functionality) used in DW_axi_x2p to cross clock boundaries.
- Appendix B, "Glossary" provides a glossary of general terms.

Related Documentation

- *Using DesignWare Library IP in coreAssembler* – Contains information on getting started with using DesignWare SIP components for AMBA 2 and AMBA 3 AXI/AMBA 4 AXI components within coreTools
- *coreAssembler User Guide* – Contains information on using coreAssembler
- *coreConsultant User Guide* – Contains information on using coreConsultant

To see a complete listing of documentation within the DesignWare Synthesizable Components for AMBA3 AXI, refer to the *Guide to Documentation for DesignWare Synthesizable Components for AMBA 2, AMBA 3 AXI and AMBA 4 AXI*.

Web Resources

- DesignWare IP product information: https://www.synopsys.com/designware-ip.html
- Your custom DesignWare IP page: https://www.synopsys.com/dw/mydesignware.php
- Documentation through SolvNetPlus: https://solvnetplus.synopsys.com (Synopsys password required)
- Synopsys Common Licensing (SCL): https://www.synopsys.com/keys

Customer Support

Synopsys provides the following various methods for contacting Customer Support:

- Prepare the following debug information, if applicable:
  - For environment set-up problems or failures with configuration, simulation, or synthesis that occur within coreConsultant or coreAssembler, select the following menu:

    File > Build Debug Tar-file

    Check all the boxes in the dialog box that apply to your issue. This option gathers all the Synopsys product data needed to begin debugging an issue and writes it to the <*core tool startup directory*>/debug.tar.gz file.
  - For simulation issues outside of coreConsultant or coreAssembler:
    - Create a waveforms file (such as VPD or VCD).
    - Identify the hierarchy path to the DesignWare instance.
    - Identify the timestamp of any signals or locations in the waveforms that are not understood.
- *For the fastest response*, enter a case through SolvNetPlus:

  a. https://solvnetplus.synopsys.com

  Note SolvNetPlus does not support Internet Explorer. Use a supported browser such as Microsoft Edge, Google Chrome, Mozilla Firefox, or Apple Safari.

  b. Click the **Cases** menu and then click **Create a New Case** (below the list of cases).

  c. Complete the mandatory fields that are marked with an asterisk and click **Save**.

  Ensure to include the following:
  - **Product L1:** DesignWare Library IP
  - **Product L2:** <name of L2>

  d. After creating the case, attach any debug files you created.

  For more information about general usage information, refer to the following article in SolvNetPlus:

  https://solvnetplus.synopsys.com/s/article/SolvNetPlus-Usage-Help-Resources

- Or, send an e-mail message to support_center@synopsys.com (your email will be queued and then, on a first-come, first-served basis, manually routed to the correct support engineer):
  - Include the Product L1 and Product L2 names, and Version number in your e-mail so it can be routed correctly.
  - For simulation issues, include the timestamp of any signals or locations in waveforms that are not understood
  - Attach any debug files you created.
- Or, telephone your local support center:
  - North America:
    Call 1-800-245-8005 from 7 AM to 5:30 PM Pacific time, Monday through Friday.
  - All other countries:
    https://www.synopsys.com/support/global-support-centers.html

Product Code

Table 1-1 lists all the components associated with the product code for DesignWare AMBA Fabric.

Table 1-1 DesignWare AMBA Fabric – Product Code: 3768-0

Component Name	Description
DW_ahb	High performance, low latency interconnect fabric for AMBA 2 AHB
DW_ahb_eh2h	High performance, high bandwidth AMBA 2 AHB to AHB bridge
DW_ahb_h2h	Area efficient, low bandwidth AMBA 2 AHB to AHB Bridge
DW_ahb_icm	Configurable multi-layer interconnection matrix
DW_ahb_ictl	Configurable vectored interrupt controllers for AHB bus systems
DW_apb	High performance, low latency interconnect fabric & bridge for AMBA 2 APB for direct connect to AMBA 2 AHB fabric
DW_apb_ictl	Configurable vectored interrupt controllers for APB bus systems
DW_axi	High performance, low latency interconnect fabric for AMBA 3 AXI and AMBA 4 AXI
DW_axi_a2x	Configurable bridge between AMBA 3 AXI/AMBA 4 AXI components and AHB components or between AMBA 3 AXI/AMBA 4 AXI components and AMBA 3 AXI/AMBA 4 AXI components.
DW_axi_gm	Simplify the connection of third party/custom master controllers to any AMBA 3 AXI or AMBA 4 AXI fabric
DW_axi_gs	Simplify the connection of third party/custom slave controllers to any AMBA 3 AXI or AMBA 4 AXI fabric
DW_axi_hmx	Configurable high performance interface from an AHB master to an AMBA 3 AXI or AMBA 4 AXI slave
DW_axi_rs	Configurable standalone pipelining stage for AMBA 3 AXI or AMBA 4 AXI subsystems

Table 1-1 DesignWare AMBA Fabric – Product Code: 3768-0 (Continued)

Component Name	Description
DW_axi_x2h	Bridge from AMBA 3 AXI or AMBA 4 AXI to AMBA 2.0 AHB, enabling easy integration of legacy AHB designs with newer AXI systems
DW_axi_x2p	High performance, low latency interconnect fabric and bridge for AMBA 2 & 3 APB for direct connect to AMBA 3 AXI or AMBA 4 AXI fabric
DW_axi_x2x	Flexible bridge between multiple AMBA 3 AXI components or busses

1 Product Overview

The DW_axi_x2p is an AXI-to-APB bridge that connects an AMBA 3 AXI/AMBA 4 AXI-compliant master to an AMBA 2-compliant or AMBA 3-compliant APB slave. Essentially, the DW_axi_x2p connects an AMBA 2/AMBA 3 APB slave to one of the following AMBA 3 AXI/AMBA4 AXI components:

- AXI master
- AXI interconnect

Note You must have a DWC-AMBA-Fabric-Source license to enable the AXI4 interface.

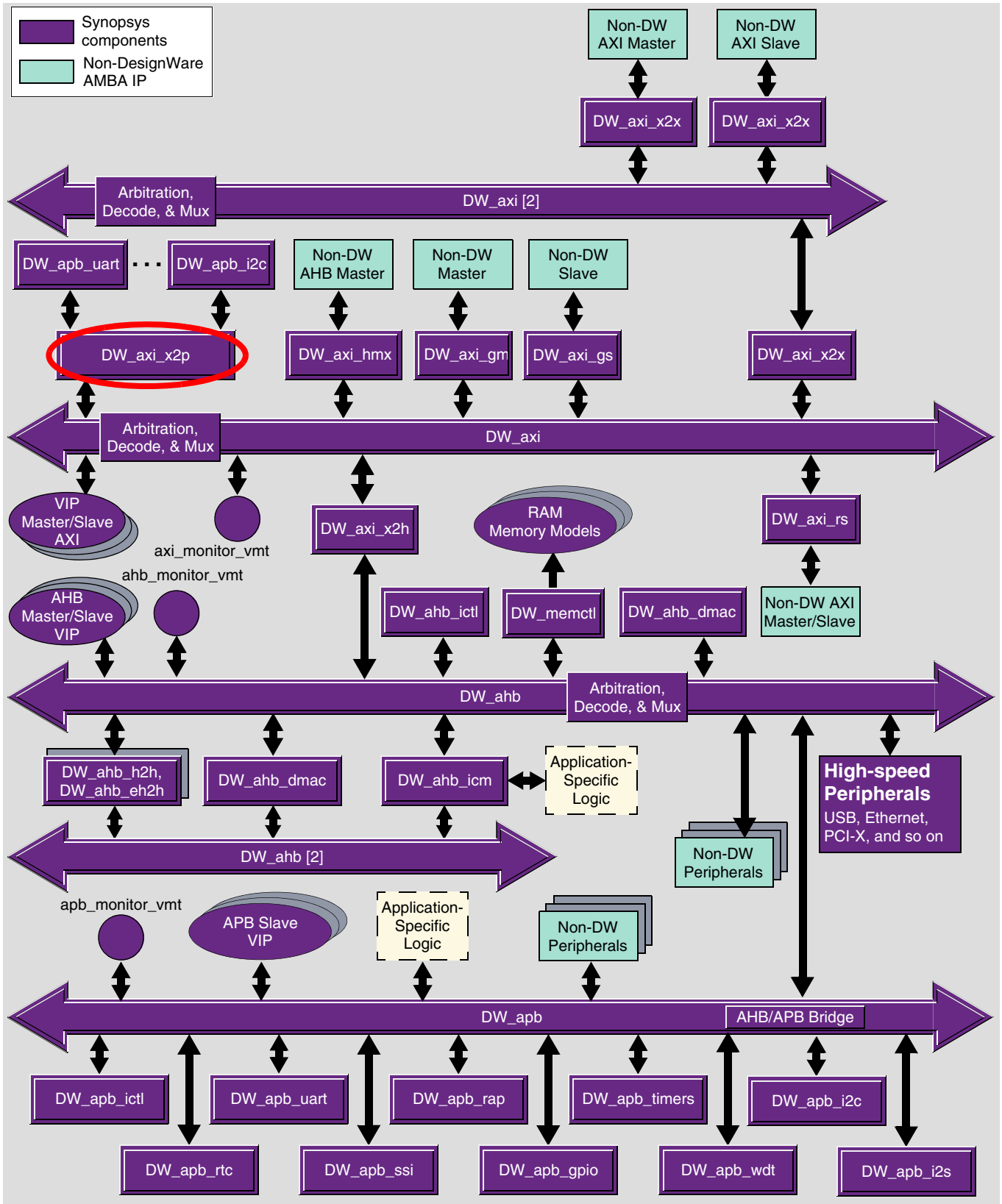
1.1 DesignWare Synthesizable Components for AMBA System Overview

The Synopsys DesignWare Synthesizable Components environment is a parameterizable bus system containing AMBA version 2.0-compliant AHB (Advanced High-performance Bus) and APB (Advanced Peripheral Bus) components, and AMBA 3 AXI/AMBA 4 AXI (Advanced eXtensible Interface) components.

Figure 1-1 illustrates one example of this environment, including the AXI bus, the AHB bus, and the APB bus. Included in this subsystem are synthesizable IP for AXI/AHB/APB peripherals, bus bridges, and an AXI interconnect and AHB bus fabric. Also included are verification IP for AXI/AHB/APB master/slave models and bus monitors. In order to display the databook for a DW_* component, click on the corresponding component object in the illustration.

Attention Links resolve only if you are viewing this databook from your $DESIGNWARE_HOME tree, and to only those components that are installed in the tree.

Figure 1-1 Example of DW_axi_x2p Bridge in a Complete System



1.2 General Product Description

The DW_axi_x2p is an AXI-to-APB bridge that allows the connection of an APB slave to an AMBA 3 AXI/AMBA 4 AXI-compliant master or interconnect. APB components attached to the DW_axi_x2p can be either AMBA 2-compliant or AMBA 3-compliant. In other words, they can include either or both of the signals, pready and pslverr, added by the AMBA 3 APB protocol.

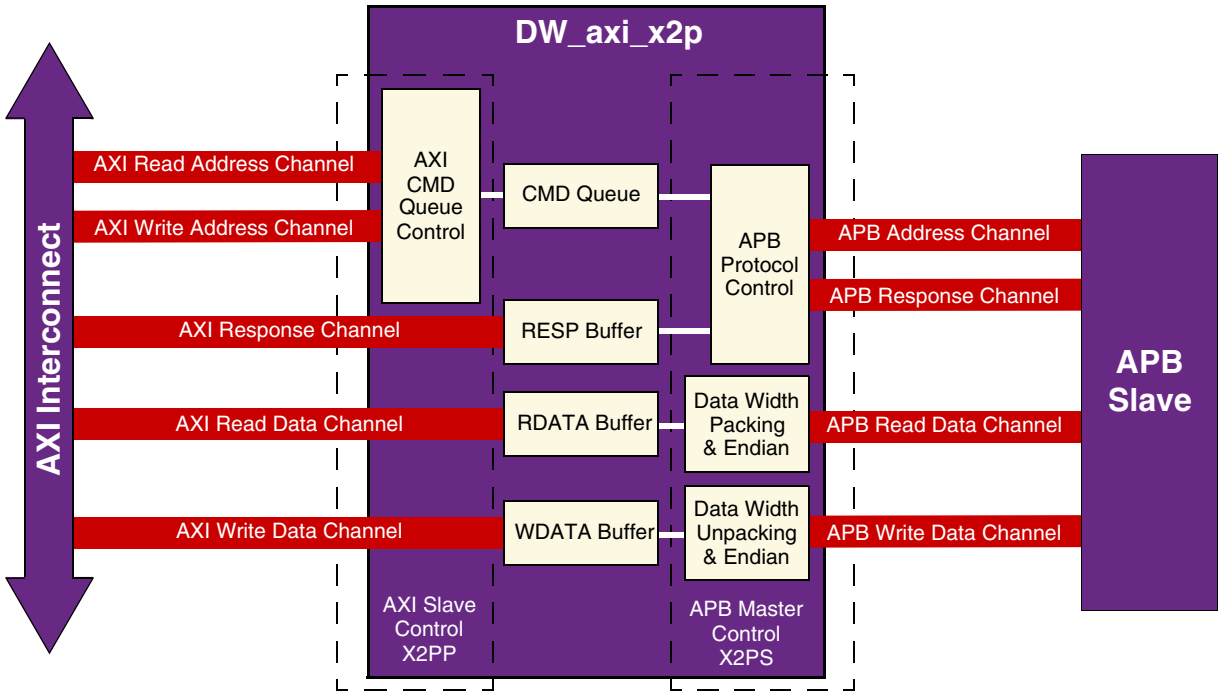
One side of the DW_axi_x2p — the primary or X2PP — is an AXI slave interface, which accepts reads and writes from an attached AXI master interface. The other side of the DW_axi_x2p — the secondary or X2PS — is an APB master interface, which provides reads and writes to attached APB slave components.

The DW_axi_x2p is a configurable component, allowing the IP integrator to include or remove logic to suit specific design requirements.

1.2.1 DW_axi_x2p Block Diagram

The general block diagram of the DW_axi_x2p is shown in Figure 1-2.

Figure 1-2 DW_axi_x2p Block Diagram



1.3 Features

The DW_axi_x2p supports the following features:

- Complies with the following specifications:
  - AMBA 3 AXI
  - AMBA 4 AXI
- Translates AXI transactions into APB transfers

  - Configuration option for AMBA 2 (APB) compatibility
  - Accepts simultaneous read/write AXI transactions; passes all AXI transactions to APB to execute one APB transfer at a time
  - Provides AXI-to-APB data congruity checks; automatically avoids APB writes with incompatible address alignment or write strobes only if X2P_ALLOW_SPARSE_TRANSFER is set to 0.
- Flexible address and data port configurations
  - AXI data ports: 8, 16, 32, 64, 128, 256, or 512 bits wide
  - APB data ports: 8, 16, or 32 bits wide
  - AXI address ports: 32 or 64 bits
  - APB address ports: 8,16, or 32 bits
  - Byte invariant endianness support — AXI can be big-endian or little-endian; APB is always little-endian
- Supports up to sixteen AMBA 2 APB slaves
- Support for different clock domains; single clock or two asynchronous clocks
- Buffers AXI transactions
  - Buffers activity on all of the AXI channels, minimizing channel stalling on AXI control, data, and response
  - Supports a wide range of user-selectable depths for the command queues, response buffer, read data, and write data buffers
- Low-power interface are:
  - Transaction activity status to an external low-power controller (LPC) for enabling or disabling the clock.
  - Configurable maximum number of clock cycles the system must remain idle before entering low-power mode.

1.4 Restrictions

The DW_axi_x2p has the following restrictions or limitations:

- No support for AXI atomic operations (exclusive and locked accesses)
- Interleaving not supported; AXI interleave depth set to 1 for both reads and writes
- No Out-of-order transaction completion
- All APB slave addresses aligned to 1 KB boundary
- Minimum address space of 1 KB allocated to an APB slave
- APB data port width must be configured to be less than or equal to the AXI data port width
- If X2P_ALLOW_SPARSE_TRANSFER is set to 0, data in writes from the AXI must be in contiguous increments of the APB bus size and AXI write strobes that correspond to the APB word contents must be either all on or all off. Otherwise, an error is returned on the write response (SLVERR).

- AXI read addressing that is unaligned to the APB word address is accepted by the X2P, but returned as an error
- AXI transactions with a size less than the APB data bus width are not issued on the APB and returned to the AXI flagged with an error response (SLVERR)
- AXI subsystem does not receive a response for a write until the APB completes writing all the AXI data.
- APB decoding is fixed to 32-bit addresses, regardless of configured APB address width or AXI address width
- When the AXI4 interface is enabled, the following features are not applicable:
  - QoS
  - Multiple region
  - User-defined signaling

1.5 DW_axi_x2p Terminology

The following terms are used throughout this databook.

- **Active read/write command** – A command that has been generated by an AXI master and accepted by an AXI slave but has not completed.

  A read command completes when the last data beat of the read burst completes; that is, when the AXI slave asserts the rlast and rvalid signals, and the AXI master asserts the rready signal.

  A write command completes when the AXI slave returns the write response and it is accepted by the AXI master; for instance, when the AXI slave asserts the bvalid signal, and the AXI master asserts the bready signal.
- **Beat** – A single data transfer.
- **Burst** – A number of data transfers in a single transaction.
- **AXI transfer** – A single sequence initiated by the valid signal and ended by the ready signal. A write command phase, read command phase, write data phase, read data phase, and write response phase are each, by themselves, a transfer.
- **APB transfer** – A single sequence initiated by the psel signal and ended by the pready signal.
- **Sparse data** – Where some of the bytes in an AXI write data transaction words are marked as not to be written.
- **Primary and Secondary** – Refers to the ports and their associated logic in a device with respect to the transaction flow. Transactions flow from the primary to the secondary side.
- **Downsizing** – Refers to the bridge configurations where the primary data port width is wider than the secondary data port width.
- **Byte reordering** – Changing the location of bytes in the data word across the Bridge (Little-Endian or Big-Endian).
- **Write transaction** – An AXI write transaction is composed of the following three independent phases:

  - A write command phase
  - A write data phase
  - A write response phase. This terminates the write transaction
- **Read Transaction –** An AXI read transaction is composed of the following two independent phases:
  - A read command phase
  - A read data phase. The signalling of the last beat of read data terminates the read transaction
- **Transaction ordering –** The order of transaction completion. A read transaction completes when the rlast and rvalid signals are asserted and accepted by the master (by the assertion of the rready signal). A write transaction completes when the bvalid signal is asserted and accepted. Out-of-order transaction completion at an AXI master occurs when transactions do not complete in the same order in which they were issued by the AXI master.
- **Write data interleaving depth –** The number of different AWID transactions that can be outstanding. Interleaving depth is controlled by the AXI slave and is the maximum amount of write data, from write transactions with different AWIDs, that can be interleaved to this slave. The AXI master must be aware of the slave's write data interleaving depth and NOT supply write data that will violate the slave's write interleaving depth rule.
- **Outstanding transaction depth –** The number of AXI outstanding transactions allowed by the AXI slave.
- **X2PS –** This is the section of the DW_axi_x2p that controls the secondary ports — the APB master interface control.
- **X2PP –** This is the section of the DW_axi_x2p that controls the primary ports — the AXI slave interface control.

1.6 Standards Compliance

The DW_axi_x2p component conforms to the *AMBA AXI and ACE Protocol Specification* from ARM. Readers are assumed to be familiar with these specifications.

1.7 Verification Environment Overview

The DW_axi_x2p includes an extensive verification environment, which sets up and invokes your selected simulation tool to execute tests that verify the functionality of the configured component. You can then analyze the results of the simulation.

The "Verification" on page 77 chapter discusses the testbench environment and provides an overview of the tests that are used to verify the DW_axi_x2p when you run component-level simulation.

1.8 Licenses

Before you begin using the DW_axi_x2p, you must have a valid license. For more information, refer to "Licenses" in the *DesignWare Synthesizable Components for AMBA 2, AMBA 3 AXI, and AMBA 4 AXI Installation Guide.*

Source code for this component is available on a per-project basis as a DesignWare Core. Please contact your local sales office for the details.

1.9 Where To Go From Here

At this point, you may want to get started working with the DW_axi_x2p component within a subsystem or by itself. Synopsys provides several tools within its coreTools suite of products for the purposes of configuration, synthesis, and verification of single or multiple synthesizable IP components — coreConsultant and coreAssembler. For information on the different coreTools, refer to *Guide to coreTools Documentation*.

For more information about configuring, synthesizing, and verifying just your DW_axi_x2p component, refer to *"Overview of the coreConsultant Configuration and Integration Process"* section in *DesignWare Synthesizable Components for AMBA 3 AXI, and AMBA 4 AXI User Guide*.

For more information about implementing your DW_axi_x2p component within a DesignWare subsystem using coreAssembler, refer to *"Overview of the coreAssembler Configuration and Integration Process"* section in *DesignWare Synthesizable Components for AMBA 3 AXI, and AMBA 4 AXI User Guide*.

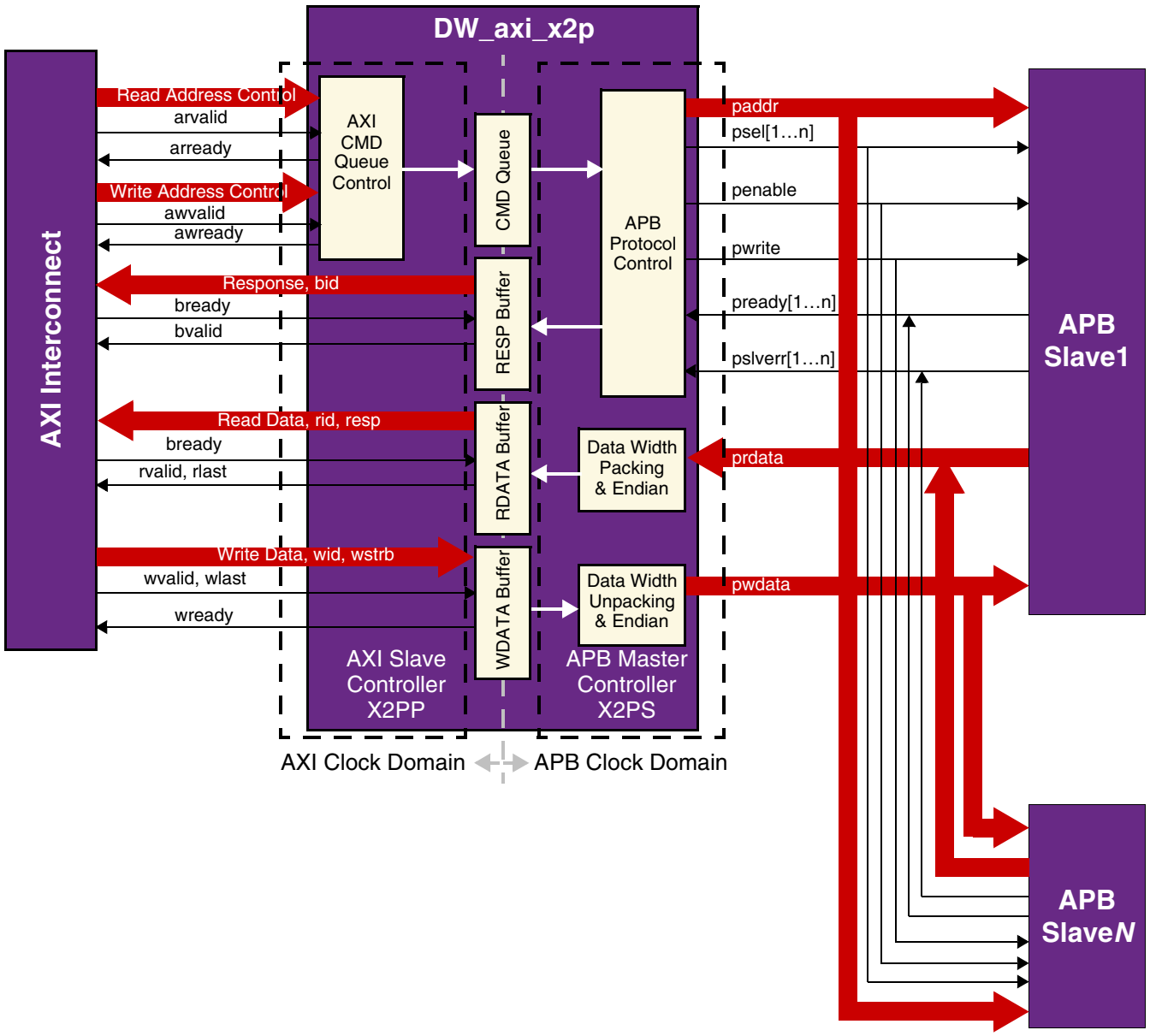
2 Functional Description

The DW_axi_x2p is a bridge that allows one to sixteen APB slave to be connected to an AXI bus. This chapter describes the functional features of the component.

2.1 Functional Overview

As illustrated in Figure 2-1, the DW_axi_x2p includes an AXI command queue controller, read/write data, response, command queue FIFOs, and an APB protocol controller.

Figure 2-1 DW_axi_x2p Block Diagram



2.1.1 Separate AXI and APB Functions

Each side of the DW_axi_x2p can service its respective protocol with no direct knowledge of the protocol on the other side. The bridge simplifies the interface between the two protocols by using four FIFOs — a common Command (CMD) Queue, a Write Data Buffer (WDATA), Read Data Buffer (RDATA), and a Write Response Buffer (RESP). The AXI slave (X2PP) and the APB master controller (X2PS) see only their side of these buffers filling and/or emptying. Thus, either side of the controller has no direct knowledge of what the other side is doing, but sees and responds to only the results of the requested transfer at the CMD Queue and data buffers' ports.

2.1.2 AXI Slave Controller (X2PP)

The X2PP monitors the AXI write and read address channels for the initiation of transactions (awvalid or arvalid). Upon detecting an AXI transaction on the write or read address channels, the X2PP loads the CMD Queue with AXI transaction information — address, type, size, beats, and direction — and responds to the transaction (awready or arready). Because the APB master controller can only exercise a single transaction at a time, the CMD Queue is shared for reads and writes and contains both read and write commands that are serially ordered from the X2PP AXI slave controller to the X2PS APB master controller. In cases where a simultaneous read and write is asserted on the AXI, the X2PP uses a simple write-then-read arbitration scheme resolving the order in which they are pushed into the common CMD Queue.

The DW_axi_x2p interface between X2PP and X2PS consists of three other FIFOs — WDATA, RDATA, and RESP buffers. Each of these FIFOs operate independently of each other.

2.1.2.1 WDATA Buffer

The assertion of wvalid from the AXI causes the following behavior:

- If the WDATA buffer is not full, the information from the write data channel is pushed into the WDATA buffer and wready is asserted.
- If the WDATA buffer is full, wready is not asserted until space becomes available in the WDATA buffer.

The information from the write data channel that is loaded into the WDATA buffer consists of data, ID, write strobes, and wlast fields.

2.1.2.2 RDATA Buffer

Upon detection of rready being asserted, the RDATA buffer is popped and data is transferred. The next read executes only when arvalid is asserted the next time.

2.1.2.3 RESP Buffer

The RESP buffer contains the responses from APB to write transfer operations that the DW_axi_x2p executes. As long as it is not empty, the RESP buffer issues to the AXI write response channel bid, bresp with bvalid, and waits for the AXI to return bready to advance the RESP buffer.

2.1.3 FIFO Usage

All communication between AXI and APB passes through these FIFOs in order to enable synchronization between the two clock domains. Additionally, FIFO depths are configurable in order to enable different system bandwidths for optimum performance.

2.1.4 APB Master Controller (X2PS)

The X2PS APB controller translates the AXI transactions seen in the CMD Queue into APB transactions. This involves the following:

- Fetches from the CMD Queue: transaction address, SIZE, LEN, BURST TYPE, and direction (read or write)
- When the CMD Queue transaction type is Write:

- A write command initiates write fetches from the WDATA buffer for the data, wstrb, wid, and bresp, and unpacks it to fit the APB bus width.
- Calculations on the APB address are performed to see if the APB address maps into a configured range or address aligned to the APB data word. If not, an error is issued (SLVERR) from the RESP data buffer and continues to pop the remaining write buffer data for the current AXI write, but stops all further APB transfers for the offending AXI transaction.
- Data ambiguity checks are performed depending on the following values for the X2P_ALLOW_SPARSE_TRANSFER parameter:
  - 0 – X2PS verifies the WSTROBE fitting the APB word - all strobes on or all off.

    Otherwise, an error (SLVERR) is issued and all further APB transfers are stopped for the offending AXI write transaction.
  - 1 – X2PS allows the sparse transfer without an error as explained in "AXI-to-APB Sparse Transfers" on page 26.
- An APB write transfer is initiated, fetches from the WDATA buffer and issuing to the APB side continue until wlast is asserted (to indicate transfer of the last beat in the WDATA buffer). When wlast and the last write APB transfer completes, a response is pushed into the RESP buffer, including the ID of the AXI transaction and the status of the APB write for the AXI transaction. If any APB write transfer for the AXI write transaction encountered an error, the SLVERR response is pushed into the RESP buffer.

For further details, refer to "AXI-to-APB Write Transfers" on page 24.

- When the CMD Queue transfer type is Read:
  - Initiates APB read transfers for which the number and addressing are derived from the read command in the CMD Queue.
  - Calculates the APB address and checks to see if the APB address maps into a configured range. If not, an error is issued and all further APB transfers for the current read transaction are stopped.
  - Issues the APB read transfer. As the data arrives from the APB, the bytes of the data are packed and arranged until the data width of the AXI read command SIZE is achieved.
  - Loads the RDATA buffer with the packed APB read transfers with the ID and the status of all APB read transfers. The X2PS returns the required number of read data beats (as indicated in the arlen in the read command).
  - Returns the SLVERR error on AXI under the following conditions for the corresponding beat:
    - If the APB transfer is encountered an PSLVERR
    - For an address unaligned to the APB data word
    - For sparse transfer if X2P_ALLOW_SPARSE_TRANSFER=0

      (in case if X2P_ALLOW_SPARSE_TRANSFER=1, X2PS allows the sparse transfer without an error as mentioned in the section "AXI-to-APB Sparse Transfers" on page 26).

2.2 Functional Features

This section provides more detail about the functional features of the DW_axi_x2p.

2.2.1 Translation of AXI Transactions into APB Transfers

This section discusses the details of AXI-to-APB transfers.

2.2.1.1 AXI-to-APB Read Transfers

The X2PP responds to the assertion of ARVALID on the AXI read address channel by pushing the AXI read channel contents into the CMD Queue. On monitoring the address channel, contents are first passed to the CMD Queue before going to the read data channel.

The X2PP, upon seeing a read on the AXI interface (arvalid asserted), resolves any possible conflicts with a simultaneous write on the write address channel, checks the CMD Queue, and if the queue is not full, pushes the read transfer information into it. If the CMD Queue is full or currently pushing a write command, the X2PP forces the read transfer on the AXI interface to wait (arready not asserted) until the current write is finished being pushed and space is available in the CMD Queue.

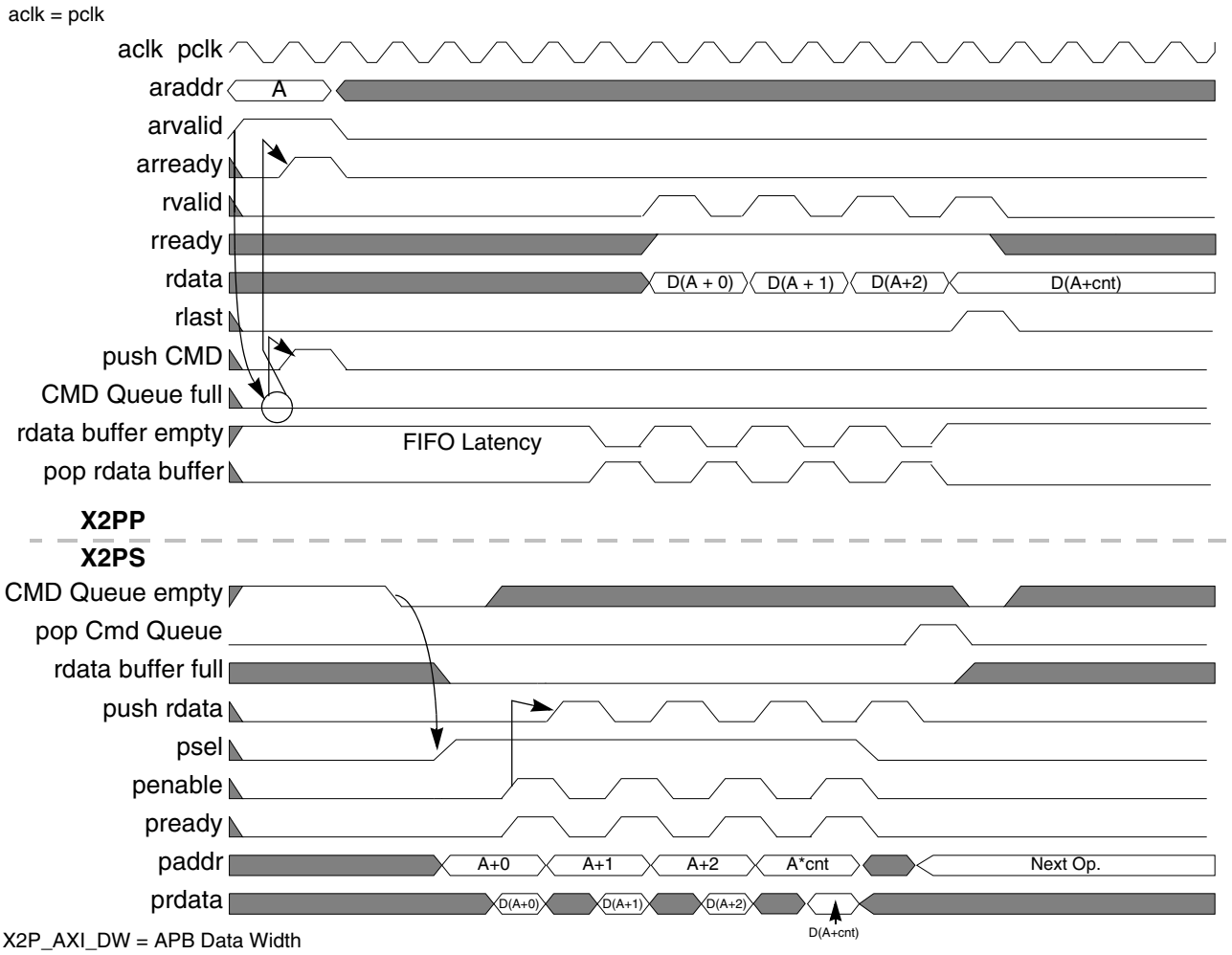
The X2PS receives transfers from the CMD Queue. If the current command popped from the CMD Queue is a read, the X2PS initiates a read transfer on the APB. The total number of APB transfers needed to fill the AXI read request is calculated based upon the LEN and SIZE fields in the CMD Queue and the configured APB data bus width.

Upon receiving the data from the APB, the X2PS positions the incoming data bytes and packs them until the AXI transfer word size is satisfied and then pushes the data beat (data, ID, status) into the RDATA buffer. When the X2PP APB master has fetched enough data to satisfy the requested AXI transaction words ((LEN * SIZE)/(APB Bus Width)), the X2PS pushes the data, ID, status, and rlast = 1 into the RDATA buffer. If the RDATA buffer is full, the controller stalls the APB transfer by delaying the start of a read transfer popped out of the CMD Queue or by holding the start of the transfer (psel) on the APB.

The X2PP passes the RDATA Buffer contents to the AXI as it becomes available in the buffer. The X2PP responds to the AXI read when the RDATA Buffer is not empty by “popping” the FIFO until it is empty. Each pop contains data, ID, and status of the active read transaction on the AXI.

The timing diagram in Figure 2-2 shows this behavior.

Figure 2-2 AXI Read Transaction Translated to APB



2.2.1.2 AXI-to-APB Write Transfers

The X2PP AXI slave and X2PS APB master controllers communicate during AXI writes via the Common CMD Queue, and the WDATA and RESP buffers.

Upon seeing the AXI write transaction on the AXI interface (awvalid asserted), the X2PP does the following:

- Arbitrates (see "Arbitration of AXI Transactions" on page 30) with any pending read transaction
- Pushes the CMD Queue with the write address, WID, and control from the AXI write address channel
- Sets the read/write bit of the CMD Queue to 1 to indicate a write transfer type

Any AXI write data, indicated by wvalid, is pushed into the WDATA Buffer. As long as the write buffer is not full, the X2PP pushes write data with its ID, data strobes, and the wlast bit into the WDATA Buffer and returns wready.

The X2PS, upon popping a write transaction from the CMD Queue, then starts accessing the WDATA Buffer or waits for the first write data to appear in the WDATA Buffer. When wlast is asserted in the data word from the WDATA Buffer, the current write transaction is considered completed by X2PS. The burst length,

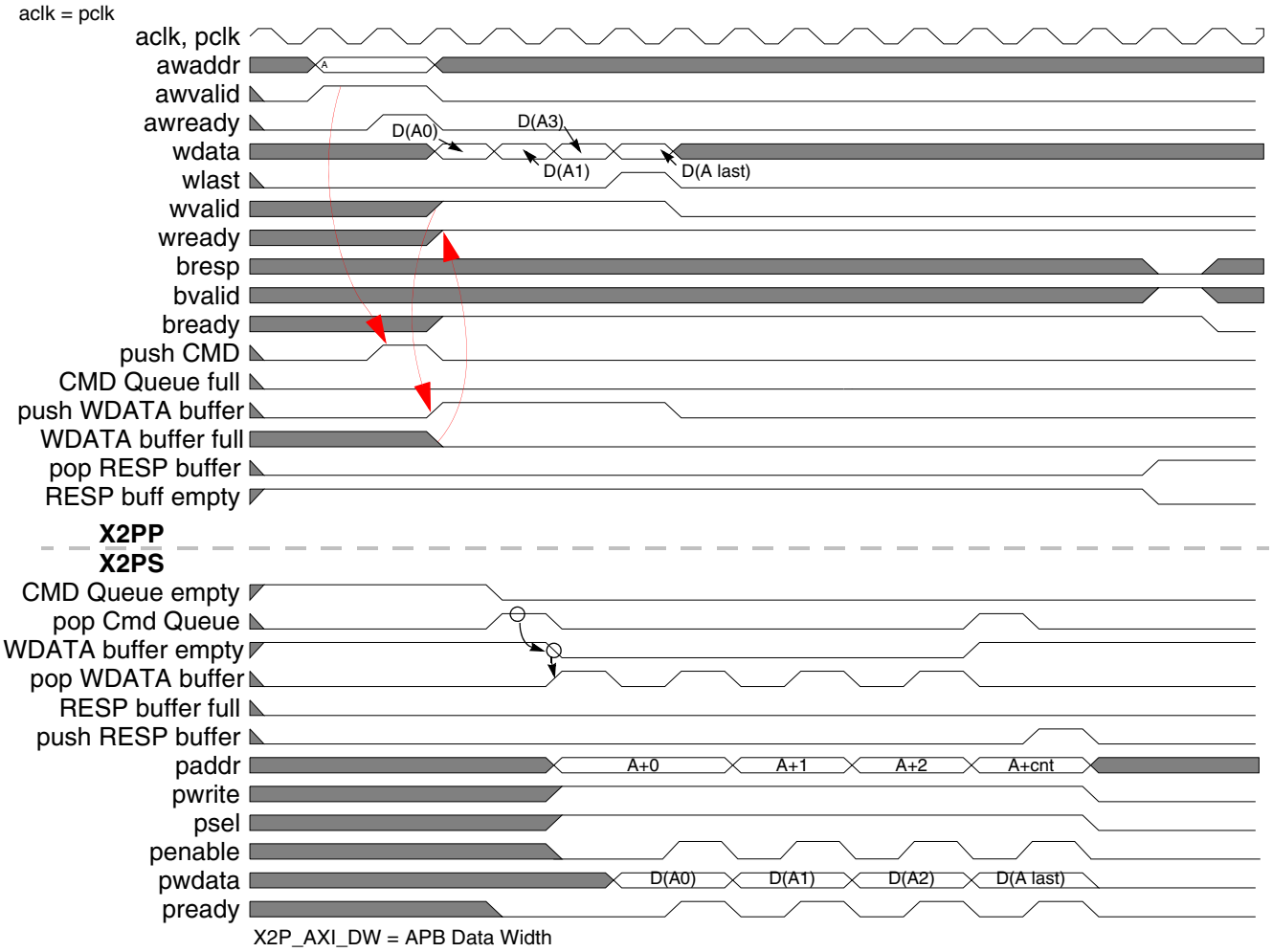
wlen, is not monitored to verify the correct number of data beats; instead, the DW_axi_x2p depends on accurate signaling with wlast.

Upon seeing wlast, and while issuing the last unpacked write transfer to the APB, the CMD Queue is checked and — if not empty — is popped, obtaining the next transaction. The X2PS comes to the end of the AXI write transaction when the data out of the WDATA buffer contains wlast and it has finished all the APB writes for the AXI write transaction. At this time, the X2PS passes the APB write completion status by pushing it into the RESP Buffer. The X2PP passes the contents of the RESP Buffer directly to the AXI write response channel.

As long as there are write transactions pending on the AXI write address channel, the X2PP continues pushing them into the CMD Queue until the CMD Queue fills. If the CMD Queue fills or is busy loading a read transaction, the X2PP stalls the AXI write address channel by holding awready inactive. The X2PP continues to hold awready inactive until the CMD Queue is free.

Figure 2-3 shows how the X2PP and X2PS handle the write transfer from AXI to APB.

Figure 2-3 AXI Write Transaction Translated to APB



2.2.1.3 Fixed Write Interleave Depth

The write interleaving depth of an AXI slave (such as the AXI-to-APB bridge) tells the AXI interconnect that it may deliver interleaved write or read data via the AXI bus for a certain number of active write or read commands (with different IDs).

The DW_axi_x2p has a write data interleaving depth of 1. This means there is no write interleaving. Once the AXI interconnect sends the first data beat of the write data transfer, then all the remaining data beats for the transfer must be sent before the next write transaction's data transfer can start.

2.2.1.4 Write Response for Write Transfers

The DW_axi_x2p provides a response to the AXI writes once all the APB write transfers have completed.

2.2.1.5 AXI-to-APB Sparse Transfers

The AXI bus supports the following features:

- Sparse writes – Indicates that certain byte lanes are enabled to be written but other byte lanes are not enabled to be written
- Transactions lesser than the bus width. – Indicates that the size of the transfer is lesser than the APB bus width

However, the APB bus does not support these features.

The DW_axi_x2p handles the sparse transfers feature by enabling the X2P_ALLOW_SPARSE_TRANSFER parameter. DW_axi_x2p will not generate an SLVERR when X2P_ALLOW_SPARSE_TRANSFER is set to 1 and allows the transfers to pass on to the APB interface for the following transactions:

- Sparse write data - If the strobes in the APB word are found to be a combination of 1's and 0's.
- AXI transaction with size less than the APB data bus width.

The DW_axi_x2p passes the AXI write transfer on the APB interface either:

- If all the write strobes for a given APB word are set to 1.
- If the strobes in the APB word are combination of 1's and 0's.

The address generated on the APB bus for the AXI transaction is always aligned to the APB bus width and you must ensure that a write transfer is not overwriting the data unintentionally in the APB interface for sparse transfers (because APB2/3 does not support write data strobes).

Figure 2-4 and Figure 2-5 show examples of converting AXI sparse transfers to the APB transfers. Each row in the figure represents a transfer. The cells highlighted in yellow indicate bytes that are valid based on the AXI address and control information. The transaction highlighted in blue is skipped in APB transfer because all strobes are zero for the APB word.

Figure 2-4 Sample Sparse Read Transfer Transactions

Primary Port
AXI Bus Width : 32
Address : 0x4
Transfer Size : 16 bits
Burst Type : Incrementing
Burst Length : 4 Transfers
Read /Write : Read
Secondary Port
APB Bus Width : 32 bits

Address Offset	AXI Bytes			
0x4	7	6	5	4
0x6	7	6	5	4
0x8	11	10	9	8
0xa	11	10	9	8

Translates to

Address Offset	APB Bytes			
0x4	7	6	5	4
0x4	7	6	5	4
0x8	11	10	9	8
0x8	11	10	9	8

Primary Port
AXI Bus Width : 32
Address : 0x4
Transfer Size : 8 bits
Burst Type : Incrementing
Burst Length : 4 Transfers
Read /Write : Read
Secondary Port
APB Bus Width : 32 bits

Address Offset	AXI Bytes			
0x4	7	6	5	4
0x5	7	6	5	4
0x6	7	6	5	4
0x7	7	6	5	4

Translates to

Address Offset	APB Bytes			
0x4	7	6	5	4
0x4	7	6	5	4
0x4	7	6	5	4
0x4	7	6	5	4

Primary Port
AXI Bus Width : 32
Address : 0x0
Transfer Size : 8 bits
Burst Type : SINGLE
Burst Length : 4 Transfers
Read /Write : Read
Secondary Port
APB Bus Width : 32 bits

Address Offset	AXI Bytes			
0x0	3	2	1	0
0x4	7	6	5	4
0x8	11	10	9	8
0xc	15	14	13	12

Translates to

Address Offset	APB Bytes			
0x0	3	2	1	0
0x4	7	6	5	4
0x8	11	10	9	8
0xc	15	14	13	12

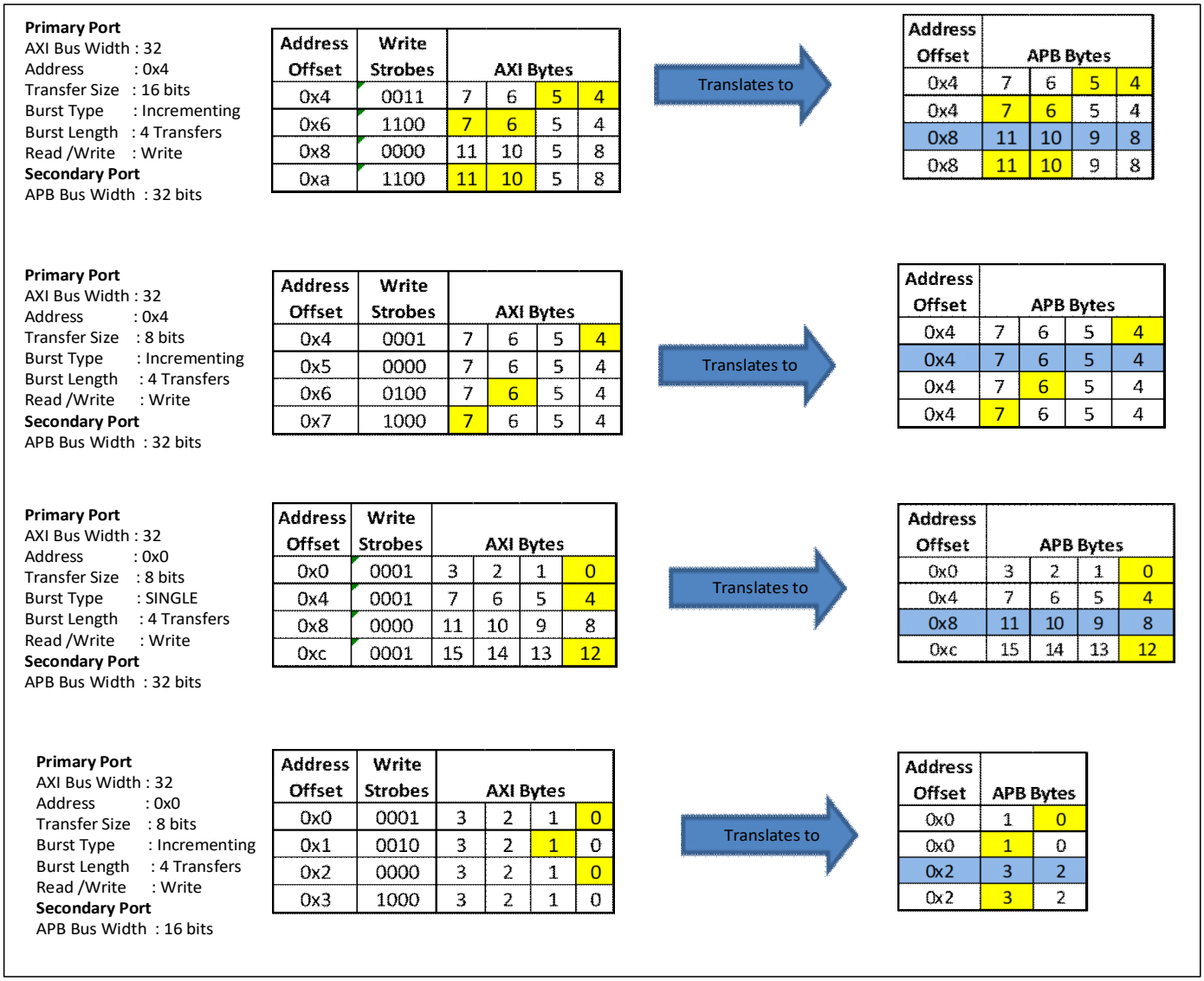
Primary Port
AXI Bus Width : 32
Address : 0x0
Transfer Size : 8 bits
Burst Type : SINGLE
Burst Length : 4 Transfers
Read /Write : Read
Secondary Port
APB Bus Width : 16 bits

Address Offset	AXI Bytes			
0x0	3	2	1	0
0x1	3	2	1	0
0x2	3	2	1	0
0x3	3	2	1	0

Translates to

Address Offset	APB Bytes	
0x0	1	0
0x0	1	0
0x2	3	2
0x2	3	2

Figure 2-5 Sample Sparse Write Transfer Transactions



2.2.1.6 AXI-to-APB Write Data Limitations

The AXI bus supports “sparse” writes in which some byte lanes are enabled to be written but others are not. The AXI write data channel has a wstrb field with one bit for each byte of write data to indicate that the data byte is valid. The APB does not support sparse writes.

The X2PS responds to the AXI write strobes in the following manner.

If X2P_ALLOW_SPARSE_TRANSFER = 0:

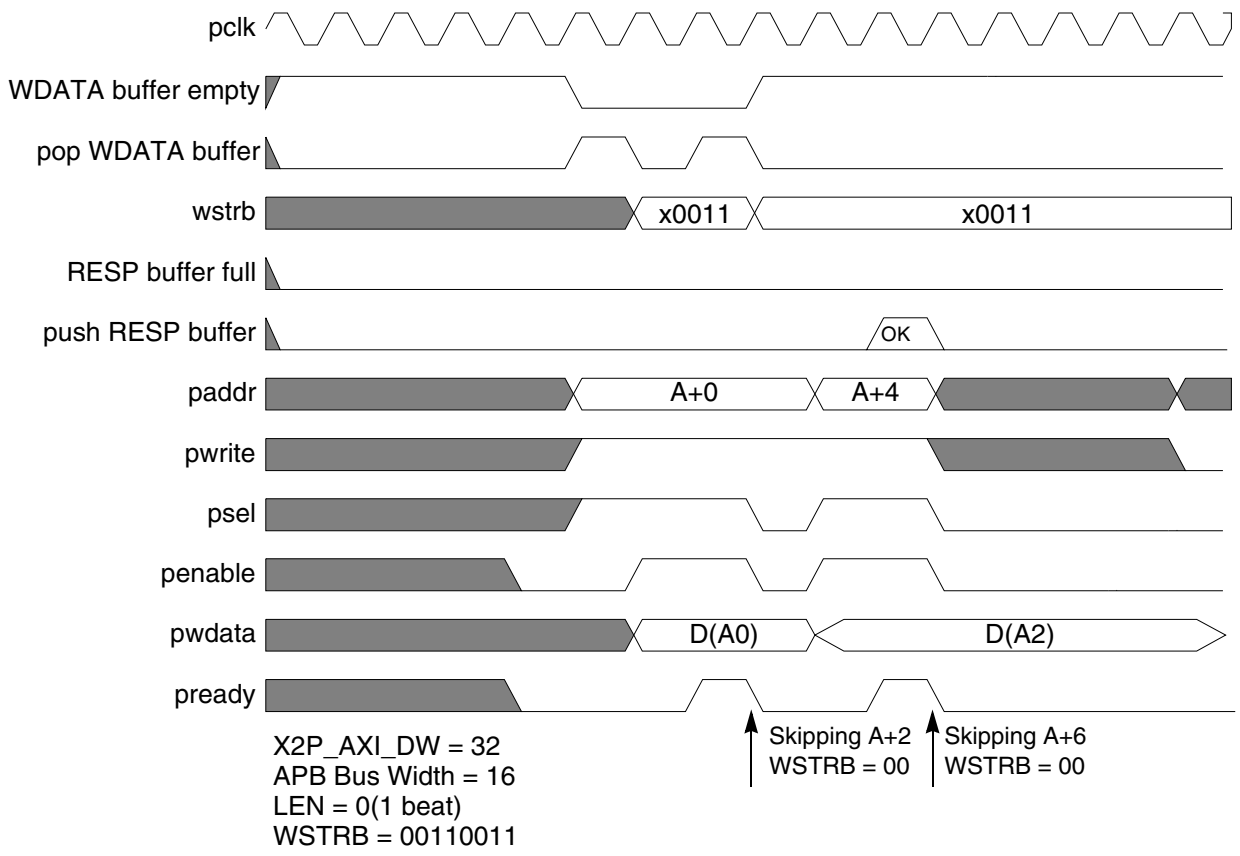
1. If all the write strobes for a given APB word are set to 1, the APB write transfer is executed.
2. If all strobes in the APB word are 0, the APB write transfer for the current word is skipped and the strobes for the next APB write word in the current write transaction are checked.

3. If the strobes in the APB word are found to be a combination of 1's and 0's, all subsequent APB transfers are terminated for the current AXI transaction and a SLVERR is returned through the AXI write response channel bresp.

If X2P_ALLOW_SPARSE_TRANSFER = 1:

1. If all the write strobes for a given APB are set to 1, the APB Write transfer is executed.
2. If atleast one strobe for a given APB word is set to 1, the APB Write transfer is executed. You must ensure that the APB Write is not unintentionally overwriting the data in APB Interface.
3. If all the strobes in the APB word are 0, the APB write transfer for the current word is skipped and the strobes for the next APB write word in the current write transaction are checked.

Figure 2-6 AXI Write Transaction with Write Strobes Selecting APB Words



2.2.1.7 AXI Command Usage on APB Transfers

All operations on the APB are a single beat transfer — that is, no bursts are supported — so the X2PS issues single read or writes on the APB. The X2PS adjusts for the AXI transfer types by conditioning the address of the APB transactions. All the AXI burst transfer types (INCR, WRAP, and FIXED) are supported.

2.2.1.7.1 AXI Increment (INCR)

For AXI transactions with the APB, the X2PS uses multiple APB commands to read or write all the data for a single AXI transaction. For example, suppose an AXI transaction performs an INCR read that is 12 beats

long with a data width of 32 bits, and that the APB has been set to have a data width of 8. The X2PS has to perform a total of 48 (12 * (32/8)) read transfers.

2.2.1.7.2 AXI Wraps (WRAP)

Since the APB is a single transfer protocol, it has no provision for WRAP. If a WRAP transfer comes in from the AXI, the X2PS issues APB transfers addressed to read up to the WRAP boundary, then adjusts the transfer addresses to start at the wrapped address to complete the required transfer.

2.2.1.7.3 AXI Fixed (FIXED)

It is also possible for a FIXED burst read or write command to come in from AXI. This specifies that the same location should be read or written several times in a row. The X2PS repeats the read or write on the APB as many times as specified by the AXI FIXED command. A sequence of reads or writes would be repeated for each data beat of a FIXED transaction if X2P_AXI_DW were greater than the APB data width. This is known as downsizing and is described further in "Address and Data Port Configuration Adjustments" on page 30.

2.2.2 Arbitration of AXI Transactions

The X2PP Arbiter selects which pending AXI transaction is to be forwarded by the X2PP. The AXI protocol supports the simultaneous issue of a read and a write transaction. Since the APB can issue only one transaction at a time, the AXI transactions are ordered such that they are pushed to the APB one at a time.

If the next accesses continue to be both read and write, the writes are issued on subsequent cycles first, alternated with reads.

2.2.3 Address and Data Port Configuration Adjustments

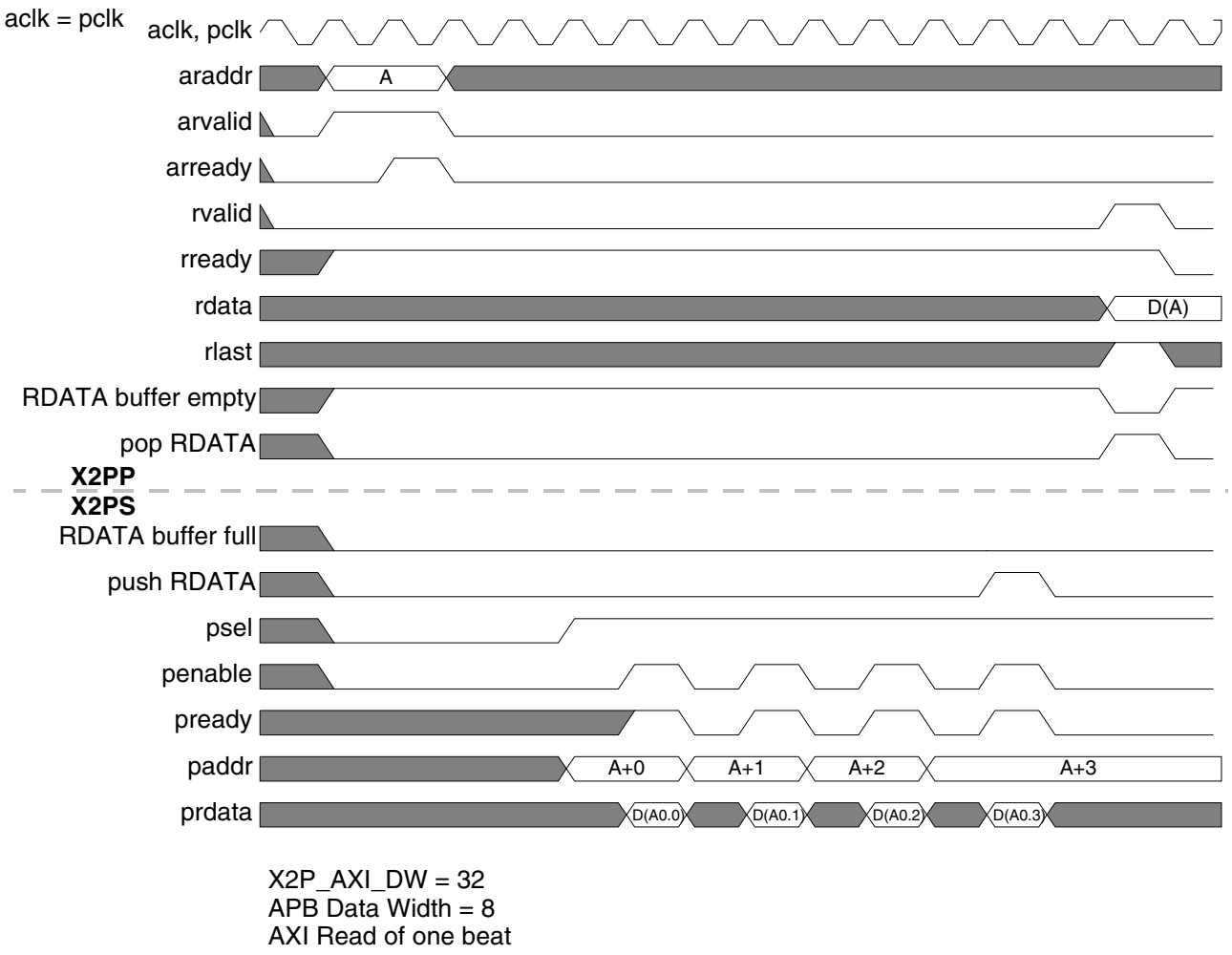
Downsizing of data ports from the AXI to the APB requires that the write data be unpacked from an AXI-sized word to APB-sized words. Reading the APB requires that the APB words are packed to form an AXI-sized word.

2.2.3.1 Read Packing

It is possible to have a mismatch in data widths between AXI read transactions and read commands issued on APB by the X2PP (X2P_AXI_DW > APB data width). In AXI read transactions where the X2P_AXI_DW is greater than the APB data width, the X2PS uses multiple APB read commands to fetch all the data for a single AXI read data transfer. For example, suppose an AXI transaction performs an INCR read that is 12 beats long with a data width of 32 bits, and that the APB has been set to have a data width of 8. The APB master controller has to perform a total of 48 (12 * (32/8)) read transfers; that is, there are four APB reads to fill one AXI data beat.

Figure 2-7 illustrates signals for this example of read packing.

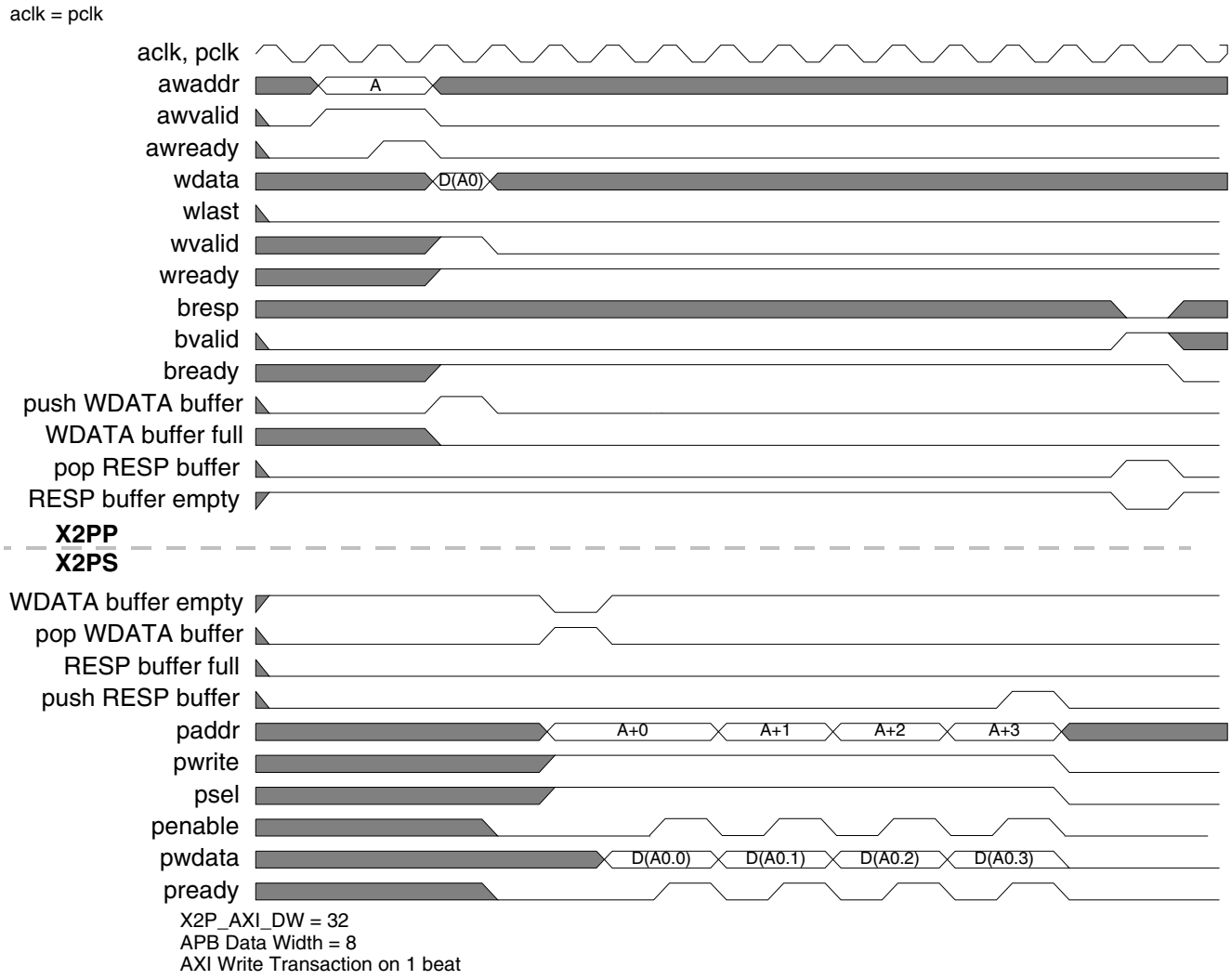
Figure 2-7 Packed AXI Read Transaction from APB



2.2.3.2 Write Unpacking

In a configuration where the AXI write transactions have an X2P_AXI_DW greater than the APB data width, the write data must be unpacked into APB data bus width for APB writes.

Figure 2-8 Unpacked AXI Write Transaction to APB

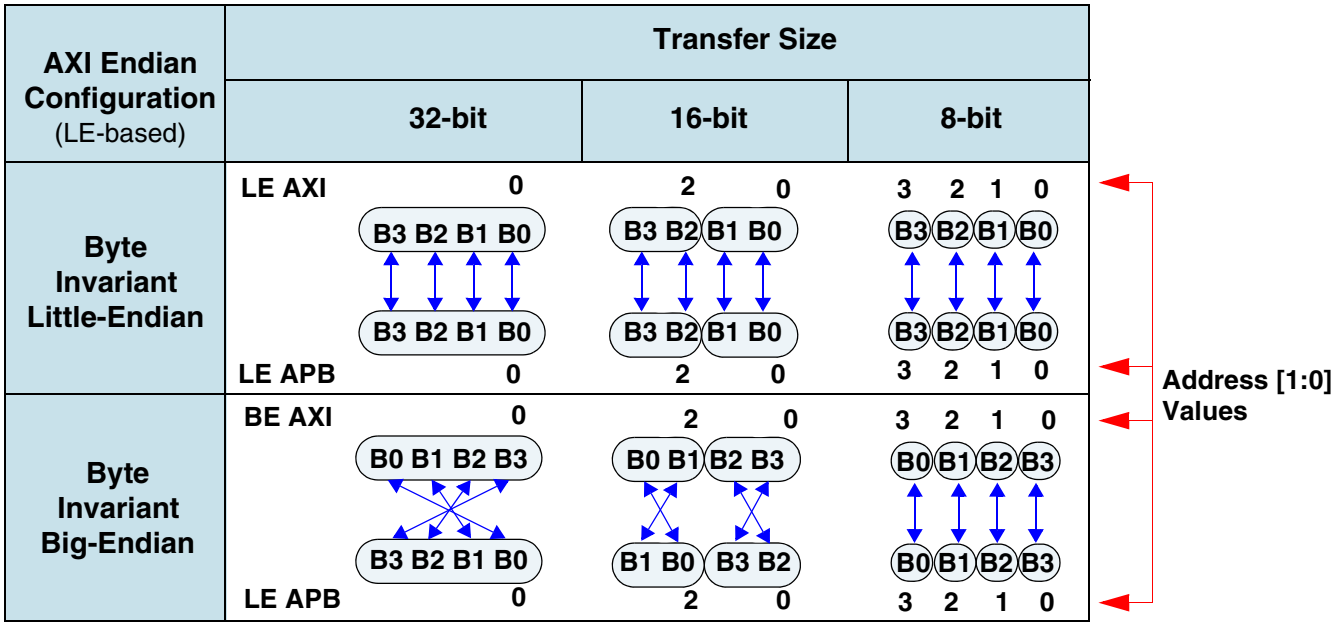


2.2.3.3 Endian Adaptation (Byte Re-ordering)

The DW_axi_x2p supports “byte-invariant little-endian” and “byte-invariant big-endian” for the AXI; the APB is fixed at little-endian byte-invariant.

For a definition of the endian variations used in the DW_axi_x2p, refer to Figure 2-9 on page 33, which illustrates the available endian transfers operating on a 32-bit data path under three different transfer sizes. This example can be extrapolated to show wider data paths and larger transfer sizes.

Figure 2-9 Endian Definitions



2.2.4 Slave Error Conditions

The following are conditions for issuing a SLVERR to the AXI. Data transfers continue on the APB until any error condition is detected. The detection of an error terminates all APB transfer activity for the current AXI write or read transaction. To satisfy the AXI protocol, the pushing and popping of the buffers continue with SLVERR inserted in the appropriate fields:

- For writes – bresp in the write response channel
- For reads – rresp in each of the beats of the read data from where the error was detected

2.2.4.1 Unaligned Transaction Addresses

When the X2PP detects an AXI start address that points to a byte not at the start of the APB word boundary, it issues an SLVERR response for the transaction and does not issue the transaction from the X2PS.

For a write transaction, this results in a single transfer on the burst response channel with a response of SLVERR.

For a read transaction, this results in a number of read beats being issued, all with the response signal set to SLVERR. The number of beats returned equal the arlen of the offending read command.

2.2.4.2 APB Memory Mapping Errors

The DW_axi_x2p has space for the allocation of up to 16 APB devices. The bridge determines an APB slave's existence by checking the APB memory map. The memory map is set up through the configuration parameters, which consists of setting an address pair for the start and finish of each APB device. For a description of the settings, refer to "Parameter Descriptions" on page 43.

The X2PS, after receiving an AXI transaction address, proceeds to initiate APB transfers, incrementing the address in APB word increments, as appropriate. Before the start of each APB transaction, the X2PS checks to see if the current APB word address is located within any of the mapped address pairs. If this condition is

violated, the X2PS considers the remaining transfers to be in error, and a SLVERR error is issued to the RDATA buffer for all of the remaining AXI data beats of the transaction.

2.2.4.3 AXI Transaction Size Less Than APB Data Bus Width

Any AXI transaction with a SIZE less than the APB data width does not issue any APB transfers. In this circumstance, the DW_axi_x2p responds to a write with a SLVERR bresp to the RESP buffer and to a read with a SLVERR on each of the AXI data beats to the RDATA buffer.

2.2.4.4 Sparse Data Errors

In cases where some of the AXI "write strobes" are disabled (a mix of strobe settings) inside an APB data word, writes to the APB for that and subsequent words are skipped, and the X2PS sets an error condition as a bresp of SLVERR to the RESP buffer.

2.2.4.5 PSLVERR from APB Slave

New to the AMBA 3 APB protocol is the PSLVERR response condition, which indicates a failed transfer. Transactions that the X2PS performs on the APB bus may return a PSLVERR response.

As the X2PS pushes data into the RDATA buffer, it also pushes in an AXI response code for each beat. An OKAY response is set if the data being pushed in came from APB with no PSLVERR. Keep in mind that the data pushed into the RDATA buffer may have been obtained from more than one APB read due to downsizing. If any of the data pushed came from APB with PSLVERR, then an AXI response of SLVERR is put into the RDATA buffer along with this data.

When the X2PS pushes in the AXI write response, if any of the writes on APB were terminated with a PSLVERR then an AXI response of SLVERR is pushed into the RESP Buffer.

2.2.5 Clocks and Resets

This section provides more information about how the clock functions. There are two clock inputs: aclk and pclk. The clocking mode is a user-configurable parameter — X2P_CLOCK_MODE — which selects either "single clock" or "dual clock." If "single clock" is chosen, then only the aclk input is used and the pclk input is ignored.

2.2.5.1 Single Clock Mode

Single clock mode is used for systems where aclk and pclk are fully synchronous; that is, when the clocks are phase-aligned and of the same frequency. No logic for crossing the clock boundary exists across the aclk to pclk domains in this configuration.

2.2.5.2 Dual Clock Mode

Dual clock mode is used for systems where aclk and pclk are either asynchronous or quasi-synchronous; that is, when the clocks are phase-aligned, but have integer frequency ratios.

2.2.5.2.1 Synchronization Depth

The X2P_DUAL_CLK_SYNC_DEPTH coreConsultant parameter is enabled only when X2P_CLK_MODE is configured as Dual Clock. In this mode, the user can select the synchronization depth when crossing clock domains between AXI and APB.

- 0 – No synchronization
- 2 – Two-stage synchronization with both stages synchronizing on positive clock edge

- 3 – Three-stage synchronization, with all stages synchronizing on positive clock edge

The number of synchronization stages directly affects the latency through the X2P bridge.

When aclk and pclk are phase-aligned and of an integer frequency ratio — that is, quasi-synchronous — then metastability stages are not required. In this case, the X2P_DUAL_CLK_SYNC_DEPTH parameter can be set to 0.

2.2.5.3 Usage of Reset Inputs

The DW_axi_x2p includes two negative-true reset inputs: aresetn to the X2PP and presetn to X2PS. The resets are used to asynchronously reset flip-flops inside the DW_axi_x2p.

The aresetn input is used to reset flip-flops that are clocked by the AXI clock, aclk, and presetn is used to reset flip-flops that are clocked by the APB clock, pclk.

Attention It is recommended that you never assert just one of these reset signals without asserting the other. To do so would likely cause serious operation failures in the FIFOs because they would have either their “write pointers” or “read pointers” cleared but not both. Always assert aresetn and presetn (low).

When releasing the DW_axi_x2p from reset, it is necessary to have aresetn go to the de-asserted state (high) synchronous to aclk, and preset go to the de-asserted state (high) synchronous to pclk.

In the case of single clock mode where the AXI and APB are driven by the same clock (aclk), only the single reset input aresetn is used to reset all of DW_axi_x2p.

2.2.6 Additional Control Information Signals

The AXI protocol specification defines AR/AWCACHE[3:0] and AR/AWPROT[2:0] as additional control signals. The APB protocol contains no equivalence to these signals. The DW_axi_x2p does not monitor AR/AWCACHE[3:0] and AR/AWPROT[2:0].

2.2.7 Atomic Accesses

As described in this section, the DW_axi_x2p does not support locked transactions and exclusive accesses.

2.2.7.1 Locked Transactions

The DW_axi_x2p does not support and ignores the ARLOCK/AWLOCK signals.

2.2.7.2 Exclusive AXI-to-APB Transactions

The DW_axi_x2p does not support exclusive accesses. It responds to exclusive reads by performing the read and returning the data with OKAY (or SLVERR), but never EXOKAY. In the case of an exclusive write, the DW_axi_x2p always performs the write normally and responds with OKAY or SLVERR, but never EXOKAY.

2.2.8 Backward Compatibility with AMBA 2 APB

The more recent AMBA 3 APB protocol added the signals ready (pready) and error (pslverr). APB slaves attached to the DW_axi_x2p can support either the AMBA 3 APB or AMBA 2 APB protocol, however. For each APB slave, you can use the X2P_IS_APB3_Sn configuration parameter to specify whether the attached

component supports AMBA 3 APB or AMBA 2 APB. This configuration determines whether or not the pready and pslverr signals from that APB slave are considered by the DW_axi_x2p during APB transactions. For more information on configuration parameters, refer to "Parameter Descriptions" on page 43.

2.2.9 Low-Power Interface

You can configure the DW_axi_x2p to include an AXI low-power handshaking interface. This interface allows the following:

- DW_axi_x2p informs the system low-power controller (LPC) when it has no outstanding transactions
- LPC requests the DW_axi_x2p to enter into a low-power state

You can include this low-power interface in your design by setting the X2P_LOWPWR_HS_IF parameter to 1.

The low power handshaking interface includes the following signals:

- csysreq input – de-asserted by the LPC to initiate a low-power state; asserted by LPC to initiate an exit from a low-power state
- csysack output – de-asserted by DW_axi_x2p to acknowledge a request to enter a low-power state; asserted by DW_axi_x2p to acknowledge a request to exit a low-power state
- cactive output – de-asserted by DW_axi_x2p when the clock can be removed after the next positive edge; csysack must also be deasserted

The sequence of events for entering a low-power state is as follows:

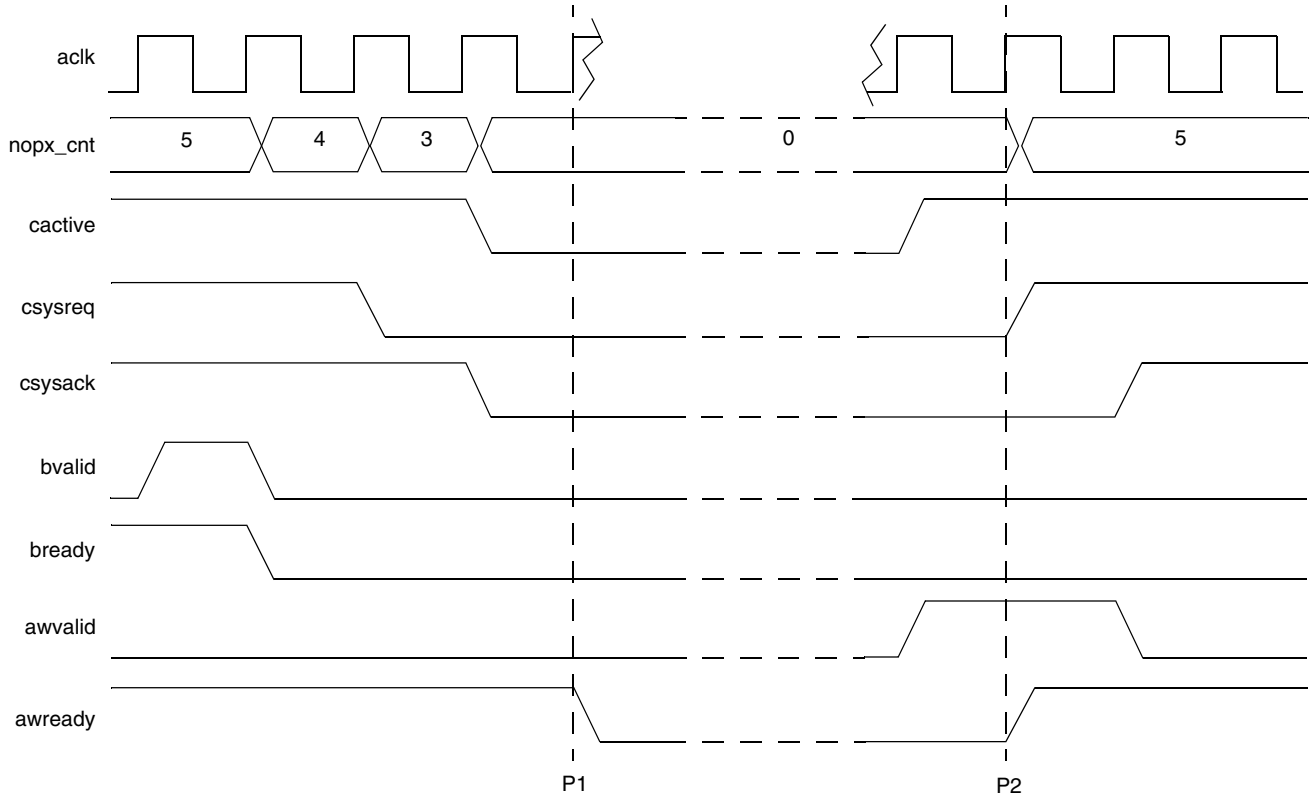
1. The LPC requests the DW_axi_x2p to enter a low-power state by de-asserting the csysreq signal.
2. Since the DW_axi_x2p does not have a power-up or power-down sequence, it always acknowledges a csysreq signal on the next cycle by asserting or de-asserting the csysack signal.

   When requested by the LPC, the low-power state depends on the value of the cactive signal at the time that the csysack signal is sampled by the LPC after the csysreq signal has been de-asserted.
3. The cactive signal de-asserts when the DW_axi_x2p has no outstanding transactions. If you set the X2P_LOWPWR_NOPX_CNT parameter to "X" in coreConsultant, the cactive signal de-asserts after *X* cycles, during which there were no outstanding transactions, have elapsed.
4. DW_axi_x2p enters a low-power state when *all* of the cactive, csysreq, and csysack signals are low (0) when sampled at the positive edge of aclk; this is illustrated at P1 in Figure 2-10 on page 37.

Note Entry to a low-power state happens at any cycle in which the above condition is satisfied.

While in a low-power state, the awready, wready, and arready signals are held low, which prevents any new transaction from starting and thus allows the clock to be safely disabled; this is illustrated between points P1 and P2 of Figure 2-10.

Figure 2-10 Low-Power State Entry and Exit (X2P_LOWPWR_NOPX_CNT = 5)



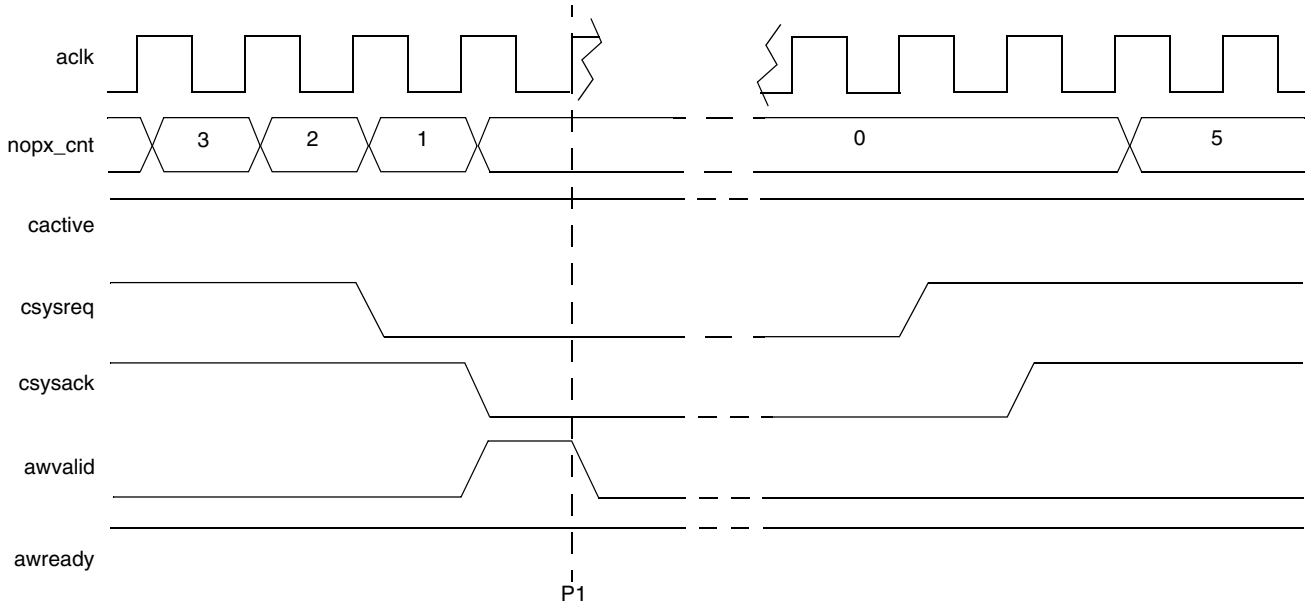
The DW_axi_x2p exits a low-power state when the cactive signal goes high and is sampled at the positive edge of aclk; illustrated at P2 of Figure 2-10.

An exit from low-power can be initiated by:

- LPC asserting the csysreq signal, which causes the DW_axi_x2p to assert the cactive signal, illustrated in Figure 2-12.
- A master asserting the awvalid or arvalid signals of the DW_axi_x2p, which causes the DW_axi_x2p to assert the cactive signal, illustrated in Figure 2-10.

Figure 2-11 shows the DW_axi_x2p rejecting a request to enter a low-power state. At P1, the no-pending-transaction (nopx_cnt) counter has reached 0, but on this same cycle the awvalid signal asserts, which keeps the cactive signal asserted. The LPC samples at P1 that the DW_axi_x2p has rejected the entry to low-power mode, and therefore must not remove the clock.

Figure 2-11 DW_axi_x2p Low-Power Interface Rejects Low-Power Entry



2.2.10 cactive Signal De-assertion

The cactive signal de-asserts the X2P_LOWPWR_NOPX_CNT parameter *aclk* cycles after the last pending transaction completes. If the csysreq signal de-asserts while the component is counting down to 0 from X2P_LOWPWR_NOPX_CNT, the cactive signal will de-assert on the next cycle.

After the positive edge of the aclk signal where the cactive signal and the csysack signal are sampled low, the low-power controller (LPC) may remove the clock(s) from the DW_axi_x2p (P1 in Figure 2-10). At this point the bready and awready signals will also be driven low.

2.2.11 cactive Signal Assertion

The cactive signal will assert combinatorially when the awvalid or arvalid signals are asserted, at which point it will then remain asserted until all outstanding transactions have completed and X2P_LOWPWR_NOPX_CNT cycles have elapsed.

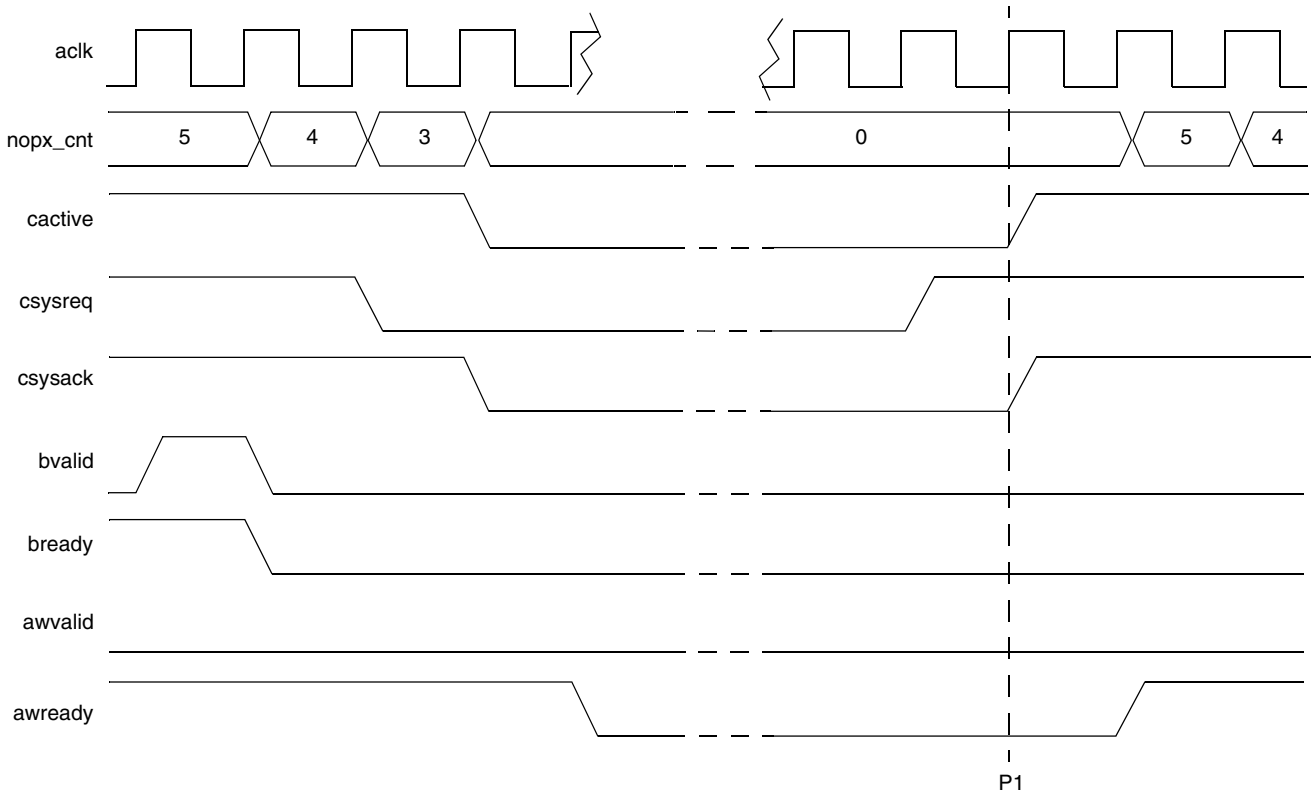
Note Early write data will not cause the cactive signal to assert or to remain asserted.

If the cactive signal is low and the csysreq signal is asserted, the DW_axi_x2p will assert the cactive signal at the same time as the csysack signal in order to complete the low-power exit handshake. After the clock edge where the cactive and csysack signals are sampled high on exit from low-power mode, the DW_axi_x2p will keep the cactive signal asserted for X2P_LOWPWR_NOPX_CNT cycles, after which it will either:

- De-assert until there are active transactions again.
- De-assert until another low-power exit handshake is completed.

This can be seen at P1 in Figure 2-12 where the cactive signal is asserted 1 clock cycle after the csysreq signal asserts, and the no-pending-transaction counter (nopx_cnt) starts to decrement again from the next cycle.

Figure 2-12 LPC Initiates Low-Power Exit



Note

When coming out of reset, if the awvalid and arvalid signals equal 0:

- cactive signal equals 0, if X2P_LOWPWR_NOPX_CNT parameter equals 0
- cactive signal equals 1, if X2P_LOWPWR_NOPX_CNT parameter is greater than 0 and will de-assert when *X2P_LOWPWR_NOPX_CNT* clock cycles have elapsed

2.2.12 Configuring Memory Map in coreConsultant

You configure the memory map using the APB Parameters page of the Specify Configuration activity in coreConsultant.

You can configure the start and end addresses of each APB slave attached to the same DW_axi_x2p with values up to X2P_AXI_AW bits wide. This enables you to organize the APB slave addresses in a memory larger than 32 bits, depending on the AXI address bus width (X2P_AXI_AW).

Figure 2-13 illustrates two APB slaves in a 33-bits AXI address map that are located at non-contiguous regions in the address map. However, since APB memory space is limited to 4 Gb, the DW_axi_x2p uses only the lower 32 bits of each address location to decode which slave is being accessed. Thus, at the APB memory space, the APB slaves are at contiguous locations.

Figure 2-13 Two APB Slaves Attached to DW_axi_x2p in 33-bit AXI Address Map

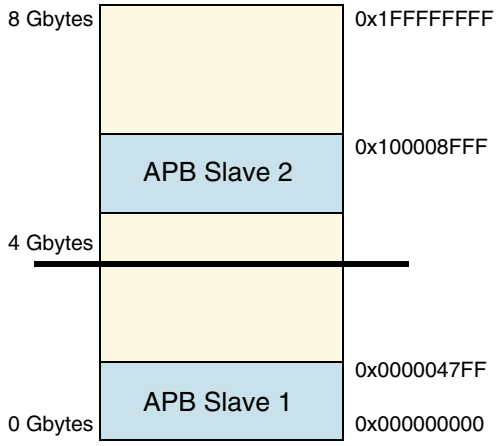
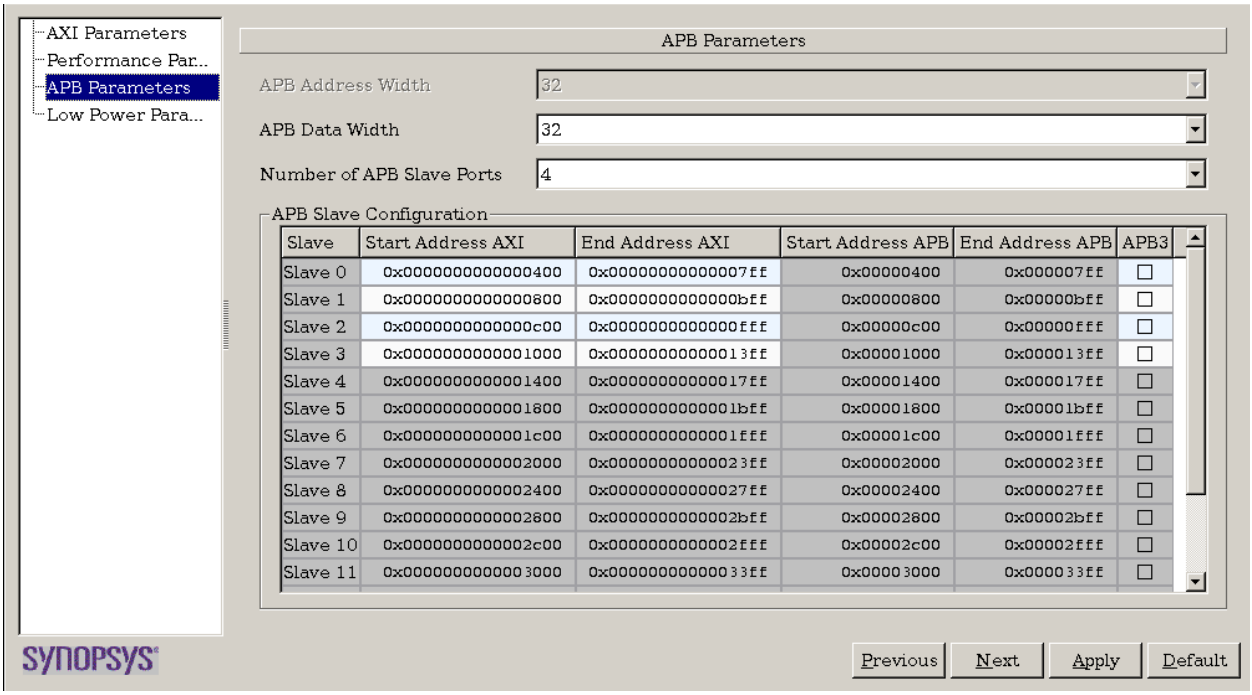


Figure 2-14 illustrates an example of configuring the memory map in coreConsultant. In the two left-most columns — Start Address AXI and End Address AXI — you define the address values as sent through AXI address buses. The next two columns — Start Address APB and End Address APB — contain the values defined in the previous columns transformed into 32-bit values. These are the values that the DW_axi_x2p uses to decode the addressed APB slave. The cells in these columns are non-editable and are only visible in order to help users to understand the address boundaries considered by DW_axi_x2p.

Figure 2-14 DW_axi_x2p Memory Map Configuration in coreConsultant



During the DW_axi_x2p configuration, coreConsultant checks for the following conditions:

- AXI address values must not exceed ($2^{X2P\_AXI\_AW}$) - 1024 for Start Addresses and ($2^{X2P\_AXI\_AW}$) - 1 for End addresses.

- There are no overlaps in the memory spaces defined by the lower 32 bits of the start and end AXI addresses.

3 Parameter Descriptions

This chapter details all the configuration parameters. **You can use the coreConsultant GUI configuration reports to determine the actual configured state of the controller.** Some expressions might refer to TCL functions or procedures (sometimes identified as **<functionof>)** that coreConsultant uses to make calculations. The exact formula used by these TCL functions is not provided in this chapter. However, when you configure the controller in coreConsultant, all TCL functions and parameters are evaluated completely; and the resulting values are displayed where appropriate in the coreConsultant GUI reports.

The parameter descriptions in this chapter include the **Enabled:** attribute which indicates the values required to be set on other parameters before you can change the value of this parameter.

These tables define all of the ~~user~~ configuration options for this component.

3.1 AXI Parameters

Table 3-1 AXI Parameters

Label	Description
AXI Parameters	
Select AXI Interface Type?	Select AXI Interface Type as AXI3 or AXI4. By default, DW_axi_x2p supports the AXI3 interface. **Values:** ■ AXI3 (0) ■ AXI4 (1) **Default Value:** AXI3 **Enabled:** DWC-AMBA-Fabric-Source License required **Parameter Name:** X2P_AXI_INTERFACE_TYPE
AXI Address Width	Address bus width of the AXI system to which the bridge is attached as an AXI slave. The legal values comprise the full range from 32 bits to 64 bits. The full range is used for the psel select signals, but only the lower 32 bits are passed on to paddr. **Values:** 32, ..., 64 **Default Value:** 32 **Enabled:** Always **Parameter Name:** X2P_AXI_AW
AXI ID Width	Read and write ID width of the AXI system to which the bridge is attached as an AXI slave. **Values:** 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16 **Default Value:** 8 **Enabled:** Always **Parameter Name:** X2P_AXI_SIDW
AXI Data Width	Read and write data bus width of the AXI system to which the bridge is attached as an AXI slave. Data bus width for the AXI slave interface must be greater than or equal to that of the APB master interface. **Values:** 8, 16, 32, 64, 128, 256, 512 **Default Value:** 32 **Enabled:** Always **Parameter Name:** X2P_AXI_DW
AXI Burst Length Width	Width used for the AWLEN and ARLEN burst count field. **Values:** 4, 5, 6, 7, 8 **Default Value:** 4 **Enabled:** Always **Parameter Name:** X2P_AXI_BLW

Table 3-1 AXI Parameters (Continued)

Label	Description
AXI Endianness	Data bus endianness of the AXI system to which the bridge is attached as an AXI slave. ■ 0: AXI bus is little-endian ■ 1: AXI bus is big-endian The APB bus is always little endian. For more information, see the "Endian Adaptation (Byte Re-ordering)" section in the DW_axi_x2p Databook. **Values:** ■ Little Endian (0) ■ Big Endian (1) **Default Value:** Little Endian **Enabled:** Always **Parameter Name:** X2P_AXI_ENDIANNESS
Allow Sparse Transfer on APB Interface?	Selects whether to generate the error response for the sparse write and read transfers or not. ■ True (1) - DW_axi_x2p does not generate the error for sparse transfers and passes the AXI transaction on to APB Interface. ■ False (0) - DW_axi_x2p generates an error resonse for sparse transfers. For more information, see the "AXI-to-APB Sparse Transfers" section in the DW_axi_x2p Databook. **Values:** ■ false (0) ■ true (1) **Default Value:** false **Enabled:** Always **Parameter Name:** X2P_ALLOW_SPARSE_TRANSFER

3.2 Performance Parameters

Table 3-2 Performance Parameters

Label	Description
Buffer Configuration	
Command Queue Depth	Number of locations in the common command queue. Depth must be at least 2 when in dual-clock mode. The depth 1 is allowed in only single-clock mode. **Values:** 1, 2, 4, 8, 16, 32 **Default Value:** 4 **Enabled:** Always **Parameter Name:** X2P_CMD_QUEUE_DEPTH
Write Data Buffer Depth	Number of locations in the write data buffer. Depth must be at least 2 when in dual-clock mode. The depth 1 is allowed in only single-clock mode. **Values:** 1, 2, 4, 8, 16, 32, 64 **Default Value:** 2 **Enabled:** Always **Parameter Name:** X2P_WRITE_BUFFER_DEPTH
Write Response Buffer Depth	Number of locations in the write response buffer. Depth must be at least 2 when in dual-clock mode. The depth 1 is allowed in only single-clock mode. **Values:** 1, 2, 4, 8, 16 **Default Value:** 2 **Enabled:** Always **Parameter Name:** X2P_WRITE_RESP_BUFFER_DEPTH
Read Data Buffer Depth	Number of locations in the read data buffer. Depth must be at least 2 when in dual-clock mode. The depth 1 is allowed in only single-clock mode. **Values:** 1, 2, 4, 8, 16, 32 **Default Value:** 2 **Enabled:** Always **Parameter Name:** X2P_READ_BUFFER_DEPTH

Table 3-2 Performance Parameters (Continued)

Label	Description
Clock Mode	
Clock Mode	Specifies the relationship between aclk and pclk. The AXI slave interface is clocked by aclk; the APB master interface is clocked by pclk. ■ Dual Clock: aclk and pclk are different (asynchronous or quasi-synchronous) using synchronization register stages to a depth of X2P_DUAL_CLK_SYNC_DEPTH. ■ Single Clock: aclk and pclk are driven by the same clock signal. **Values:** ■ Dual Clock (0) ■ Single Clock (2) **Default Value:** Dual Clock **Enabled:** Always **Parameter Name:** X2P_CLK_MODE
Dual clock mode Synchronisation Depth	Number of synchronization register stages in the internal channel fifos for signals crossing clock domains between AXI and APB. When aclk and pclk are quasi-synchronous it should be possible to set this parameter to 0 to reduce the latency across the bridge. This parameter is enabled only if dual clock mode is selected. If dual clock mode is not selected this parameter is irrelevant. **Values:** 0, ~~1,~~ 2, 3 **Default Value:** 2 **Enabled:** X2P_CLK_MODE == 0 **Parameter Name:** X2P_DUAL_CLK_SYNC_DEPTH

3.3 APB Parameters

Table 3-3 APB Parameters

Label	Description
APB Parameters	
APB Address Width	Address bus width of the APB system to which the bridge is attached as an APB master. **Values:** 32 **Default Value:** 32 **Enabled:** Always **Parameter Name:** X2P_APB_ADDR_WIDTH
APB Data Width	Read and write data bus width of the APB system to which the bridge is attached as an APB master. This width must be equal or smaller than the width of the AXI data bus. **Values:** 8, 16, 32 **Default Value:** 32 **Enabled:** Always **Parameter Name:** X2P_APB_DATA_WIDTH
Number of APB Slave Ports	Number of APB Slave Ports. **Values:** 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16 **Default Value:** 4 **Enabled:** Always **Parameter Name:** X2P_NUM_APB_SLAVES

Table 3-3 APB Parameters (Continued)

Label	Description
Slave i	
Start Address for APB Slave i (for i = 0; i <= X2P_NUM_APB_SLAVES)	Start address for APB Slave 0. Following are the default values: ■ i = 0: 0x0000000000000400 ■ i = 1: 0x0000000000000800 ■ i = 2: 0x0000000000000c00 ■ i = 3: 0x0000000000001000 ■ i = 4: 0x0000000000001400 ■ i = 5: 0x0000000000001800 ■ i = 6: 0x0000000000001c00 ■ i = 7: 0x0000000000002000 ■ i = 8: 0x0000000000002400 ■ i = 9: 0x0000000000002800 ■ i = 10: 0x0000000000002c00 ■ i = 11: 0x0000000000003000 ■ i = 12: 0x0000000000003400 ■ i = 13: 0x0000000000003800 ■ i = 14: 0x0000000000003c00 ■ i = 15: 0x0000000000004000 **Values:** 0x0000000000000000, ..., (2**X2P_AXI_AW)-1024 **Default Value:** 0x0000000000000400 **Enabled:** Always **Parameter Name:** X2P_START_PADDR_S(i)

Table 3-3 APB Parameters (Continued)

Label	Description
End Address for APB Slave i (for i = 0; i <= X2P_NUM_APB_SLAVES)	End address for APB Slave 0. Following are the default values: ■ i = 0: 0x00000000000007ff ■ i = 1: 0x0000000000000bff ■ i = 2: 0x0000000000000fff ■ i = 3: 0x00000000000013ff ■ i = 4: 0x00000000000017ff ■ i = 5: 0x0000000000001bff ■ i = 6: 0x0000000000001fff ■ i = 7: 0x00000000000023ff ■ i = 8: 0x00000000000027ff ■ i = 9: 0x0000000000002bff ■ i = 10: 0x0000000000002fff ■ i = 11: 0x00000000000033ff ■ i = 12: 0x00000000000037ff ■ i = 13: 0x0000000000003bff ■ i = 14: 0x0000000000003fff ■ i = 15: 0x00000000000043ff **Values:** 0x00000000000003ff, ..., (2**X2P_AXI_AW)-1 **Default Value:** See Description **Enabled:** Always **Parameter Name:** X2P_END_PADDR_S(i)
Start Address of APB Slave i (Limited to 32 bits) (for i = 0; i <= X2P_NUM_APB_SLAVES)	Start Address for APB Slave #0 that the component will use to decode the addressed APB Slave. This parameter is non-editable and is used for visual purposes only in the coreConsultant GUI. **Values:** 0x00000000, ..., 0xfffffc00 **Default Value:** X2P_START_PADDR_S0 & 0x00000000ffffffff **Enabled:** 0 **Parameter Name:** X2P_START_PADDR_32_S(i)
End Address of APB Slave i (Limited to 32 bits) (for i = 0; i <= X2P_NUM_APB_SLAVES)	End Address for APB Slave #0 that the component will use to decode the addressed APB Slave. This parameter is non-editable and is used for visual purposes only in the coreConsultant GUI. **Values:** 0x000003ff, ..., 0xffffffff **Default Value:** X2P_END_PADDR_S0 & 0x00000000ffffffff **Enabled:** 0 **Parameter Name:** X2P_END_PADDR_32_S(i)

Table 3-3 APB Parameters (Continued)

Label	Description
APB3 (for i = 0; i <= X2P_NUM_APB_SLAVES)	AMBA 3 support for APB slave. Slave 0 has the additional ports PREADY and PSLVERR according to the AMBA3 specification. **Values:** ■ false (0) ■ true (1) **Default Value:** false **Enabled:** Always **Parameter Name:** X2P_IS_APB3_S(i)

3.4 Low Power Parameters

Table 3-4 Low Power Parameters

Label	Description
Low Power	
Low Power Interface Enable	If true, the low-power handshaking interface (csysreq, csysack and cactive signals) and associated logic is implemented. If false, support for low-power handshaking interface is not provided. **Values:** ■ false (0) ■ true (1) **Default Value:** false **Enabled:** Always **Parameter Name:** X2P_LOWPWR_HS_IF
Number of inactive clock cycles before component requests power down	Number of AXI clock cycles to wait before cactive signal de-asserts, when there are no pending transactions. Note that if csysreq de-asserts while waiting this number of cycles, cactive will immediately de-assert. If a new transaction is initiated during the wait period, the counting will be halted, cactive will not de-assert, and the counting will be re-initiated when there are no pending transactions. Available only if X2P_LOWPWR_HS_IF is true **Values:** 0, ..., 4294967295 **Default Value:** 0 **Enabled:** X2P_LOWPWR_HS_IF == 1 **Parameter Name:** X2P_LOWPWR_NOPX_CNT

4 Signal Descriptions

This chapter details all possible I/O signals in the controller. For configurable IP titles, your actual configuration might not contain all of these signals.

Inputs are on the left of the signal diagrams; outputs are on the right.

Attention: For configurable IP titles, do not use this document to determine the exact I/O footprint of the controller. It is for reference purposes only.

When you configure the controller in coreConsultant, you must access the I/O signals for your actual configuration at workspace/report/IO.html or workspace/report/IO.xml after you have completed the report creation activity. That report comes from the exact same source as this chapter but removes all the I/O signals that are not in your actual configuration. This does not apply to non-configurable IP titles. In addition, all parameter expressions are evaluated to actual values. Therefore, the widths might change depending on your actual configuration.

Some expressions might refer to TCL functions or procedures (sometimes identified as **<functionof>)** that coreConsultant uses to make calculations. The exact formula used by these TCL functions is not provided in this chapter. However, when you configure the controller in coreConsultant, all TCL functions and parameters are evaluated completely; and the resulting values are displayed where appropriate in the coreConsultant GUI reports.

In addition to describing the function of each signal, the signal descriptions in this chapter include the following information:

Active State: Indicates whether the signal is active high or active low. When a signal is not intended to be used in a particular application, then this signal needs to be tied or driven to the inactive state (opposite of the active state).

Registered: Indicates whether or not the signal is registered directly inside the IP boundary without intervening logic (excluding simple buffers). A value of No does not imply that the signal is not synchronous, only that there is some combinatorial logic between the signal's origin or destination register and the boundary of the controller. A value of N/A indicates that this information is not provided for this IP title.

Synchronous to: Indicates which clocks in the IP sample this input (drive for an output) when considering all possible configurations. A particular configuration might not have all of the clocks listed. This clock might not be the same as the clock that your application logic should use to clock (sample/drive) this pin. For more details, consult the clock section in the databook.

Exists: Names of configuration parameters that populate this signal in your configuration.

Validated by: Assertion or de-assertion of signals that validates the signal being described.

Attributes used with Synchronous To

- Clock name - The name of the clock that samples an input or drive and output.
- None - This attribute may be used for clock inputs, hard-coded outputs, feed-through (direct or combinatorial), dangling inputs, unused inputs and asynchronous outputs.
- Asynchronous - This attribute is used for asynchronous inputs and asynchronous resets.

The I/O signals are grouped as follows:

4.1 Clock and Resets Signals

Table 4-1 Clock and Resets Signals

Port Name	I/O	Description
aclk	I	AXI clock signal. All AXI interface input signals are sampled on the rising edge of aclk. **Exists:** Always **Synchronous To:** None **Registered:** N/A **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
pclk	I	APB clock signal. For more information about the clocking mode, refer to "Clock Adaptation" in the DW_axi_x2p Databook. **Exists:** (X2P_CLK_MODE != 2) **Synchronous To:** None **Registered:** N/A **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
aresetn	I	AXI reset signal. Used to reset the AXI portion of the DW_axi_x2p. The default value of this input is high to indicate normal operation. Asynchronous assertion, synchronous de-assertion. The reset must be synchronously de-asserted after rising edge of aclk. The DW_axi_x2p does not contain logic to perform this synchronization, so it must be provided externally. **Exists:** Always **Synchronous To:** None **Registered:** N/A **Power Domain:** SINGLE_DOMAIN **Active State:** Low

Table 4-1 Clock and Resets Signals (Continued)

Port Name	I/O	Description
presetn	I	APB reset signal. Asynchronous assertion, synchronous de-assertion. The reset must be synchronously de-asserted after rising edge of pclk. The DW_axi_x2p does not contain logic to perform this synchronization, so it must be provided externally. **Exists:** (X2P_CLK_MODE != 2) **Synchronous To:** None **Registered:** N/A **Power Domain:** SINGLE_DOMAIN **Active State:** Low

4.2 AXI Write Address Channel Signals

Table 4-2 AXI Write Address Channel Signals

Port Name	I/O	Description
awready	O	AXI write address ready. Indicates that slave is ready to accept address and associated control signals. ■ 0: Slave not ready ■ 1: Slave ready **Exists:** Always **Synchronous To:** aclk **Registered:** ((X2P_LOWPWR_HS_IF==0) || (X2P_LOWPWR_LEGACY_IF==1)) ?"Yes":"No" **Power Domain:** SINGLE_DOMAIN **Active State:** High
awaddr[(X2P_AXI_AW-1):0]	I	AXI write address. Specifies address of first transfer in write burst transaction. Associated control signals used to determine addresses of remaining transfers in burst. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

Table 4-2 AXI Write Address Channel Signals (Continued)

Port Name	I/O	Description
awvalid	I	AXI write address valid. Indicates valid write address and control information are available. Address and control information remain stable until awready signal is high. ■ 0: Address and control information not available ■ 1: Address and control in-formation available **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
awburst[1:0]	I	AXI Write Burst. Combined with size information, shows how address for each transfer with burst is calculated. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
awlock[(X2P_AXI_LTW-1):0]	I	AXI write lock. Provides additional information about characteristics of transfer. This signal is unused. It is included for interface consistency only. **Exists:** Always **Synchronous To:** None **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
awsize[2:0]	I	AXI write burst size. Indicates size of each transfer in burst. Byte lane strobes indicate exactly which byte lanes to update. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
awprot[2:0]	I	AXI write protection. Indicates normal, privileged, or secure protection level of transaction and whether transaction is data access or instruction access. This signal is unused. It is included for interface consistency only. **Exists:** Always **Synchronous To:** None **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

Table 4-2 AXI Write Address Channel Signals (Continued)

Port Name	I/O	Description
awid[(X2P_AXI_SIDW-1):0]	I	AXI write address identification. Gives identification tag for write address signals. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
awlen[(X2P_AXI_BLW-1):0]	I	AXI Burst length. Specifies exact number of transfers in burst; determines number of data transfers associated with address. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
awcache[3:0]	I	AXI write cache. Indicates bufferable, cacheable, write-through, and write-back attributes of transaction. This signal is unused. It is included for interface consistency only. **Exists:** Always **Synchronous To:** None **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

4.3 AXI Write Data Channel Signals

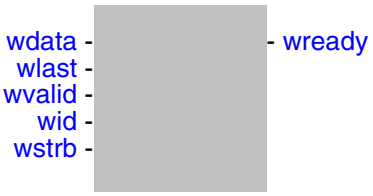


Table 4-3 AXI Write Data Channel Signals

Port Name	I/O	Description
wready	O	AXI write ready. Indicates that the slave can accept the write data. ■ 0: Slave not ready ■ 1: Slave ready **Exists:** Always **Synchronous To:** aclk **Registered:** ((X2P_LOWPWR_HS_IF==0) || (X2P_LOWPWR_LEGACY_IF==1)) ?"Yes":"No" **Power Domain:** SINGLE_DOMAIN **Active State:** High
wdata[(X2P_AXI_DW-1):0]	I	AXI write data. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
wlast	I	AXI write last. Indicates the last transfer in a write burst. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High

Table 4-3 AXI Write Data Channel Signals (Continued)

Port Name	I/O	Description
wvalid	I	AXI write valid. This signal indicates that valid write data and strobes are available. ■ 0: Write data and strobes not available ■ 1: Write data and strobes available **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
wid[(X2P_AXI_SIDW-1):0]	I	AMBA AXI3 write identification tag of write data transfer. This value must match awid value of write transaction. This signal is unused. It is included for interface consistency only. **Exists:** X2P_AXI_INTERFACE_TYPE == 0 **Synchronous To:** None **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
wstrb[((X2P_AXI_DW/8)-1):0]	I	AXI write strobes. Indicates which byte lanes to update in memory. There is one write strobe for each eight bits of the write data bus. The wstrb signal is associated with wdata[(8 x n)+7:(8 x n)]. Only the strobes for the configured data width are generated. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High

4.4 AXI Write Response Channel Signals

Table 4-4 AXI Write Response Channel Signals

Port Name	I/O	Description
bresp[1:0]	O	AXI write response. Indicates status of write transaction. The responses supported by the DW_axi_x2p are OKAY and SLVERR. This signal is driven from the RESP buffer. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
bid[(X2P_AXI_SIDW-1):0]	O	AXI write response identification tag. This value must match awid value of write transaction to which slave responds. This signal is driven from the RESP buffer. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
bvalid	O	AXI write response valid. Indicates that valid write response is available. This signal is driven from the RESP buffer status output. ■ 0: Write response not available ■ 1: Write response available **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High

Table 4-4 AXI Write Response Channel Signals (Continued)

Port Name	I/O	Description
bready	I	AXI response ready. Indicates master can accept response information. ■ 0: Master not ready ■ 1: Master ready **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High

4.5 AXI Read Address Channel Signals

Table 4-5 AXI Read Address Channel Signals

Port Name	I/O	Description
aready	O	AXI read address ready. Indicates that slave is ready to accept address and associated control signals. ■ 0: Slave not ready ■ 1: Slave ready **Exists:** Always **Synchronous To:** aclk **Registered:** ((X2P_LOWPWR_HS_IF==0) || (X2P_LOWPWR_LEGACY_IF==1)) ?"Yes":"No" **Power Domain:** SINGLE_DOMAIN **Active State:** High
araddr[(X2P_AXI_AW-1):0]	I	AXI read address. Specifies address of first transfer in read burst transaction. Associated control signals used to determine addresses of remaining transfers in burst. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

Table 4-5 AXI Read Address Channel Signals (Continued)

Port Name	I/O	Description
arvalid	I	AXI read address valid. Indicates valid read address and control information are available. When high, indicates read is valid and remains stable until arready is high. ■ 0: Address and control information not valid. ■ 1: Address and control infor-mation valid. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
arburst[1:0]	I	AXI read Burst. Combined with size information, shows how address for each transfer with burst is calculated. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
arlock[(X2P_AXI_LTW-1):0]	I	AXI read lock. Encoded value indicates whether the transfer is a normal, exclusive, or locked access. This signal is unused. It is included for interface consistency only. **Exists:** Always **Synchronous To:** None **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
arsize[2:0]	I	AXI read burst size. Indicates size of each transfer in burst. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
arprot[2:0]	I	AXI read protection. Indicates normal, privileged, or secure protection level of transaction and whether transaction is data access or instruction access. This signal is unused. It is included for interface consistency only. **Exists:** Always **Synchronous To:** None **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

Table 4-5 AXI Read Address Channel Signals (Continued)

Port Name	I/O	Description
arid[(X2P_AXI_SIDW-1):0]	I	AXI read address identification. Gives identification tag for read address signals. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
arlen[(X2P_AXI_BLW-1):0]	I	AXI read burst length. Gives exact number of transfers in the burst and determines the number of data transfers associated with an address. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
arcache[3:0]	I	AXI read cache. Indicates bufferable, cacheable, read-through, and read-back attributes of transaction. This signal is unused. It is included for interface consistency only. **Exists:** Always **Synchronous To:** None **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

4.6 AXI Read Data Channel Signals

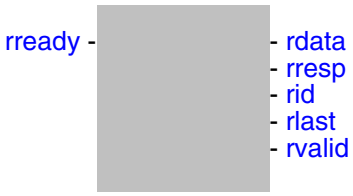


Table 4-6 AXI Read Data Channel Signals

Port Name	I/O	Description
rdata[(X2P_AXI_DW-1):0]	O	AXI read data. This signal is driven from the RDATA buffer. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
rresp[1:0]	O	AXI read response. Indicates status of read transaction. The responses supported by the DW_axi_x2p are OKAY and SLVERR. This signal is driven from the RDATA buffer. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
rid[(X2P_AXI_SIDW-1):0]	O	AXI Read identification tag. This value is generated by slave and must match arid value of read transaction to which it responds. This signal is driven from the RDATA buffer. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
rlast	O	AXI read last. Indicates last transfer in read burst. This signal is driven from the RDATA buffer. **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High

Table 4-6 AXI Read Data Channel Signals (Continued)

Port Name	I/O	Description
rvalid	O	AXI read valid. Indicates that required read data is available and read transfer can complete. This signal is driven from the RDATA buffer. ■ 0: Read data not available ■ 1: Read data available **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
rready	I	AXI read ready. Indicates master can accept read data and response information. ■ 0: Master not ready ■ 1: Master ready **Exists:** Always **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High

4.7 APB Inputs (for x = 0; x <= X2P_NUM_APB_SLAVES) Signals

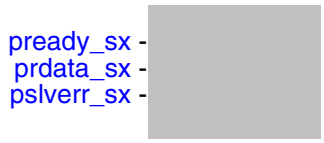
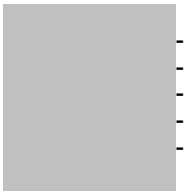


Table 4-7 APB Inputs (for x = 0; x <= X2P_NUM_APB_SLAVES) Signals

Port Name	I/O	Description
pready_sx	I	Indicates whether a request cycle was accepted. Supported only on AMBA 3 APB devices. **Exists:** (X2P_IS_APB3_Sx==1) **Synchronous To:** (X2P_CLK_MODE == 2)? "aclk" : "pclk" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
prdata_sx[(X2P_APB_DATA_WIDTH-1):0]	I	Read data from the APB slave. **Exists:** Always **Synchronous To:** (X2P_CLK_MODE == 2)? "aclk" : "pclk" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
pslverr_sx	I	Flag for the slave error response from APB. Support for AMBA 3 APB devices. **Exists:** (X2P_IS_APB3_Sx==1) **Synchronous To:** (X2P_CLK_MODE == 2)? "aclk" : "pclk" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High

4.8 APB Outputs Signals



- psel_sx (for x = 0; x <= X2P_NUM_APB_SLAVES-1)
- paddr
- penable
- pwdata
- pwrite

Table 4-8 APB Outputs Signals

Port Name	I/O	Description
psel_sx (for x = 0; x <= X2P_NUM_APB_SLAVES-1)	O	Selects the APB slave. Must be active for requests to be accepted. **Exists:** Always **Synchronous To:** (X2P_CLK_MODE == 2)? "aclk" : "pclk" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
paddr[(X2P_APB_ADDR_WIDTH-1):0]	O	Address on APB. **Exists:** Always **Synchronous To:** (X2P_CLK_MODE == 2)? "aclk" : "pclk" **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** N/A
penable	O	Indicates access phase. **Exists:** Always **Synchronous To:** (X2P_CLK_MODE == 2)? "aclk" : "pclk" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
pwdata[(X2P_APB_DATA_WIDTH-1):0]	O	Write data on APB Slave. **Exists:** Always **Synchronous To:** (X2P_CLK_MODE == 2)? "aclk" : "pclk" **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** N/A

Table 4-8 APB Outputs Signals (Continued)

Port Name	I/O	Description
pwrite	O	Write Select signal when active request is write request. **Exists:** Always **Synchronous To:** (X2P_CLK_MODE == 2)? "aclk" : "pclk" **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High

4.9 AXI Low Power Interface Signals

Table 4-9 AXI Low Power Interface Signals

Port Name	I/O	Description
nopx	O	Low power signal. **Exists:** ((X2P_LOWPWR_HS_IF==1)&&(X2P_LOWPWR_LEGACY_IF==1)) **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** Low
cactive	O	Clock active request. De-asserted by DW_axi_x2p to tell system low-power controller (LPC) that clock can be removed. ■ 1: Peripheral clock required ■ 0: Peripheral clock not required **Note**: Clock must be removed on the positive edge after cactive and csysack signals are sampled low (0). **Exists:** ((X2P_LOWPWR_HS_IF==1)&&(X2P_LOWPWR_LEGACY_IF==0)) **Synchronous To:** aclk **Registered:** No **Power Domain:** SINGLE_DOMAIN **Active State:** High
csysack	O	Low power request acknowledgement. ■ De-asserted by DW_axi_x2p to acknowledge request to enter low-power state. ■ Asserted by DW_axi_x2p to acknowledge request to exit low-power state. **Exists:** ((X2P_LOWPWR_HS_IF==1)&&(X2P_LOWPWR_LEGACY_IF==0)) **Synchronous To:** aclk **Registered:** Yes **Power Domain:** SINGLE_DOMAIN **Active State:** High

Table 4-9 AXI Low Power Interface Signals (Continued)

Port Name	I/O	Description
csysreq	I	System low-power request from system clock controller. ■ De-asserted by system low power controller (LPC) to initiate entry into a low-power state. ■ Asserted to initiate exit from a low-power state. **Exists:** ((X2P_LOWPWR_HS_IF==1)&&(X2P_LOWPWR_LEGACY_IF==0)) **Synchronous To:** aclk **Registered:** (X2P_LOWPWR_NOPX_CNT == 0) ? "Yes":"No" **Power Domain:** SINGLE_DOMAIN **Active State:** High

5 Internal Parameter Descriptions

Provides a description of the internal parameters that might be indirectly referenced in expressions in the Signals, Parameters, or Registers chapters. These parameters are not visible in the coreConsultant GUI and most of them are derived automatically from visible parameters. **You must not set any of these parameters directly.**

Some expressions might refer to TCL functions or procedures (sometimes identified as **function_of**) that coreConsultant uses to make calculations. The exact formula used by these TCL functions is not provided in this chapter. However, when you configure the core in coreConsultant, all TCL functions and parameters are evaluated completely; and the resulting values are displayed where appropriate in the coreConsultant GUI reports.

Table 5-1 Internal Parameters

Parameter Name	Equals To
X2P_AXI_LTW	{ [~~function_of:~~ ::DW_axi_x2p::calc_lock_width X2P_AXI_INTERFACE_TYPE] }
X2P_LOWPWR_LEGACY_IF	0

6 Verification

This chapter provides an overview of the testbench available for the DW_axi_x2p verification. After the DW_axi_x2p has been configured and the verification is setup, simulations can be run automatically. For information on running simulations for DW_axi_x2p in coreAssembler or coreConsultant, see the "Running the Simulation" section in the user guide.

Note The DW_axi_x2p verification testbench is built using Synopsys SVT Verification IP (VIP). Ensure that you have the supported version of the VIP components for this release, otherwise, you may experience some tool compatibility problems. For more information about supported tools in this release, see the "Supported Versions of Tools and Libraries" section in the installation guide.

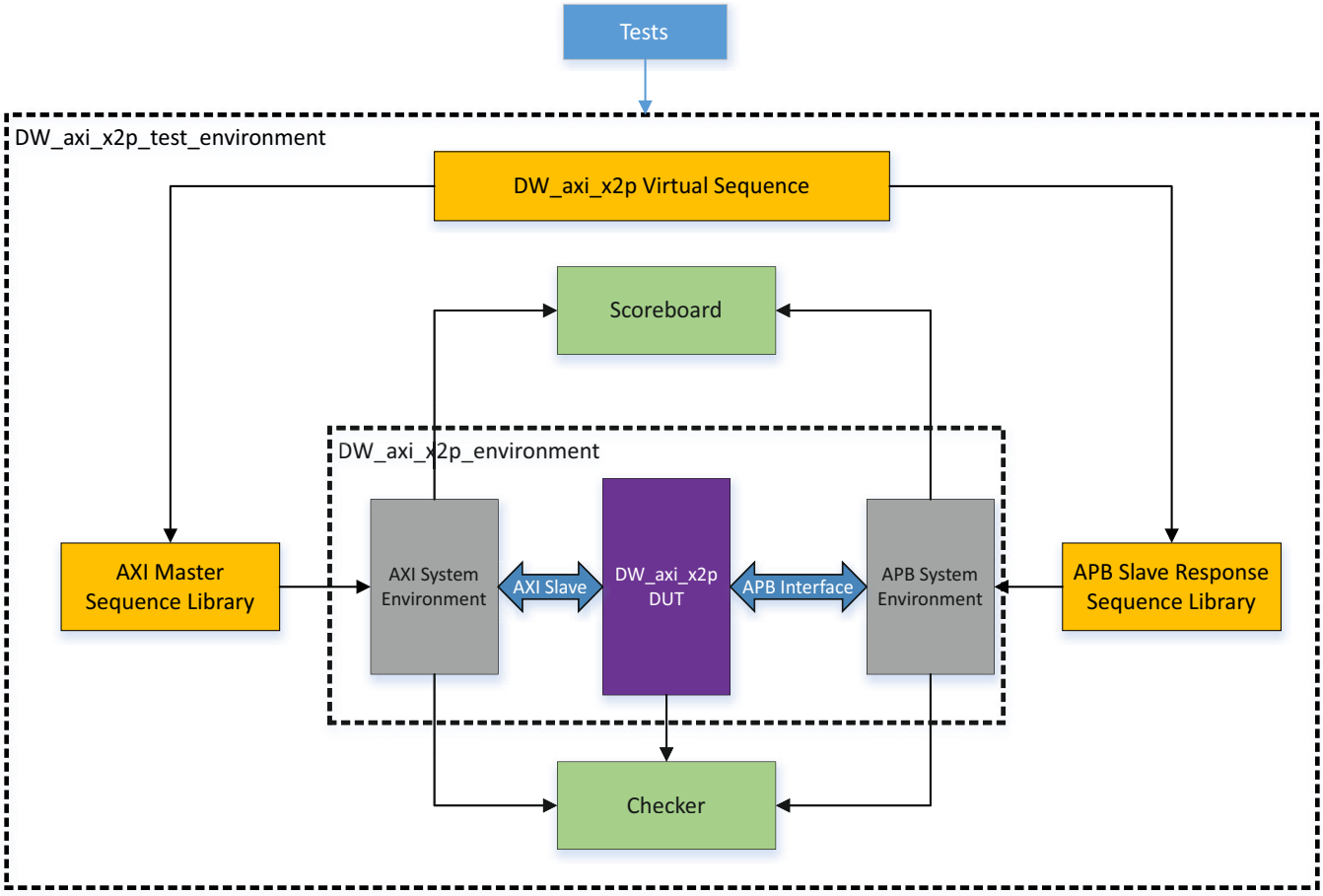
This chapter discusses the following sections:

6.1 Verification Environment

DW_axi_x2p is verified using a UVM-methodology-based constrained random verification environment. The environment can generate random scenarios and the test case has hooks to control the scenarios to be generated.

Figure 6-1 shows the verification environment of the DW_axi_x2p testbench:

Figure 6-1 DW_axi_x2p UVM Verification Environment



The testbench consists of the following elements:

- Testbench makes use of the standard SVT VIP for the protocol interfaces:
  - AMBA SVT VIP
    - AXI VIP for the interface with AXI Master Data transfer interface
    - AXI Master VIP model is used to generate random sequences
    - APB VIP model is used to generate response for the incoming APB transfers. The VIP are auto-configurable for the APB2/APB3/APB4 traffic based on the DUT configuration

6.2 Testbench Directories and Files

The DW_axi_x2p verification environment contains the following directories and associated files.

Table 6-1 shows the various directories and associated files:

Table 6-1 DW_axi_x2p Testbench Directory Structure

Directory	Description
<configured workspace>/sim/testbench	Top level testbench module (test_top.sv) and the DUT to the testbench wrapper (dut_sv_wrapper.sv) exist in this folder.
<configured workspace>/sim/testbench/env	Contains testbench files. For example, scoreboard, sequences, VIP, environment, sequencers, and agents.
<configured workspace>/sim/	Primarily contains the supporting files to compile and run the simulation. After the completion of the simulation, the log files are present here.
<configured workspace>/sim/test_*	Contains individual test cases. After the completion of the simulation, the test specific log files and if applicable the waveform files are stored here.

6.3 Packaged Testcases

The simulation environment that comes as a package file includes some demonstrative tests. Some or all of the packaged demonstrative tests, depending upon their applicability to the chosen configuration are displayed in **Setup and Run Simulations > Testcases** in the coreConsultant GUI.

The associated test cases shipped and their description is explained in Table 6-2:

Table 6-2 DW_axi_x2p Test Description

Test Name	Test Description
test_x2p_random	This test is aimed at generating random traffic which is AXI and APB compliant.The traffic is generated randomly based on DW_axi_x2p configuration and VIP is auto-constrained to generate the traffic within the protocol limits. The traffic is generated in an outstanding fashion and sent towards DW_axi_x2p. AXI Master Sequence from the VIP generates various possible AXI transfers for both Write and Read requests in parallel. The AXI transfer control attributes are randomized within the protocol and as per the DW_axi_x2p requirement. The Primary port is monitored for the correctness of the AXI protocol across different IP configuration. APB Slave Sequence from the VIP generates various possible APB responses for the requests from secondary port of the DW_axi_x2p. The response, delay, and other attributes are generated randomly. The secondary port is monitored for the correctness of the APB protocol across different IP configuration.

7 Integration Considerations

After you have configured, tested, and synthesized your component with the coreTools flow, you can integrate the component into your own design environment.

7.1 Hardware Considerations

The following subsections discuss considerations you should take into account when integrating the DW_axi_x2p in your design.

7.1.1 Usage Requirements

Signals from the AXI read address channel, write address channel, and write data channel to the DW_axi_x2p are sent directly to the DW_axi_x2p FIFO memories.

When interfacing to the DW_axi_x2p in order to best ensure that the setup and hold times for the configured FIFOs are met, the designer should consider registering the following signals:

- awaddr
- awlen
- awsize
- awburst
- araddr
- arlen
- arsize
- arburst
- wlast
- wdata
- wstrb

For more information on registering signals, refer to refer to the AXI register slice in the *DesignWare DW_axi_rs Databook*.

Signals to the AXI read data channel and the write response buffer are output directly from the FIFOs on their respective AXI channels. In this case, the designer should consider that these are accesses from the FIFO memories and that their delays vary, depending on the selected configuration.

7.2 Performance

This section discusses performance and the hardware configuration parameters that affect performance of the DW_axi_x2p.

7.2.1 Power Consumption, Frequency, Area, and DFT Coverage

Table 7-1 provides information about the synthesis results (power consumption, frequency, area) and DFT coverage of the DW_axi_x2p using the industry standard 16nm technology library.

Table 7-1 Synthesis Results for DW_axi_x2p

Configuration	Operating Frequency	Gate Count	Power Consumption		Tetramax (%)		SpyGlass StuckAtCov(%)
			Static Power	Dynamic Power	StuckAt Test	Transition	
Default Configuration	aclk=400 MHz pclk=100 MHz	7953	6 nW	0.219 mW	99.75	96.5	98.2
Typical Configuration - 1 X2P_APB_DATA_WIDTH = 32 X2P_AXI_AW = 32 X2P_AXI_BLW = 8 X2P_AXI_DW = 32 X2P_AXI_ENDIANNESS = 1 X2P_AXI_SIDW = 4 X2P_CLK_MODE = 2 X2P_CMD_QUEUE_DEPTH = 8 X2P_NUM_APB_SLAVES = 16 X2P_READ_BUFFER_DEPTH = 8 X2P_WRITE_BUFFER_DEPTH = 8 X2P_WRITE_RESP_BUFFER_DEPTH = 8	aclk=400 MHz	15962	10 nW	0.659 mW	99.93	98.22	98.8

Table 7-1 Synthesis Results for DW_axi_x2p

Configuration	Operating Frequency	Gate Count	Power Consumption		Tetramax (%)		SpyGlass StuckAtCov(%)
			Static Power	Dynamic Power	StuckAt Test	Transition	
Typical Configuration - 2 AXI_TEST_INACTIVE_SIGNALS = 8 SIM_A_CLK_PERIOD = 75 SIM_B_CLK_PERIOD = 100 X2P_APB_DATA_WIDTH = 16 X2P_AXI_AW = 36 X2P_AXI_BLW = 4 X2P_AXI_DW = 256 X2P_AXI_ENDIANNESS = 1 X2P_AXI_INTERFACE_TYPE = 1 X2P_ALLOW_SPARSE_TRANSFER = 1 X2P_AXI_SIDW = 7 X2P_CLK_MODE = 0 X2P_CMD_QUEUE_DEPTH = 32 X2P_DUAL_CLK_SYNC_DEPTH = 3 X2P_END_PADDR_S0 = 0x0000000000001fff X2P_END_PADDR_S1 = 0x0000000000002fff X2P_END_PADDR_S10 = 0x0000000000000bff X2P_END_PADDR_S11 = 0x00000000000047ff X2P_END_PADDR_S12 = 0x00000000000023ff X2P_END_PADDR_S13 = 0x0000000000003bff X2P_END_PADDR_S2 = 0x00000000000033ff X2P_END_PADDR_S3 = 0x0000000000004fff X2P_END_PADDR_S4 = 0x00000000000053ff X2P_END_PADDR_S5 = 0x00000000000017ff X2P_END_PADDR_S6 = 0x0000000000000fff X2P_END_PADDR_S7 = 0x0000000000003fff X2P_END_PADDR_S8 = 0x00000000000003ff X2P_END_PADDR_S9 = 0x0000000000002bff	aclk=400 MHz pclk=100 MHz	251574	200 nW	8.870 mW	99.99	92.88	100

Table 7-1 Synthesis Results for DW_axi_x2p

Configuration	Operating Frequency	Gate Count	Power Consumption		Tetramax (%)		SpyGlass StuckAtCov(%)
			Static Power	Dynamic Power	StuckAt Test	Transition	
X2P_IS_APB3_S0 = 1 X2P_IS_APB3_S1 = 1 X2P_IS_APB3_S10 = 1 X2P_IS_APB3_S3 = 1 X2P_IS_APB3_S5 = 1 X2P_IS_APB3_S7 = 1 X2P_LOWPWR_HS_IF = 1 X2P_LOWPWR_LEGACY_IF = 0 X2P_LOWPWR_NOPX_CNT = 15 X2P_NUM_APB_SLAVES = 14 X2P_READ_BUFFER_DEPTH = 8 X2P_START_PADDR_S0 = 0x0000000000001c00 X2P_START_PADDR_S1 = 0x0000000000002c00 X2P_START_PADDR_S10 = 0x0000000000000800 X2P_START_PADDR_S11 = 0x0000000000004400 X2P_START_PADDR_S12 = 0x0000000000002000 X2P_START_PADDR_S13 = 0x0000000000003800 X2P_START_PADDR_S2 = 0x0000000000003000 X2P_START_PADDR_S3 = 0x0000000000004c00 X2P_START_PADDR_S4 = 0x0000000000005000 X2P_START_PADDR_S5 = 0x0000000000001400 X2P_START_PADDR_S6 = 0x0000000000000c00 X2P_START_PADDR_S7 = 0x0000000000003c00 X2P_START_PADDR_S8 = 0x0000000000000000 X2P_START_PADDR_S9 = 0x0000000000002800 X2P_WRITE_BUFFER_DEPTH = 64 X2P_WRITE_RESP_BUFFER_DEPTH = 8							

7.2.2 Latency

Table 7-2 and Table 7-3 provide information about the latency of the DW_axi_x2p and how it affects performance.

Table 7-2 Best Case, One Clock to Both Sides of FIFO Write Data Path from WVALID to PSEL = 4 Clocks

Cycle	Description of Clock Cycle	Latency
T1	Clock from WVALID to push the Write Data Buffer	+ 1 clock
T2	Clock through the write buffer FIFO; best case; same clock	+ 1 clock
T3	Write data byte swapped and then registered	+ 1 clock
T4	Write data selected; registered as pwdata	+ 1 clock

Table 7-3 Read Data Path from CMD on AXI to First RVALID (1:1 size)

Cycle	Description of Clock Cycle	Latency
T1	AWVALID to push the CMD queue	+ 1 clock
T2	Register address and send as paddr	+ 1 clock
T3	APB transfer read data packed and registered	+ 2 clocks[a]
T4	Packed data pushed through FIFO and RVALID set active	+ 1 clock[b]

a. For each APB transfer, clocks = AXI SIZE/(APB data width) + APB waits.
b. For each APB transfer, clocks = AXI SIZE/(APB data width)

A Basic Core Module (BCM) Library

The Basic Core Module (BCM) Library is a library of commonly used blocks for the Synopsys DesignWare IP development. These BCMs are configurable on an instance-by-instance basis and, for the majority of BCM designs, there is an equivalent (or nearly equivalent) DesignWare Building Block (DWBB) component.

This chapter provides more information about the BCMs used in DW_axi_x2p.

Note The DesignWare Building Blocks (DWBB) contains several synchronizer components with functionality similar to methods documented in this appendix. For more information about the DWBB synchronizer components go to:
https://www.synopsys.com/dw/buildingblock.php

A.1 BCM Library Components

Table A-1 describes the list of BCM library components used in DW_axi_x2p.

Table A-1 List of BCM Library Components used in the Design

BCM Module Name	BCM Description	DWBB Equivalent
DW_axi_x2p_bcm05	FIFO controller interface for one of two clock domains instantiated in DW_axi_x2p_bcm07	DW_fifoctl_if Submodule of DW_fifoctl_s2_sf
DW_axi_x2p_bcm06	Synchronous (Single Clock) FIFO Controller with Dynamic Flags	DW_fifoctl_s1_df
DW_axi_x2p_bcm07	Synchronous (Dual-Clock) FIFO Controller with Static Flags	DW_fifoctl_s2_sf
DW_axi_x2p_bcm21	Single Clock Data Bus Synchronizer	DW_sync
DW_axi_x2p_bcm57	Synchronous Write-Port, Asynchronous. Read-Port RAM (Flip-Flop-Based)	DW_ram_r_w_s_dff

A.2 Synchronizer Methods

This section also describes the synchronizer methods (blocks of synchronizer functionality) that are used in the DW_axi_x2p to cross clock boundaries.

This section contains the following:

A.2.1 Synchronizers used in DW_axi_x2p

Each of the synchronizers and synchronizer sub-modules are comprised of verified DesignWare Basic Core (BCM) RTL designs. The BCM synchronizer designs are identified by the synchronizer type. The corresponding RTL files comprising the BCM synchronizers used in the DW_axi_x2p are listed and cross referenced to the synchronizer type in Table A-2. Note that certain BCM modules are contained in other BCM modules, as they are used in a building block fashion.

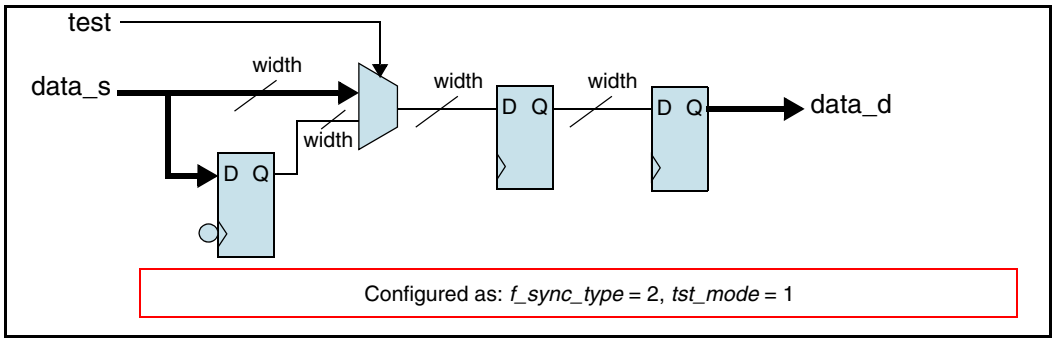
Table A-2 Synchronizer used in DW_axi_x2p

Synchronizer module file	Sub Module File	Synchronizer Type and Number
DW_axi_x2p_bcm21.v		Synchronizer 1: Simple Multiple register synchronizer
DW_axi_x2p_bcm07.v	DW_axi_x2p_bcm05.v DW_axi_x2p_bcm21.v	Synchronizer 2: Synchronous dual clock FIFO Controller with Static Flags

A.2.2 Synchronizer 1: Simple Double Register Synchronizer (DW_axi_x2p)

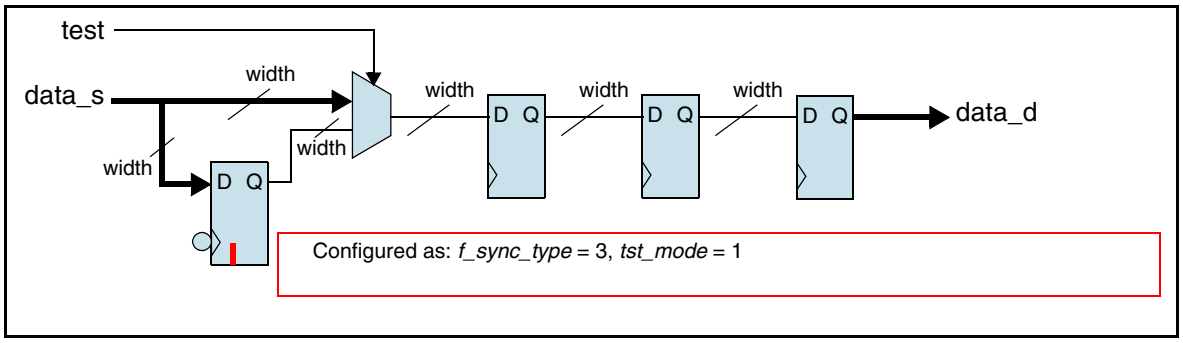
This is a single clock data bus synchronizer for synchronizing data that crosses asynchronous clock boundaries. The synchronization scheme depends on core configuration. If aclk and pclk are asynchronous (X2P_CLK_MODE=0) then DW_axi_x2p_bcm21 is instantiated inside the core for synchronization. The number of stages of synchronization is configurable through the parameter X2P_DUAL_CLK_SYNC_DEPTH. The following example shows the two stage synchronization process (Figure A-1) both using positive edge of clock.

Figure A-1 Block diagram of Synchronizer 1 with Two Stage Synchronization (both positive edge)



The following example shows the three stage synchronization process (Figure A-2) both using positive edge of clock.

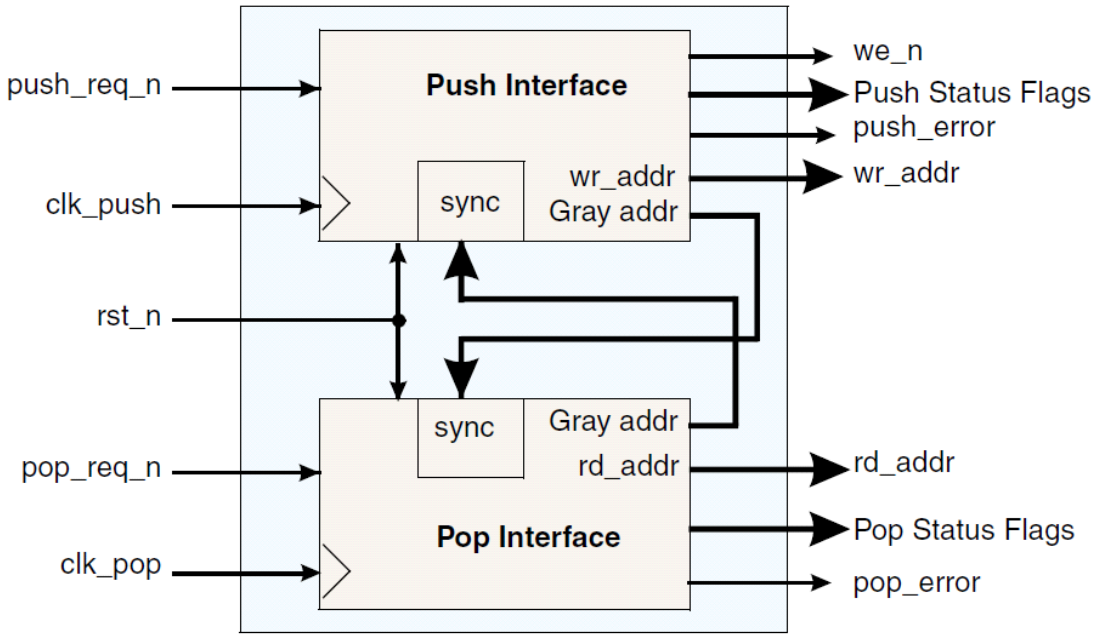
Figure A-2 Block diagram of Synchronizer 1 with Three Stage Synchronization (both positive edge)



A.2.3 Synchronizer 2: Synchronous (Dual-clock) FIFO Controller with Static Flags (DW_axi_x2p)

DW_axi_x2p_bcm07 is a dual independent clock FIFO RAM controller. It is designed to interface with a dual-port synchronous RAM. The FIFO controller provides address generation, write-enable logic, flag logic, and operational error detection logic. Figure A-3 shows the block diagram of Synchronizer 2.

Figure A-3 Synchronizer 2 Block diagram



B
Glossary

application design	Overall chip-level design into which a subsystem or subsystems are integrated.
BFM	Bus-Functional Model — A simulation model used for early hardware debug. A BFM simulates the bus cycles of a device and models device pins, as well as certain on-chip functions. See also Full-Functional Model.
big-endian	Data format in which most significant byte comes first; normal order of bytes in a word.
blocked command stream	A command stream that is blocked due to a blocking command issued to that stream; see also command stream, blocking command, and non-blocking command.
blocking command	A command that prevents a testbench from advancing to next testbench statement until this command executes in model. Blocking commands typically return data to the testbench from the model.
command channel	Manages command streams. Models with multiple command channels execute command streams independently of each other to provide full-duplex mode function.
command stream	The communication channel between the testbench and the model.
component	A generic term that can refer to any synthesizable IP or verification IP in the DesignWare Library. In the context of synthesizable IP, this is a configurable block that can be instantiated as a single entity (VHDL) or module (Verilog) in a design.
configuration	The act of specifying parameters for a core prior to synthesis; can also be used in the context of VIP.
configuration intent	Range of values allowed for each parameter associated with a reusable core.
cycle command	A command that executes and causes HDL simulation time to advance.
decoder	Software or hardware subsystem that translates from and “encoded” format back to standard format.

design context	Aspects of a component or subsystem target environment that affect the synthesis of the component or subsystem.
design creation	The process of capturing a design as parameterized RTL.
DesignWare Library	A collection of synthesizable IP and verification IP components that is authorized by a single DesignWare license. Products include SmartModels, VMT model suites, DesignWare Memory Models, Building Block IP, and the DesignWareSynthesizable Components for AMBA.
dual role device	Device having the capabilities of function and host (limited).
endian	Ordering of bytes in a multi-byte word; see also little-endian and big-endian.
Full-Functional Mode	A simulation model that describes the complete range of device behavior, including code execution. See also BFM.
GPIO	General Purpose Input Output.
GTECH	A generic technology view used for RTL simulation of encrypted source code by non-Synopsys simulators.
hard IP	Non-synthesizable implementation IP.
HDL	Hardware Description Language – examples include Verilog and VHDL.
IIP	Implementation Intellectual Property — A generic term for synthesizable HDL and non-synthesizable “hard” IP in all of its forms (coreKit, component, core, MacroCell, and so on).
implementation view	The RTL for a core. You can simulate, synthesize, and implement this view of a core in a real chip.
instantiate	The act of placing a core or model into a design.
interface	Set of ports and parameters that defines a connection point to a component.
IP	Intellectual property — A term that encompasses simulation models and synthesizable blocks of HDL code.
little-endian	Data format in which the least-significant byte comes first.
master	Device or model that initiates and controls another device or peripheral.
model	A Verification IP component or a Design View of a core.
monitor	A device or model that gathers performance statistics of a system.
non-blocking command	A testbench command that advances to the next testbench statement without waiting for the command to complete.
peripheral	Generally refers to a small core that has a bus connection, specifically an APB interface.

RTL	Register Transfer Level. A higher level of abstraction that implies a certain gate-level structure. Synthesis of RTL code yields a gate-level design.
SDRAM	Synchronous Dynamic Random Access Memory; high-speed DRAM adds a separate clock signal to control signals.
SDRAM controller	A memory controller with specific connections for SDRAMs.
slave	Device or model that is controlled by and responds to a master.
SoC	System on a chip.
soft IP	Any implementation IP that is configurable. Generally referred to as synthesizable IP.
sparse data	Data bus data which is individually byte-enabled. The AXI bus allows sparse data transfers using the WSTRB signal, each byte lane individually enabled. The AHB bus does not allow sparse data transfers.
static controller	Memory controller with specific connections for Static memories such as asynchronous SRAMs, Flash memory, and ROMs.
synthesis intent	Attributes that a core developer applies to a top-level design, ports, and core.
synthesizable IP	A type of Implementation IP that can be mapped to a target technology through synthesis. Sometimes referred to as Soft IP.
technology-independent	Design that allows the technology (that is, the library that implements the gate and via widths for gates) to be specified later during synthesis.
Testsuite Regression Environment (TRE)	A collection of files for stand-alone verification of the configured component. The files, tests, and functionality vary from component to component.
VIP	Verification Intellectual Property — A generic term for a simulation model in any form, including a Design View.
wrap, wrapper	Code, usually VHDL or Verilog, that surrounds a design or model, allowing easier interfacing. Usually requires an extra, sometimes automated, step to create the wrapper.
zero-cycle command	A command that executes without HDL simulation time advancing.

Index

Z