



DesignWare® DW_apb Databook

DW_apb – *Product Code*

Copyright Notice and Proprietary Information

© 2020 Synopsys, Inc. All rights reserved. This Synopsys software and all associated documentation are proprietary to Synopsys, Inc. and may only be used pursuant to the terms and conditions of a written license agreement with Synopsys, Inc. All other use, reproduction, modification, or distribution of the Synopsys software or the associated documentation is strictly prohibited.

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	5
Preface	9
Organization	9
Related Documentation	9
Web Resources	10
Customer Support	10
Product Code	11
Chapter 1	
Product Overview	13
1.1 DesignWare System Overview	13
1.1.1 DesignWare System Block Diagram	13
1.2 General Product Description	15
1.3 Features	15
1.3.1 Notes and Restrictions	16
1.3.2 Features Not Supported	16
1.4 Standards Compliance	16
1.5 Verification Environment Overview	16
1.6 Licenses	17
1.7 Where To Go From Here	17
Chapter 2	
Functional Description	19
2.1 Overview	19
2.1.1 Block Diagram	19
2.2 Transfers	20
2.2.1 Burst Transfers	21
2.3 PCLK versus HCLK	21
2.4 Optional External Decoder	21
2.5 Endianness	21
2.6 APB Slave Interface	22
2.7 Memory Map	23
2.8 Backward Compatibility with AMBA 2 APB and AMBA 3 APB	23
2.9 Timing Diagrams	23
2.10 Back-to-Back Transfer Support on an APB Interface	35
2.11 APB4 Protocol Feature	37
2.11.1 Write Strobing	37
2.11.2 Protection	38
2.12 AMBA 5 AHB Features	38

2.12.1 Secure Transfers	38
2.12.2 Endianness	39
2.12.3 User Signals	42
Chapter 3	
Parameter Descriptions	49
3.1 Top Level Parameters	50
3.2 User Signal Configuration Parameters	53
3.3 Address Map Parameters	55
Chapter 4	
Signal Descriptions	57
4.1 Clocks and Resets Signals	59
4.2 AHB Slave Interface Signals	60
4.3 APB Interface Signals	64
Chapter 5	
Verification	69
5.1 Verification Environment	69
5.2 Testbench Directories and Files	71
5.3 Packaged Testcases	72
Chapter 6	
Integration Considerations	73
6.1 Performance	74
6.1.1 Power Consumption, Frequency, Area and DFT Coverage	74
6.2 Accessing Top-level Constraints	75
6.3 Reading and Writing from an APB Slave	75
6.3.1 Reading From Unused Locations	75
6.3.2 32-bit Bus System	76
6.3.3 16-bit Bus System	77
6.3.4 8-bit Bus System	77
6.4 Write Timing Operation	77
6.5 Read Timing Operation	79
6.6 Coherency	79
6.6.1 Writing Coherently	80
6.6.2 Reading Coherently	86
6.7 Timing Exceptions	89
Appendix A	
DesignWare Constants	91
Chapter B	
Internal Parameter Descriptions	93
Appendix C	
Glossary	95

Revision History

This section tracks the significant documentation changes that occur from release-to-release and during a release from version 1.02d onward.

Version	Date	Description
3.03a	December 2020	Added <ul style="list-style-type: none"> ■ “AMBA 5 AHB Features” on page 38 ■ “Timing Exceptions” on page 89 Updated: <ul style="list-style-type: none"> ■ Version changed for 2020.12a release ■ Chapter 5, “Verification” ■ “Performance” on page 74 ■ “Parameter Descriptions” on page 49, “Signal Descriptions” on page 57, and “Internal Parameter Descriptions” on page 93 are auto-extracted with change bars from the RTL Removed: <ul style="list-style-type: none"> ■ Index chapter
3.02a	July 2018	Updated: <ul style="list-style-type: none"> ■ Version changed for 2018.07a release ■ “Performance” on page 74 ■ “Parameter Descriptions” on page 49, “Signal Descriptions” on page 57, and “Internal Parameter Descriptions” on page 93 are auto-extracted with change bars from the RTL Removed: <ul style="list-style-type: none"> ■ Chapter 2, “Building and Verifying a Component or Subsystem” and added the contents in the newly created user guide.

Version	Date	Description
3.01a	October 2016	<ul style="list-style-type: none"> Version changed for 2016.10a release Added “Back-to-Back Transfer Support on an APB Interface” on page 35 “Parameter Descriptions” on page 49 auto-extracted from the RTL Removed the “Running Leda on Generated Code with coreConsultant” section, and reference to Leda directory in Table 2-1 Removed the “Running Leda on Generated Code with coreAssembler” section, and reference to Leda directory in Table 2-4 Recreated Figure 2-20 on page 37 Added “Running VCS XPROP Analyzer” Moved “Internal Parameter Descriptions” to Appendix
3.00a	June 2015	<ul style="list-style-type: none"> Added “Running SpyGlass® Lint and SpyGlass® CDC” Added “Running SpyGlass on Generated Code with coreAssembler” “Signal Descriptions” on page 57 auto-extracted from the RTL Added “Internal Parameter Descriptions” on page 93 Added “APB4 Protocol Feature” on page 37 Updated area and power numbers in “Performance” on page 74
2.03a	June 2014	<ul style="list-style-type: none"> Version change for 2014.06a release Updated “Performance” section in the “Integration Considerations” chapter Corrected Default Input/Output Delays in Signals chapter
2.02c	May 2013	<ul style="list-style-type: none"> Version change for 2013.05a release Updated the template
2.02b	October 2012	Added the product code on the cover and in Table 1-1
2.02b	October 2011	Version change for 2011.10a release
2.02a	June 2011	Updated: <ul style="list-style-type: none"> Figure 3-14 to reflect current hrdata functionality System diagram in Figure 1-1 “Related Documents” section in Preface.
2.01a	May 2011	Corrected Figures 3-7 and 3-9.
2.01a	April 2011	Version change for 2011.03a release.
2.00a	December 2010	Version change for 2010.12a release.
1.04a	September 2010	<ul style="list-style-type: none"> Corrected names of include files and vcs command used for simulation Included additional information about AMBA 3 APB protocol
1.03a	December 2009	Updated databook to new template for consistency with other IIP/VIP/PHY databooks
1.03a	July 2009	Enhanced with PRDATA sample timing

Version	Date	Description
1.03a	May 2009	Removed references to QuickStarts, as they are no longer supported
1.03a	October 2008	Version change for 2008.10a release
1.02e	July 2008	Added “Burst Transfers” subsection
1.02e	June 2008	Version change for 2008.06a release
1.02d	December 2007	<ul style="list-style-type: none">■ Updated for revised installation guide and consolidated release notes titles■ Changed references of “Designware AMBA” to simply “DesignWare”
1.02d	June 2007	Description added under Figure 11

Preface

This databook provides information about the DW_apb, which is an AMBA APB Protocol Specification v2.0-compliant Advanced Peripheral Bus component. The DW_apb is a part of the DesignWare® Synthesizable Components for AMBA APB Protocol Specification v2.0. The databook also supplies descriptions of tests used to verify the DW_apb component, synthesis information, and user options unique to the DW_apb.

This databook is intended for designers who plan to use the DW_apb with Synopsys tools and supported third-party simulators. Readers are assumed to be familiar with the [AMBA Specification, Revision 2.0](#) from Arm.

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_apb.
- Chapter 3, “[Parameter Descriptions](#)” identifies the configurable parameters supported by the DW_apb.
- Chapter 4, “[Signal Descriptions](#)” provides a list and description of the DW_apb signals.
- Chapter 5, “[Verification](#)” provides information on verifying the configured DW_apb.
- Chapter 6, “[Integration Considerations](#)” includes information you need to integrate the configured DW_apb into your design.
- Appendix A, “[DesignWare Constants](#)” includes the contents of the DesignWare Synthesizable Components bus constants file.
- Appendix B, “[Internal Parameter Descriptions](#)” provides a list of internal parameter descriptions that might be indirectly referenced in expressions in the Signals chapter.
- Appendix C, “[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 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 AMBA APB Protocol Specification v2.0, see the *Guide to Documentation for DesignWare Synthesizable Components for AMBA 2 and AMBA 3 AXI (Documentation Overview)*.

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:** AMBA
- 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_ah2h	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 APB4 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
DW_axi_a2x	Configurable bridge between AXI and AHB components or AXI and AXI components.
DW_axi_gm	Simplify the connection of third party/custom master controllers to any AMBA 3 AXI fabric
DW_axi_gs	Simplify the connection of third party/custom slave controllers to any AMBA 3 AXI fabric
DW_axi_hmx	Configurable high performance interface from and AHB master to an AXI slave
DW_axi_rs	Configurable standalone pipelining stage for AMBA 3 AXI subsystems
DW_axi_x2h	Bridge from AMBA 3 AXI to AMBA 2.0 AHB, enabling easy integration of legacy AHB designs with newer AXI systems

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

Component Name	Description
DW_axi_x2p	High performance, low latency interconnect fabric and bridge for AMBA 2 & 3 APB for direct connect to AMBA 3 AXI fabric
DW_axi_x2x	Flexible bridge between multiple AMBA 3 AXI components or buses

Product Overview

This chapter describes the DesignWare APB, which provides a bridge between the AHB bus and a set of APB peripherals.

1.1 DesignWare System Overview

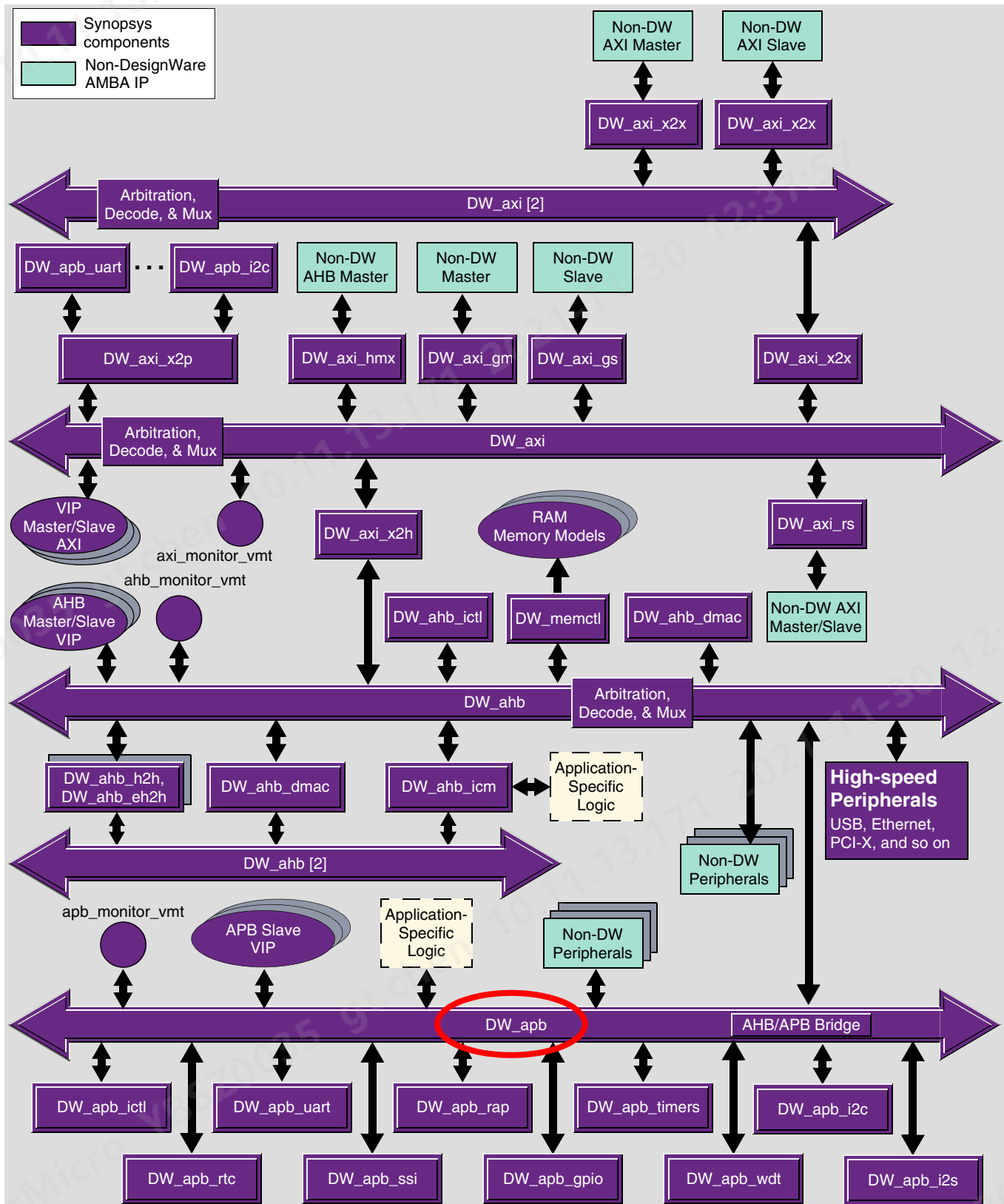
The Synopsys DesignWare Synthesizable Components environment is a parameterizable bus system containing components for the following:

- AMBA version 2.0-compliant AHB (Advanced High-performance Bus)
- AMBA APB Protocol Specification v2.0 (Advanced Peripheral Bus)
- AMBA version 3.0-compliant AXI (Advanced eXtensible Interface)

1.1.1 DesignWare System Block Diagram

[Figure 1-1](#) illustrates one example of this environment, including the AXI bus, the AHB bus, and an 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. To access the product page and documentation for AMBA components, see the [DesignWare IP Solutions for AMBA Interconnect](#) page. (SolvNetPlus ID required)

Figure 1-1 Example of DW_apb in a Complete System



You can connect, configure, synthesize, and verify the DW_apb within a DesignWare subsystem using coreAssembler, documentation for which is available on the web in the *coreAssembler User Guide*.

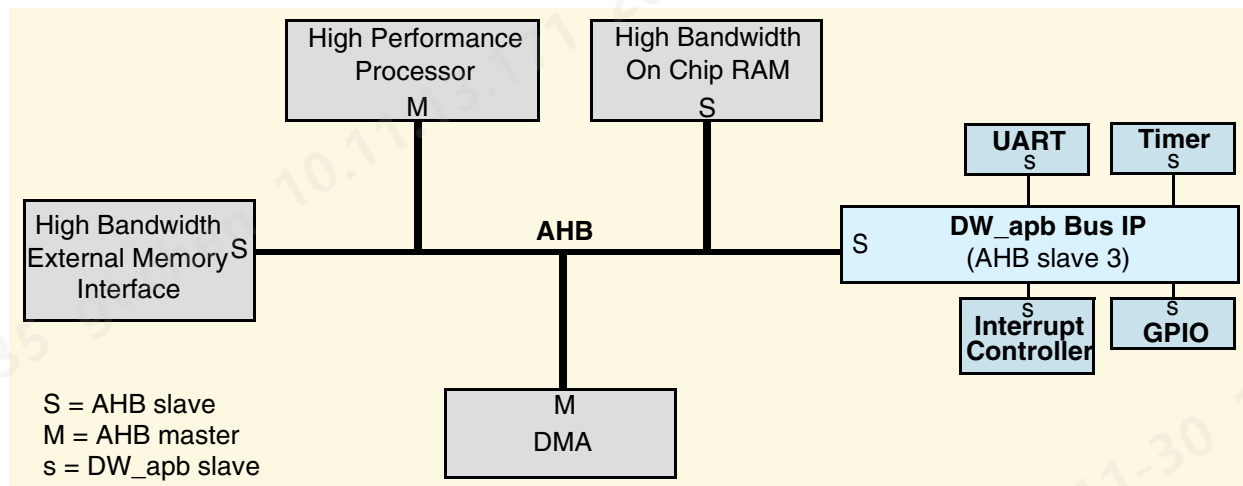
If you want to configure, synthesize, and verify a single component such as the DW_apb component, you might prefer to use coreConsultant, documentation for which is available in the *coreConsultant User Guide*.

1.2 General Product Description

The DW_apb is a parameterizable, synthesizable, and programmable component that implements the APB functionality of the *AMBA APB Protocol Specification v2.0* from Arm®.

The DW_apb provides a bridge between the AHB bus and a set of APB peripherals. All communication between masters on the AHB and slaves on the APB pass through the DW_apb. From the point of view of the AHB system, the DW_apb appears as a slave, as illustrated in [Figure 1-2](#).

Figure 1-2 DW_apb in an Example System



1.3 Features

The DW_apb includes the following features:

- Compliance with the *AMBA Specification, Revision 2.0* from Arm®
- Compliance with the *AMBA 3 APB Specification, Revision 1.0* from Arm®
- Compliance with the *AMBA APB Protocol Specification, v2.0* from Arm®

Support for the following:

- Up to 16 APB slaves
- Big- and little-endian AHB systems
- Little-endian APB slaves
- 32, 64, 128, and 256-bit AHB data buses
- 8, 16, and 32-bit APB data buses
- Single and burst AHB transfers

- Synchronous hclk/pclk; hclk is an integer multiple of pclk
- Optional external decoder
- AMBA 5 AHB support
 - Secure transfers
 - Endianness
 - User Signals support on each channel

1.3.1 Notes and Restrictions

- Slave numbers are configured consecutively – 0, 1, 2, 3; not 0, 3, 5, 9.
- All slaves must have their address spaces aligned to a 1 KB boundary.
- Minimum address space allocated to a configured slave is 1 KB.
- There is support for only little-endian APB slaves.
- The APB data bus width must be less than or equal to the AHB data bus width.
- The APB clock must be equal to, or a submultiple of and synchronous to, the AHB clock.

1.3.2 Features Not Supported

The following features are not supported in this release:

- Independent AHB clock (*hclk*) and APB clock (*pclk*) (APB bus must be synchronous with AHB bus)
- No support for the following AHB features when an AHB slave:
 - SPLIT transfers
 - RETRY responses
- Big-endian APB peripherals

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

1.4 Standards Compliance

The DW_apb component conforms to the *AMBA Specification, Revision 2.0*, *AMBA 3 APB Protocol Specification v1.0*, and *AMBA APB Protocol Specification v2.0* from Arm®. Readers are assumed to be familiar with these specifications.

1.5 Verification Environment Overview

The DW_apb 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 69 chapter discusses the specific procedures for verifying the DW_apb.

1.6 Licenses

Before you begin using the DW_apb, you must have a valid license. For more information, see “Licenses” in the *DesignWare Synthesizable Components for AMBA 2/AMBA 3 AXI Installation Guide*.

1.7 Where To Go From Here

At this point, you may want to get started working with the DW_apb 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, see *Guide to coreTools Documentation*.

For more information about configuring, synthesizing, and verifying just your DW_apb component, see “Overview of the coreConsultant Configuration and Integration Process” in *DesignWare Synthesizable Components for AMBA 2 User Guide*.

For more information about implementing your DW_apb component within a DesignWare subsystem using coreAssembler, see “Overview of the coreAssembler Configuration and Integration Process” in *DesignWare Synthesizable Components for AMBA 2 User Guide*.

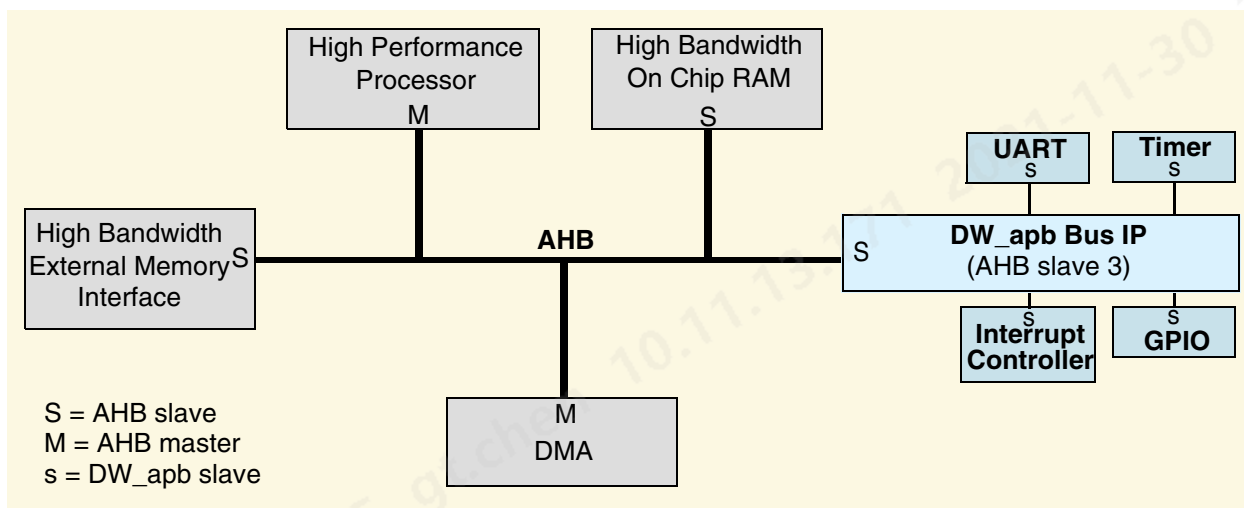
Functional Description

The DW_apb is a parameterizable, synthesizable, and programmable component that implements the APB functionality of the [AMBA Specification \(Rev. 2.0\)](#).

2.1 Overview

The DW_apb provides the interconnect fabric to connect an AHB bus to APB peripherals, which are compliant with *AMBA 2 APB Specification*, *AMBA 3 APB Protocol Specification v1.0*, or *AMBA APB Protocol Specification v2.0*. The interconnect fabric is referred to as the APB Bridge in the AMBA 2 APB specification and *AMBA APB Protocol Specification v2.0*, and simply as APB in the *AMBA 3 APB Protocol Specification v1.0*. The bridge is the only master on the APB. From the point of view of the AHB system, the DW_apb appears as a slave, as illustrated in [Figure 2-1](#).

Figure 2-1 DW_apb in an Example System



2.1.1 Block Diagram

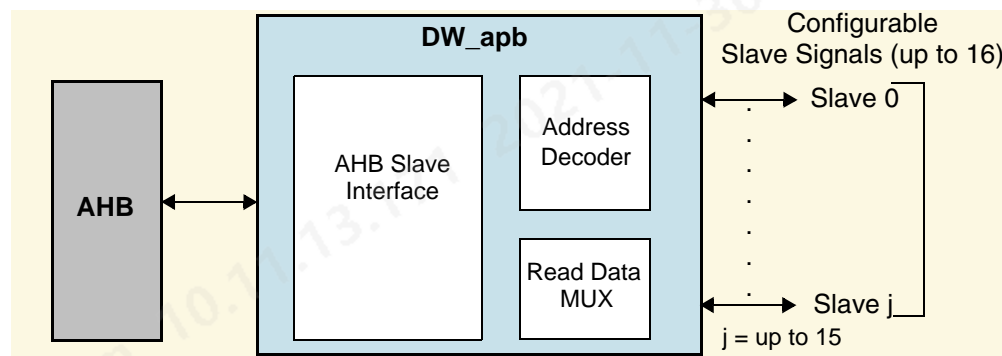
The DW_apb is configurable, synthesizable, and performs the following functions:

- Monitors and responds to AHB transactions for the DW_apb
- Generates APB control, address, and write data signals

- Generates APB peripheral select lines
- Matches wide AHB write data bus to narrow APB write data bus
- Converts big-endian AHB write data to little-endian APB write data
- Matches narrow APB read data buses to wide AHB read data bus
- Converts little-endian APB read data to big-endian AHB read data

A block diagram is illustrated in [Figure 2-2](#).

Figure 2-2 DW_apb Block Diagram



2.2 Transfers

If an AHB master wants to communicate with an APB slave, it does this by selecting the DW_apb and driving the necessary address, data, and control information to it. The DW_apb presents the data it receives from the APB peripherals onto the AHB data bus. The DW_apb cannot initiate any transfers on the AHB itself; it only responds to the requests from AHB masters.

A write transfer on the APB has the address, control, and data signals aligned, unlike the AHB where data and addresses are pipelined. The transfer on the APB takes a minimum of two cycles to complete. A write transfer from the AHB to the APB does not require the AHB system bus to stall until the transfer on the APB has completed. This means a write to the APB can be followed directly by a read from an AHB peripheral (not DW_apb).

While the APB transfer is being aligned, started, and executed, a read from an AHB peripheral can be performed. If the system were held until the write is completed, then for a system with a very slow APB, it would be the APB that would control the system performance. If another write occurs to the APB immediately following the first, the address and control is taken, the instruction is pipelined, and other transfers are stalled by bringing hready low. When the pipeline is cleared, any additional instructions for the APB are then processed. However if the first write transfer targets an AMBA APB4 slave, the AHB cannot issue any new transfer while the first does not complete on DW_apb.

Regarding reads, once a read is started, it is completed and the AHB bus held (by bringing hready low) until the data is returned from the slave. For more information about read and write transfers to or from the APB, see [“Timing Diagrams”](#) on page 23.



Note

If a transfer is initiated with a BUSY or IDLE transfer, DW_apb ignores the transfer.

2.2.1 Burst Transfers

The DW_apb supports all AHB burst accesses. Since the DW_apb is a relatively simple slave, it processes all AHB beats on a cycle-by-cycle basis. Since an AHB master is required to generate an address for every beat of a burst, the DW_apb can support AHB bursts without internally sampling the hburst signal. The hburst is necessary for only more advanced slaves that do prefetching, cache line fills, and so on.

The hburst input is still included in the DW_apb for I/O signal compatibility with later releases that may include functionality that uses the hburst information.

2.3 PCLK versus HCLK

The DW_apb uses only hclk and pclk_en, and it treats a rising edge of hclk and pclk_en = 1 as an indication of a rising edge on pclk. This means that if pclk_en is active, then the next rising edge on hclk is also a rising edge of pclk. The design of the DW_apb assumes that the clocks hclk and pclk are synchronous; they do not have to be the same frequency. The pclk_en should be generated from an hclk register.

When pclk is the same as hclk, pclk_en must be always high. (The data rate on the APB is half that on the AHB, due to the how the AMBA standard is defined.)

When pclk is not the same as hclk, the data rate on the APB depends on the frequency of the pclk_en signal, which pulses once every n hclk cycles. When addresses and data come from an AHB master, they are saved. Only when pclk_en is high are addresses and data presented to the APB slave.

APB peripherals use the pclk signal as the clock, whereas the APB bridge uses hclk and the pclk_en signal in order to gauge pclk in relation to hclk.

**Note**

When pclk is not equal to hclk, prdata is sampled on the first positive hclk edge after assertion of penable, *not* on the first pclk edge after assertion of penable. For more details, see the text associated with [Figure 2-8](#) on page 26.

2.4 Optional External Decoder

During configuration of DW_apb, you can choose to have an external decoder. By having the decoder external to DW_apb, you can connect any decoder with any number of remap options. When this option is chosen, the internal decoder is not included. There are inputs for the peripheral selects from the external decoder, which pass through the bridge and drive the peripheral select outputs of DW_apb.

2.5 Endianness

APB slave subsystems are little-endian; the DW_apb performs the conversion from a big-endian AHB to the little-endian APB peripheral by swapping the bytes. However, there is no support for converting a big-endian AHB to a big-endian APB slave peripheral. You have to manually perform this process by swapping the bytes as illustrated in [Figure 2-3](#).

Figure 2-3 Converting Big-Endian AHB to Big-Endian APB Peripheral

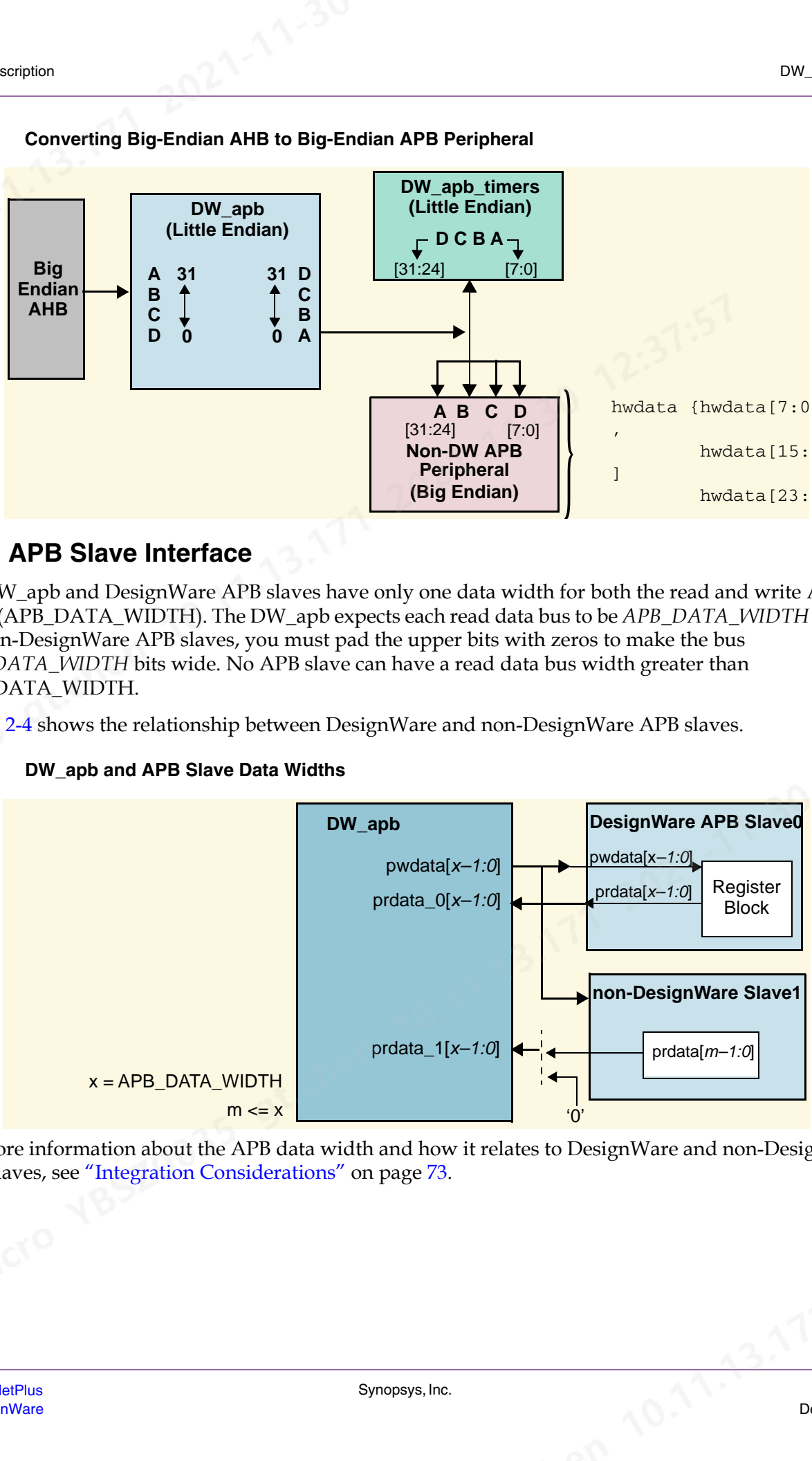
2.6 APB Slave Interface

The DW_apb and DesignWare APB slaves have only one data width buses (APB_DATA_WIDTH). The DW_apb expects each read data to be APB_DATA_WIDTH bits wide. For non-DesignWare APB slaves, you must pad the upper bits with zeros to make the read data APB_DATA_WIDTH bits wide. No APB slave can have a read data width less than APB_DATA_WIDTH.

Figure 2-4 shows the relationship between DesignWare and non-DesignWare APB slaves.

Figure 2-4 DW_apb and APB Slave Data Widths

For more information about the APB data width and how it relates to APB slaves, see “[Integration Considerations](#)” on page 73.



Functional Description

Figure 2-3 Converting Big-Endian AHB to Little-Endian APB

2.6 APB Slave Interface

The DW_apb and DesignWare APB slave interfaces use the same data bus width (APB_DATA_WIDTH). The APB slave interface is designed to be compatible with the APB master interface. For non-DesignWare APB slaves, the APB slave interface is designed to be compatible with the APB master interface. The APB slave interface is designed to be compatible with the APB master interface. The APB slave interface is designed to be compatible with the APB master interface.

Figure 2-4 shows the relationship between the APB slave interface and the APB master interface. The APB slave interface is designed to be compatible with the APB master interface. The APB slave interface is designed to be compatible with the APB master interface. The APB slave interface is designed to be compatible with the APB master interface.

Figure 2-4 DW_apb and APB Slave Data Width

$x = \text{APB_DATA_WIDTH}$

$m < x$

For more information about the APB slave interface, see “Integration Considerations”.

22 SolvNetPlus DesignWare

ctional Description
DW_apb Databook

Figure 2-3 Converting Big-Endian AHB to Big-Endian APB Peripheral

The diagram illustrates the connection between a Big-Endian AHB bus and a Non-DW APB Peripheral (Big Endian) via a DW_apb (Little Endian) interface. The DW_apb block contains four registers labeled A, B, C, and D. Register A has bit ranges [31:0] and [7:0], while register D has [31:24] and [7:0]. The DW_apb_timers (Little Endian) block also has registers A, B, C, and D with similar bit ranges. The Non-DW APB Peripheral (Big Endian) receives data from the DW_apb registers and outputs it to the hwdatas of the peripheral.

```

hwdatas {
    hwdatas[7:0] = ...
    hwdatas[15:0] = ...
    hwdatas[23:0] = ...
}

```

6 APB Slave Interface

The DW_apb and DesignWare APB slaves have only one data width for both the read and write APB data buses (`APB_DATA_WIDTH`). The DW_apb expects each read data bus to be `APB_DATA_WIDTH` bits wide. For non-DesignWare APB slaves, you must pad the upper bits with zeros to make the bus `APB_DATA_WIDTH` bits wide. No APB slave can have a read data bus width greater than `APB_DATA_WIDTH`.

Figure 2-4 shows the relationship between DesignWare and non-DesignWare APB slaves.

Figure 2-4 DW_apb and APB Slave Data Widths

The diagram shows the DW_apb block interfacing with two APB slaves. The DW_apb block has three output ports: `pdata[x-1:0]`, `prdata_0[x-1:0]`, and `prdata_1[x-1:0]`. The DesignWare APB Slave0 block has a `Register Block` and interfaces with the DW_apb via `pdata[x-1:0]` and `prdata[x-1:0]`. The non-DesignWare Slave1 block interfaces with the DW_apb via `prdata_1[x-1:0]` and provides its own `prdata[m-1:0]` output. The diagram specifies that `x = APB_DATA_WIDTH` and `m ≤ x`.

For more information about the APB data width and how it relates to DesignWare and non-DesignWare APB slaves, see “[Integration Considerations](#)” on page 73.

SolvNetPlus
DesignWare

Synopsys, Inc.

3.03a
December 2020

[illegible]

Functional Description

Figure 2-3 Converting Big-Endian AHB to Big-Endian

The diagram illustrates the conversion of a Big-Endian AHB bus to a DW_apb (Little Endian) bus. On the left, a grey box labeled "Big Endian AHB" has an arrow pointing to a blue box labeled "DW_apb (Little Endian)". Inside the blue box, the bit ordering for a 32-bit transaction is shown. On the left side of the box, the bits are labeled A (31), B (30), C (29), and D (28). On the right side, the bits are labeled D (31), C (30), B (29), and A (28). Arrows indicate the mapping: A (31) maps to D (31), B (30) maps to C (30), C (29) maps to B (29), and D (28) maps to A (28).

2.6 APB Slave Interface

The DW_apb and DesignWare APB slaves have a data bus width of `APB_DATA_WIDTH`. The DW_apb slave interface is designed to be compatible with the APB data bus width. For non-DesignWare APB slaves, you must pad the data bus to `APB_DATA_WIDTH` bits wide. No APB slave should be wider than `APB_DATA_WIDTH`.

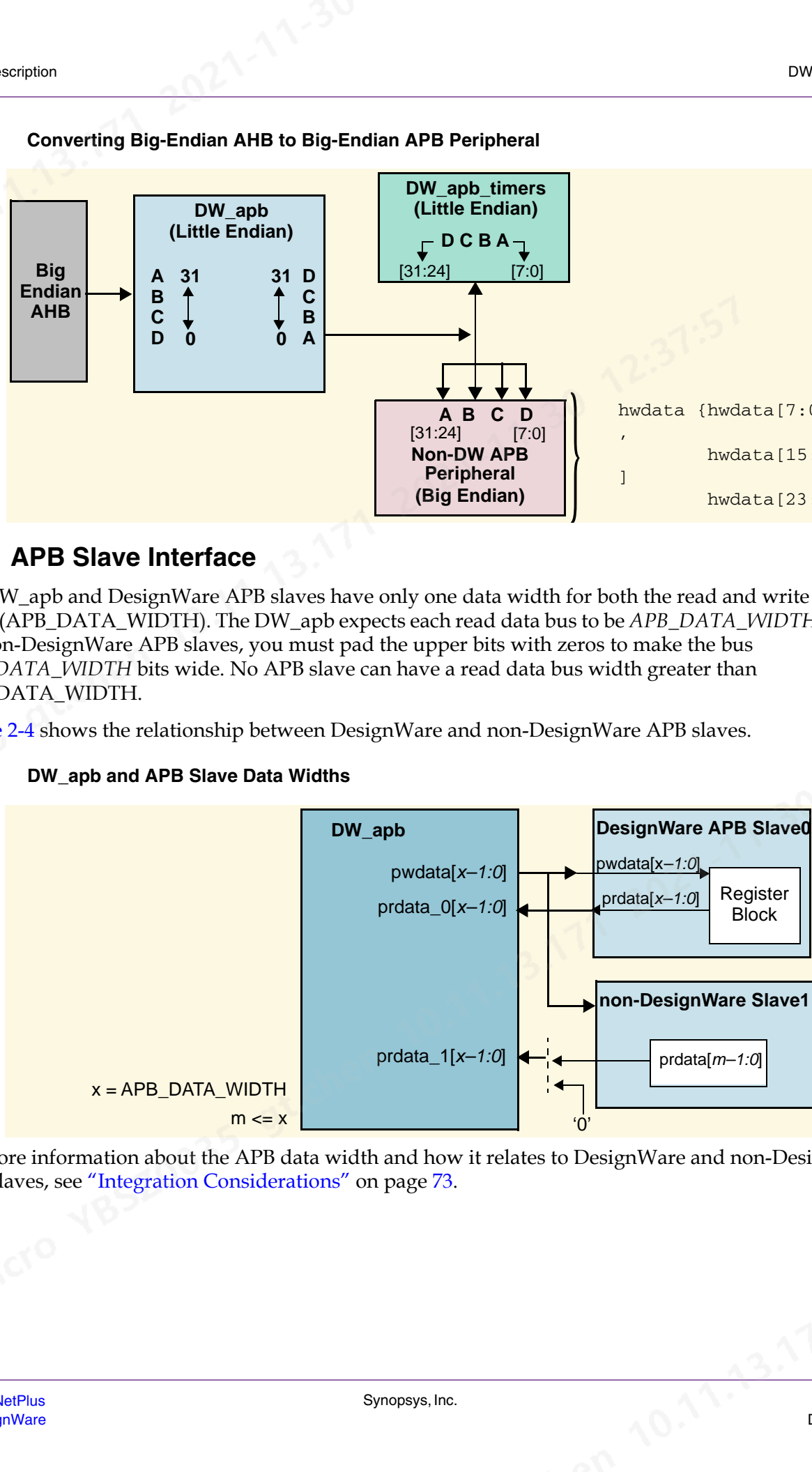
Figure 2-4 shows the relationship between DesignWare APB slaves and the APB data bus width.

Figure 2-4 DW_apb and APB Slave Data Widths

The diagram shows a yellow box with the text `x = APB_DATA_WIDTH` and a blue box with the text `m <= x`. The blue box is partially visible on the right side of the diagram.

For more information about the APB data width and how to integrate DesignWare APB slaves, see “[Integration Considerations](#)”.

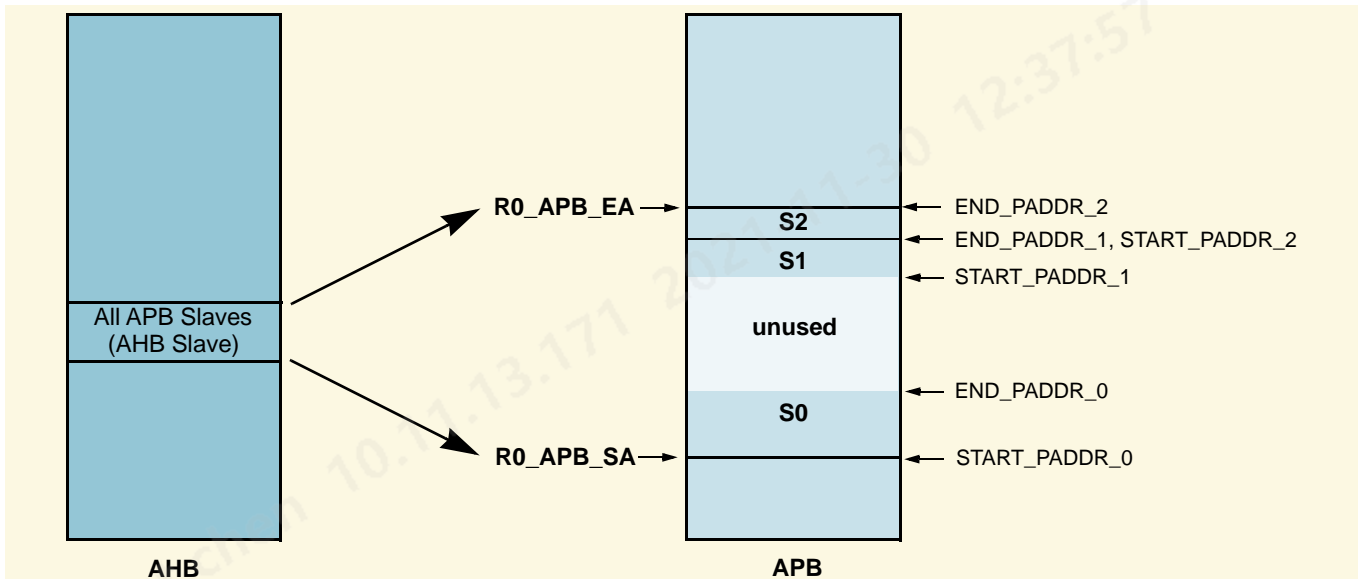
22 SolvNetPlus
DesignWare

[illegible]

2.7 Memory Map

Figure 2-5 illustrates a DW_apb memory map for a system with three slaves. Notice that the starting and ending address space (R0_APB_SA, R0_APB_EA) of the APB corresponds to an address space on the AHB for all APB slaves.

Figure 2-5 DW_apb Memory Map



2.8 Backward Compatibility with AMBA 2 APB and AMBA 3 APB

The AMBA 3 APB protocol has added the signals ready (pready) and error (pslverr) to the previous protocol, and the AMBA APB Protocol Specification v2.0 protocol has added the signals write-strobing (pstrb) and protection (pprot). However, APB slaves attached to the DW_apb can support either the AMBA APB Protocol Specification v2.0, AMBA 3 APB or AMBA 2 APB protocol. For each APB slave, you can use the `APB_INTERFACE_TYPE_SLAVE_i` configuration parameter to specify whether the attached component supports AMBA APB Protocol Specification v2.0, AMBA 3 APB or AMBA 2 APB. This configuration determines whether or not the freshly introduced signals are added on the I/O of the DW_apb instance. For more information on configuration parameters, see “Parameter Descriptions” on page 49.

2.9 Timing Diagrams

For timing, refer to the following diagrams:

- Read Transfer from AHB to AMBA 2 APB Slave (hclk = pclk): [Figure 2-6](#)
- Read Transfer from AHB to AMBA 3 APB Slave (hclk = pclk): [Figure 2-7](#)
- Read Transfer from AHB to AMBA 2 APB Slave (hclk != pclk): [Figure 2-8](#)
- Read Transfer from AHB to AMBA 3 APB Slave (hclk != pclk): [Figure 2-9](#)
- Write Transfer from AHB to AMBA 2 APB Slave (hclk = pclk): [Figure 2-10](#)
- Write Transfer from AHB to AMBA 3 APB Slave (hclk = pclk): [Figure 2-11](#)
- Write Transfer from AHB to AMBA 2 APB Slave (hclk != pclk): [Figure 2-12](#)

- Write Transfer from AHB to AMBA 3 APB Slave (hclk != pclk): [Figure 2-13](#)
- Read Transfer from AHB to AMBA 3 APB Slave (hclk != pclk) completed with an error: [Figure 2-14](#)
- Write Transfer from AHB to AMBA 3 APB Slave (hclk != pclk) completed with an error: [Figure 2-15](#)
- Back-to-back write transfer (hclk = pclk): [Figure 2-16](#)
- Back-to-back write transfer (hclk != pclk): [Figure 2-17](#)

Figure 2-6 DW_apb Read Transfer from AHB to AMBA 2 APB Slave (hclk = pclk)

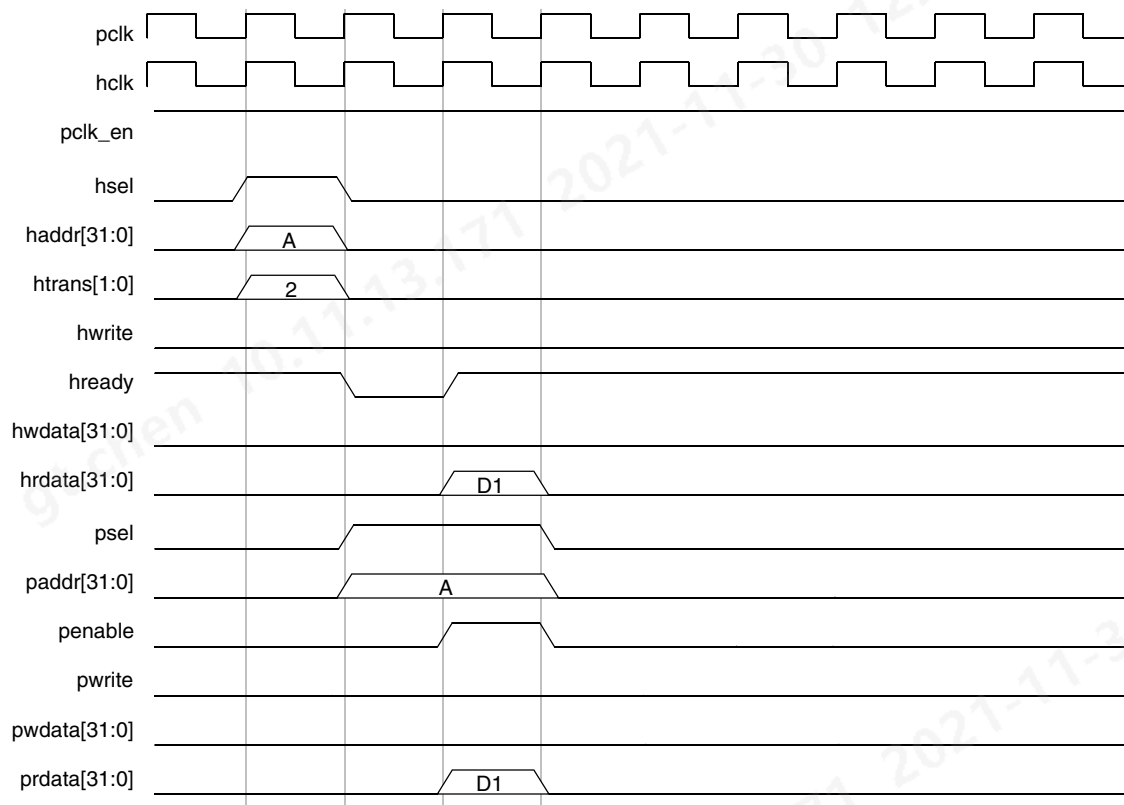


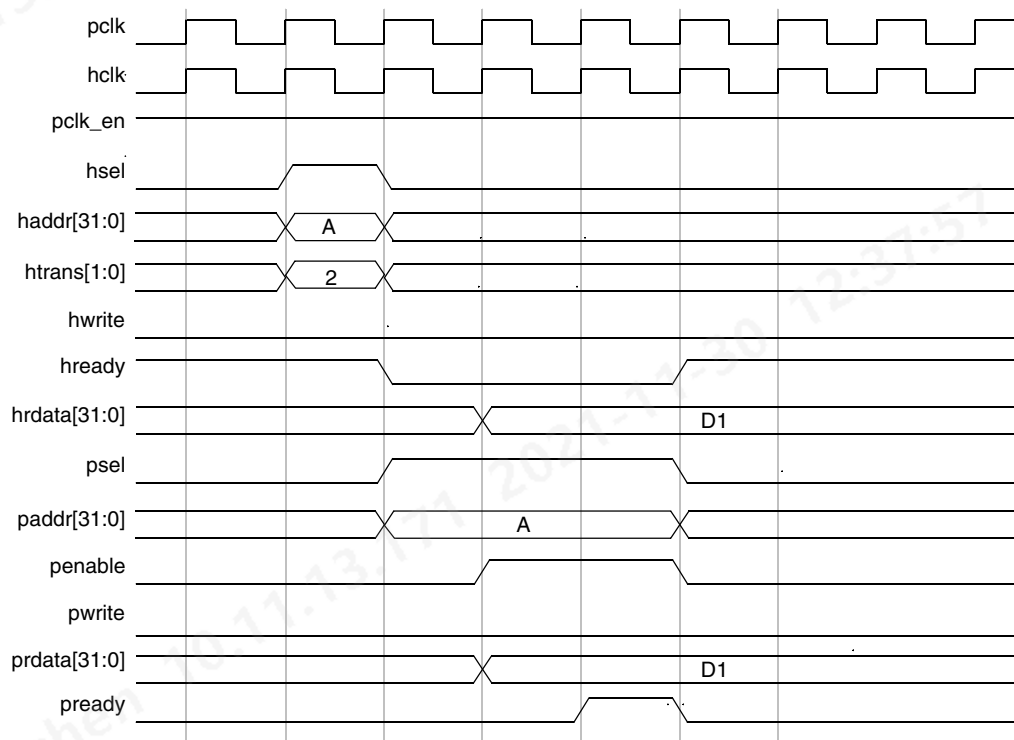
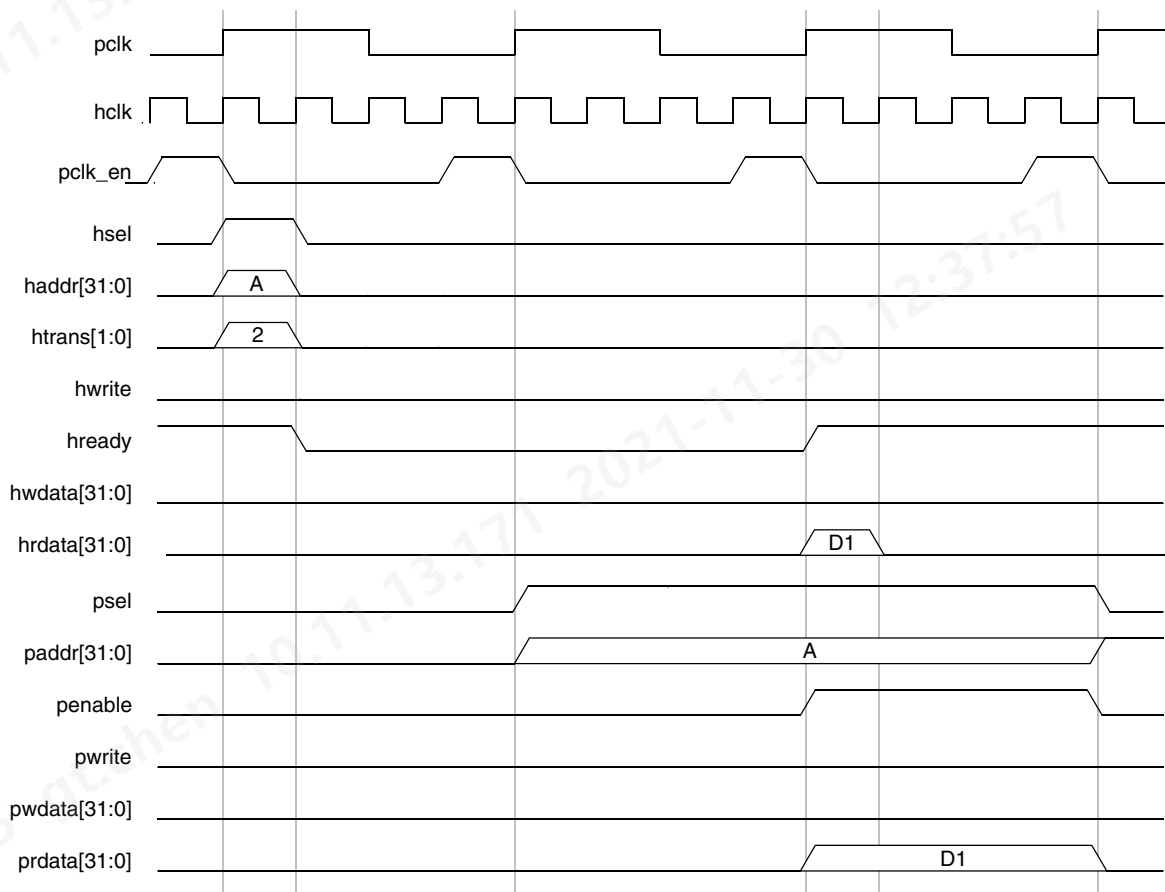
Figure 2-7 DW_apb Read Transfer from from AHB to AMBA 3 APB Slave (hclk = pclk)

Figure 2-8 DW_apb Read Transfer from AHB to AMBA 2 APB Slave(hclk != pclk)

The DW_apb registers the hready_resp output to prevent long combinatorial paths in the AHB bus system. The pclk_en signal is used to ensure that the hready_resp output can be registered from the DW_apb, regardless of the frequency ratio between hclk and pclk.

This implementation results in read data from the APB slave being sampled by the AHB master one hclk cycle after being driven by the APB slave.

The DW_apb bridge expects the prdata input from the APB slave to be registered. As the prdata input to the DW_apb bridge is driven from a register, it returns the read data to the AHB master before the end of the PENABLE phase without negatively affecting the timing closure of the system.

This architecture results in a high performance AMBA-compliant APB bridge.

The AMBA protocol specification gives designers two choices when interfacing APB and AHB; refer to 5-15 of the [AMBA Specification, Revision 2.0](#).

1. Route prdata directly to the AHB (hclk domain).
2. Register prdata at the end of the ENABLE cycle.

Because option 1 does not require a wait state for APB reads, and since prdata is assumed to come from a register, this is the best option for high performance. This is how the DW_apb bridge is implemented.

In systems where pclk is not equal to hclk, this means that prdata is sampled on the first hclk edge after penable is asserted. This requires that the prdata signals from the APB slaves – attached to the prdata_s(j)

ports — must be constrained to be stable one hclk after transitioning. This is already taken care of in the packaged synthesis intent of the DW_apb, but is your responsibility to ensure if synthesis is done outside of coreConsultant or coreAssembler.

Figure 2-9 DW_apb Read Transfer from AHB to AMBA 3 APB Slave (hclk != pclk)

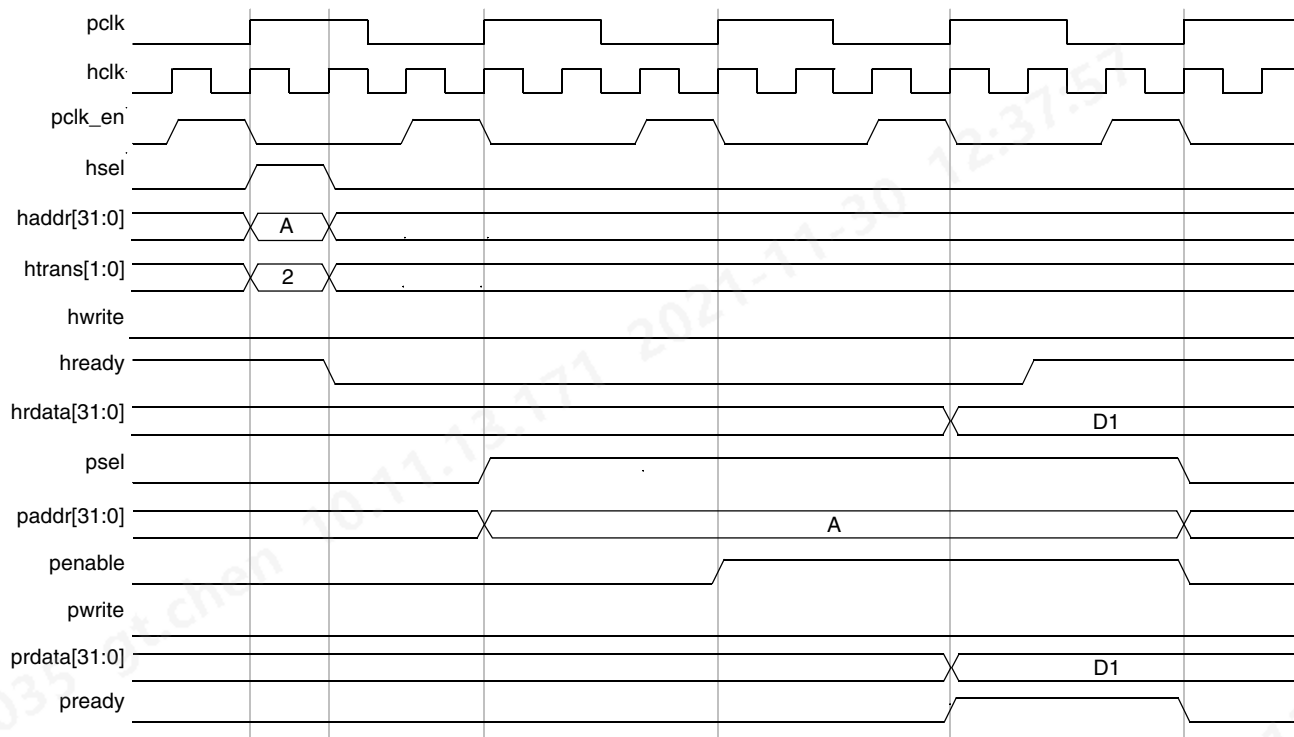


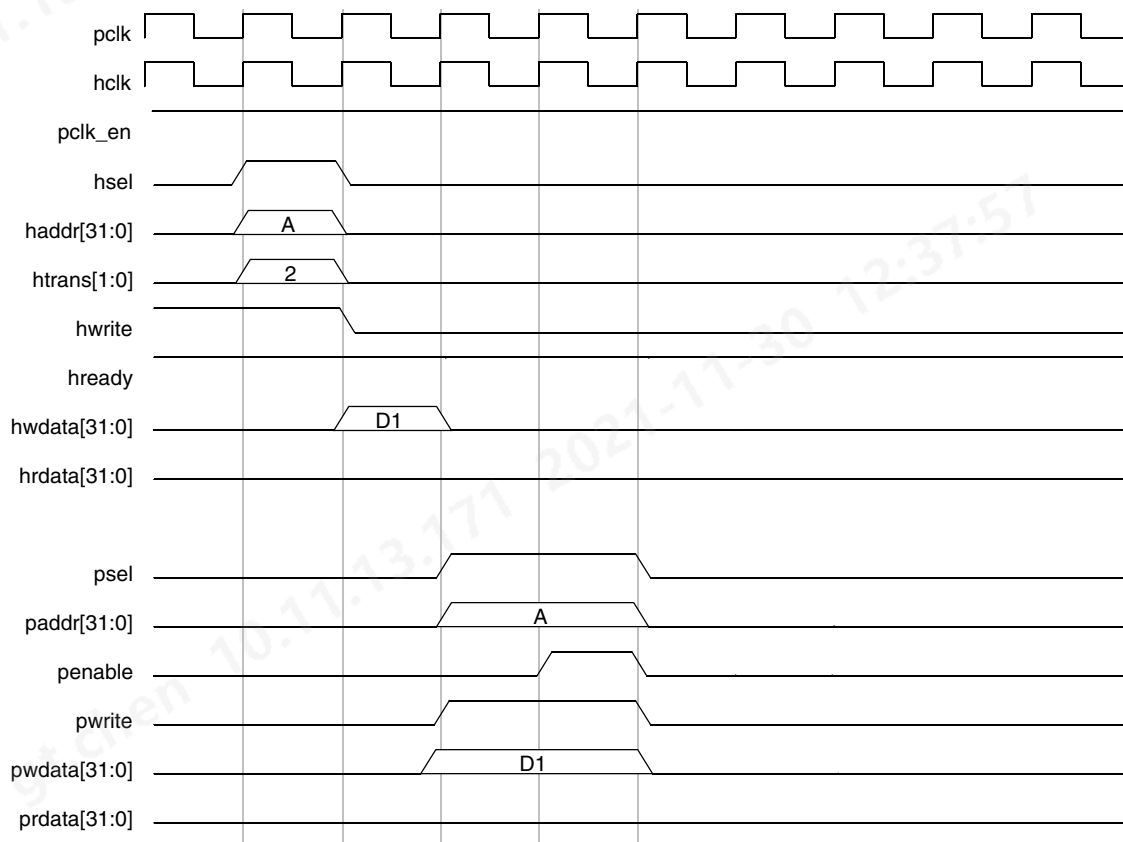
Figure 2-10 DW_apb Write Transfer from AHB to AMBA 2 APB Slave (hclk = pclk)

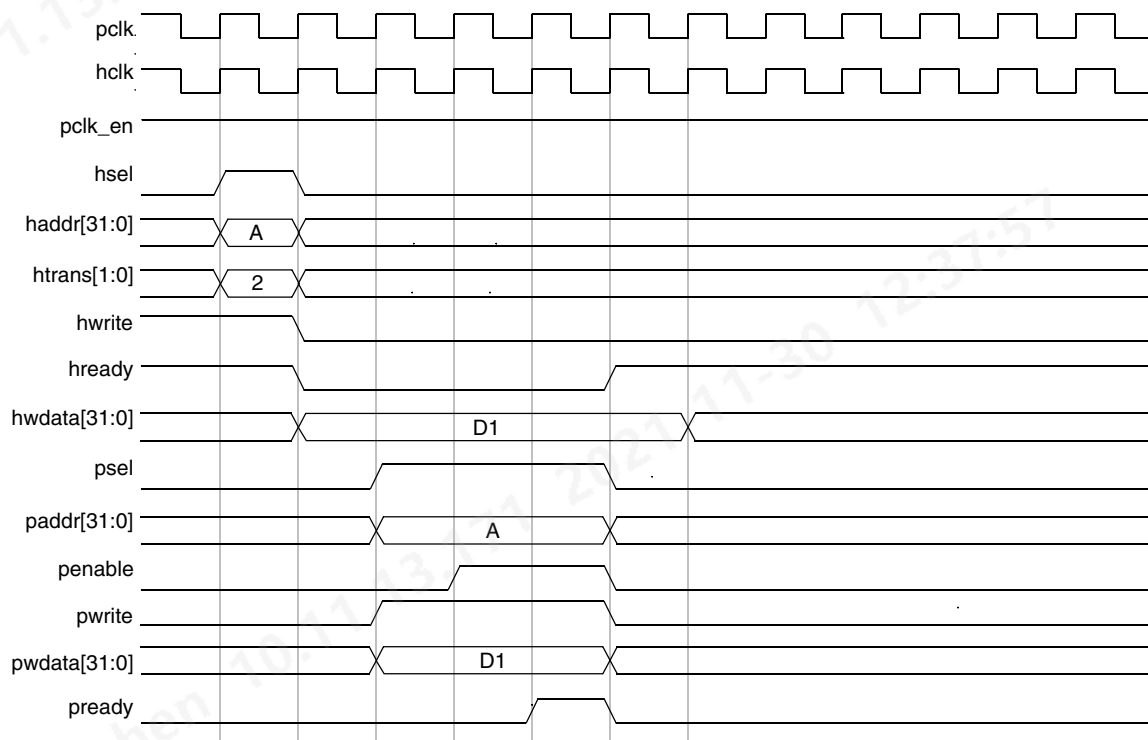
Figure 2-11 DW_apb Write Transfer from AHB to AMBA 3 APB Slave (hclk = pclk)

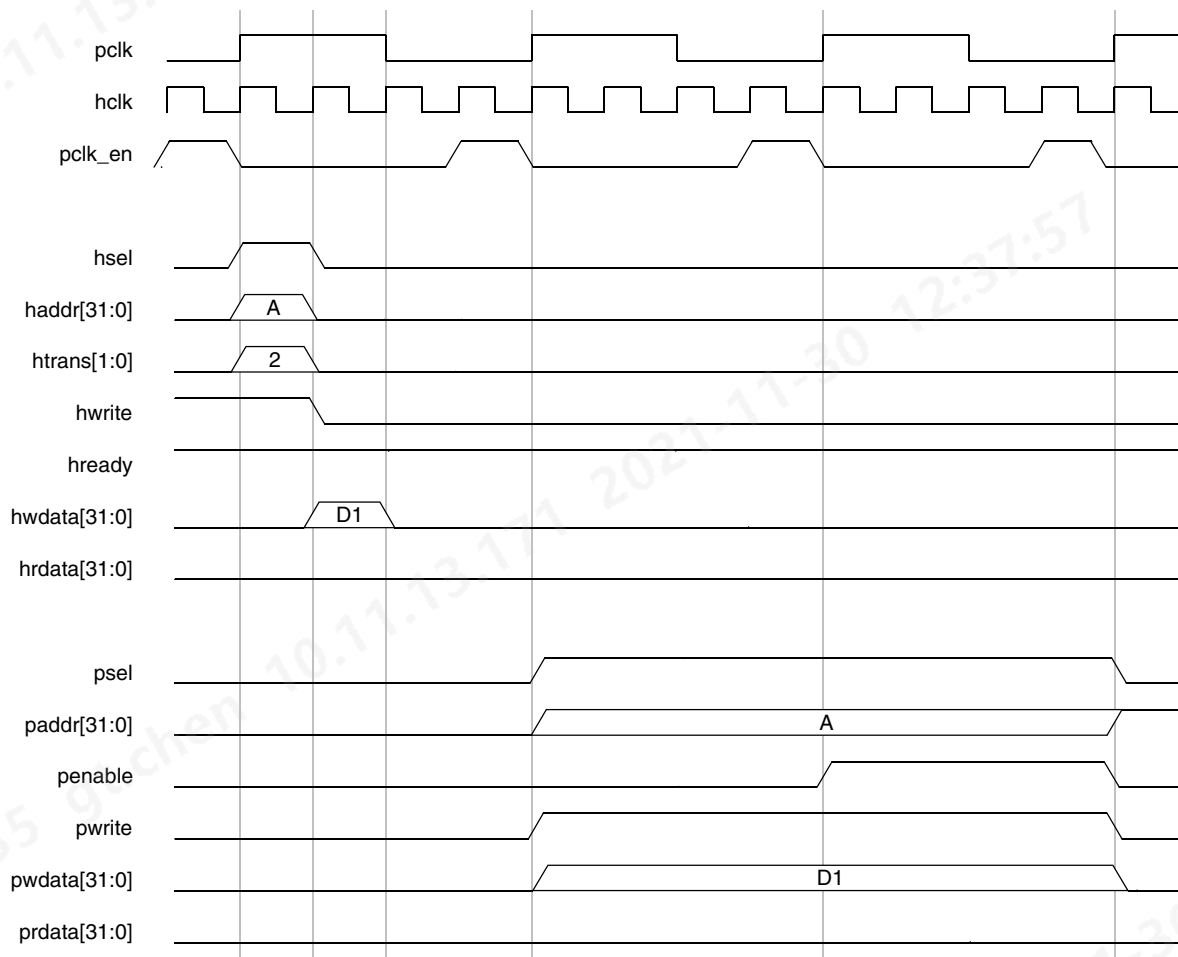
Figure 2-12 DW_apb Write Transfer from AHB to AMBA 2 APB Slave (hclk != pclk)

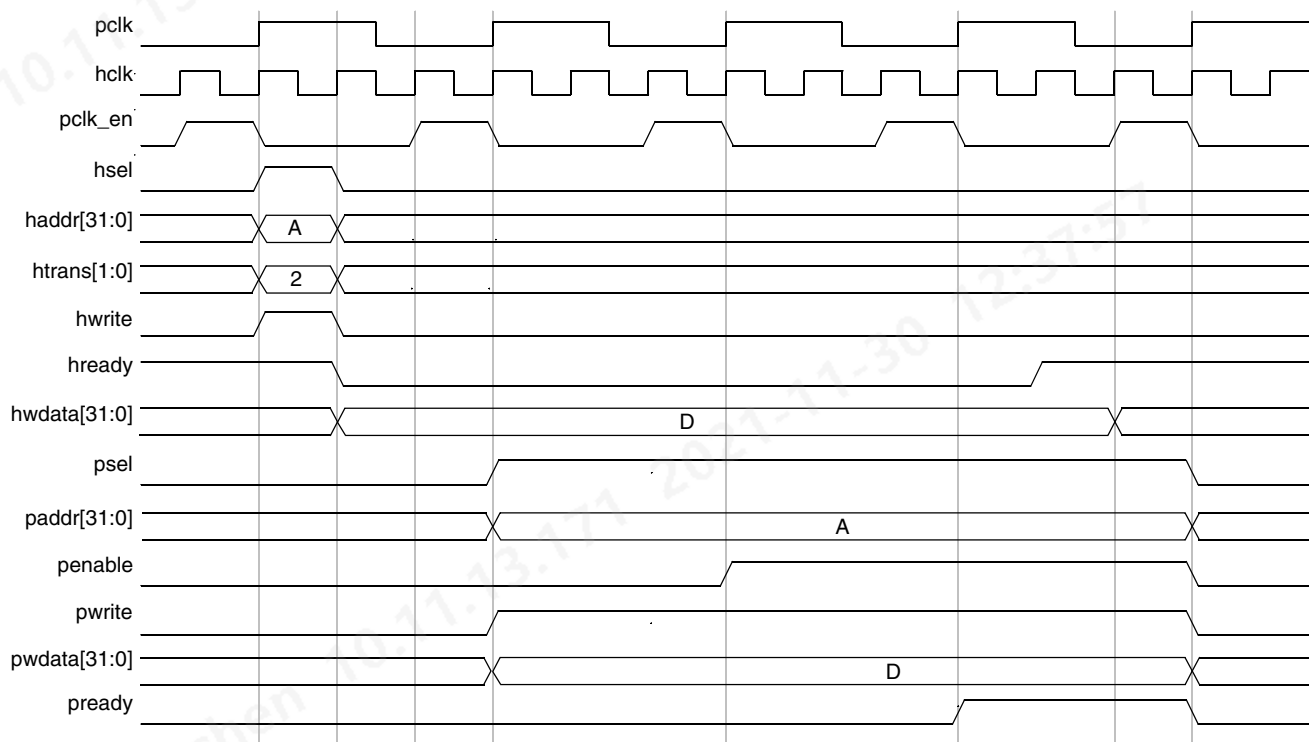
Figure 2-13 DW_apb Write Transfer from AHB to AMBA 3 APB Slave (hclk != pclk)

Figure 2-14 DW_apb Read Transfer from AHB to AMBA 3 APB Slave (hclk != pclk) Completed with Error

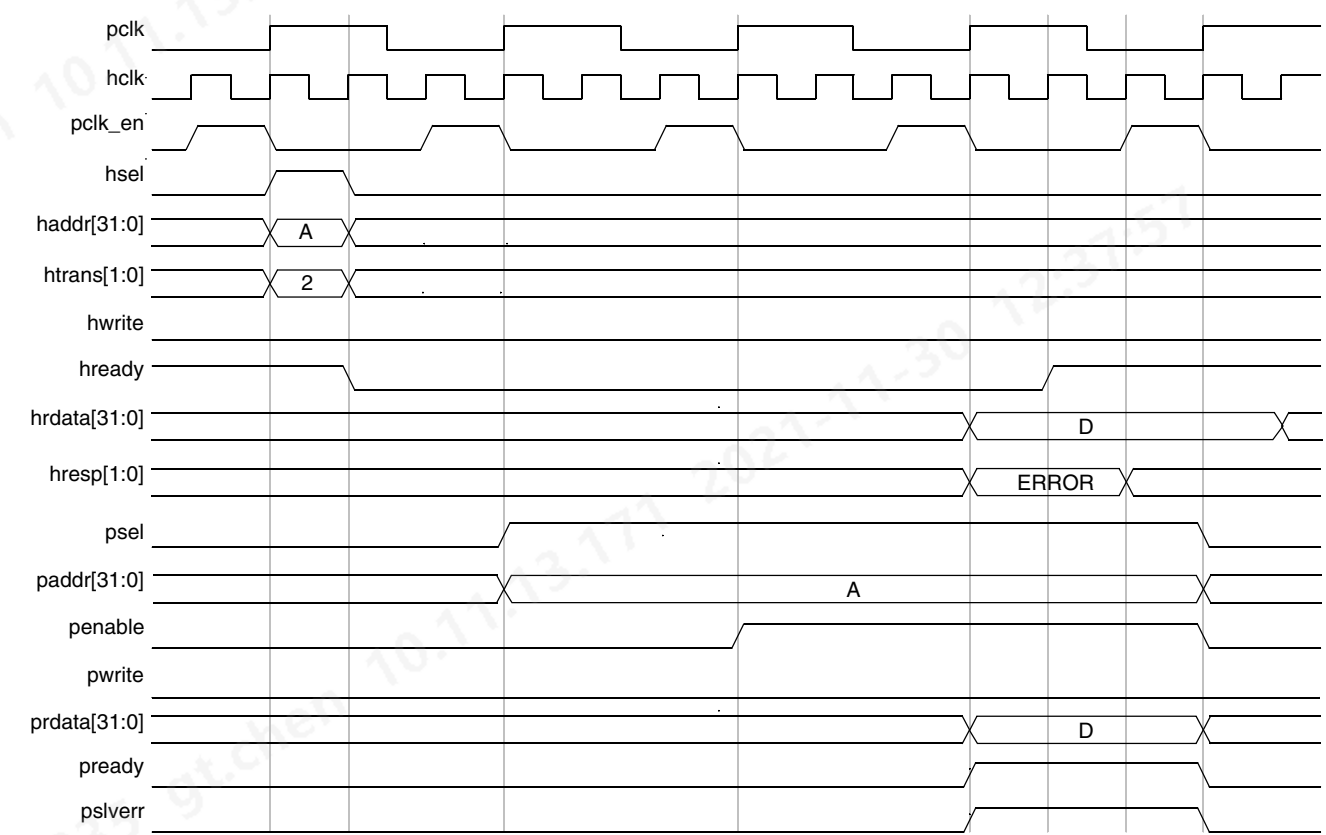
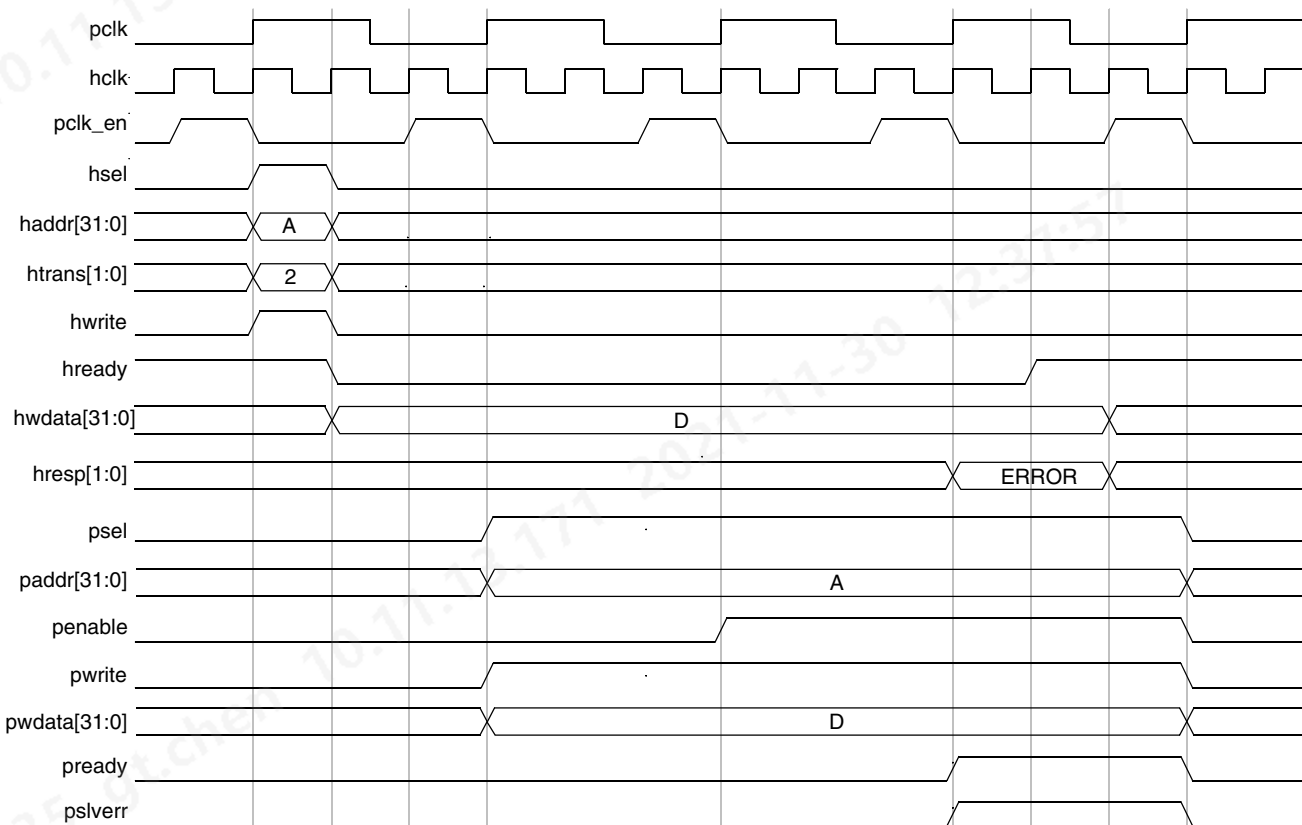


Figure 2-15 DW_apb Write Transfer from AHB to AMBA 3 APB Slave (hclk != pclk) Completed with Error

The DW_apb bridge expects the pslverr input from the APB slave to be registered. As the pslverr input to the DW_apb bridge is driven from a register, it returns hresp to the AHB master before the end of the PENABLE phase without negatively affecting the timing closure of the system. Thus pslverr is directly routed to the AHB (hclk domain) by mapping to hresp=ERROR (when pready is high) as suggested in the *AMBA 3 APB Specification, Revision 1.0*. Note that the paths from pready_sX to hready_resp are always registered.

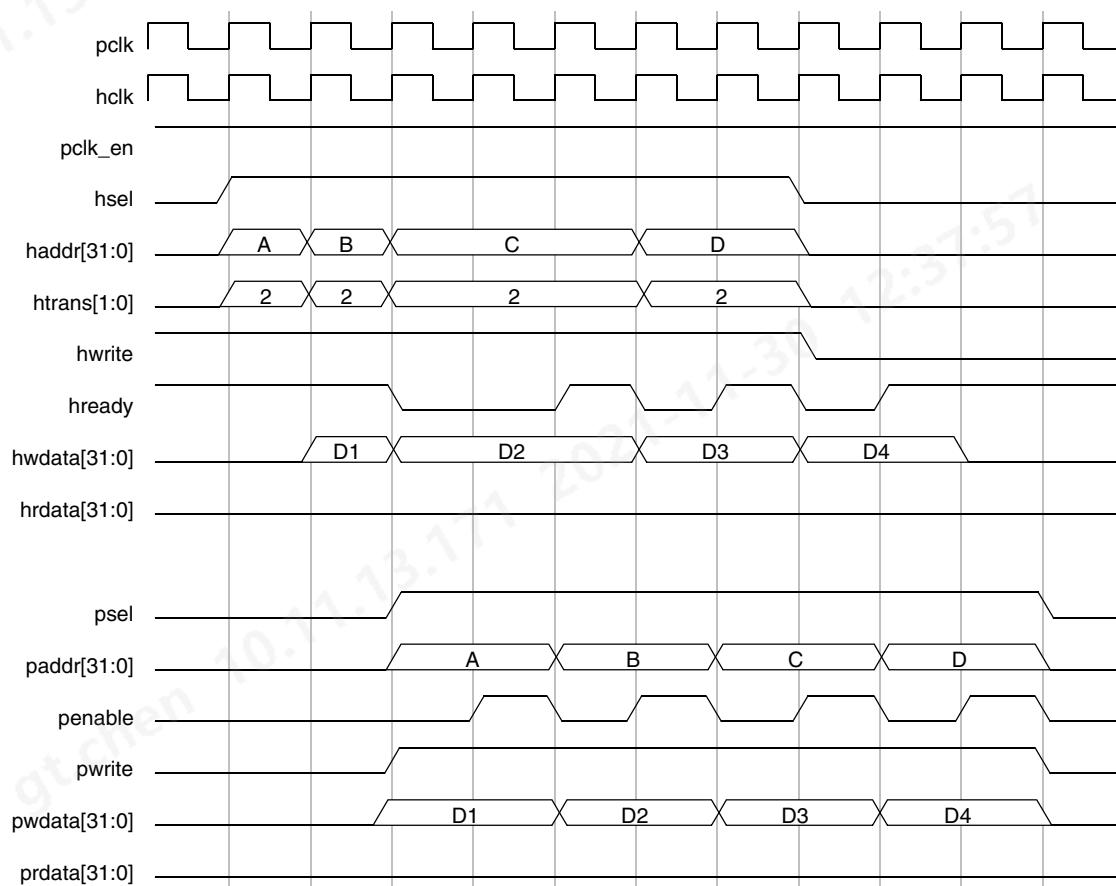
Figure 2-16 Back-to-Back Write Transfer (hclk = pclk)

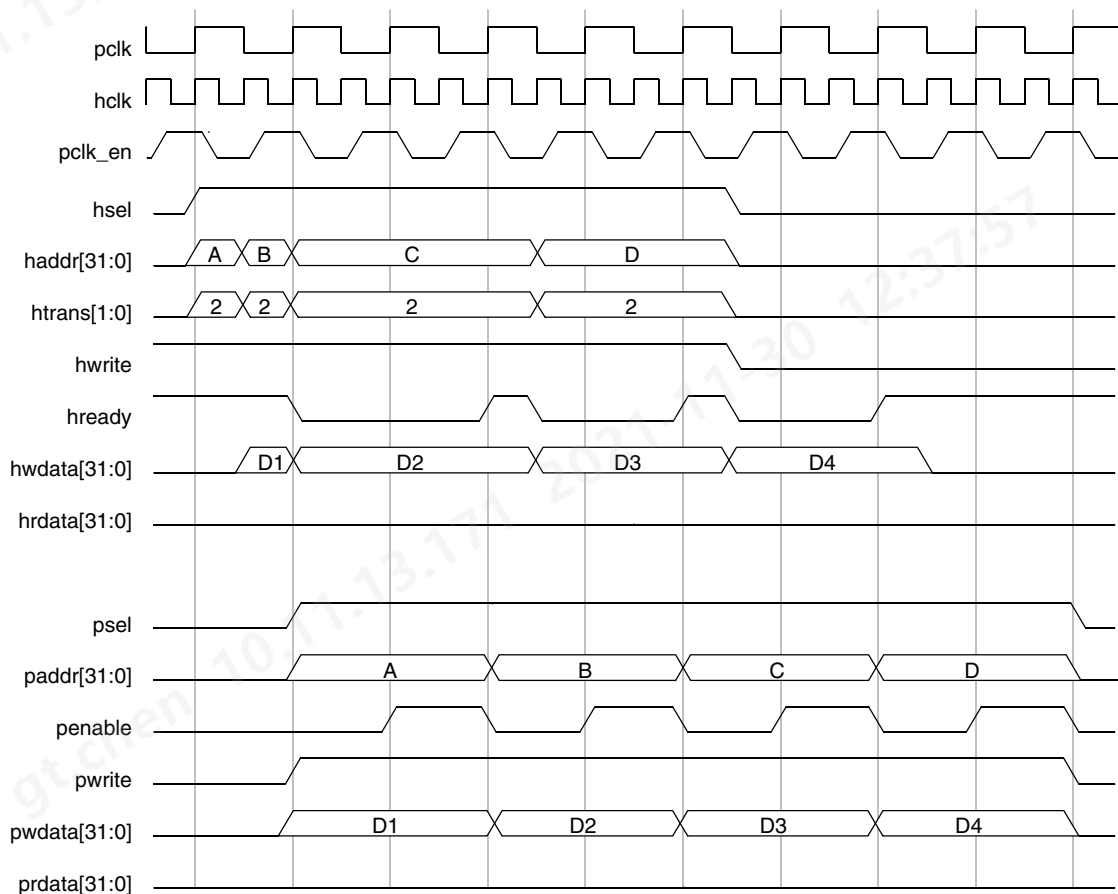
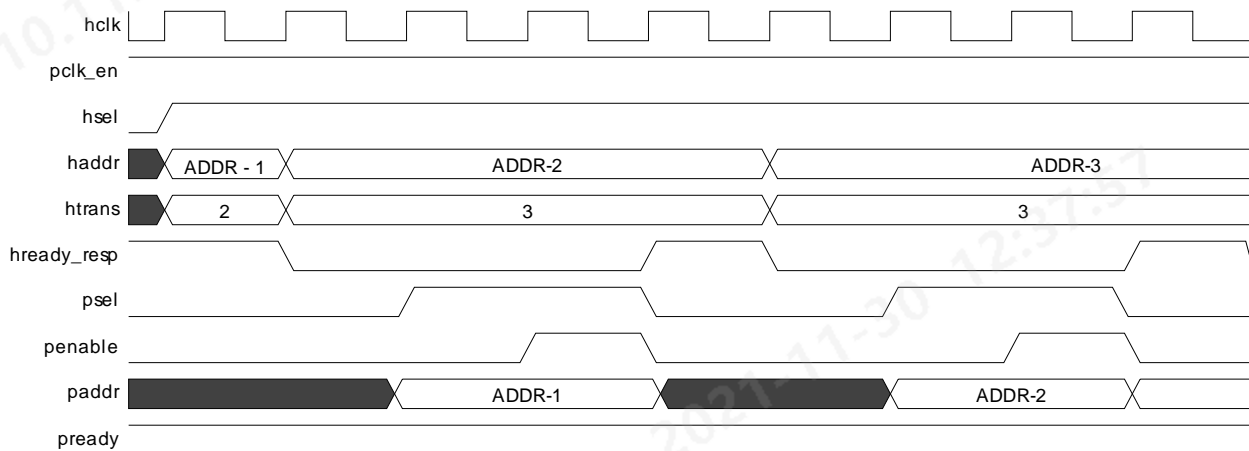
Figure 2-17 Back-to-Back Write Transfer (hclk != pclk)**Note**

Figure 2-16 and Figure 2-17 show the AHB issuing consecutive write transfers on the DW_apb, which are targeting AMBA 2 APB slaves. If any transfer targets an AMBA 3 APB slave, the bus brings hready low and the system stalls until each transfer completes on the APB bus.

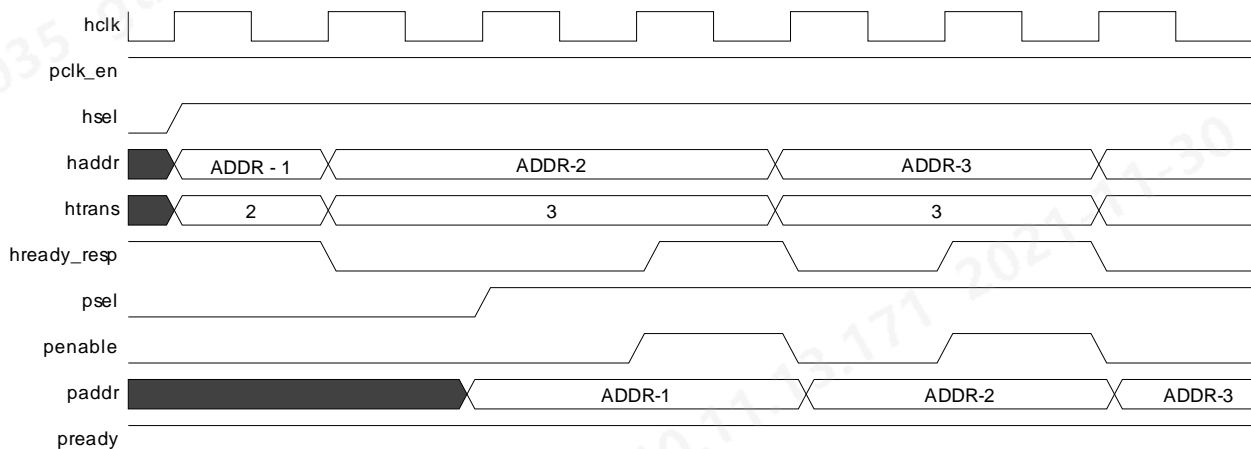
2.10 Back-to-Back Transfer Support on an APB Interface

By default, DW_apb waits for an APB transfer to complete to provide the response to an AHB interface and thereby introduces an additional cycle (wait cycle) on the APB interface. The additional cycle reduces the performance on the APB interface. DW_apb supports the APB_ENH_THROUGHPUT parameter to enable back-to-back transfers such that the wait cycle is removed and a back-to-back transfer occurs on an APB interface.

Figure 2-18 shows the three transfers on an APB when APB_ENH_THROUGHPUT is not set to 1 and clock ratio of pclk and hclk is 1. DW_apb issues next transfer only after the current transfer is over and there is always a 2-cycle delay between every APB transfer.

Figure 2-18 APB 3 Transfer When APB_ENH_THROUGHPUT_EN = 0

When APB_ENH_THROUGHPUT is set to 1, the 2-cycle delay between every APB transfer is removed and back-to-back transfer occurs on the APB interface that includes the overall bandwidth. [Figure 2-19](#) shows the APB 3 transfer when APB_ENH_THROUGHPUT is set to 1 and clock ratio of pclk and hclk is 1.

Figure 2-19 APB 3 Transfer When APB_ENH_THROUGHPUT_EN = 1**Note**

APB transactions are not back-to-back when:

- APB transfers return with an ERROR response.
- AHB Master inserts the next transfer after an APB slave receives a READY response for the current transfer.

2.11 APB4 Protocol Feature

2.11.1 Write Strobing

Write strobing allows AHB master to send data with width lesser than that defined by APB_DATA_WIDTH. The active data bytes are communicated to the APB peripheral by the pstrb signal. The mapping between hwdata and pwdata in case of varying haddr, hsize, and APB_DATA_WIDTH is shown independently in Figure 2-20, for APB2/APB3 and APB4 modes.

Figure 2-20 Mapping Between hwdata and pwdata

HADDR [1:0]	HSIZE (Decoded)	HWDAT A[31:0]	PADDR [1:0]	PSTRB	PWDATA
2'b00	32	7 6 5 4 →	2'b00	N/A	7 6 5 4
2'b00	16	7 6 5 4 →	2'b00	N/A	5 4
2'b10	16	7 6 5 4 →	2'b10	N/A	7 6
2'b00	8	7 6 5 4 →	2'b00	N/A	4
2'b01	8	7 6 5 4 →	2'b01	N/A	5
2'b10	8	7 6 5 4 →	2'b10	N/A	6
2'b11	8	7 6 5 4 →	2'b11	N/A	7

NOTE: APB_SLAVE_INTERFACE_x = APB2/APB3: HSIZE must match `APB_DATA_WIDTH`

HADDR [1:0]	HSIZE (Decoded)	HWDAT A[31:0]	PADDR [1:0]	PSTRB	PWDATA
2'b00	32	7 6 5 4 →	2'b00	4'b1111	7 6 5 4
2'b00	16	7 6 5 4 →	2'b00	4'b0011	7 6 5 4
2'b00	16	7 6 5 4 →	2'b00	4'b1100	7 6 5 4
2'b00	8	7 6 5 4 →	2'b00	4'b0001	7 6 5 4
2'b01	8	7 6 5 4 →	2'b00	4'b0010	7 6 5 4
2'b10	8	7 6 5 4 →	2'b00	4'b0100	7 6 5 4
2'b11	8	7 6 5 4 →	2'b00	4'b1000	7 6 5 4
2'b00	16	7 6 5 4 →	2'b00	2'b11	5 4
2'b10	16	7 6 5 4 →	2'b10	2'b11	7 6
2'b00	8	5 4 →	2'b00	2'b01	5 4
2'b01	8	7 6 5 4 →	2'b00	2'b10	5 4
2'b10	8	7 6 5 4 →	2'b10	2'b01	7 6
2'b11	8	7 6 5 4 →	2'b10	2'b10	7 6
2'b00	8	7 6 5 4 →	2'b00	1'b1	4
2'b01	8	7 6 5 4 →	2'b01	1'b1	5
2'b10	8	7 6 5 4 →	2'b10	1'b1	6
2'b11	8	7 6 5 4 →	2'b11	1'b1	7

NOTE: APB_SLAVE_INTERFACE_x = APB4: HSIZE can be less than or equal to `APB_DATA_WIDTH`

2.11.2 Protection

Setting APB_INTERFACE_TYPE_SLAVE_x to APB4 adds a three-bit pprot signal to its interface. Values to the signal are mapped in the following manner:

Table 2-1 pprot Signal Value Mapping

APB signal	AHB Signal
pprot [0]	hprot[1], if EXT_PROT_EN = 1 0, if EXT_PROT_EN=0
pprot [1]	hnonsec, if AHB_HAS_SECURE_XFER = 1 0, if AHB_HAS_SECURE_XFER = 0
pprot [2]	~hprot[0], if EXT_PROT_EN = 1 0, if EXT_PROT_EN=0

In any case, hprot[3:2] (cacheable, bufferable bits) are unused in design.

2.12 AMBA 5 AHB Features



Note

You must have DWC-AMBA-AHB5-Fabric-Source add-on license to access AHB5 features (AHB_INTERFACE_TYPE==1).

This section describes the following features supported by DW_apb that complies with AMBA 5 AHB protocol specification.

- “Secure Transfers” on page 38
- “Endianness” on page 39
- “User Signals” on page 42

These features are enabled when the configuration parameter AHB_INTERFACE_TYPE is set to 1.

2.12.1 Secure Transfers

DW_apb supports secure transfers feature when the AHB_HAS_SECURE_XFER parameter is set to 1. When this feature is selected, the DW_apb adds the hnonsec signal to its interface. This signal indicates whether the transfer is secure or non-secure. APB4 signal pprot[1] indicates the secure or non-secure transfer type to the slave.

- When APB slaves are configured for APB4 and the AHB_HAS_SECURE_XFER parameter is set to 1, DW_apb maps the hnonsec signal to pprot[1].
- When no slave is configured for APB4 and the AHB_HAS_SECURE_XFER parameter is set to 1, the hnonsec signal is not used by DW_apb.

2.12.2 Endianness

AHB5 introduces the Endian feature to define which form of big-endian data access is supported. When the BIG_ENDIAN parameter is set to 1, DW_apb supports the following big-endian data formats on the AHB data and user bus:

- Address-invariant (BE-A)
- Word-invariant (BE-32)
- Byte-invariant (BE-8)

2.12.2.1 Memory Access Table

Table 2-2 shows the effect of LE, BE-8, BE-32, and BE-A on a 64-bit-wide bus. It outlines the differences between word access, half-word access, and byte address for the different endian implementations. Definitions for items in Table 2-2 are as follows:

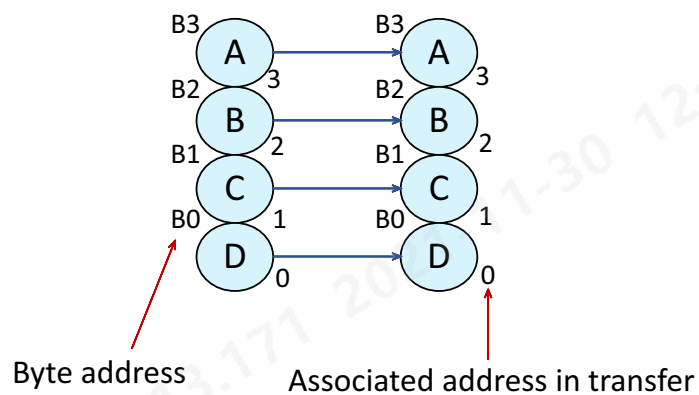
- ANum – Byte Access to address[2:0] = num
- ANum-Byte – Byte of word/half-word access to address[2:0] = num
- MS – Most significant byte
- MS-1 – Second most significant byte
- LS+1 – Second least significant byte
- LS – Least significant byte

Table 2-2 Endian Memory Access Table

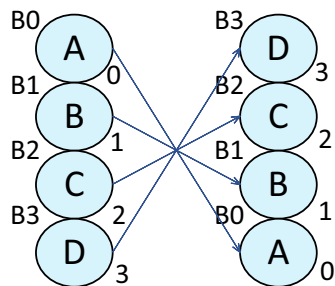
	Byte Access				Half-word Access				Word Access			
	LE	BE-8	BE-32	BE-A	LE	BE-8	BE-32	BE-A	LE	BE-8	BE-32	BE-A
63:56	A7	A7	A4	A0	A6:MS	A6:LS	A4:MS	A0	A4:MS	A4:LS	A4:MS	A0
55:48	A6	A6	A5	A1	A6:LS	A6:MS	A4:LS	A1	A4:MS-1	A4:LS+1	A4:MS-1	A1
47:40	A5	A5	A6	A2	A4:MS	A4:LS	A6:MS	A2	A4:LS+1	A4:MS-1	A4:LS+1	A2
39:32	A4	A4	A7	A3	A4:LS	A4:MS	A6:LS	A3	A4:LS	A4:MS	A4:LS	A3
31:24	A3	A3	A0	A4	A2:MS	A2:LS	A0:MS	A4	A0:MS	A0:LS	A0:MS	A4
23:16	A2	A2	A1	A5	A2:LS	A2:MS	A0:LS	A5	A0:MS-1	A0:LS+1	A0:MS-1	A5
15:8	A1	A1	A2	A6	A0:MS	A0:LS	A2:MS	A6	A0:LS+1	A0:MS-1	A0:LS+1	A6
7:0	A0	A0	A3	A7	A0:LS	A0:MS	A2:LS	A7	A0:LS	A0:MS	A0:LS	A7

2.12.2.2 Endian Transformations

The following diagrams show transformations for different endian representations. Figure 2-21 shows how to interpret the endian transformation diagrams.

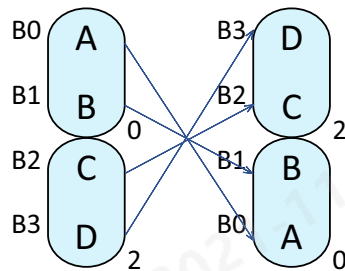
Figure 2-21 Key for Endian Transformation Diagrams**Key for Endian Transformation Diagrams**

8-bit

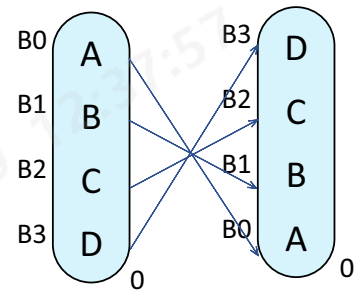


16-bit

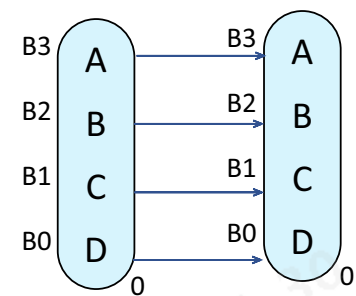
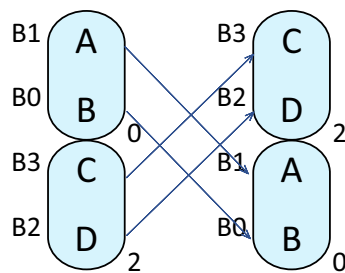
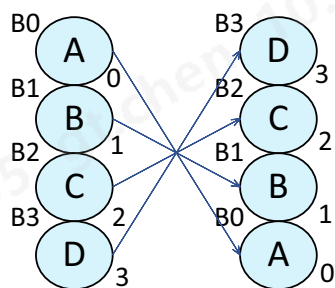
BE-A to LE



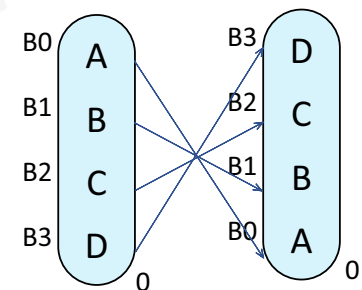
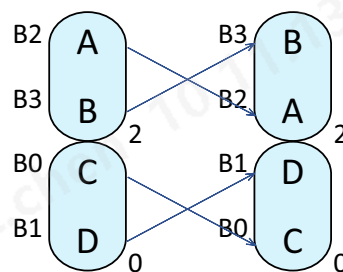
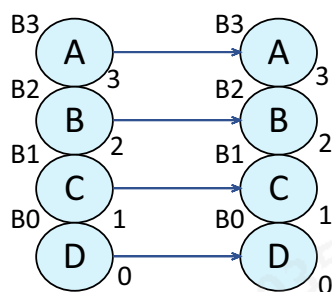
32-bit



BE-32 to LE



BE-8 to LE



2.12.3 User Signals

DW_apb supports user signals on each channel. The configuration parameters APB_R_UBW, APB_W_UBW, and APB_A_UBW determine the existence and width of the user signals on read data, write data and address channels. The AHB user signals `huser` are mapped to `hwuser` and `pwuser` respectively. `hruser` is mapped with `pruser` from the selected slave (`pruser_sN`).

The address channel user signals are transported unmodified across DW_apb.

The read and write data channel user signals can be transferred through DW_apb in two ways.

- “Pass-through (`USER_SIGNAL_XFER_MODE=0`)” on page 42
- “Aligned to Data (`USER_SIGNAL_XFER_MODE=1`)” on page 46

2.12.3.1 Pass-through (`USER_SIGNAL_XFER_MODE=0`)

In this mode, the width of write data and read data channel user signals is configured through the parameters APB_W_UBW and APB_R_UBW respectively and these signals are transported unmodified across DW_apb.

For timing of the user signals in pass-through mode, refer to the following timing diagrams:

- Figure 2-22 shows the write transfer from AHB to AMBA 3 APB with address and write data channel user signals (`hclk = pclk`)
- Figure 2-23 shows the read transfer from AHB to AMBA 3 APB with address and read data channel user signal (`hclk = pclk`)
- Figure 2-24 shows the write transfer from AHB to AMBA 3 APB with address and write data channel user signals (`hclk != pclk`)
- Figure 2-25 shows the read transfer from AHB to AMBA 3 APB with address and read data and Address Channel User Signal (`hclk != pclk`)

Figure 2-22 Write Transfer from AHB to AMBA 3 APB with Address and Write Data Channel User Signals (hclk = pclk)

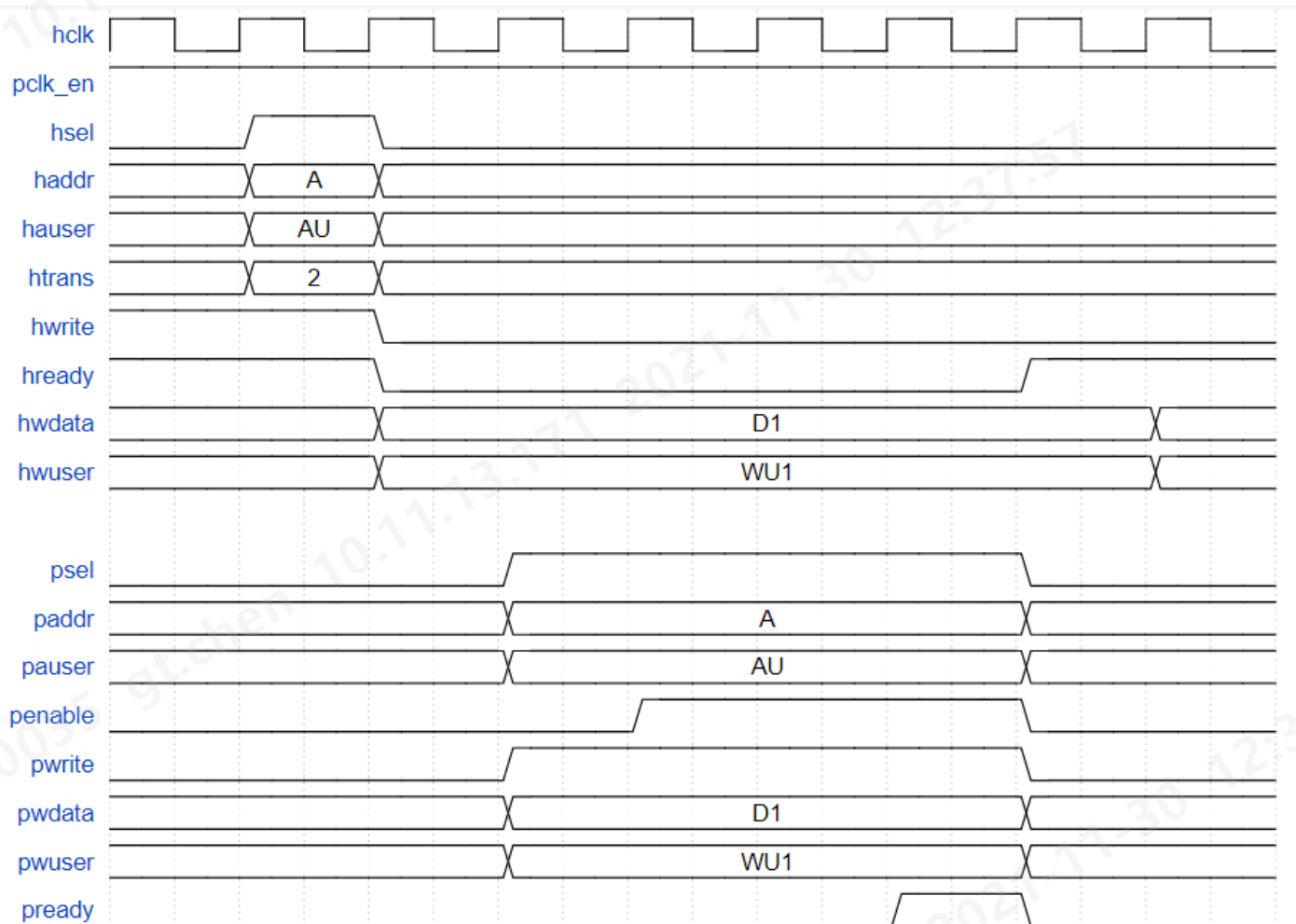


Figure 2-23 Read Transfer From AHB to AMBA 3 APB with Address and Read Data Channel User Signal (hclk = pclk)

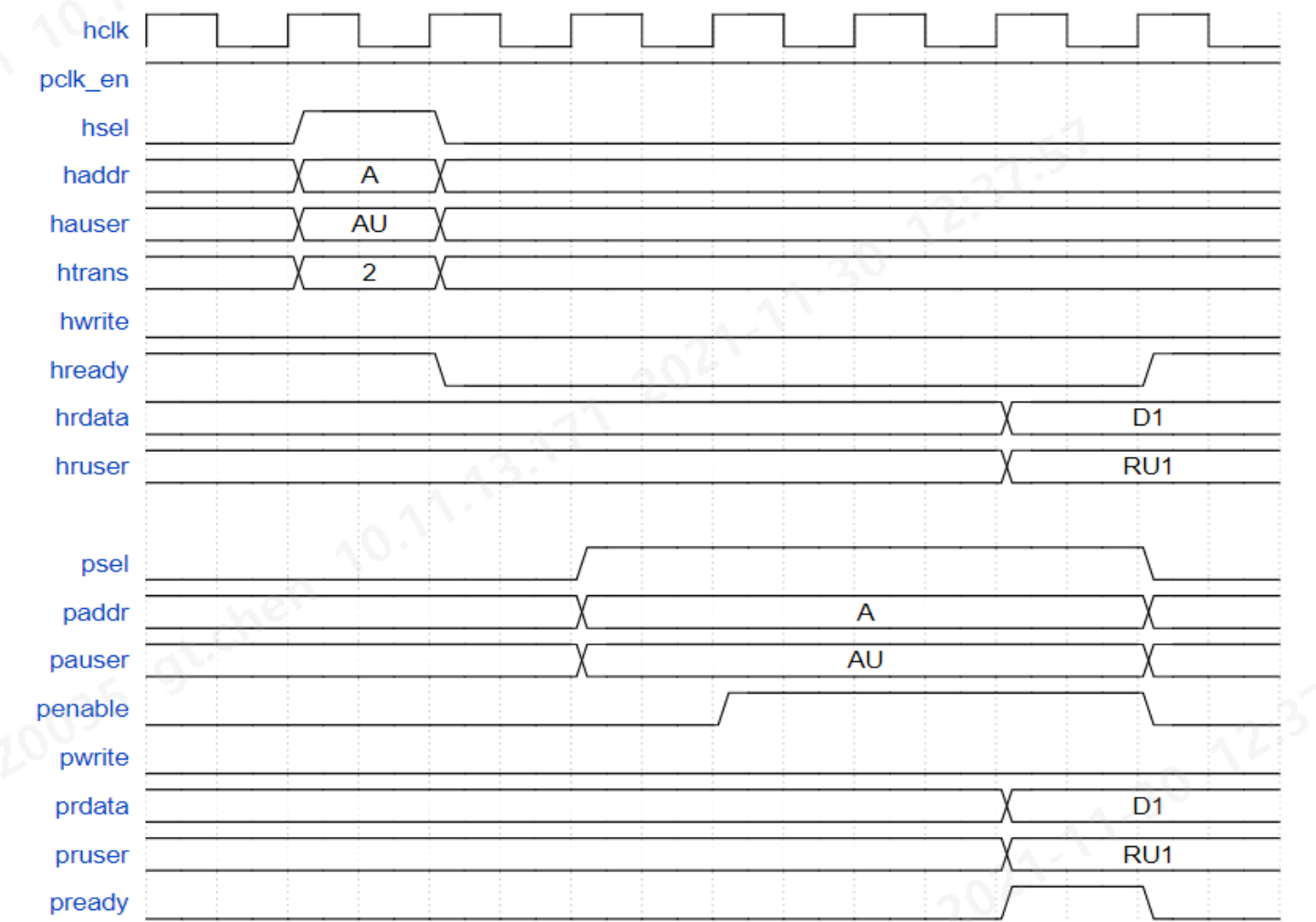


Figure 2-24 Write Transfer From AHB to AMBA 3 APB with Address and Write Data Channel User Signals (hclk != pclk)

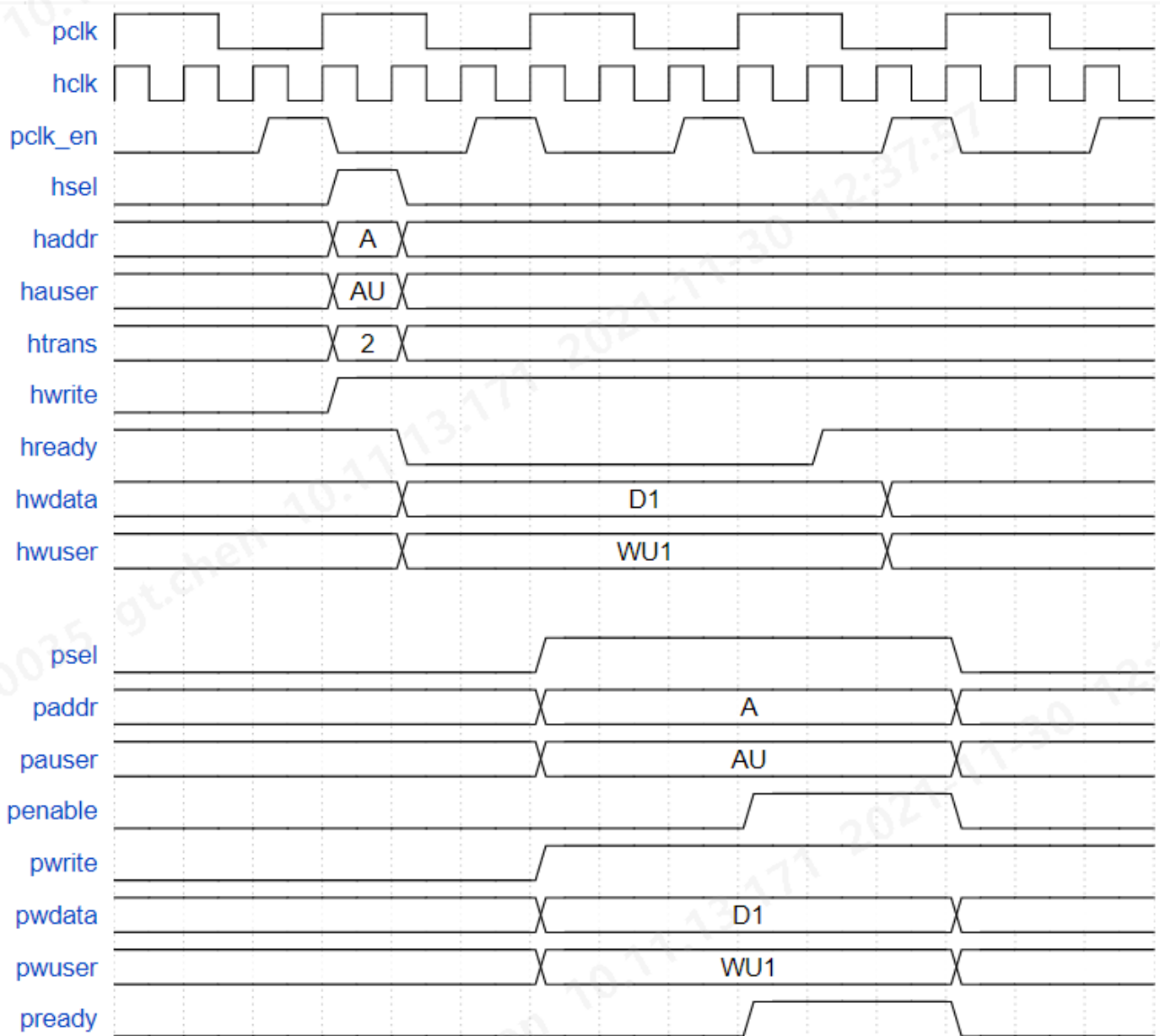
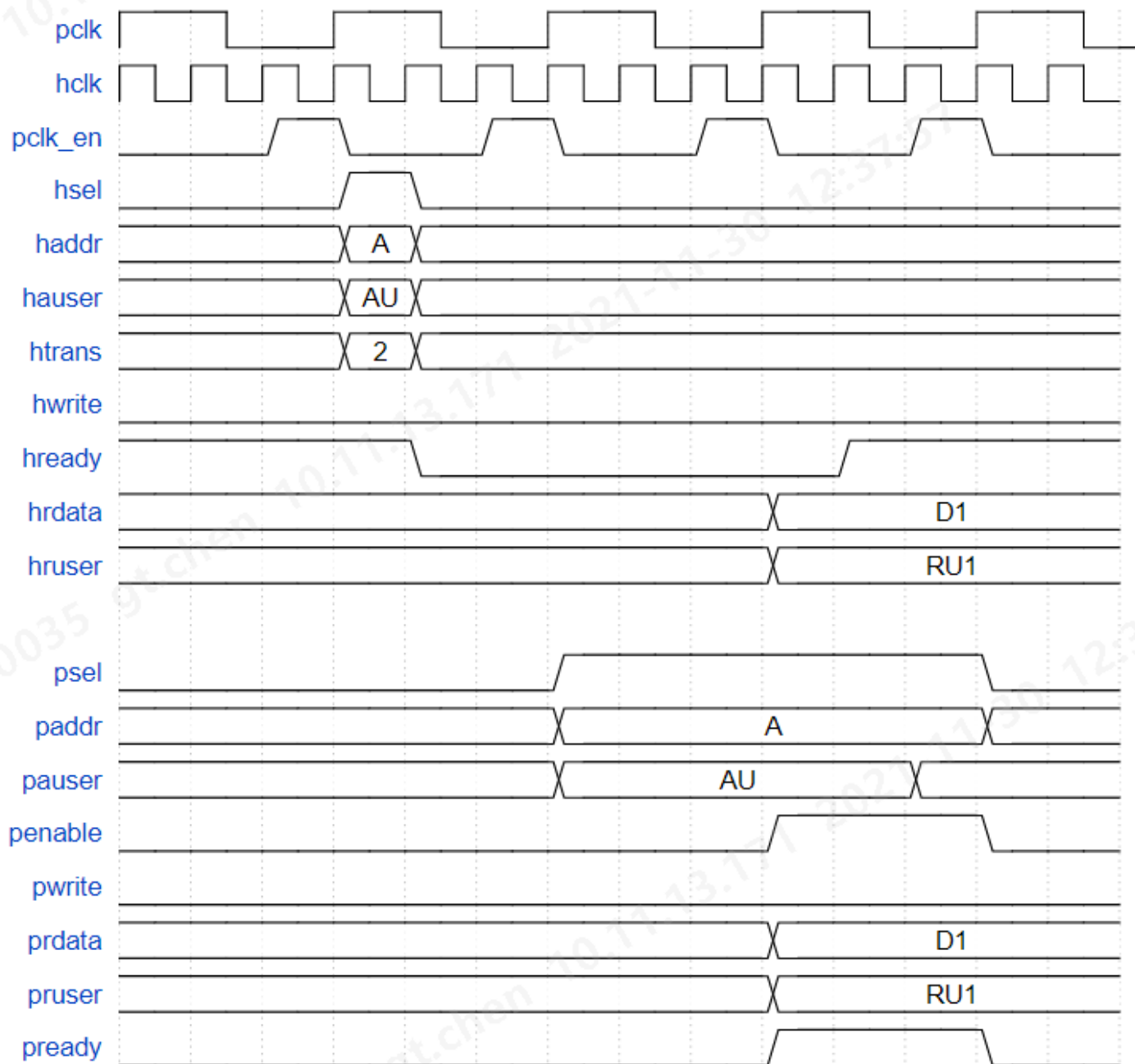


Figure 2-25 Read Transfer From AHB to AMBA 3 APB with Address and Read Data Channel User Signals (hclk != pclk)

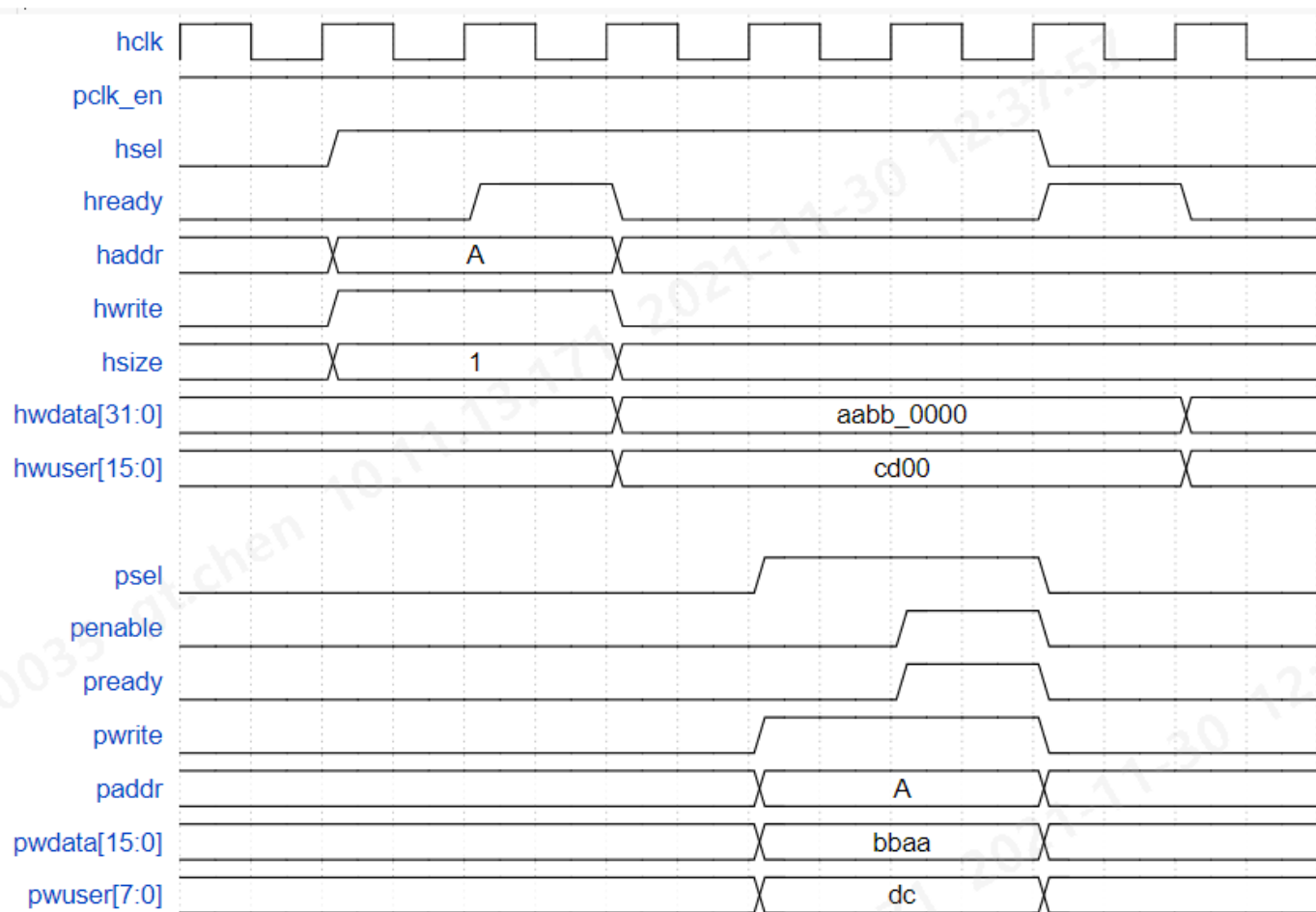


2.12.3.2 Aligned to Data (USER_SIGNAL_XFER_MODE=1)

In this mode, the number of read data and write data user signals bits per byte can be configured through the RUSER_BITS_PER_BYTE and the WUSER_BITS_PER_BYTE parameters respectively. These parameters determine the width of data channel user signals. As the data is transported across DW_apb, the user signal bits corresponding to the data bytes are transmitted. Endian conversion is also done on the user signals.

Figure 2-26 shows the timing of write data user signals in aligned to data mode for AHB_DATA_WIDTH=32, APB_DATA_WIDTH=16, WUSER_BITS_PER_BYTE=4 and AHB data is Address-invariant Big-Endian.

Figure 2-26 Write Transfer From AHB to AMBA 3 APB with User Signals in Aligned to Data Mode (hclk=pclk)



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 configuration options for this component.

- Top Level Parameters on [page 50](#)
- User Signal Configuration on [page 53](#)
- Address Map on [page 55](#)

3.1 Top Level Parameters

Table 3-1 Top Level Parameters

Label	Description
Top Level Parameters	
Select AHB Interface Type	<p>This parameter selects the AHB interface type.</p> <p>Values:</p> <ul style="list-style-type: none"> ■ AHB (0) ■ AHB5 (1) <p>Default Value: AHB</p> <p>Enabled: DWC-AMBA-AHB5-Fabric-Source Add on Source license exists.</p> <p>Parameter Name: AHB_INTERFACE_TYPE</p>
AHB System Address Width	<p>The address width of the AHB system.</p> <p>Values:</p> <ul style="list-style-type: none"> ■ 32 (32) ■ 64 (64) <p>Default Value: 32</p> <p>Enabled: Always</p> <p>Parameter Name: HADDR_WIDTH</p>
APB System Address Width	<p>The address width of the APB system.</p> <p>Note: The current DW_apb design supports APB address width of 32-bits only.</p> <p>Values:</p> <ul style="list-style-type: none"> ■ 32 (32) ■ 64 (64) <p>Default Value: 32</p> <p>Enabled: Always</p> <p>Parameter Name: PADDR_WIDTH</p>
AHB Data Bus Width	<p>The data width of the AHB bus.</p> <p>Values: 32, 64, 128, 256</p> <p>Default Value: 32</p> <p>Enabled: Always</p> <p>Parameter Name: AHB_DATA_WIDTH</p>
APB Data Bus Width	<p>The data width of the APB bus.</p> <p>Values: 8, 16, 32</p> <p>Default Value: 32</p> <p>Enabled: Always</p> <p>Parameter Name: APB_DATA_WIDTH</p>

Table 3-1 Top Level Parameters (Continued)

Label	Description
Single-bit AHB Response?	When set to true, hresp is a single-bit signal, else it is of 2 bits. Values: 0, 1 Default Value: 0 Enabled: Always Parameter Name: AHB_SCALAR_HRESP
AHB Endianness	The endianness of the AHB system. The APB subsystem is always little-endian. Values: <ul style="list-style-type: none"> Little-Endian (0) Big-Endian (1) Default Value: Little-Endian Enabled: Always Parameter Name: BIG_ENDIAN
Select the Type of Endian Conversion	This parameter selects the AHB big-endian type. Values: <ul style="list-style-type: none"> Address-invariant(BE-A) (0) Word-invariant(BE-32) (1) Byte-invariant(BE-8) (2) Default Value: Address-invariant(BE-A) Enabled: BIG_ENDIAN==1 && AHB_INTERFACE_TYPE==1 Parameter Name: BIG_ENDIAN_TYPE
Number of APB Slave Ports	The 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: NUM_APB_SLAVES
External Decoder?	If this parameter is set to True (1), the decoder is external to DW_apb. If False (0), the decoder is internal to DW_apb. For an internal decoder, the addresses needs to be supplied by DW_apb during configuration. An external decoder allows users to connect to any decoder. Values: <ul style="list-style-type: none"> false (0) true (1) Default Value: false Enabled: Always Parameter Name: APB_HAS_XDCDR

Table 3-1 Top Level Parameters (Continued)

Label	Description
Include HPROT signal to interface?	<p>Enabling this parameter includes the HPROT signal to the AHB interface.</p> <p>Values: 0, 1</p> <p>Default Value: 0</p> <p>Enabled: APB_INTERFACE_TYPE_SLAVE(x) configured for APB4 on any Slave</p> <p>Parameter Name: EXT_PROT_EN</p>
Include AHB5 Secure Transfers Property?	<p>Select this parameter to include AHB5 Secure Transfers Property in DW_apb. When set to 1, DW_apb adds hnonsec signal to its interface to support this property.</p> <p>Values:</p> <ul style="list-style-type: none"> ■ false (0) ■ true (1) <p>Default Value: false</p> <p>Enabled: AHB_INTERFACE_TYPE==1</p> <p>Parameter Name: AHB_HAS_SECURE_XFER</p>
Enable enhanced throughput on APB bus?	<p>If configured in this mode, DW_apb performs back-to-back transfers on the APB bus, if AHB master is providing back-to-back transfers. This increases the overall throughput.</p> <p>Values:</p> <ul style="list-style-type: none"> ■ No (0) ■ Yes (1) <p>Default Value: No</p> <p>Enabled: Always</p> <p>Parameter Name: APB_ENH_THROUGHPUT_EN</p>

3.2 User Signal Configuration Parameters

Table 3-2 User Signal Configuration Parameters

Label	Description
User Signal Configuration	
Width of Address Channel User Bus	<p>This parameter specifies the width of address channel user signal bus. When set to 0, the address channel user signals are removed from the interface.</p> <p>Values: 0, ..., 256</p> <p>Default Value: 0</p> <p>Enabled: USER_SIGNAL_XFER_MODE==0 and DWC-AMBA-AHB5-Fabric-Source Add on Source license exists.</p> <p>Parameter Name: APB_A_UBW</p>
Select Data Channel User Signal Transfer Mode	<p>This parameter selects whether the data channel user signals are to be transported as pass through or aligned to data.</p> <p>Values:</p> <ul style="list-style-type: none"> ■ Pass Through (0) ■ Aligned to data (1) <p>Default Value: Pass Through</p> <p>Enabled: DWC-AMBA-AHB5-Fabric-Source Add on Source license exists.</p> <p>Parameter Name: USER_SIGNAL_XFER_MODE</p>
Number of Write User Signal bits per Data Byte	<p>This parameter specifies the number of user signal bits corresponding to each byte of write data bus.</p> <p>Values: 0, ..., 8</p> <p>Default Value: 0</p> <p>Enabled: USER_SIGNAL_XFER_MODE==1</p> <p>Parameter Name: WUSER_BITS_PER_BYTE</p>
Number of Read User Signal bits per Data Byte	<p>This parameter specifies the number of user signal bits corresponding to each byte of read data bus.</p> <p>Values: 0, ..., 8</p> <p>Default Value: 0</p> <p>Enabled: USER_SIGNAL_XFER_MODE==1</p> <p>Parameter Name: RUSER_BITS_PER_BYTE</p>
Width of Write Data Channel User Bus	<p>This parameter specifies the width of write data channel user signal bus. When set to 0, the write data channel user signals are removed from the interface.</p> <p>Values: 0, ..., (USER_SIGNAL_XFER_MODE==1) ? ((AHB_DATA_WIDTH/8)*WUSER_BITS_PER_BYTE) : 256</p> <p>Default Value: (USER_SIGNAL_XFER_MODE==1) ? ((AHB_DATA_WIDTH/8)*WUSER_BITS_PER_BYTE) : 0</p> <p>Enabled: USER_SIGNAL_XFER_MODE==0 and DWC-AMBA-AHB5-Fabric-Source Add on Source license exists.</p> <p>Parameter Name: APB_W_UBW</p>

Table 3-2 User Signal Configuration Parameters (Continued)

Label	Description
Width of Read Data Channel User Bus	<p>This parameter specifies the width of read data channel user signal bus. When set to 0, the read data channel user signals are removed from the interface.</p> <p>Values: 0, ..., (USER_SIGNAL_XFER_MODE==1) ? ((AHB_DATA_WIDTH/8)*RUSER_BITS_PER_BYTE) : 256</p> <p>Default Value: (USER_SIGNAL_XFER_MODE==1) ? ((AHB_DATA_WIDTH/8)*RUSER_BITS_PER_BYTE) : 0</p> <p>Enabled: USER_SIGNAL_XFER_MODE==0 and DWC-AMBA-AHB5-Fabric-Source Add on Source license exists.</p> <p>Parameter Name: APB_R_UBW</p>
Width of APB Write Data Channel User Bus	<p>Width of APB write data channel user bus.</p> <p>Values: 0, ..., (USER_SIGNAL_XFER_MODE==1) ? ((APB_DATA_WIDTH/8)*WUSER_BITS_PER_BYTE) : 256</p> <p>Default Value: (USER_SIGNAL_XFER_MODE==1) ? ((APB_DATA_WIDTH/8)*WUSER_BITS_PER_BYTE) : APB_W_UBW</p> <p>Enabled: 0</p> <p>Parameter Name: APBS_W_UBW</p>
Width of APB Read Data Channel User Bus	<p>Width of APB read data channel user bus.</p> <p>Values: 0, ..., (USER_SIGNAL_XFER_MODE==1) ? ((APB_DATA_WIDTH/8)*RUSER_BITS_PER_BYTE) : 256</p> <p>Default Value: (USER_SIGNAL_XFER_MODE==1) ? ((APB_DATA_WIDTH/8)*RUSER_BITS_PER_BYTE) : APB_R_UBW</p> <p>Enabled: 0</p> <p>Parameter Name: APBS_R_UBW</p>

3.3 Address Map Parameters

Table 3-3 Address Map Parameters

Label	Description
Slave j	
Start Address of APB Slave x (for x = 0; x <= NUM_APB_SLAVES-1)	<p>The Start Address for APB Slave x. This is an absolute address value.</p> <p>Values: 0x00000000, ..., 0xfffffc00</p> <p>Default Value: For N=0: 0x00000400; for N=1: 0x00000800; for N=2: 0x00000c00; for N=3: 0x00001000; for N=4: 0x00001400; for N=5: 0x00001800; for N=6: 0x00001c00; for N=7: 0x00002000; for N=8: 0x00002400; for N=9: 0x00002800; for N=10: 0x00002c00; for N=11: 0x00003000; for N=12: 0x00003400; for N=13: 0x00003800; for N=14: 0x00003c00; for N=15: 0x00004000</p> <p>Enabled: NUM_APB_SLAVES > x && !APB_HAS_XDCDR</p> <p>Parameter Name: START_PADDR_(x)</p>
End Address of APB Slave x (for x = 0; x <= NUM_APB_SLAVES-1)	<p>The End Address for APB Slave x. This is an absolute address value.</p> <p>Values: 0x000003ff, ..., 0xffffffff</p> <p>Default Value: For N=0: 0x000007ff; for N=1: 0x00000bff; for N=2: 0x00000fff; for N=3: 0x000013ff; for N=4: 0x000017ff; for N=5: 0x00001bff; for N=6: 0x00001fff; for N=7: 0x000023ff; for N=8: 0x000027ff; for N=9: 0x00002bff; for N=10: 0x00002fff; for N=11: 0x000033ff; for N=12: 0x000037ff; for N=13: 0x00003bff; for N=14: 0x00003fff; for N=15: 0x000043ff</p> <p>Enabled: NUM_APB_SLAVES > x && !APB_HAS_XDCDR</p> <p>Parameter Name: END_PADDR_(x)</p>
APB Slave Interface Type (for x = 0; x <= NUM_APB_SLAVES-1)	<p>Select between AMBA 4 APB slave (APB4), AMBA 3 APB slave (APB3), and AMBA 2 APB slave (APB2) for Slave x. If AMBA 3 APB Slave is selected, the additional ports PREADY and PSLVERR are included. If AMBA 4 APB Slave is selected, the additional ports PSTRB and PPROT are included.</p> <p>Values:</p> <ul style="list-style-type: none"> ■ APB2 (0) ■ APB3 (1) ■ APB4 (2) <p>Default Value: APB2</p> <p>Enabled: NUM_APB_SLAVES > x && !APB_HAS_XDCDR</p> <p>Parameter Name: APB_INTERFACE_TYPE_SLAVE_(x)</p>

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:

- Clocks and Resets on [page 59](#)
- AHB Slave Interface Signals on [page 60](#)
- APB Interface Signals on [page 64](#)

4.1 Clocks and Resets Signals

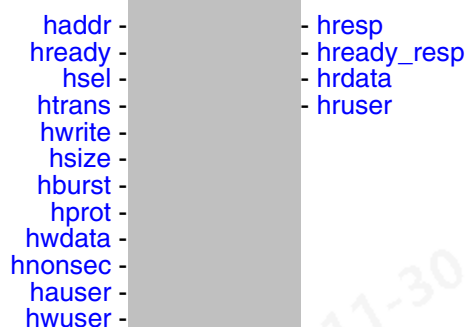
hclk -
hresetn -
pclk_en -



Table 4-1 Clocks and Resets Signals

Port Name	I/O	Description
hclk	I	<p>AHB Clock Signal. This clock times all bus transfers. All signal timings are related to the rising edge of hclk.</p> <p>Exists: Always</p> <p>Synchronous To: None</p> <p>Registered: N/A</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: N/A</p>
hresetn	I	<p>AHB Reset Signal. This signal is used to reset the system and the bus on the DesignWare interface. The reset must be synchronously deasserted after the rising edge of hclk. Since DW_apb does not contain logic to perform this synchronization, it must be provided externally. During reset, masters must ensure that address and control signals are at valid levels, and that htrans indicates the IDLE state.</p> <p>Exists: Always</p> <p>Synchronous To: Asynchronous</p> <p>Registered: N/A</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: Low</p>
pclk_en	I	<p>APB Clock Enable Strobe. This signal is of one hclk cycle duration and identifies the hclk rising edge that corresponds with a pclk rising edge. Tied high by the user if pclk = hclk.</p> <p>Exists: Always</p> <p>Synchronous To: hclk</p> <p>Registered: No</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: High</p>

4.2 AHB Slave Interface Signals



haddr -
 hready -
 hsel -
 htrans -
 hwrite -
 hsize -
 hburst -
 hprot -
 hwdata -
 hnonsec -
 hauser -
 hwuser -

hresp
 hready_resp
 hrdata
 hruser

Table 4-2 AHB Slave Interface Signals

Port Name	I/O	Description
haddr[(HADDR_WIDTH-1):0]	I	Address bus from AHB master. Exists: Always Synchronous To: hclk Registered: No Power Domain: SINGLE_DOMAIN Active State: N/A
hready	I	Ready response for DW_apb. Exists: Always Synchronous To: hclk Registered: No Power Domain: SINGLE_DOMAIN Active State: High
hsel	I	When asserted, the signal indicates that the DW_apb has been selected. Each AHB slave has its own hsel line. This is driven by the AHB decoder block. Exists: Always Synchronous To: hclk Registered: No Power Domain: SINGLE_DOMAIN Active State: High

Table 4-2 AHB Slave Interface Signals (Continued)

Port Name	I/O	Description
htrans[1:0]	I	Transfer type from selected master. Exists: Always Synchronous To: hclk Registered: No Power Domain: SINGLE_DOMAIN Active State: N/A
hwrite	I	When hwrite is high, there is a write transfer and the master broadcasts data on the write data bus (hwdata). When hwrite is low, a read transfer is performed, and the DW_apb must generate the data on the read data bus (hrdata). Exists: Always Synchronous To: hclk Registered: No Power Domain: SINGLE_DOMAIN Active State: High
hsize[2:0]	I	Transfer size. Indicates the size of the transfer. This signal is used in the component only when the component is configured to have an APB4 enabled interface. In this case, hsize can supply values which decode to a width less than or equal to APB_DATA_WIDTH. Otherwise, each transfer requires a hsize of APB_DATA_WIDTH, and the input value is assumed to decode to the same value. The signal is left unconnected in this case. Exists: Always Synchronous To: hclk Registered: No Power Domain: SINGLE_DOMAIN Active State: N/A
hburst[(HBURST_WIDTH-1):0]	I	Burst type indication from selected AHB master. This signal is left unconnected on the interface because it is not required. Exists: Always Synchronous To: None Registered: No Power Domain: SINGLE_DOMAIN Active State: N/A

Table 4-2 AHB Slave Interface Signals (Continued)

Port Name	I/O	Description
hprot[(HPROT_WIDTH-1):0]	I	<p>AHB Protection type signal. HPROT values are mapped to relevant pprot signal if the input signal is included. Otherwise, default values (3'b000) are copied to the pprot signals upon access.</p> <p>When DW_apb is configured for AHB5 interface (AHB_INTERFACE_TYPE=1), the width of hprot signal is increased to 7-bits when Extended Memory Types property is enabled, in order to maintain interface consistency. The additional bits, hprot[6:4] are not used by the DW_apb.</p> <p>Exists: (EXT_PROT_EN==1)</p> <p>Synchronous To: hclk</p> <p>Registered: No</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: N/A</p>
hresp[(HRESP_WIDTH-1):0]	O	<p>Transfer Response. When all the slaves are configured as APB2, this signal always indicates an OKAY response. When hready_resp is High, this shows the transfer has completed successfully.</p> <p>When slaves are configured as APB3 or APB4, hresp can give OKAY or ERROR response, indicating whether the transfer has completed successfully or not.</p> <p>SPLIT and RETRY responses are not supported by DW_apb.</p> <p>Exists: Always</p> <p>Synchronous To: hclk</p> <p>Registered: No</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: N/A</p>
hready_resp	O	<p>Response from DW_apb. Asserted when current transfer has completed.</p> <p>Exists: Always</p> <p>Synchronous To: hclk</p> <p>Registered: APB_ENH_THROUGHPUT_EN==0 ? Yes : (APB_HAS_APB3==0 ? Yes : No)</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: High</p>
hwddata[(AHB_DATA_WIDTH-1):0]	I	<p>Write data bus from selected AHB master.</p> <p>Exists: Always</p> <p>Synchronous To: hclk</p> <p>Registered: No</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: N/A</p>

Table 4-2 AHB Slave Interface Signals (Continued)

Port Name	I/O	Description
hnonsec	I	Secure or non-secure transfer type from AHB Master. When asserted, this signal indicates that the current transfer is a Non-secure transfer. When de-asserted the transfer is a Secure transfer. Exists: AHB_HAS_SECURE_XFER==1 Synchronous To: hclk Registered: No Power Domain: SINGLE_DOMAIN Active State: High
hauser[(APB_A_UBW-1):0]	I	Address channel user bus from AHB master. When the parameter APB_A_UBW is set to 0, these signals are removed from the interface. Exists: APB_A_UBW>0 Synchronous To: hclk Registered: No Power Domain: SINGLE_DOMAIN Active State: N/A
hwuser[(APB_W_UBW-1):0]	I	Write data channel user bus from AHB master. When the parameter APB_W_UBW is set to 0, these signals are removed from the interface. Exists: APB_W_UBW>0 Synchronous To: hclk Registered: No Power Domain: SINGLE_DOMAIN Active State: N/A
hrdata[(AHB_DATA_WIDTH-1):0]	O	Transfer read data. The read data bus is used to transfer data from DW_apb to the bus master during read operations. Exists: Always Synchronous To: hclk Registered: No Power Domain: SINGLE_DOMAIN Active State: N/A
hruser[(APB_R_UBW-1):0]	O	Read data channel user bus. When the parameter APB_R_UBW is set to 0, these signals are removed from the DW_apb. Exists: APB_R_UBW>0 Synchronous To: hclk Registered: No Power Domain: SINGLE_DOMAIN Active State: N/A

4.3 APB Interface Signals

`xpsel_sN` (for $N = 0; N \leq \text{NUM_APB_SLAVES}-1$) - `paddr`
`prdata_sN` (for $N = 0; N \leq \text{NUM_APB_SLAVES}-1$) - `penable`
`pready_sN` (for $N = 0; N \leq \text{NUM_APB_SLAVES}-1$) - `pwrite`
`pslverr_sN` (for $N = 0; N \leq \text{NUM_APB_SLAVES}-1$) - `pprot`
`pruser_sN` (for $N = 0; N \leq \text{NUM_APB_SLAVES}-1$) - `pstrb`
`psel_sN` (for $N = 0; N \leq \text{NUM_APB_SLAVES}-1$)
`pwuser`
`pauser`
`pwdata`

Table 4-3 APB Interface Signals

Port Name	I/O	Description
<code>paddr[(PADDR_WIDTH-1):0]</code>	O	<p>APB address bus. Can change on only a <code>pclk_en</code> active edge. It retains its last value, even though there may be no activity on the APB bus, until it is overwritten by a new address.</p> <p>Exists: Always Synchronous To: <code>hclk</code> Registered: Yes Power Domain: SINGLE_DOMAIN Active State: N/A</p>
<code>penable</code>	O	<p>Enable Strobe. Asserted to validate APB transfer. It is always driven low at the end of each APB access.</p> <p>Exists: Always Synchronous To: <code>hclk</code> Registered: Yes Power Domain: SINGLE_DOMAIN Active State: High</p>
<code>pwrite</code>	O	<p>APB Read/Write Signal. When <code>pwrite</code> is high, there is a write transfer and data is broadcast on the write data bus (<code>pwdata</code>). When <code>pwrite</code> is low, a read transfer is performed, and the slave must generate the data on its read data bus (<code>prdata</code>).</p> <p>Exists: Always Synchronous To: <code>hclk</code> Registered: Yes Power Domain: SINGLE_DOMAIN Active State: High</p>

Table 4-3 APB Interface Signals (Continued)

Port Name	I/O	Description
pprot[2:0]	O	<p>APB4 Protection type signal. HPROT values are mapped to relevant pprot signal if the input signal is included. When AHB5 secure transfers property is selected, hnonsec signal is mapped to pprot[1]. Else, default values are copied to the pprot signals upon access.</p> <p>Exists: APB_HAS_APB4==1</p> <p>Synchronous To: hclk</p> <p>Registered: Yes</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: N/A</p>
pstrb[((APB_DATA_WIDTH/8)-1):0]	O	<p>APB4 Write strobe bus. HSIZE of an incoming transaction may now be lower than APB_DATA_WIDTH. This value along with address offset is used to generate strobe signals on APB side to selectively write to certain bytes in the apb data bus.</p> <p>Exists: APB_HAS_APB4==1</p> <p>Synchronous To: hclk</p> <p>Registered: Yes</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: High</p>
psel_sN (for N = 0; N <= NUM_APB_SLAVES-1)	O	<p>Select lines for APB slaves (one per slave)</p> <p>Exists: N < NUM_APB_SLAVES</p> <p>Synchronous To: hclk</p> <p>Registered: No</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: High</p>
xpsel_sN (for N = 0; N <= NUM_APB_SLAVES-1)	I	<p>Slave select line for APB.</p> <p>Exists: N < NUM_APB_SLAVES</p> <p>Synchronous To: hclk</p> <p>Registered: No</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: N/A</p>
prdata_sN[(APB_DATA_WIDTH-1):0] (for N = 0; N <= NUM_APB_SLAVES-1)	I	<p>Read Data from APB slaves. The width of each bus is APB_DATA_WIDTH.</p> <p>Exists: N < NUM_APB_SLAVES</p> <p>Synchronous To: hclk</p> <p>Registered: No</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: N/A</p>

Table 4-3 APB Interface Signals (Continued)

Port Name	I/O	Description
pready_sN (for N = 0; N <= NUM_APB_SLAVES-1)	I	Indicates whether a request cycle was accepted. Exists only on slave interfaces configured as APB3 or APB4. Exists: N < NUM_APB_SLAVES Synchronous To: hclk Registered: No Power Domain: SINGLE_DOMAIN Active State: High
pslverr_sN (for N = 0; N <= NUM_APB_SLAVES-1)	I	Flag for the slave error response from APB. Exists only on slave interfaces configured as APB3 or APB4. Exists: N < NUM_APB_SLAVES Synchronous To: hclk Registered: No Power Domain: SINGLE_DOMAIN Active State: High
pruser_sN[(APBS_R_UBW-1):0] (for N = 0; N <= NUM_APB_SLAVES-1)	I	Read data channel user bus. When the parameter APB_R_UBW is set to 0, these ports are removed from the DW_apb. Exists: N < NUM_APB_SLAVES && APB_R_UBW > 0 Synchronous To: hclk Registered: No Power Domain: SINGLE_DOMAIN Active State: N/A
pwuser[(APBS_W_UBW-1):0]	O	Write data channel user bus to slave. When the parameter APB_W_UBW is set to 0, these signals are removed from the interface. Exists: APB_W_UBW>0 Synchronous To: hclk Registered: APB_ENH_THROUGHPUT_EN == 0 ? "Yes" : "No" Power Domain: SINGLE_DOMAIN Active State: N/A
pauser[(APB_A_UBW-1):0]	O	Address channel user bus to slave. When the parameter APB_A_UBW is set to 0, these signals are removed from the interface. Exists: APB_A_UBW>0 Synchronous To: hclk Registered: Yes Power Domain: SINGLE_DOMAIN Active State: N/A

Table 4-3 APB Interface Signals (Continued)

Port Name	I/O	Description
pwwdata[(APB_DATA_WIDTH-1):0]	O	APB transfer write data bus shared by all slaves. Exists: Always Synchronous To: hclk Registered: No Power Domain: SINGLE_DOMAIN Active State: N/A

5

Verification

This chapter provides an overview of the testbench available for the DW_apb verification. After the DW_apb has been configured and the verification setup, simulations can be run automatically. For information on running simulations for DW_apb in coreAssembler or coreConsultant, see the “Simulating the Core” section in *DesignWare Synthesizable Components for AMBA 2 User Guide*.

**Note**

The DW_apb verification testbench is built with DesignWare Verification IP (VIP). Make sure 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, refer to the *DesignWare Synthesizable Components for AMBA 2/AMBA 3 AXI Installation Guide*.

**Note**

The packaged test benches are only for validating the IP configuration in coreConsultant GUI. It is not for system level validation.
IPs that have the Vera test bench packaged, these test benches are encrypted.

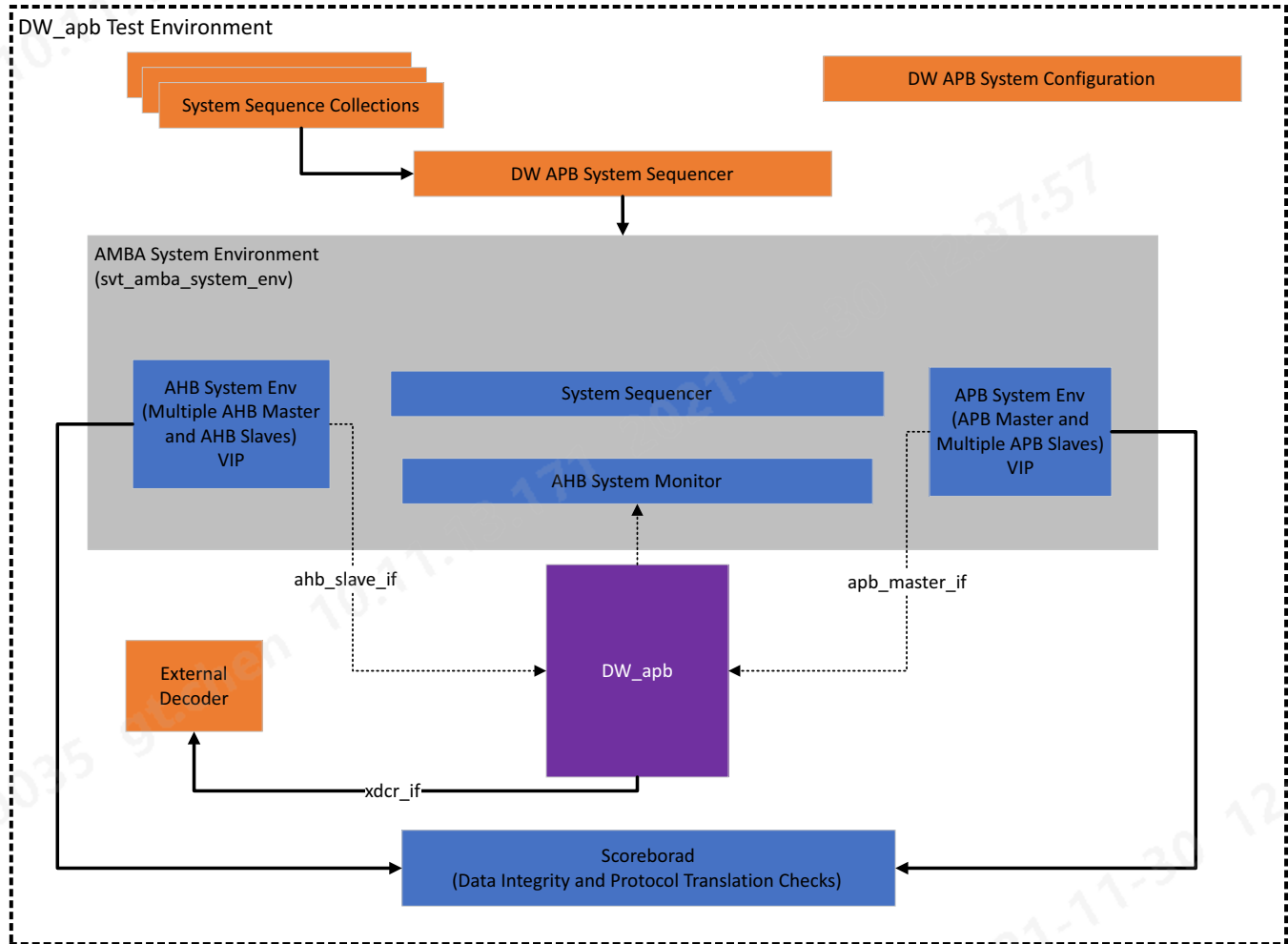
This chapter discusses the following sections:

- “[Verification Environment](#)” on page 69
- “[Testbench Directories and Files](#)” on page 71
- “[Packaged Testcases](#)” on page 72

5.1 Verification Environment

DW_apb is verified using a UVM-methodology-based constrained random verification environment. The environment is capable of generating random scenarios and the test case has hooks to control the scenarios to be generated.

[Figure 5-1](#) shows the verification environment of the DW_apb testbench:

Figure 5-1 DW_apb UVM Verification Environment

The testbench consists of the following elements:

- Testbench uses the standard SVT VIP for the protocol interfaces:
 - AMBA SVT VIP
 - SVT_AHB VIP Interface for connecting master ports of DUT.
 - SVT_APB VIP Interface for connecting slave ports of DUT.
 - AHB system environment is configured with two active master agents (master[0] as the dummy master and master[1] to drive the DUT AHB slave interface), AHB bus agent and 2 slave agents (an active slave agent slave[0], as the default slave and a passive slave agent slave[1]). The active AHB master agent is responsible for creating AHB traffic on the AHB slave interface of DW_APB.
 - APB system environment is configured to support N number of APB slave agents. These slaves models the APB slaves, which gets connected to DW_APB's APB slave ports.
- Testbench uses the custom components:

□ External Decoder

DW_apb supports an RTL configuration to have an external decoder to drive the peripheral slave outputs. When this option is chosen, by setting the APB_HAS_XDCDR parameter, the internal decoder is not included and the design will have inputs for the peripheral selects(xpsel_s0..xpsel_s(i)), which needs to be driven by the external decoder. These external decoder signals (xpsels) pass through the bridge and drive the peripheral select outputs of DW_apb.

□ Sequence library

The system sequence library consists of a set of sequence from VIP as well as a set of sequence developed to create traffic from design side.

□ Scoreboard

The data integrity, AHB to APB protocol conversion and correctness of slave error mapping is accomplished with a UVM based scoreboard class. The scoreboard component supports 2 analysis exports, one for AHB system's slave[1] monitor and the other for the APB system's N number of slave monitors. Logic in the scoreboard has been implemented in a generic way to support all APB protocol versions, any AHB/APB address_width and data_widths.

□ System Configuration

This object takes care of configuring the complete system, which includes the design and verification IPs. As there are no register configurations available in the design, only VIP configuration handles are present in the System configuration class.

- i. Design/DUT configuration class: Consists of the design cc_constants mapped to dw_apb_config class fields.
- ii. AMBA system configuration class: This is the configuration class provided along with the amba system environment. This provides separate handles to the AHB system configuration and APB system configuration classes.

5.2 Testbench Directories and Files

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

Table 5-1 shows the various directories and associated files:

Table 5-1 DW_apb Testbench Directory Structure

Directory	Description
<configured workspace>/sim/testbench	Contains the 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/	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.

5.3 Packaged Testcases

The simulation environment that comes as a package files 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 shipped test cases and their description is explained in [Table 5-2](#).

Table 5-2 DW_apb Test Description

Test Name	Test Description
test_100_random	<p>This test subjects the DUT to random AHB traffic from the master and randomized response is reported from the slave. The test incorporates all possible scenarios under a single umbrella testing scheme.</p> <ul style="list-style-type: none"> ■ Randomized stimulus including single write (test_1_single_write) ■ Single read(test_2_single_read) ■ Single write immediately followed by single read(test_3_single_wr_rd) ■ Write bursts and read bursts (test_4_burst_write&test_5_burst_read) ■ Continuous single write (test_6_bk2bk_single_write) ■ Continuous single read (test_7_bk2bk_single_read)
test_101_master_connectivity	<p>The test ensures that all possible slaves are connected to each master and randomized stimulus is driven to each one of them to confirm correctness of functionality of all the possible connected masters and slaves.</p>
test_102_pslverr	<p>The test includes ERROR response scenarios initiated by different APB slaves for random sequences issued by the AHB master.</p>

6

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.

6.1 Performance

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

6.1.1 Power Consumption, Frequency, Area and DFT Coverage

Table 6-1 provides information about the synthesis results (power consumption, frequency and area) and DFT coverage of the DW_apb using the industry standard 7nm technology library.

Table 6-1 Synthesis Results for DW_apb

Configuration	Operating Frequency	Gate Count	Power Consumption		TetraMax Coverage (%)		SpyGlass StuckAtCov(%)
			Static Power	Dynamic Power	StuckAtTest	Transition	
Default Configuration	pclk= 100 MHz	2172	6 nW	0.040 mW	99.9	99.58	100
Typical Configuration 1 AHB_DATA_WIDTH = 32 APB_DATA_WIDTH = 16 NUM_APB_SLAVES = 8 APB_INTERFACE_TYPE_SLAVE_1 = 2 APB_INTERFACE_TYPE_SLAVE_3 = 2 APB_INTERFACE_TYPE_SLAVE_5 = 2 APB_INTERFACE_TYPE_SLAVE_7 = 2 BIG_ENDIAN = 1 APB_HAS_XDCDR = 0 EXT_PROT_EN = 1 APB_ENH_THROUGHPUT_EN = 1 AHB_INTERFACE_TYPE = 1 AHB_HAS_SECURE_XFER = 1 USER_SIGNAL_XFER_MODE = 1 APB_A_UBW = 4 WUSER_BITS_PER_BYTE 5 RUSER_BITS_PER_BYTE 4 BIG_ENDIAN_TYPE =1	pclk=100 MHz	2190	6 nW	0.047 mW	99.96	99.7	100

6.2 Accessing Top-level Constraints

To get SDC constraints out of coreConsultant, you need to first complete the synthesis activity and then use the “write_sdc” command to write out the results:

1. This cC command sets synthesis to write out scripts only, without running DC:

```
set_activity_parameter Synthesize ScriptsOnly 1
```

2. This cC command autocompletes the activity:

```
autocomplete_activity Synthesize
```

3. Finally, this cC command writes out SDC constraints:

```
write_sdc <filename>
```

6.3 Reading and Writing from an APB Slave

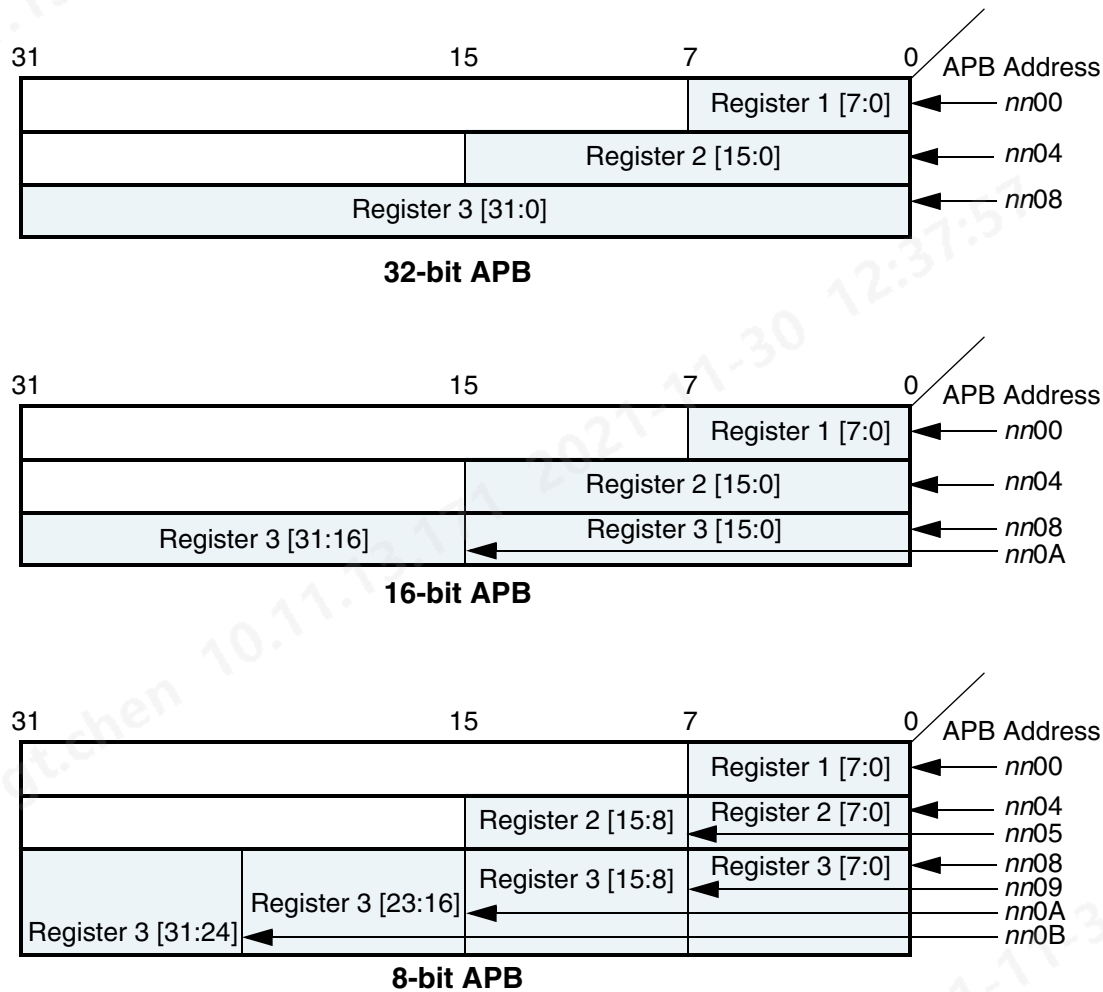
When writing to and reading from DesignWare APB slaves, you should consider the following:

- The size of the APB peripheral should always be set equal to the size of the APB data bus, if possible.
- The APB bus has no concept of a transfer size or a byte lane, unlike the DW_ahb.
- The APB slave subsystem is little endian; the DW_apb performs the conversion from a big-endian AHB to the little-endian APB.
- All APB slave programming registers are aligned on 32-bit boundaries, irrespective of the APB bus size.
- The maximum APB_DATA_WIDTH is 32 bits. Registers larger than this occupy more than one location in the memory map.
- The DW_apb does not return any SPLIT or RETRY response; it always returns an OKAY or ERROR response to the AHB.
- For all bus widths:
 - In the case of a read transaction, registers less than the full bus width returns zeros in the unused upper bits.
 - Writing to bit locations larger than the register width does not have any effect. Only the pertinent bits are written to the register.
- The APB slaves do not need the full 32-bit address bus, paddr. The slaves include the lower bits even though they are not actually used in a 32- or 16-bit system.

6.3.1 Reading From Unused Locations

Reading from an unused location or unused bits in a particular register always returns zeros. The following sections show the relationship between the register map and the read/write operations for the three possible APB_DATA_WIDTH values: 8-, 16-, and 32-bit APB buses.


Figure 6-1 Read/Write Locations for Different APB Bus Data Widths



6.3.2 32-bit Bus System

For 32-bit bus systems, all programming registers can be read or written with one operation, as illustrated in the previous figure.

Because all registers are on 32-bit boundaries, paddr[1:0] is not actually needed in the 32-bit bus case. But these bits still exist in the configured code for usability purposes.

 **Note** If you write to an address location not on a 32-bit boundary, the bottom bits are ignored/not used.

6.3.3 16-bit Bus System

For 16-bit bus systems, two scenarios exist, as illustrated in the previous picture:

1. The register to be written to or read from is less than or equal to 16 bits

In this case, the register can be read or written with one transaction. In the case of a read transaction, registers less than 16 bits wide returns zeros in the un-used bits. Writing to bit locations larger than the register width causes nothing to happen, i.e. only the pertinent bits are written to the register.

2. The register to be written to or read from is >16 and ≤ 32 bits

In this case, two AHB transactions are required, which in turn creates two APB transactions, to read or write the register. The first transaction should read/write the lower two bytes (half-word) and the second transaction the upper half-word.

Because the bus is reading a half-word at a time, `paddr[0]` is not actually needed in the 16-bit bus case. But these bits still exist in the configured code for connectivity purposes.



Note

If you write to an address location not on a 16-bit boundary, the bottom bits are ignored/not used.

6.3.4 8-bit Bus System

For 8-bit bus systems, three scenarios exist, as illustrated in the previous picture:

1. The register to be written to or read from is less than or equal to 8 bits

In this case, the register can be read or written with one transaction. In the case of a read transaction, registers less than 8 bits wide returns zeros in the unused bits. Writing to bit locations larger than the register width causes nothing to happen, that is, only the pertinent bits are written to the register.

2. The register to be written to or read from is >8 and ≤ 16 bits

In this case, two AHB transactions are required, which in turn creates two APB transactions, to read or write the register. The first transaction should read/write the lower byte and the second transaction the upper byte.

3. The register to be written to or read from is >16 and ≤ 32 bits

In this case, four AHB transactions are required, which in turn creates four APB transactions, to read or write the register. The first transaction should read/write the lower byte and the second transaction the second byte, and so on.

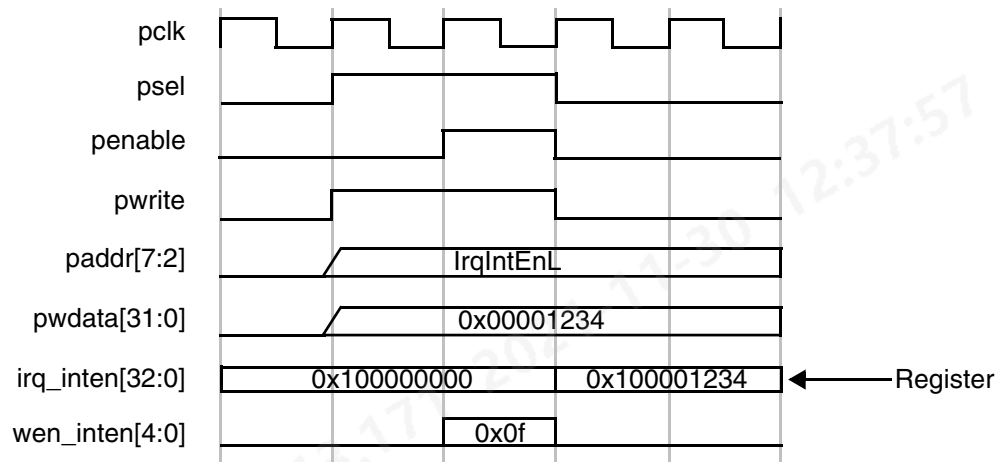
Because the bus is reading a byte at a time, all lower bits of `paddr` are decoded in the 8-bit bus case.

6.4 Write Timing Operation

A timing diagram of an APB write transaction for an APB peripheral register (an earlier version of the DW_apb_ictl) is shown in the following figure. Data, address, and control signals are aligned. The APB

frame lasts for two cycles when `psel` is high, unless an APB3 enabled slave delays the transfer completion by pulling `pready` low.

Figure 6-2 APB Write Transaction



A write can occur after the first phase with `penable` low, or after the second phase when `penable` is high. The second phase is preferred and is used in all APB slave components. The timing diagram is shown with the write occurring after the second phase. Whenever the address on `paddr` matches a corresponding address from the memory map and provided `psel`, `pwrite`, and `penable` are high, then the corresponding register write enable is generated.

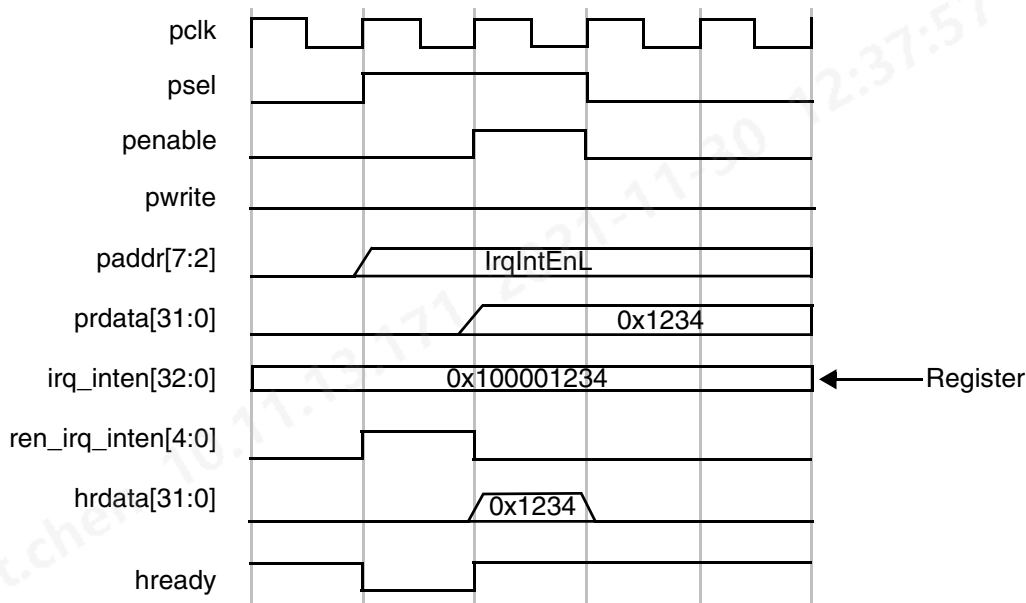
A write from the AHB to the APB does not require the AHB system bus to stall until the transfer on the APB has completed. A write to the APB can be followed by a read transaction from another AHB peripheral (not the DW_apb).

The timing example is a 33-bit register and a 32-bit APB data bus. To write this, 5 byte enables would be generated internally. The example shows writing to the first 32 bits with one write transaction.

6.5 Read Timing Operation

A timing diagram of an APB read transaction for an APB peripheral (an earlier version of the DW_apb_ictl) is shown in the following figure. The APB frame lasts for two cycles, when psel is high, unless an APB3 enabled slave delays the transfer completion by pulling pready low.

Figure 6-3 APB Read Transaction



Whenever the address on paddr matches the corresponding address from the memory map – psel is high, pwrite and penable are low – then the corresponding read enable is generated. The read data is registered within the peripheral before passing back to the master through the DW_apb and DW_ahb.

The qualification of the read-back data with hready from the bridge is shown in the timing diagram, but this does not form part of the APB interface. The read happens in the first APB cycle and is passed straight back to the AHB master in the same cycles as it passes through the bridge. By returning the data immediately to the AHB bus, the bridge can release control of the AHB data bus faster. This is important for systems where the APB clock is slower than the AHB clock.

Once a read transaction has started, it is completed and the AHB bus is held either until the data is returned from the slave, or until it responds with an ERROR message.



Note

If a read enable is not active, then the previously read data is maintained on the read-back data bus.

6.6 Coherency

Coherency is where bits within a register are logically connected. For instance, part of a register is read at time 1 and another part is read at time 2. Being coherent means that the part read at time 2 is at the same

value it was when the register was read at time 1. The unread part is stored into a shadow register and this is read at time 2. When there is no coherency, no shadow registers are involved.

A bus master may need to be able to read the contents of a register, regardless of the data bus width, and be guaranteed of the coherency of the value read. A bus master may need to be able to write a register coherently regardless of the data bus width and use that register only when it has been fully programmed. This may need to be the case regardless of the relationship between the clocks.

Coherency enables a value to be read that is an accurate reflection of the state of the counter, independent of the data bus width, the counter width, and even the relationship between the clocks. Additionally, a value written in one domain is transferred to another domain in a seamless and coherent fashion.

Throughout this appendix the following terms are used:

- **Writing.** A bus master programs a configuration register. An example is programming the load value of a counter into a register.
- **Transferring.** The programmed register is in a different clock domain to where it is used, therefore, it needs to be transferred to the other clock domain.
- **Loading.** Once the programmed register is transferred into the correct clock domain, it needs to be loaded or used to perform its function. For example, once the load value is transferred into the counter domain, it gets loaded into the counter.

6.6.1 Writing Coherently

Writing coherently means that all the bits of a register can be written at the same time. A peripheral may have programmable registers that are wider than the width of the connected APB data bus, which prevents all the bits being programmed at the same time unless additional coherency circuitry is provided.

The programmable register could be the load value for a counter that may exist in a different clock domain. Not only does the value to be programmed need to be coherent, it also needs to be transferred to a different clock domain and then loaded into the counter. Depending on the function of the programmable register, a qualifier may need to be generated with the data so that it knows when the new value is currently transferred and when it should be loaded into the counter.

Depending on the system and on the register being programmed, there may be no need for any special coherency circuitry. One example that requires coherency circuitry is a 32-bit timer within an 8-bit APB system. The value is entirely programmed only after four 8-bit wide write transfers. It is safe to transfer or use the register when the last byte is currently written. An example where no coherency is required is a 16-bit wide timer within a 16-bit APB system. The value is entirely programmed after a single 16-bit wide write transfer.

Coherency circuitry enables the value to be loaded into the counter only when fully programmed and crossed over clock domains if the peripheral clock is not synchronous to the processor clock. While the load register is being programmed, the counter has access to the previous load value in case it needs to reload the counter.

Coherency circuitry is only added in cores where it is needed. The coherency circuitry incorporates an upper byte method that requires users to program the load register in LSB to MSB order when the peripheral width is smaller than the register width. When the upper byte is programmed, the value can be transferred and loaded into the load register. When the lower bytes are being programmed, they need to be stored in shadow registers so that the previous load register is available to the counter if it needs to reload.

When the upper byte is programmed, the contents of the shadow registers and the upper byte are loaded into the load register.

The upper byte is the top byte of a register. A register can be transferred and loaded into the counter only when it has been fully programmed. A new value is available to the counter once this upper byte is written into the register. The following table shows the relationship between the register width and the peripheral bus width for the generation of the correct upper byte. The numbers in the table represent bytes, Byte 0 is the LSB and Byte 3 is the MSB. NCR means that no coherency circuitry is required, as the entire register is written with one access.

Table 6-2 Upper Byte Generation

	Upper Byte Bus Width		
Load Register Width	8	16	32
1 - 8	NCR	NCR	NCR
9 - 16	1	NCR	NCR
17 - 24	2	2	NCR
25 - 32	3	2 (or 3)	NCR

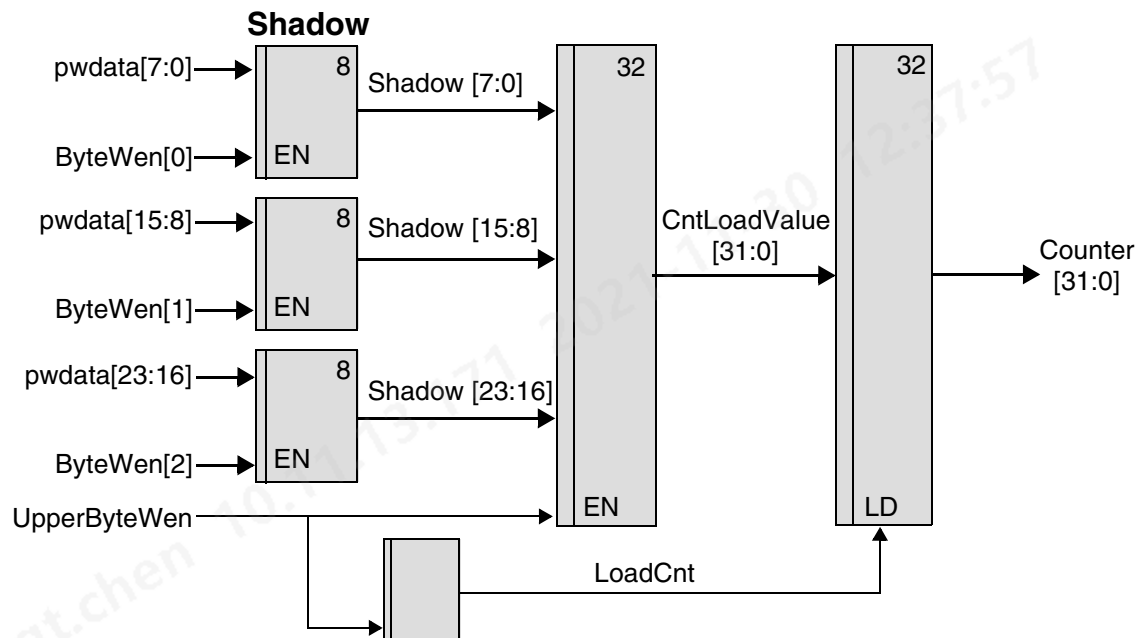
There are three relationship cases to be considered for the processor and peripheral clocks:

- Identical
- Synchronous (phase coherent but of an integer fraction)
- Asynchronous

6.6.1.1 Identical Clocks

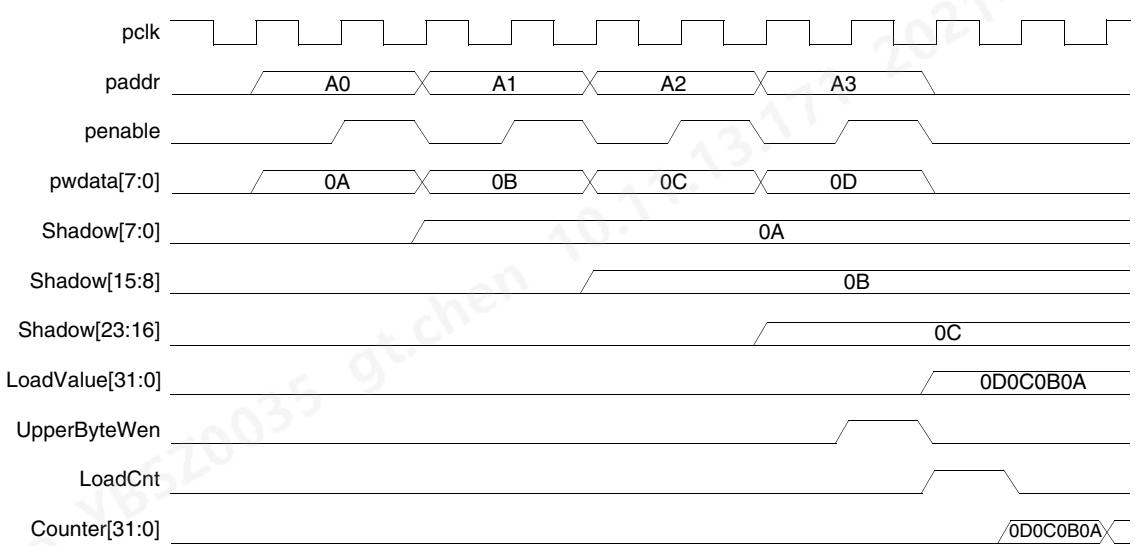
The following figure illustrates an RTL diagram for the circuitry required to implement the coherent write transaction when the APB bus clock and peripheral clocks are identical.

Figure 6-4 Coherent Loading – Identical Synchronous Clocks



The following figure shows a 32-bit register that is written over an 8-bit data bus, as well as the shadow registers being loaded and then loaded into the counter when fully programmed. The LoadCnt signal lasts for one cycle and is used to load the counter with CntLoadValue.

Figure 6-5 Coherent Loading – Identical Synchronous Clocks



Each of the bytes that make up the load register are stored into shadow registers until the final byte is written. The shadow register is up to three bytes wide. The contents of the shadow registers and the final

byte are transferred into the CntLoadValue register when the final byte is written. The counter uses this register to load/initialize itself. If the counter is operating in a periodic mode, it reloads from this register each time the count expires.

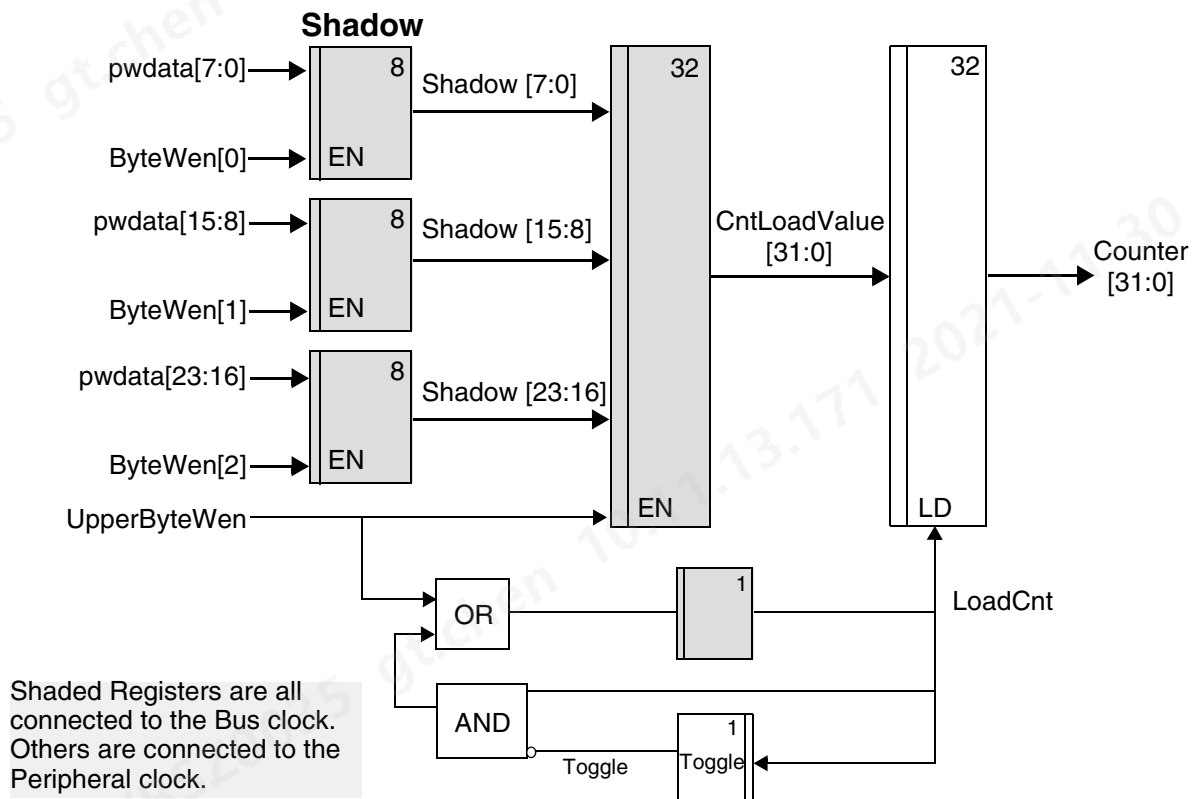
By using the shadow registers, the CntLoadValue is kept stable until it can be changed in one cycle. This allows the counter to be loaded in one access and the state of the counter is not affected by the latency in programming it. When there is a new value to be loaded into the counter initially, this is signaled by LoadCnt = 1. After the upper byte is written, the LoadCnt goes to zero.

6.6.1.2 Synchronous Clocks

When the clocks are synchronous but do not have identical periods, the circuitry needs to be extended so that the LoadCnt signal is kept high until a rising edge of the counter clock occurs. This extension is necessary so that the value can be loaded, using LoadCnt, into the counter on the first counter clock edge. At the rising edge of the counter clock if LoadCnt is high, then a register clocked with the counter clock toggles, otherwise it keeps its current value. A circuit detecting the toggling is used to clear the original LoadCnt by looking for edge changes. The value is loaded into the counter when a toggle has been detected. Once it is loaded, the counter should be free to increment or decrement by normal rules.

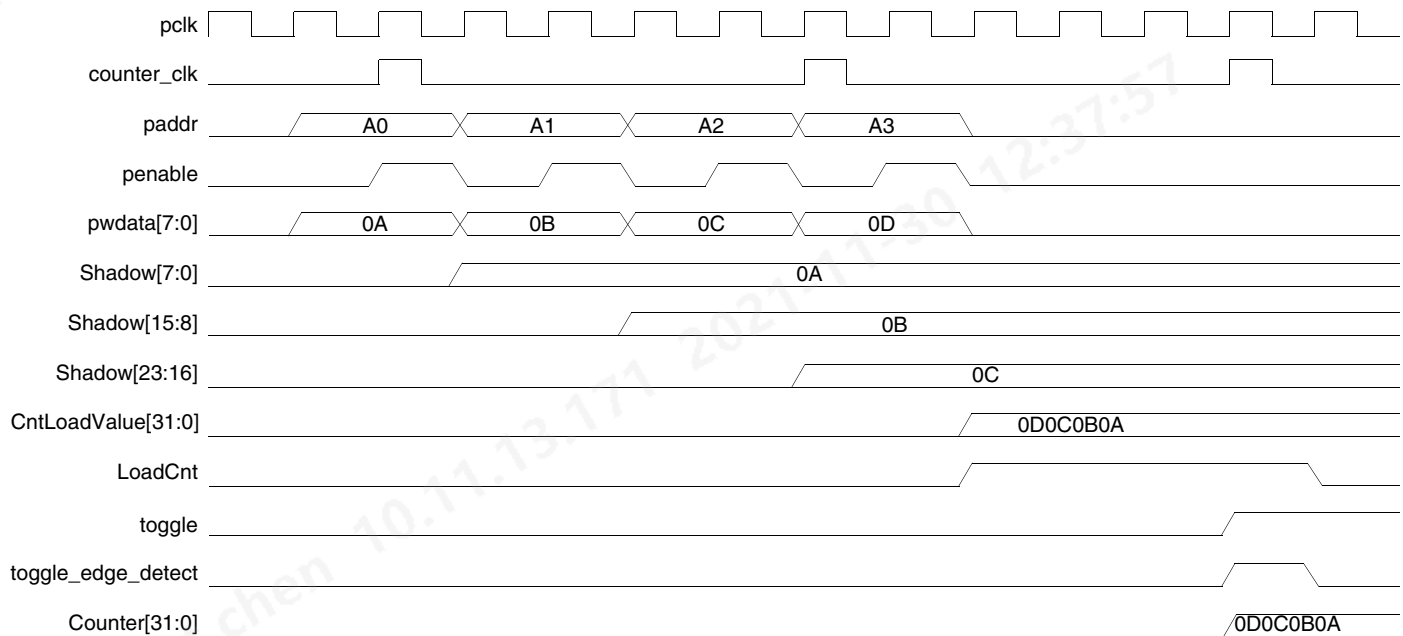
The following figure shows an RTL diagram for the circuitry required to implement the coherent write when the bus and peripheral clocks are synchronous.

Figure 6-6 Coherent Loading – Synchronous Clocks



The following figure shows a 32-bit register being written over an 8-bit data bus, as well as the shadow registers being loaded and then loaded into the counter when fully programmed. The LoadCnt signal is extended until a change in the toggle is detected and is used to load the counter.

Figure 6-7 Coherent Loading – Synchronous Clocks

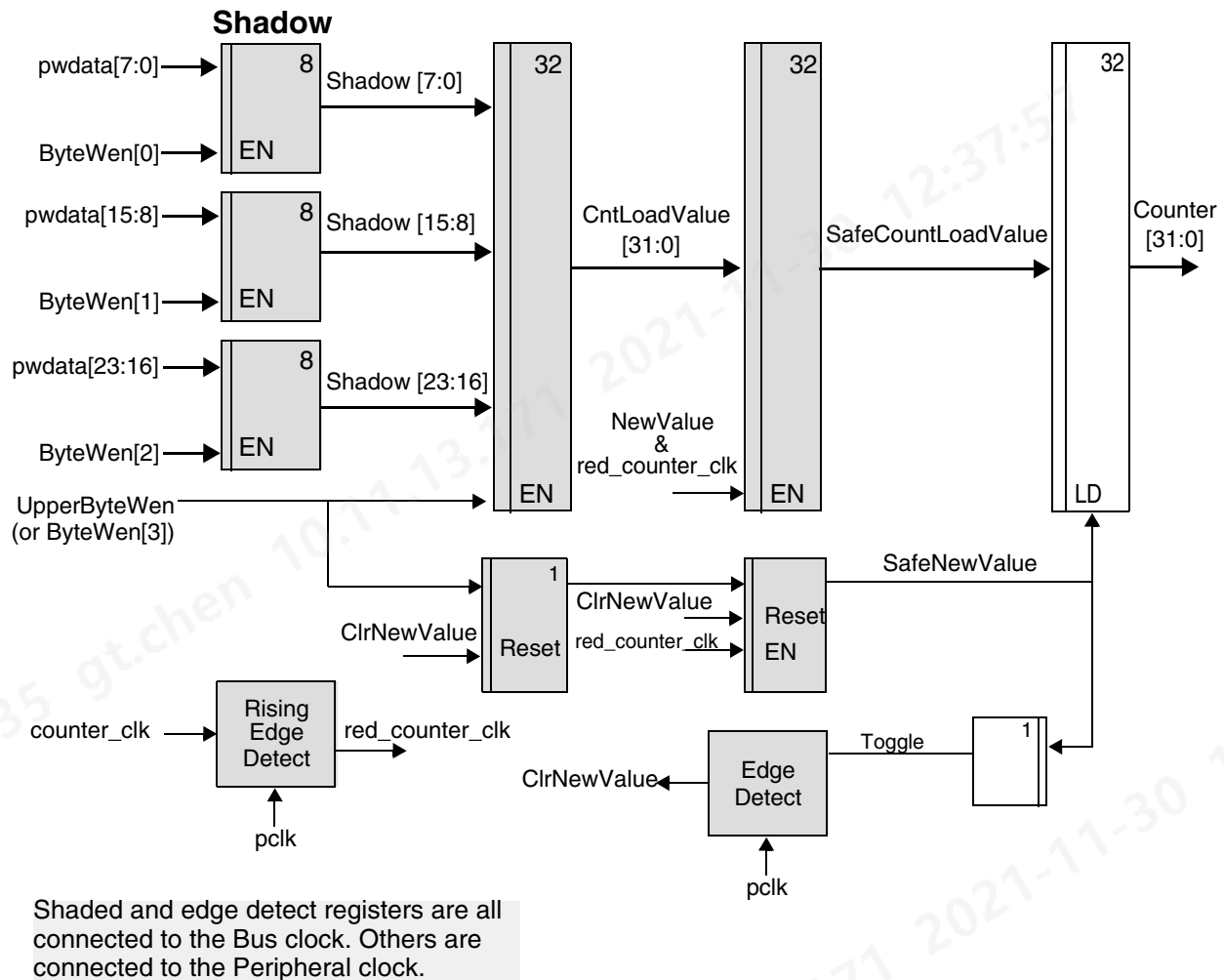


6.6.1.3 Asynchronous Clocks

When the clocks are asynchronous, the processor clock needs to be three-times the speed of the peripheral clock for the re-timing to operate correctly. The high pulse time of the peripheral clock needs to be greater

than the period of the processor clock. The following figure shows an RTL diagram for the circuitry required to implement the coherent write when the bus and peripheral clocks are asynchronous.

Figure 6-8 Coherent Loading – Asynchronous Clocks

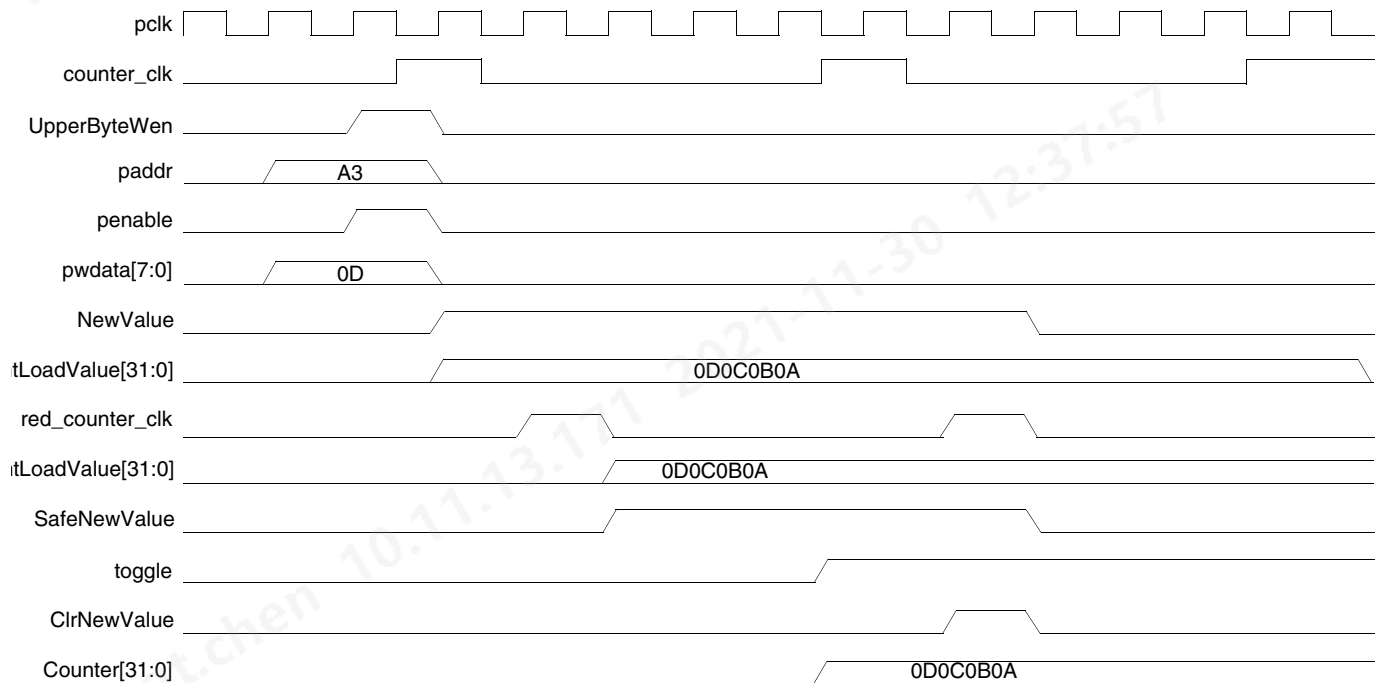


When the clocks are asynchronous, you need to transfer the contents of the register from one clock domain to another. It is not desirable to transfer the entire register through meta-stability registers, as coherency is not guaranteed with this method. The circuitry needed requires the processor clock to be used to re-time the peripheral clock. Upon a rising edge of the re-timed clock, the new value signal, NewValue, is transferred into a safe new value signal, SafeNewValue, which happens after the edge of the peripheral clock has occurred.

Every time there is a rising edge of the peripheral clock detected, the CntLoadValue is transferred into a SafeCntLoadValue. This value is used to transfer the load value across the clock domains. The SafeCntLoadValue only changes a number of bus clock cycles after the peripheral clock edge changes. A counter running on the peripheral clock is able to use this value safely. It could be up to two peripheral clock periods before the value is loaded into the counter. Along with this loaded value, there also is a single bit transferred that is used to qualify the loading of the value into the counter.

The timing diagram depicted in the following figure does not show the shadow registers being loaded. This is identical to the loading for the other clock modes.

Figure 6-9 Coherent Loading – Asynchronous Clocks



The NewValue signal is extended until a change in the toggle is detected and is used to update the safe value. The SafeNewValue is used to load the counter at the rising edge of the peripheral clock. Each time a new value is written the toggle bit is flipped and the edge detection of the toggle is used to remove both the NewValue and the SafeNewValue.

6.6.2 Reading Coherently

For writing to registers, an upper-byte concept is proposed for solving coherency issues. For read transactions, a lower-byte concept is required. The following table provides the relationship between the register width and the bus width for the generation of the correct lower byte.

Table 6-3 Lower Byte Generation

	Lower Byte Bus Width		
Counter Register Width	8	16	32
1 - 8	NCR	NCR	NCR
9 - 16	0	NCR	NCR
17 - 24	0	0	NCR
25 - 32	0	0	NCR

Depending on the bus width and the register width, there may be no need to save the upper bits because the entire register is read in one access, in which case there is no problem with coherency. When the lower byte is read, the remaining upper bytes within the counter register are transferred into a holding register. The holding register is the source for the remaining upper bytes. Users must read LSB to MSB for this solution to operate correctly. NCR means that no coherency circuitry is required, as the entire register is read with one access.

There are two cases regarding the relationship between the processor and peripheral clocks to be considered as follows:

- Identical and/or synchronous
- Asynchronous

6.6.2.1 Synchronous Clocks

When the clocks are identical and/or synchronous, the remaining unread bits (if any) need to be saved into a holding register once a read is started. The first read byte must be the lower byte provided in the previous table, which causes the other bits to be moved into the holding register, *SafeCntVal*, provided that the register cannot be read in one access. The upper bytes of the register are read from the holding register rather than the actual register so that the value read is coherent. This is illustrated in the following figure and in the timing diagram after it.

Figure 6-10 Coherent Registering – Synchronous Clocks

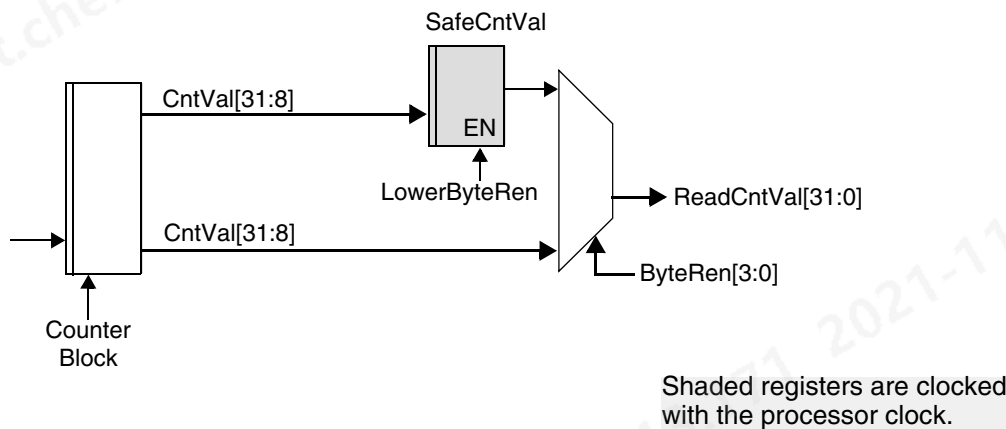
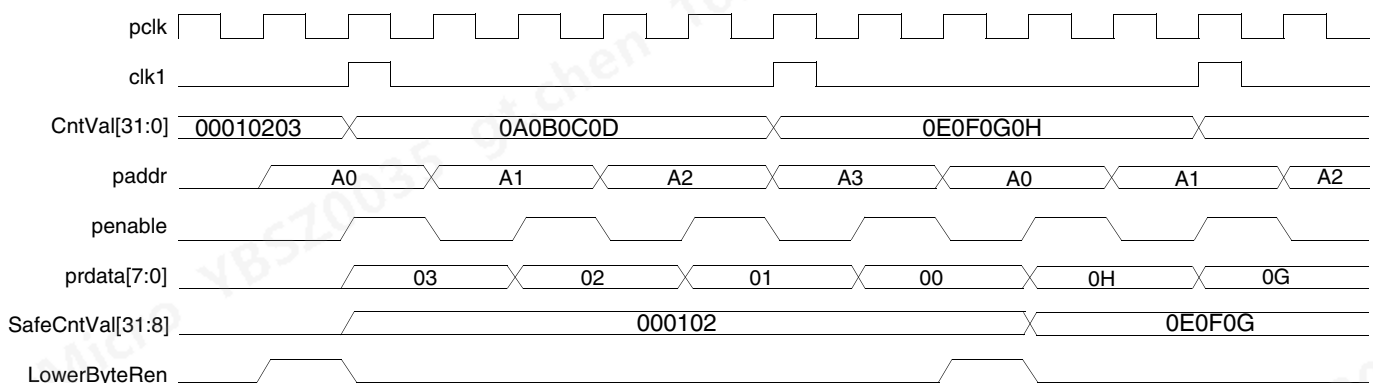


Figure 6-11 Coherent Registering – Synchronous Clocks



6.6.2.2 Asynchronous Clocks

When the clocks are asynchronous, the processor clock needs to be three times the speed of the peripheral clock for the re-timing to operate correctly. The high pulse time of the peripheral clock needs to be greater than the period of the processor clock.

To safely transfer a counter value from the counter clock domain to the bus clock domain, the counter clock signal should be transferred to the bus clock domain. When the rising edge detect of this re-timed counter clock signal is detected, it is safe to use the counter value to update a shadow register that holds the current value of the counter.

While reading the counter contents it may take multiple APB transfers to read the value.



Note

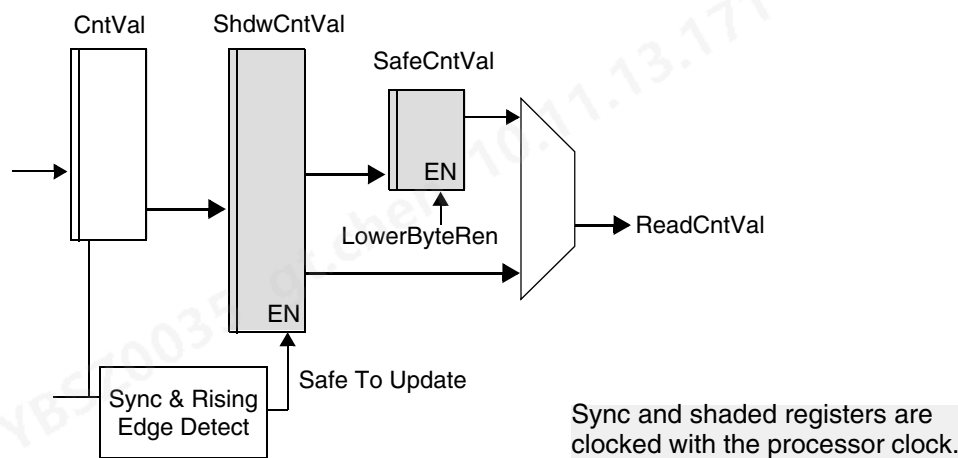
You must read LSB to MSB when the bus width is narrower than the counter width.

Once a read transaction has started, the value of the upper register bits need to be stored into a shadow register so that they can be read with subsequent read accesses. Storing these upper bits preserves the coherency of the value that is being read. When the processor reads the current value it actually reads the contents of the shadow register instead of the actual counter value. The holding register is read when the bus width is narrower than the counter width. When the LSB is read, the value comes from the shadow register; when the remaining bytes are read they come from the holding register. If the data bus width is wide enough to read the counter in one access, then the holding registers do not exist.

The counter clock is registered and successively pipelined to sense a rising edge on the counter clock. Having detected the rising edge, the value from the counter is known to be stable and can be transferred into the shadow register. The coherency of the counter value is maintained before it is transferred, because the value is stable.

The following figure illustrates the synchronization of the counter clock and the update of the shadow register.

Figure 6-12 Coherency and Shadow Registering – Asynchronous Clocks



6.7 Timing Exceptions

For details on multi cycle paths in the design, see the DW_apb.sdc file generated by the Perform ASIC Synthesis activity of the coreConsultant.

A

DesignWare Constants

Table A-1 provides the DesignWare bus constant definitions. These definitions can also be found in DW_amba_constants.v file in the src directory of your DW_apb coreKit.

Table A-1 DesignWare Bus Constant Definitions

DesignWare Constant	Value
HBURST_WIDTH	3
HMASTER_WIDTH	4
HPROT_WIDTH	4
HRESP_WIDTH	2
HSIZE_WIDTH	3
HSPLIT_WIDTH	16
HTRANS_WIDTH	2
HBURST Values	
SINGLE	3'b000
INCR	3'b001
WRAP4	3'b010
INCR4	3'b011
WRAP8	3'b100
INCR8	3'b101
WRAP16	3'b110
INCR16	3'b111
HRESP Values	
OKAY	2'b00

Table A-1 DesignWare Bus Constant Definitions (Continued)

DesignWare Constant	Value
ERROR	2'b01
RETRY	2'b10
SPLIT	2'b11
HSIZE Values	
BYTE	3'b000
WORD	3'b001
WORD	3'b010
LWORD	3'b011
DWORD	3'b100
WORD4	3'b101
WORD8	3'b110
WORD16	3'b111
HTRANS Values	
IDLE	2'b00
BUSY	2'b01
NONSEQ	2'b10
SEQ	2'b11
HWRITE/PWRITE Values	
READ	1'b0
WRITE	1'b1
Generic Definitions	
TRUE	1'b1
FALSE	1'b0
zero8	8'b0
zero16	16'b0
zero32	32'b0
KBYTE	1024

B

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 B-1 Internal Parameters

Parameter Name	Equals To
AHB_HAS_EXTD_MEMTYPE	0
APB_HAS_APB3	=(APB_INTERFACE_TYPE_SLAVE_0 != 0 APB_INTERFACE_TYPE_SLAVE_1 != 0 APB_INTERFACE_TYPE_SLAVE_2 != 0 APB_INTERFACE_TYPE_SLAVE_3 != 0 APB_INTERFACE_TYPE_SLAVE_4 != 0 APB_INTERFACE_TYPE_SLAVE_5 != 0 APB_INTERFACE_TYPE_SLAVE_6 != 0 APB_INTERFACE_TYPE_SLAVE_7 != 0 APB_INTERFACE_TYPE_SLAVE_8 != 0 APB_INTERFACE_TYPE_SLAVE_9 != 0 APB_INTERFACE_TYPE_SLAVE_10 != 0 APB_INTERFACE_TYPE_SLAVE_11 != 0 APB_INTERFACE_TYPE_SLAVE_12 != 0 APB_INTERFACE_TYPE_SLAVE_13 != 0 APB_INTERFACE_TYPE_SLAVE_14 != 0 APB_INTERFACE_TYPE_SLAVE_15 != 0)

Table B-1 Internal Parameters (Continued)

Parameter Name	Equals To
APB_HAS_APB4	=(APB_INTERFACE_TYPE_SLAVE_0 ==2 APB_INTERFACE_TYPE_SLAVE_1 ==2 APB_INTERFACE_TYPE_SLAVE_2 ==2 APB_INTERFACE_TYPE_SLAVE_3 ==2 APB_INTERFACE_TYPE_SLAVE_4 ==2 APB_INTERFACE_TYPE_SLAVE_5 ==2 APB_INTERFACE_TYPE_SLAVE_6 ==2 APB_INTERFACE_TYPE_SLAVE_7 ==2 APB_INTERFACE_TYPE_SLAVE_8 ==2 APB_INTERFACE_TYPE_SLAVE_9 ==2 APB_INTERFACE_TYPE_SLAVE_10 ==2 APB_INTERFACE_TYPE_SLAVE_11 ==2 APB_INTERFACE_TYPE_SLAVE_12 ==2 APB_INTERFACE_TYPE_SLAVE_13 ==2 APB_INTERFACE_TYPE_SLAVE_14 ==2 APB_INTERFACE_TYPE_SLAVE_15 ==2)
ERROR	2'b01
HBURST_WIDTH	3
HPROT_WIDTH	{AHB_HAS_EXTD_MEMTYPE ==0 ? 4 : 7}
HRESP_WIDTH	{AHB_SCALAR_HRESP ==1 ? 1 : 2}
IDLE	2'b00
OKAY	2'b00
RETRY	2'b10
SPLIT	2'b11

C

Glossary

active command queue	Command queue from which a model is currently taking commands; see also command queue.
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 DesignWare Synthesizable Components.
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.
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.

