

Cortex[™]-M System Design Kit

Revision: r1p0

Technical Reference Manual



Cortex-M System Design Kit

Technical Reference Manual

Copyright © 2011, 2013 ARM. All rights reserved.

Release Information

The following changes have been made to this document:

| Change history | | | |
|----------------|-------|------------------|-------------------------|
| Date | Issue | Confidentiality | Change |
| 14 March 2011 | A | Non-Confidential | First release for r0p0 |
| 16 June 2011 | B | Non-Confidential | Second release for r0p0 |
| 19 April 2013 | C | Non-Confidential | First release for r1p0 |

Proprietary Notice

Words and logos marked with ® or ™ are registered trademarks or trademarks of ARM® in the EU and other countries, except as otherwise stated below in this proprietary notice. Other brands and names mentioned herein may be the trademarks of their respective owners.

Neither the whole nor any part of the information contained in, or the product described in, this document may be adapted or reproduced in any material form except with the prior written permission of the copyright holder.

The product described in this document is subject to continuous developments and improvements. All particulars of the product and its use contained in this document are given by ARM in good faith. However, all warranties implied or expressed, including but not limited to implied warranties of merchantability, or fitness for purpose, are excluded.

This document is intended only to assist the reader in the use of the product. ARM shall not be liable for any loss or damage arising from the use of any information in this document, or any error or omission in such information, or any incorrect use of the product.

Where the term ARM is used it means “ARM or any of its subsidiaries as appropriate”.

Confidentiality Status

This document is Non-Confidential. The right to use, copy and disclose this document may be subject to license restrictions in accordance with the terms of the agreement entered into by ARM and the party that ARM delivered this document to.

Product Status

The information in this document is final, that is for a developed product.

Web Address

<http://www.arm.com>

Contents

Cortex-M System Design Kit Technical Reference Manual

| | | |
|------------------|---|------|
| | Preface | |
| | About this book | vi |
| | Feedback | ix |
| Chapter 1 | Introduction | |
| | 1.1 About the Cortex-M System Design Kit | 1-2 |
| | 1.2 Product revisions | 1-4 |
| Chapter 2 | Functional Description | |
| | 2.1 About the Cortex-M System Design Kit components | 2-2 |
| | 2.2 Design components | 2-3 |
| | 2.3 ID registers in programmable components | 2-5 |
| | 2.4 Use of OVL | 2-6 |
| Chapter 3 | Basic AHB-Lite Components | |
| | 3.1 AHB default slave | 3-2 |
| | 3.2 AHB example slave | 3-3 |
| | 3.3 AHB slave multiplexer | 3-6 |
| | 3.4 AHB master multiplexer | 3-9 |
| | 3.5 AHB GPIO | 3-11 |
| | 3.6 AHB to APB sync-down bridge | 3-18 |
| | 3.7 AHB to SRAM interface module | 3-20 |
| | 3.8 AHB to flash interface modules | 3-22 |
| | 3.9 AHB timeout monitor | 3-25 |
| | 3.10 AHB to external SRAM interface | 3-27 |
| | 3.11 AHB bit-band wrapper | 3-31 |

| | | |
|-------------------|--|------|
| Chapter 4 | APB Components | |
| 4.1 | APB example slaves | 4-2 |
| 4.2 | APB timer | 4-5 |
| 4.3 | APB UART | 4-8 |
| 4.4 | APB dual-input timers | 4-11 |
| 4.5 | APB watchdog | 4-20 |
| 4.6 | APB slave multiplexer | 4-26 |
| 4.7 | APB subsystem | 4-27 |
| 4.8 | APB timeout monitor | 4-33 |
| Chapter 5 | Advanced AHB-Lite Components | |
| 5.1 | AHB bus matrix | 5-2 |
| 5.2 | AHB upsizer | 5-14 |
| 5.3 | AHB downsizer | 5-17 |
| 5.4 | AHB to APB asynchronous bridge | 5-25 |
| 5.5 | AHB to AHB and APB asynchronous bridge | 5-27 |
| 5.6 | AHB to AHB synchronous bridge | 5-30 |
| 5.7 | AHB to AHB sync-down bridge | 5-32 |
| 5.8 | AHB to AHB sync-up bridge | 5-37 |
| Chapter 6 | Behavioral Memory Models | |
| 6.1 | ROM model wrapper | 6-2 |
| 6.2 | RAM model wrapper | 6-6 |
| 6.3 | Behavioral SRAM model with AHB interface | 6-10 |
| 6.4 | 32-bit flash ROM behavioral model | 6-11 |
| 6.5 | 16-bit flash ROM behavioral model | 6-12 |
| 6.6 | FPGA SRAM synthesizable model | 6-13 |
| 6.7 | FPGA ROM | 6-14 |
| 6.8 | External asynchronous 8-bit SRAM | 6-15 |
| 6.9 | External asynchronous 16-bit SRAM | 6-16 |
| Chapter 7 | Verification Components | |
| 7.1 | AHB-Lite protocol checker | 7-2 |
| 7.2 | APB protocol checker | 7-5 |
| 7.3 | AHB FRBM | 7-7 |
| Appendix A | Revisions | |

Preface

This preface introduces the *Cortex-M System Design Kit Technical Reference Manual*. It contains the following sections:

- [About this book on page vi.](#)
- [Feedback on page ix.](#)

About this book

This is the *Technical Reference Manual* (TRM) for the Cortex-M System Design Kit.

Product revision status

The *rn**pn* identifier indicates the revision status of the product described in this book, where:

rn Identifies the major revision of the product.

pn Identifies the minor revision or modification status of the product.

Intended audience

This book is written for system designers to design products with the ARM Cortex-M processors.

Using this book

This book is organized into the following chapters:

Chapter 1 *Introduction*

Read this for an introduction to the Cortex-M System Design Kit.

Chapter 2 *Functional Description*

Read this for an overview of the major functional blocks and the operation of the Cortex-M System Design Kit.

Chapter 3 *Basic AHB-Lite Components*

Read this for a description of the AHB-Lite components that the Cortex-M System Design Kit uses.

Chapter 4 *APB Components*

Read this for a description of the APB components that the Cortex-M System Design Kit uses.

Chapter 5 *Advanced AHB-Lite Components*

Read this for a description of the advanced AHB-Lite components that the Cortex-M System Design Kit uses.

Chapter 6 *Behavioral Memory Models*

Read this for a description of the behavioral memory models that the Cortex-M System Design Kit uses.

Chapter 7 *Verification Components*

Read this for a description of the verification components in the Cortex-M System Design Kit.

Appendix A *Revisions*

Read this for a description of the technical changes between released issues of this book.

Glossary

The *ARM Glossary* is a list of terms used in ARM documentation, together with definitions for those terms. The *ARM Glossary* does not contain terms that are industry standard unless the ARM meaning differs from the generally accepted meaning.

See *ARM Glossary*, <http://infocenter.arm.com/help/topic/com.arm.doc.aeg0014-/index.html>.

Typographical Conventions

This book uses the conventions that are described in:

- [Typographical conventions](#).
- [Timing diagrams](#).
- [Signals on page viii](#).

Typographical conventions

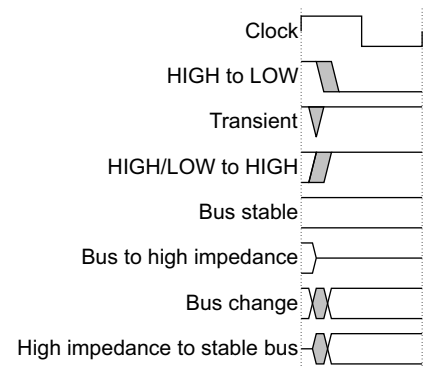
The following table describes the typographical conventions:

| Style | Purpose |
|-------------------------|--|
| <i>italic</i> | Introduces special terminology, denotes cross-references, and citations. |
| bold | Highlights interface elements, such as menu names. Denotes signal names. Also used for terms in descriptive lists, where appropriate. |
| monospace | Denotes text that you can enter at the keyboard, such as commands, file and program names, and source code. |
| <u>monospace</u> | Denotes a permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name. |
| monospace <i>italic</i> | Denotes arguments to monospace text where the argument is to be replaced by a specific value. |
| monospace bold | Denotes language keywords when used outside example code. |
| <and> | Encloses replaceable terms for assembler syntax where they appear in code or code fragments. For example: MRC p15, 0 <Rd>, <CRn>, <CRm>, <Opcode_2> |
| SMALL CAPITALS | Used in body text for a few terms that have specific technical meanings, that are defined in the <i>ARM glossary</i> . For example, IMPLEMENTATION DEFINED, IMPLEMENTATION SPECIFIC, UNKNOWN, and UNPREDICTABLE. |

Timing diagrams

The figure named [Key to timing diagram conventions on page viii](#) explains the components used in timing diagrams. Variations, when they occur, have clear labels. You must not assume any timing information that is not explicit in the diagrams.

Shaded bus and signal areas are undefined, so the bus or signal can assume any value within the shaded area at that time. The actual level is unimportant and does not affect normal operation.



Key to timing diagram conventions

Signals

The signal conventions are:

Signal-level

The level of an asserted signal depends on whether the signal is active-HIGH or active-LOW. Asserted means:

- HIGH for active-HIGH signals.
- LOW for active-LOW signals.

Lower-case n

At the start or end of a signal name denotes an active-LOW signal.

Additional reading

This section lists publications by ARM and by third parties.

See Infocenter, <http://infocenter.arm.com>, for access to ARM documentation.

ARM publications

This book contains information that is specific to this product. See the following documents for other relevant information:

- *Cortex-M0 Technical Reference Manual* (ARM DDI 0432).
- *Cortex-M0+ Technical Reference Manual* (ARM DDI 0484).
- *Cortex-M3 Technical Reference Manual* (ARM DDI 0337).
- *Cortex-M4 Technical Reference Manual* (ARM DDI 0439).
- *CoreSight™ Architecture Specification* (ARM IHI 0029).

The following confidential books are only available to licensees:

- *Cortex-M0 and Cortex-M0+ System Design Kit Example System Guide* (ARM DUI 0559).
- *Cortex-M System Design Kit Example System Guide* (ARM DUI 0594).

Other publications

This section lists relevant documents published by third parties:

- JEDEC website, www.jedec.org.
- Accellera website, www.accellera.org.

Feedback

ARM welcomes feedback on this product and its documentation.

Feedback on this product

If you have any comments or suggestions about this product, contact your supplier and give:

- The product name.
- The product revision or version.
- An explanation with as much information as you can provide. Include symptoms and diagnostic procedures if appropriate.

Feedback on content

If you have comments on content then send an e-mail to errata@arm.com. Give:

- The title.
- The number, ARM DDI 0479C.
- The page numbers to which your comments apply.
- A concise explanation of your comments.

ARM also welcomes general suggestions for additions and improvements.

———— **Note** —————

ARM tests the PDF only in Adobe Acrobat and Acrobat Reader, and cannot guarantee the quality of the represented document when used with any other PDF reader.

Chapter 1

Introduction

This chapter describes the Cortex-M System Design Kit. It contains the following sections:

- [*About the Cortex-M System Design Kit*](#) on page 1-2
- [*Product revisions*](#) on page 1-4.

1.1 About the Cortex-M System Design Kit

The Cortex-M System Design Kit helps you design products using ARM Cortex-M processors.

The design kit contains the following:

- A selection of AHB-Lite and APB components, including several peripherals such as GPIO, timers, watchdog, and UART.
- An example system for supported processor products.
- Example synthesis scripts for the example system.
- Example compilation and simulation scripts for the Verilog environment that supports ModelSim, VCS, and NC Verilog.
- Example code for software drivers.
- Example test code to demonstrate various operations of the systems.
- Example compilation scripts and example software project files that support:
 - ARM *Development Studio 5* (DS-5).
 - ARM RealView Development Suite.
 - Keil *Microcontroller Development Kit* (MDK).
 - GNU Tools for ARM Embedded Processors (ARM GCC).
- Documentation including:
 - *Cortex-M System Design Kit Technical Reference Manual*.
 - *Cortex-M0 and Cortex-M0+ System Design Kit Example System Guide*.
 - *Cortex-M System Design Kit Example System Guide*.

Figure 1-1 shows the use of the design kit in various stages of a design process.

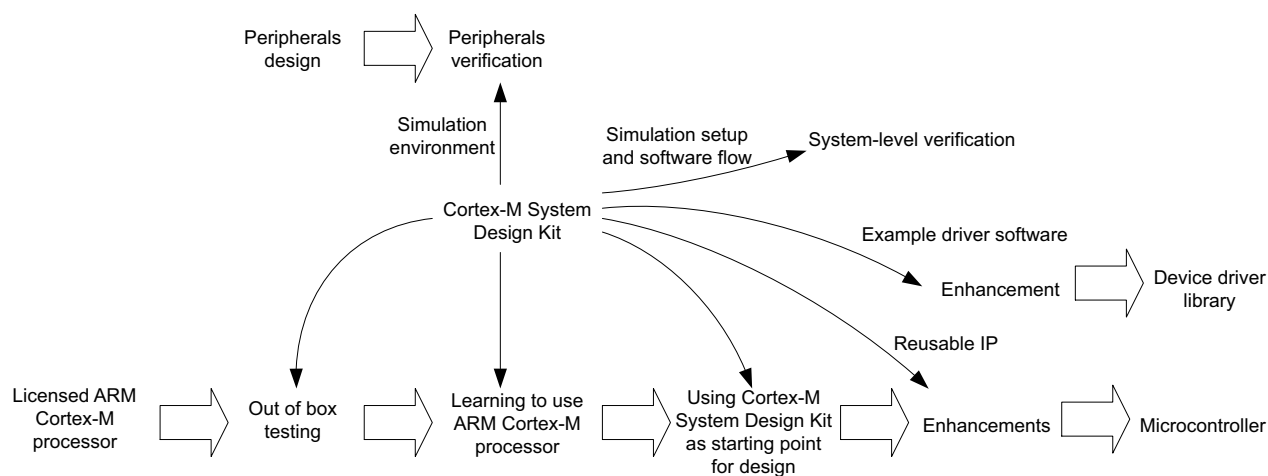


Figure 1-1 Cortex-M System Design Kit usage in various stages of a design process

Table 1-1 shows the Cortex-M System Design Kit usage in various stages of a design process.

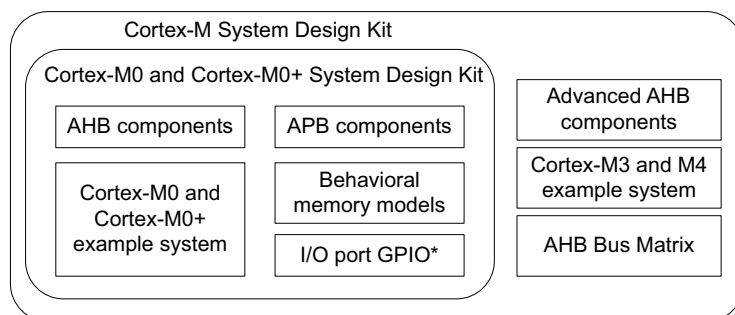
Table 1-1 Cortex-M System Design Kit usage in various stages of a design process

| Area | Description |
|-----------------------------------|---|
| <i>Out of Box</i> (OoB) testing | When you license the Cortex-M System Design Kit and a Cortex-M processor, you can use it for OoB testing and benchmarking |
| Learning | Using the example systems, you can learn how to integrate the Cortex-M processor, and carry out various operations |
| Starting point of design | You can use the Cortex-M System Design Kit as a starting point to design your microcontroller or <i>System-on-Chip</i> (SoC) products |
| Verification | You can use the example system in the Cortex-M System Design Kit as a verification environment to carry out system-level verification |
| Starting point of software driver | You can use the example software code in the Cortex-M System Design Kit as a starting point for software driver development |
| Reusable IP | You can reuse the various components of the Cortex-M System Design Kit in microcontroller or SoC design projects |

The Cortex-M System Design Kit is available as:

- Cortex-M0 and Cortex-M0+ System Design Kit. This supports Cortex-M0, Cortex-M0 DesignStart, and Cortex-M0+.
- Cortex-M System Design Kit, full version. This supports Cortex-M0, Cortex-M0 DesignStart, Cortex-M0+, Cortex-M3, and Cortex-M4.

The other differences between the Cortex-M0 and Cortex-M0+ version, and the Cortex-M version of the design kit are the example systems, and the components provided. See Figure 1-2.



* For use with the Cortex-M0+ directly, or as a subcomponent within AHB GPIO module.

Figure 1-2 Difference between the two versions of the design kit

The design supports the following bus protocols:

- AHB-Lite or AMBA 3 AHB-Lite Protocol v1.0. In this document, AHB signifies AHB-Lite.
- APB2 or AMBA 2 APB Protocol.
- APB3 or AMBA 3 APB Protocol v1.0.
- APB4 or AMBA APB Protocol v2.0.

1.2 Product revisions

This section describes the differences in functionality between product revisions of the Cortex-M System Design Kit:

r0p0 First release.

r0p0-r1p0 Functional changes are:

- Support for Cortex-M0+ processor.
- Added `cmsdk_` prefix to module names.
- CMSIS updated to version 3.2.
- Changed **HRESP** width in some components.
- AHB slave multiplexer changed from eight ports to ten ports. See [AHB slave multiplexer on page 3-6](#).
- Addition of I/O port GPIO for Cortex-M0+. See [AHB GPIO on page 3-11](#).
- Additional parameter in AHB to APB synchronous bridge. See [AHB to APB sync-down bridge on page 3-18](#).
- Addition of AHB to AHB and APB asynchronous bridge. See [AHB to AHB and APB asynchronous bridge on page 5-27](#).
- Addition of 16-bit flash ROM behavioral model. See [16-bit flash ROM behavioral model on page 6-12](#).

Chapter 2

Functional Description

This chapter describes the major functional blocks of the Cortex-M System Design Kit. It contains the following sections:

- *About the Cortex-M System Design Kit components on page 2-2.*
- *Design components on page 2-3.*
- *Verification components on page 2-4.*
- *ID registers in programmable components on page 2-5.*
- *Use of OVL on page 2-6.*

2.1 About the Cortex-M System Design Kit components

The Cortex-M System Design Kit provides example systems with AHB and APB components designed for low-power and low-latency designs.

The preconfigured and validated examples enable you to develop devices in very short design cycles. In addition, you can reuse the components in future designs.

2.2 Design components

The example systems consist of the following components and models:

- [Basic AHB-Lite components](#).
- [APB components](#).
- [Advanced AHB-Lite components](#).
- [Behavioral memory models on page 2-4](#).
- [Verification components on page 2-4](#).

2.2.1 Basic AHB-Lite components

The basic AHB-Lite components are:

- AHB default slave.
- AHB example slave.
- AHB slave multiplexer.
- AHB master multiplexer.
- AHB *General Purpose Input/Output* (GPIO), including I/O port GPIO.
- AHB to APB sync-down bridge.
- AHB to SRAM interface module.
- AHB to flash interface modules.
- AHB timeout monitor.
- AHB to external SRAM interface.
- AHB bit-band wrapper for Cortex-M0 and Cortex-M0+.

See [Chapter 3 Basic AHB-Lite Components](#) for more information.

2.2.2 APB components

The APB components are:

- APB example slave.
- APB timer.
- APB UART.
- APB dual timer.
- APB watchdog .
- APB slave multiplexer.
- APB subsystem.
- APB timeout monitor.

See [Chapter 4 APB Components](#) for more information.

2.2.3 Advanced AHB-Lite components

The advanced AHB-Lite components are:

- AHB bus matrix.
- AHB upsizer.
- AHB downsizer.
- AHB to APB asynchronous bridge.
- AHB to AHB and APB asynchronous bridge.
- AHB to AHB synchronous bridge.
- AHB to AHB sync-down bridge.
- AHB to AHB sync-up bridge.

Note

The advanced AHB-Lite components are available only with the full version of the Cortex-M System Design Kit. They are not included in the Cortex-M0 and Cortex-M0+ System Design Kit.

See [Chapter 5 *Advanced AHB-Lite Components*](#) for more information.

2.2.4 Behavioral memory models

The memory models are:

- ROM model wrapper.
- RAM model wrapper.
- Behavioral SRAM model with AHB interface.
- 32-bit flash ROM behavioral model.
- 16-bit flash ROM behavioral model.
- SRAM synthesizable (for FPGA) model.
- FPGA ROM.
- External asynchronous 8-bit SRAM.
- External asynchronous 16-bit SRAM.

See [Chapter 6 *Behavioral Memory Models*](#) for more information.

2.2.5 Verification components

The verification components are:

- AHB-Lite protocol checker.
- APB protocol checker.
- AHB *File Reader Bus Master* (FRBM).

See [Chapter 7 *Verification Components*](#) for more information.

2.3 ID registers in programmable components

In the Cortex-M System Design Kit, some of the peripherals contain a number of read-only *Identification* (ID) registers. These ID registers enable software to extract the component type and revision information. In some cases, these registers are required to enable device driver software to work with different versions of the same peripherals.

One of the ID registers, PID3, contains an *Engineering Change Order* (ECO) bit field generated from the **ECOREVNUM[3:0]** input signal. The ECO operation enables you to carry out minor design changes in the late stage of a chip design process, for example, at silicon mask level. Connect **ECOREVNUM[3:0]** to tie-off cells to support ECO revision maintenance.

The ID registers are not strictly required for peripheral operation. In ultra low-power designs, you can remove these ID registers to reduce gate count and power consumption.

When you modify a peripheral from the Cortex-M System Design Kit, modify the JEDEC ID value and the part number in the ID registers to indicate that the peripheral is no longer identical to the original version from ARM. Alternatively, you can remove these ID registers.

The JEDEC standard describes the JEDEC ID value allocation.

2.3.1 Modification of components

In some applications, it is necessary to modify the design of some components. If this is required, ARM recommends that you do the following:

- Change the component name and filename to avoid confusion, especially if you are running multiple projects using Cortex-M System Design Kit components.
- Update the ID register values. See [ID registers in programmable components](#).
- Perform your own verification and testing.

2.4 Use of OVL

The components in the Cortex-M System Design Kit contain instantiations of *Open Verification Library* (OVL) assertion components. The OVL assertions enable errors to be detected during Verilog simulation.

The instantiation of OVL assertions is conditional:

AHB components This is controlled by the ARM_AHB_ASSERT_ON macro.

APB components This is controlled by the ARM_APB_ASSERT_ON macro.

You can download the OVL source code from Accellera, www.accellera.org, if you use the OVL assertion feature.

Chapter 3

Basic AHB-Lite Components

This chapter describes the basic AHB-Lite components provided in the Cortex-M System Design Kit. It contains the following sections:

- [AHB default slave on page 3-2.](#)
- [AHB example slave on page 3-3.](#)
- [AHB slave multiplexer on page 3-6.](#)
- [AHB master multiplexer on page 3-9.](#)
- [AHB GPIO on page 3-11.](#)
- [AHB to APB sync-down bridge on page 3-18.](#)
- [AHB to SRAM interface module on page 3-20.](#)
- [AHB to flash interface modules on page 3-22.](#)
- [AHB timeout monitor on page 3-25.](#)
- [AHB to external SRAM interface on page 3-27.](#)
- [AHB bit-band wrapper on page 3-31.](#)

3.1 AHB default slave

The AHB default slave, `cmsdk_ahb_default_slave.v`, responds to transfers when the bus master accesses an undefined address. A zero wait state OKAY response is generated for IDLE or BUSY transfers, and an ERROR response is generated for NONSEQUENTIAL or SEQUENTIAL transfers. [Figure 3-1](#) shows the AHB default slave module.

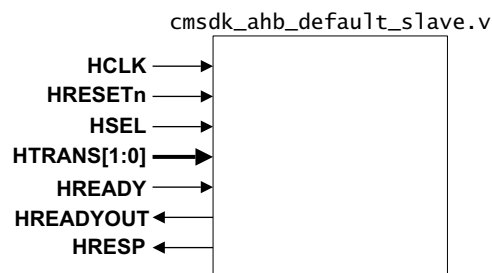


Figure 3-1 AHB default slave component

[Table 3-1](#) shows the characteristics of the AHB default slave module.

Table 3-1 AHB default slave characteristics

| Element name | Description |
|--------------|--|
| Filename | <code>cmsdk_ahb_default_slave.v</code> |
| Parameters | None |
| Clock domain | HCLK |

3.2 AHB example slave

The AHB example slave, `cmsdk_ahb_eg_slave.v`, demonstrates the implementation of a simple AHB slave, and consists of `cmsdk_ahb_eg_slave_interface.v` and `cmsdk_ahb_eg_slave_reg.v`. Figure 3-2 shows the AHB example slave module.

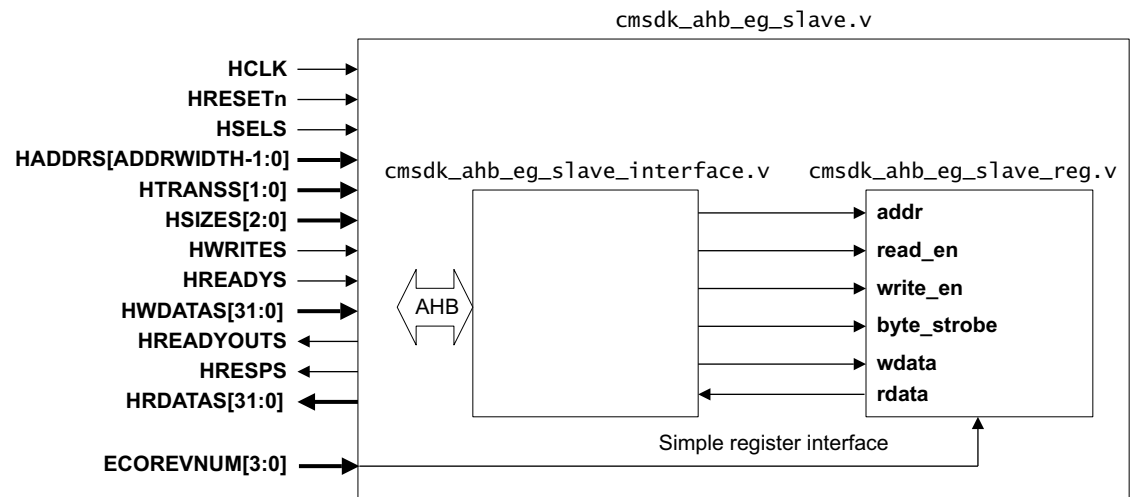


Figure 3-2 AHB example slave

The AHB example slave has the following features:

- 16 bytes of hardware RW registers organized as 4 words.
- Register accesses in byte, halfword, and word transfers
- Optional read-only Component ID and Peripheral ID registers. You must modify the following in these registers:
 - Part number, 12 bits.
 - JEDEC ID value, 7 bits.
- The **ECOREVNUM** input signal is connected to the ECO revision number in Peripheral ID Register 3.
- The interface block converts the AHB protocol to a simple non-pipelined bus protocol. You can reuse it for porting simple peripherals from 8-bit or 16-bit products to an ARM-based system.

You can use the AHB example slave as a starting point for creating your own AHB peripherals, as follows:

1. Copy the AHB example slave to a new directory, and rename the files to names of your choice.
2. Remove the register block inside the AHB example slave, and replace with your own peripheral register set.
3. Add the additional peripheral functionality and I/O pins to the design.
4. Instantiate the peripheral design in the system, and develop verification tests.

Table 3-2 shows the characteristics of the AHB example slave module.

Table 3-2 AHB example slave characteristics

| Element name | Description | |
|--------------|----------------------|--|
| Filename | cmsdk_ahb_eg_slave.v | |
| Parameters | ADDRWIDTH | Width of the AHB address bus. The default is 12. |
| Clock domain | HCLK | |

3.2.1 Programmers model

Table 3-3 shows the AHB example slave memory map.

Table 3-3 AHB example slave memory map

| Name | Base offset | Type | Width | Reset value | Description |
|-------------------|-------------|------|-------|-------------|--|
| DATA0 | 0x0000 | RW | 32 | 0x00000000 | Simple Data Register. |
| DATA1 | 0x0004 | RW | 32 | 0x00000000 | Simple Data Register. |
| DATA2 | 0x0008 | RW | 32 | 0x00000000 | Simple Data Register. |
| DATA3 | 0x000C | RW | 32 | 0x00000000 | Simple Data Register. |
| PID4 | 0xFD0 | RO | 8 | 0x04 | Peripheral ID Register 4: [7:4] 4KB block count. [3:0] jep106_c_code. |
| PID5 ^a | 0xFD4 | RO | 8 | 0x00 | Peripheral ID Register 5. |
| PID6 ^a | 0xFD8 | RO | 8 | 0x00 | Peripheral ID Register 6. |
| PID7 ^a | 0xFDC | RO | 8 | 0x00 | Peripheral ID Register 7. |
| PID0 | 0xFE0 | RO | 8 | 0x17 | Peripheral ID Register 0: [7:0] Part number[7:0]. |
| PID1 | 0xFE4 | RO | 8 | 0xB8 | Peripheral ID Register 1: [7:4] jep106_id_3_0. [3:0] Part number[11:8]. |
| PID2 | 0xFE8 | RO | 8 | 0x1B | Peripheral ID Register 2: [7:4] Revision. [3] jedec_used. [2:0] jep106_id_6_4. |
| PID3 | 0xFEC | RO | 8 | 0x00 | Peripheral ID Register 3: [7:4] ECO revision number. [3:0] Customer modification number. |
| CID0 | 0xFF0 | RO | 8 | 0x0D | Component ID Register 0. |
| CID1 | 0xFF4 | RO | 8 | 0xF0 | Component ID Register 1. |
| CID2 | 0xFF8 | RO | 8 | 0x05 | Component ID Register 2. |
| CID3 | 0xFFC | RO | 8 | 0xB1 | Component ID Register 3. |

- a. The PID5, PID6, and PID7 registers are not used.

Note

Signals such as **HPROT[3:0]**, **HMASTLOCK**, and **HBURST[2:0]** are not used in the design, so they do not appear in the AHB interface component.

3.3 AHB slave multiplexer

The AHB slave multiplexer, `cmsdk_ahb_slave_mux.v`, supports up to ten AHB slaves. It uses parameters to define the slave port usage so that the synthesis process does not generate unnecessary additional logic. Figure 3-3 shows the AHB slave multiplexer.

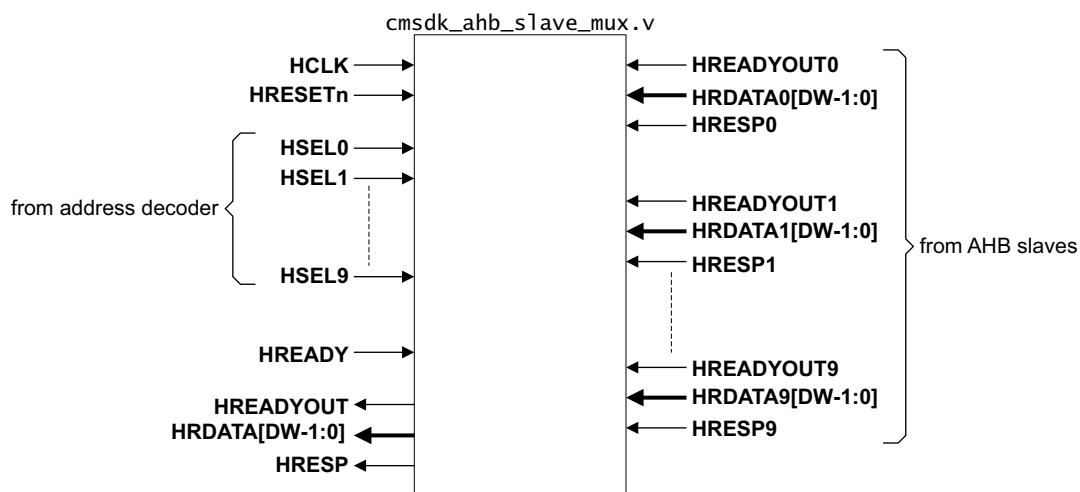


Figure 3-3 AHB slave multiplexer

The slave to master multiplexer controls the routing of read data and response signals from the system bus slaves to the bus masters. An address decoder determines the slave that is currently selected, and generates the `HSEL` signals to the AHB slave multiplexer and the AHB slaves. The multiplexer uses a registered version of the slave select signals, because the read data and response signals are valid during the data phase of a transfer, to connect the outputs of the selected slave to the inputs of the bus masters.

When slaves are added to, or removed from, the system, you must modify the input connections and update the corresponding Verilog parameters to this module to adapt for the changes.

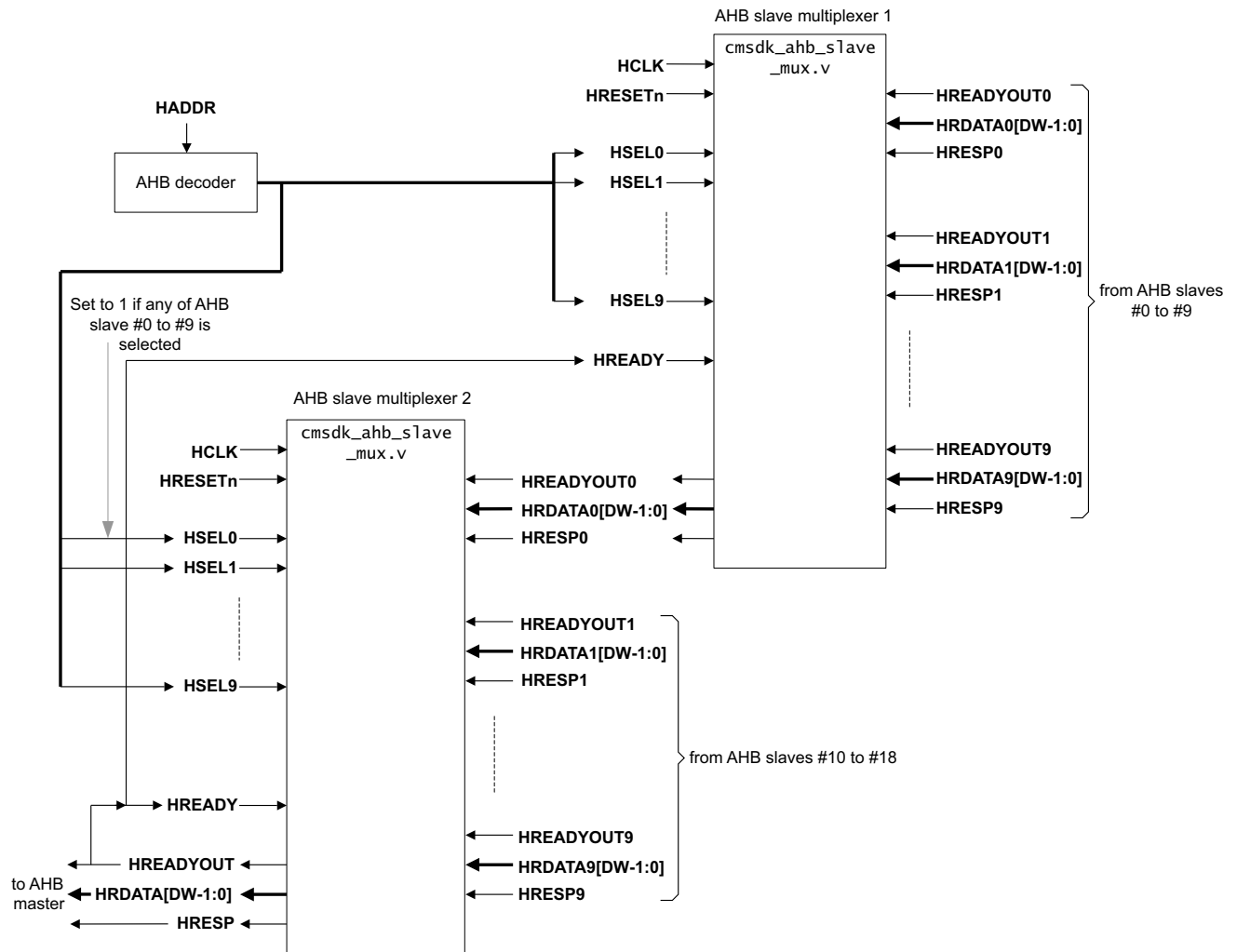
Table 3-4 shows the characteristics of the AHB slave multiplexer module.

Table 3-4 AHB slave multiplexer characteristics

| Element name | Description |
|--------------|---|
| Filename | cmsdk_ahb_slave_mux.v |
| Parameters | <p>PORT0_ENABLE The supported parameter values are: 0 Disable port 0. 1 Enable port 0.</p> <p>PORT1_ENABLE The supported parameter values are: 0 Disable port 1. 1 Enable port 1.</p> <p>PORT2_ENABLE The supported parameter values are: 0 Disable port 2. 1 Enable port 2.</p> <p>PORT3_ENABLE The supported parameter values are: 0 Disable port 3. 1 Enable port 3.</p> <p>PORT4_ENABLE The supported parameter values are: 0 Disable port 4. 1 Enable port 4.</p> <p>PORT5_ENABLE The supported parameter values are: 0 Disable port 5. 1 Enable port 5.</p> <p>PORT6_ENABLE The supported parameter values are: 0 Disable port 6. 1 Enable port 6.</p> <p>PORT7_ENABLE The supported parameter values are: 0 Disable port 7. 1 Enable port 7.</p> <p>PORT8_ENABLE The supported parameter values are: 0 Disable port 8. 1 Enable port 8.</p> <p>PORT9_ENABLE The supported parameter values are: 0 Disable port 9. 1 Enable port 9.</p> <p>———— Note ———— All PORTn_ENABLE are set to 1 by default.</p> <p>DW Data width. You can configure the width to either 64 bits or 32 bits. This is set to 32 by default.</p> |
| Clock domain | HCLK |

If you require more AHB slave ports, you can either cascade two AHB slave multiplexers, or expand the design.

Figure 3-4 on page 3-8 shows the cascade connection of two AHB slave multiplexers in which the **HSEL** signals for slaves 10-18 are connected to **HSEL1** to **HSEL9** of the AHB slave multiplexer 2. The **HSEL0** of the AHB slave multiplexer 2 is an OR function of the **HSEL** signal for the AHB slaves 0-9.

**Figure 3-4 Cascade connection**

Instead of using multiple slave multiplexers, you can modify the design as follows:

- Copy and rename the module.
- Add ports for AHB slave connections.
- Add Verilog parameters, such as `PORTn_ENABLE` if required.
- Add the data phase select register, `reg_hsel`, and its next state logic.
- Add ports to the slave signal multiplexing logic.
- Adjust the optional OVL assertion code.

3.4 AHB master multiplexer

The AHB master multiplexer, `cmsdk_ahb_master_mux.v`, permits up to three AHB masters to share an AHB connection. It uses parameters to define the master port usage. Therefore, the synthesis process does not generate unnecessary additional logic. Figure 3-5 shows the AHB master multiplexer.

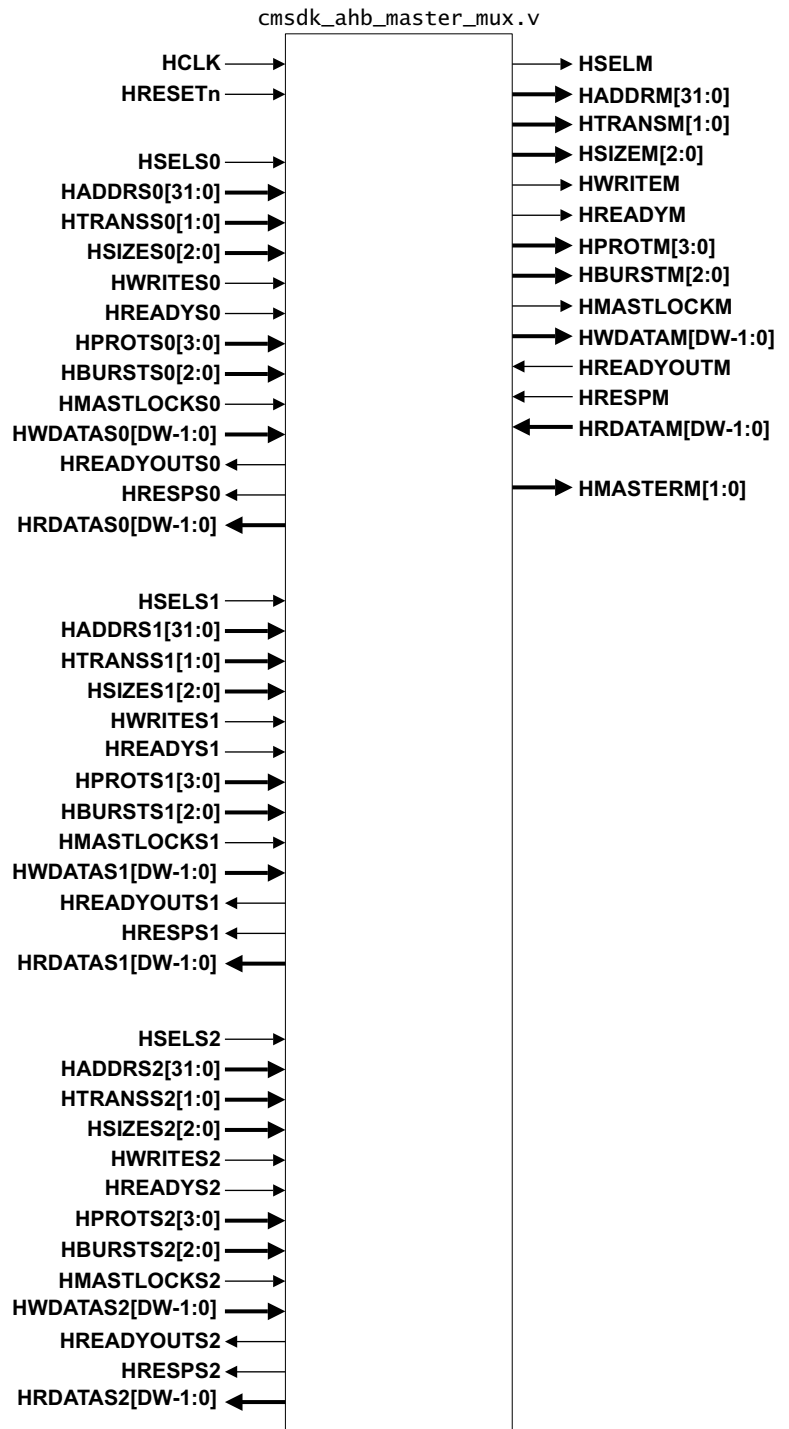


Figure 3-5 AHB master multiplexer

Table 3-5 shows the characteristics of the AHB master multiplexer.

Table 3-5 AHB master multiplexer characteristics

| Element name | Description |
|--------------|---|
| Filename | cmsdk_ahb_master_mux.v |
| Parameters | <p>PORT0_ENABLE The supported parameter values are:</p> <p> 0 Disable port 0.</p> <p> 1 Enable port 0.</p> <p>PORT1_ENABLE The supported parameter values are:</p> <p> 0 Disable port 1.</p> <p> 1 Enable port 1.</p> <p>PORT2_ENABLE The supported parameter values are:</p> <p> 0 Disable port 2.</p> <p> 1 Enable port 2.</p> <p>———— Note ————</p> <p>All PORTn_ENABLE are set to 1 by default.</p> <p>DW Data width. You can configure the width to either 64 bits or 32 bits. This is set to 32 by default.</p> |
| Clock domain | HCLK |

3.4.1 Arbitration scheme

The AHB master multiplexer uses a fixed arbitration scheme as follows:

- Port 0** Same priority as port 1, round-robin scheme.
- Port 1** Same priority as port 0, round-robin scheme.
- Port 2** Higher priority master.

Switch-over between different masters is disabled during a fixed length burst, locked transfers, or if a transfer is indicated to the AHB slaves at the same time as a wait state occurs on the bus.

You can break an incrementing burst with an unspecified length into multiple parts as a result of arbitration. The master multiplexer forces **HTRANS** to **NONSEQUENTIAL** for the first transfer after switching to ensure that the AHB protocol operates correctly.

3.4.2 Limitations

The AHB master multiplexer has the following limitations:

- The downstream slave must respond with **HREADYOUTM** HIGH and **HRESPM** OKAY when it is not selected.

3.4.3 HMASTERM output

The AHB master multiplexer provides an **HMASTERM[1:0]** output signal that indicates which port a transfer originated from:

- 2'b00 Port 0.
- 2'b01 Port 1.
- 2'b10 Port 2.
- 2'b11 None.

3.5 AHB GPIO

The AHB GPIO, `cmsdk_ahb_gpio.v`, is a general-purpose I/O interface unit.

The AHB GPIO provides a 16-bit I/O interface with the following properties:

- Programmable interrupt generation capability.
- Bit masking support using address values.
- Registers for alternate function switching with pin multiplexing support.
- Thread safe operation by providing separate set and clear addresses for control registers.
- Inputs are sampled using a double flip-flop to avoid metastability issues.

Figure 3-6 shows the control circuit and external interface of the AHB GPIO.

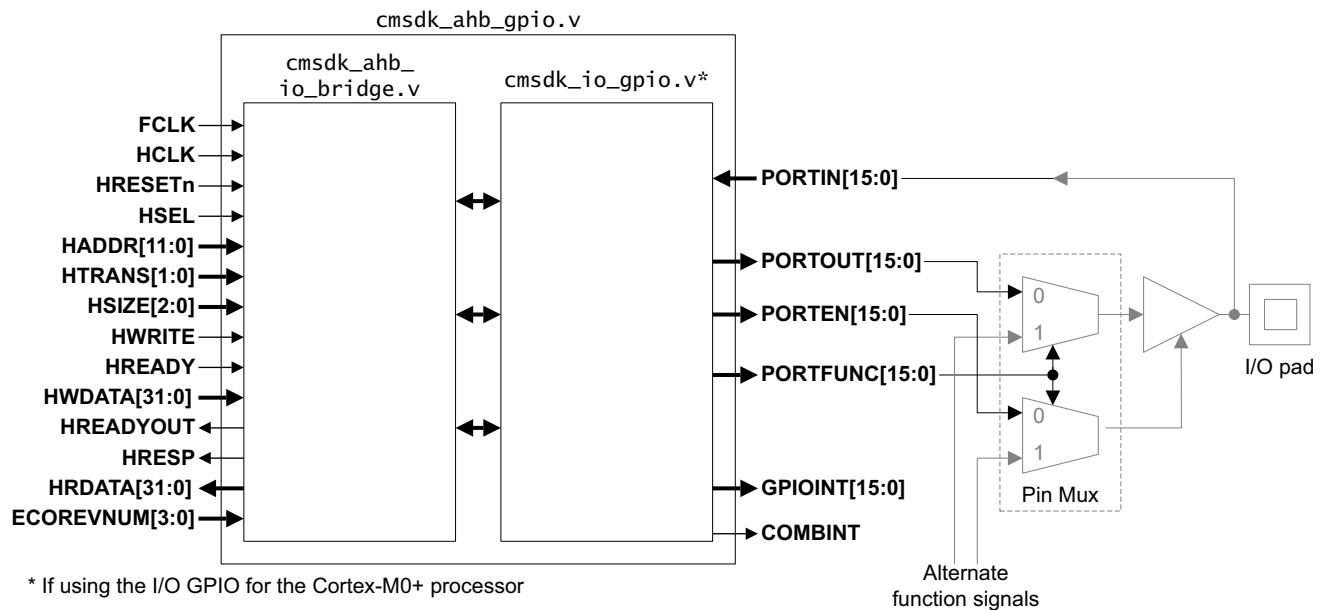


Figure 3-6 AHB GPIO control circuit and external interface

Table 3-6 shows the characteristics of the AHB GPIO.

Table 3-6 AHB GPIO characteristics

| Element name | Description |
|---------------|---|
| Filename | <code>cmsdk_ahb_gpio.v</code> |
| Parameters | <p>ALTERNATE_FUNC_MASK</p> <p>Indicates the pin that can have an alternate function. This parameter is set to 16'hFFFF by default. This means that all 16 pins can have alternate functions.</p> <p>ALTERNATE_FUNC_DEFAULT</p> <p>Default value for alternate function setting. This parameter is set to 16'h0000 by default. This means that all pins are used for the GPIO function after reset.</p> <p>BE</p> <p>Big-endian. The default value is 0 for little-endian. Set the value to 1 for big-endian configuration.</p> |
| Clock domains | <p>The clock domains are as follows:</p> <p>HCLK AHB-Lite system clock. Can be gated off during sleep mode.</p> <p>FCLK Free running clock, in same phase as HCLK. Must be running to generate edge trigger interrupt.</p> |

3.5.1 Features of the GPIO

The following sections describe the features of the GPIO:

- [Interrupt generation](#)
- [Masked access](#).

Interrupt generation

The AHB GPIO provides programmable interrupt generation features. Three registers control this, and each register has separate set and clear addresses. You can configure each bit of the I/O pins to generate interrupts based on these three registers. See [Table 3-7](#).

Table 3-7 Interrupt generation

| Interrupt enable[n] | Interrupt polarity[n] | Interrupt type[n] | Interrupt feature |
|---------------------|-----------------------|-------------------|-------------------|
| 0 | - | - | Disabled |
| 1 | 0 | 0 | Low-level |
| 1 | 0 | 1 | Falling edge |
| 1 | 1 | 0 | High-level |
| 1 | 1 | 1 | Rising edge |

After an interrupt is triggered, the corresponding bit in the INTSTATUS Register is set. This also causes the corresponding bit of the **GPIOINT[15:0]** signal to be asserted. As a result, the combined interrupt signal, **COMBINT**, is also asserted. You can clear the interrupt status using an interrupt handler that writes 1 to the corresponding bit of the INTCLEAR Register, the same address as the INTSTATUS Register.

———— Note ————

The free running clock signal, **FCLK**, must be active during interrupt detection, because of the double flip-flop synchronization logic. There is also a three cycle latency for the interrupt generation that consists of two cycles for input signal synchronization, and one cycle for registering of the interrupt status.

Masked access

The masked access feature permits individual bits or multiple bits to be read from or written to in a single transfer. This avoids software-based read-modify-write operations that are not thread safe. With the masked access operations, the 16-bit I/O is divided into two halves, lower byte and upper byte. The bit mask address spaces are defined as two arrays, each containing 256 words.

For example, to set bits[1:0] to 1 and clear bits[7:6] in a single operation, you can carry out the write to the lower byte mask access address space. The required bit mask is 0xC3, and you can write the operation as MASKLOWBYTE[0xC3] = 0x03 as [Figure 3-7 on page 3-13](#) shows.

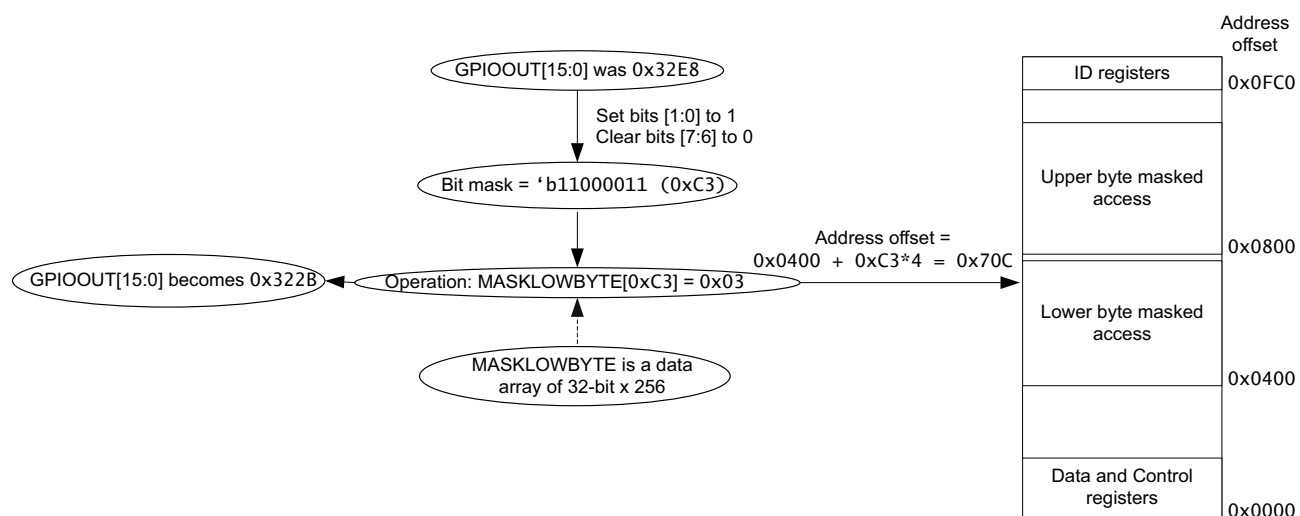


Figure 3-7 Masked access 1

Similarly, to update some of the bits in the upper eight bits of the GPIO port, you can use the MASKHIGHBYTE array as [Figure 3-8](#) shows.

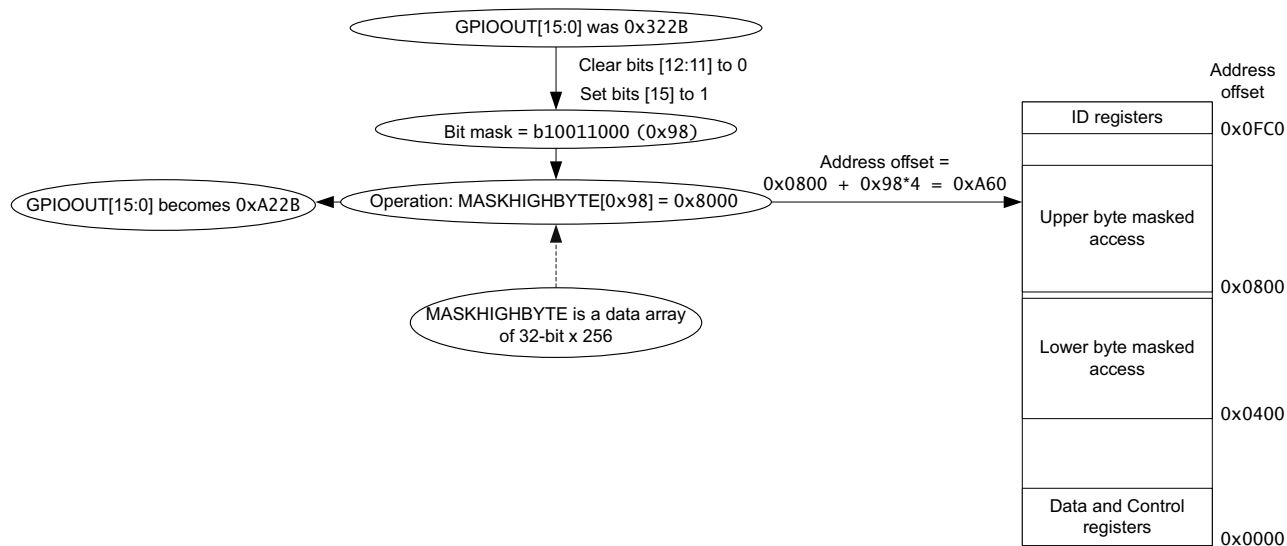


Figure 3-8 Masked access 2

3.5.2 Programmers model

Table 3-8 shows the software programmable registers in the example AHB GPIO.

Table 3-8 GPIO memory map

| Name | Base offset | Type | Width | Reset value | Description |
|------------|---------------|------|-------|-------------|---|
| DATA | 0x0000 | RW | 16 | 0x---- | Data value [15:0]: Read Sampled at pin. Write To data output register. Read back value goes through double flip-flop synchronization logic with a delay of two cycles. |
| DATAOUT | 0x0004 | RW | 16 | 0x0000 | Data output Register value [15:0]: Read Current value of data output register. Write To data output register. |
| Reserved | 0x0008-0x000C | - | - | - | Reserved. |
| OUTENSET | 0x0010 | RW | 16 | 0x0000 | Output enable set [15:0]: Write 1 Set the output enable bit. 0 No effect. Read back 0 Indicates the signal direction as input. 1 Indicates the signal direction as output. |
| OUTENCLR | 0x0014 | RW | 16 | 0x0000 | Output enable clear [15:0]: Write 1 Clears the output enable bit. 0 No effect. Read back 0 Indicates the signal direction as input. 1 Indicates the signal direction as output. |
| ALTFUNCSET | 0x0018 | RW | 16 | 0x0000 | Alternative function set [15:0]: Write 1 Sets the ALTFUNC bit. 0 No effect. Read back 0 For I/O. 1 For an alternate function. |
| ALTFUNCCLR | 0x001C | RW | 16 | 0x0000 | Alternative function clear [15:0]: Write 1 Clears the ALTFUNC bit. 0 No effect. Read back 0 For I/O. 1 For an alternate function. |
| INTENSET | 0x0020 | RW | 16 | 0x0000 | Interrupt enable set [15:0]: Write 1 Sets the enable bit. 0 No effect. Read back 0 Interrupt disabled. 1 Interrupt enabled. |

Table 3-8 GPIO memory map (continued)

| Name | Base offset | Type | Width | Reset value | Description |
|------------------------|-------------|------|-------|-------------|--|
| INTENCLR | 0x0024 | RW | 16 | 0x0000 | Interrupt enable clear [15:0]: |
| | | | | | Write 1 Clear the enable bit. |
| | | | | | 0 No effect. |
| | | | | | Read back 0 Interrupt disabled. |
| | | | | | 1 Interrupt enabled. |
| INTTYPESET | 0x0028 | RW | 16 | 0x0000 | Interrupt type set [15:0]: |
| | | | | | Write 1 Sets the interrupt type bit. |
| | | | | | 0 No effect. |
| | | | | | Read back 0 For LOW or HIGH level. |
| | | | | | 1 For falling edge or rising edge. |
| INTTYPECLR | 0x002C | RW | 16 | 0x0000 | Interrupt type clear [15:0]: |
| | | | | | Write 1 Clears the interrupt type bit. |
| | | | | | 0 No effect. |
| | | | | | Read back 0 For LOW or HIGH level. |
| | | | | | 1 For falling edge or rising edge. |
| INTPOLSET | 0x0030 | RW | 16 | 0x0000 | Polarity-level, edge IRQ configuration [15:0]: |
| | | | | | Write 1 Sets the interrupt polarity bit. |
| | | | | | 0 No effect. |
| | | | | | Read back 0 For LOW level or falling edge. |
| | | | | | 1 For HIGH level or rising edge. |
| INTPOLCLR | 0x0034 | RW | 16 | 0x0000 | Polarity-level, edge IRQ configuration [15:0]: |
| | | | | | Write 1 Clears the interrupt polarity bit. |
| | | | | | 0 No effect. |
| | | | | | Read back 0 For LOW level or falling edge. |
| | | | | | 1 For HIGH level or rising edge. |
| INTSTATUS, INTCLEAR | 0x0038 | RW | 16 | 0x0000 | Write one to clear interrupt request: |
| | | | | | Write [15:0] IRQ status clear Register. |
| | | | | | Write: |
| | | | | | 1 To clear the interrupt request. |
| | | | | | 0 No effect. |
| | | | | | Read back [15:0] IRQ status Register. |

Table 3-8 GPIO memory map (continued)

| Name | Base offset | Type | Width | Reset value | Description |
|-------------------|-----------------|------|-------|-------------|---|
| MASKLOWBYTE | 0x0400-0x07FC | RW | 16 | 0x---- | Lower eight bits masked access. Bits[9:2] of the address value are used as enable bit mask for the access: [15:8] Not used. RAZ/WI. [7:0] Data for lower byte access, with bits[9:2] of address value used as enable mask for each bit. |
| MASKHIGHBYTE | 0x0800-0x0BFC | RW | 16 | 0x---- | Higher eight bits masked access. Bits[9:2] of the address value are used as enable bit mask for the access: [15:8] Data for higher byte access, with bits[9:2] of address value used as enable mask for each bit. [7:0] Not used. RAZ/WI. |
| Reserved | 0x0C00 - 0x0FCF | - | - | - | Reserved. |
| PID4 | 0x0FD0 | RO | 8 | 0x04 | Peripheral ID Register 4: [7:4] Block count. [3:0] jep106_c_code. |
| PID5 ^a | 0x0FD4 | RO | - | 0x00 | Peripheral ID Register 5. |
| PID6 ^a | 0x0FD8 | RO | - | 0x00 | Peripheral ID Register 6. |
| PID7 ^a | 0x0FDC | RO | - | 0x00 | Peripheral ID Register 7. |
| PID0 | 0x0FE0 | RO | 8 | 0x20 | Peripheral ID Register 0: [7:0] Part number[7:0]. |
| PID1 | 0x0FE4 | RO | 8 | 0xB8 | Peripheral ID Register 1: [7:4] jep106_id_3_0. [3:0] Part number[11:8]. |
| PID2 | 0x0FE8 | RO | 8 | 0x1B | Peripheral ID Register 2: [7:4] Revision. [3] jedec_used. [2:0] jep106_id_6_4. |
| PID3 | 0x0FEC | RO | 8 | 0x00 | Peripheral ID Register 3: [7:4] ECO revision number. [3:0] Customer modification number. |
| CID0 | 0x0FF0 | RO | 8 | 0x0D | Component ID Register 0. |
| CID1 | 0x0FF4 | RO | 8 | 0xF0 | Component ID Register 1. |
| CID2 | 0x0FF8 | RO | 8 | 0x05 | Component ID Register 2. |
| CID3 | 0x0FFC | RO | 8 | 0xB1 | Component ID Register 3. |

a. The PID5, PID6, and PID7 registers are not used.

3.5.3 Component dependency

For use with the Cortex-M0+ processor, the AHB GPIO contains an AHB to single-cycle I/O interface adaptor, and a GPIO module with a single-cycle I/O interface. To use this module in your design, add `cmsdk_ahb_gpio/verilog` and `cmsdk_iop_gpio/verilog` in the search path, or explicitly include the Verilog RTL files in these two directories in your project.

3.6 AHB to APB sync-down bridge

The AHB to APB sync-down bridge, `cmsdk_ahb_to_apb.v`, has the following features:

- Supports APB2, APB3, and APB4.
- Runs the APB interface semi-synchronously slower than the AHB interface.

Figure 3-9 shows the AHB to APB sync-down bridge.

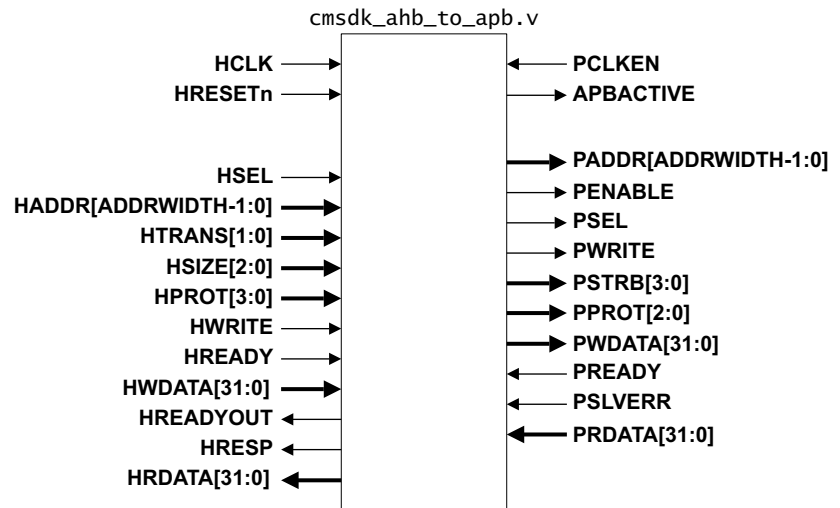


Figure 3-9 AHB to APB sync-down bridge

Table 3-9 shows the characteristics of the AHB to APB sync-down bridge module.

Table 3-9 AHB to APB sync-down bridge characteristics

| Element name | Description |
|--------------|--|
| Filename | <code>cmsdk_ahb_to_apb.v</code> |
| Parameters | <p>ADDRWIDTH APB address width. The default value is 16, that is, 64K byte APB address space.</p> <p>REGISTER_RDATA 1 Registered read data path. 0 Combinational read data path. The default value is 1.</p> <p>REGISTER_WDATA 1 Registered write data path. 0 Combinational write data path. Registering write data can help reduce timing issues caused by large fanouts. The default value is 0.</p> |
| Clock domain | HCLK |

The AHB to APB bridge has an output called **APBACTIVE** that controls the clock gating cell for generation of a gated **PCLK**. The gated **PCLK** is called **PCLKG** in the example system. When there is no APB transfer, this signal is LOW and stops **PCLKG**. Peripherals designed with separate clock pins for bus logic and peripheral operation can use the gated **PCLK** to reduce power consumption.

This block requires an APB clock synchronized to **HCLK**. **PCLK** can be divided or the same as **HCLK** by using **PCLKEN**.

When developing a system for AMBA 2.0, you can tie **PSLVERR** LOW, and **PREADY** HIGH.

When using APB2 and APB3 peripheral systems, you can ignore the **PPROT[2:0]** and the **PSTRB[3:0]** signals.

For systems that do not require a high operating frequency, you can override the **REGISTER_RDATA** Verilog parameter to 0 to reduce the latency of APB accesses. This results in the read data from the APB slaves, **PRDATA**, being directly output to the AHB read data output, **HRDATA**, and reduces the wait states in addition to the gate counts. By default, the **REGISTER_RDATA** parameter is set to 1 to include a registering stage.

For a system with **HCLK** equal to **PCLK**, and if there is no error response from APB slaves, the minimum number of cycles for each RW is as follows:

- Three **HCLK** cycles when **REGISTER_RDATA** is 1.
- Two **HCLK** cycles when **REGISTER_RDATA** is 0.

For systems that require a high operating frequency, set the **REGISTER_WDATA** Verilog parameter to 1 to register the AHB master write data. This breaks the path between **HWDATA** and **PWDATA** but increases the latency of write transfers by one cycle.

3.7 AHB to SRAM interface module

The AHB to SRAM interface module, `cmsdk_ahb_to_sram.v`, enables on-chip synchronous SRAM blocks to attach to an AHB interface. It performs read and write operations with zero wait states. The design supports 32-bit SRAM only. The SRAM must support byte writes. You can also use this module in FPGA development for connecting FPGA block RAM to the AHB. [Figure 3-10](#) shows the AHB to SRAM interface module.

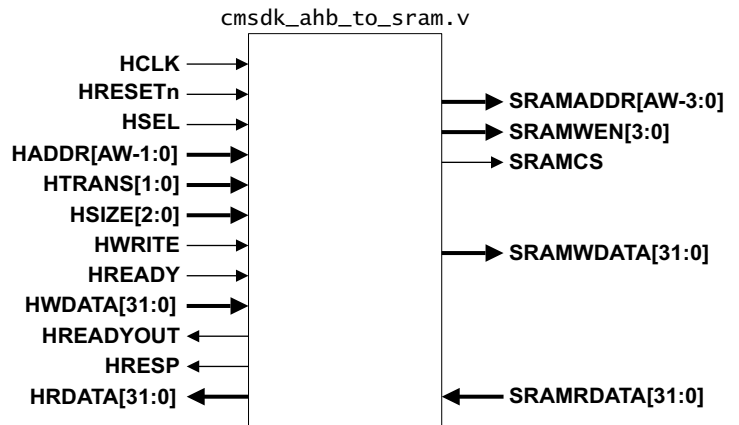


Figure 3-10 AHB to SRAM interface module

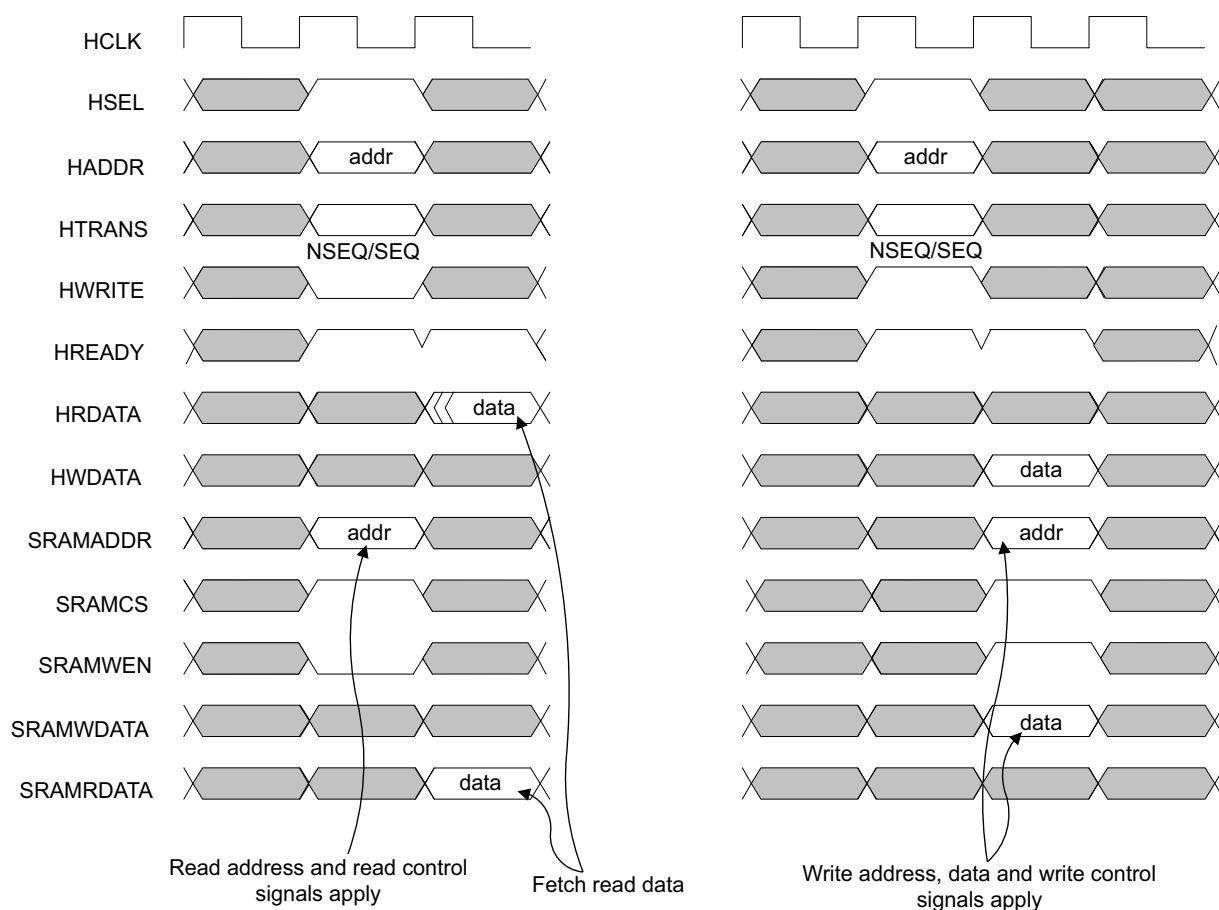
[Table 3-10](#) shows the characteristics of the AHB to SRAM interface module.

Table 3-10 AHB to SRAM interface module characteristics

| Element name | Description |
|--------------|--|
| Filename | <code>cmsdk_ahb_to_sram.v</code> |
| Parameters | AW Address width. The default value is 16, that is, 64KB. For example, if the SRAM is 8KB, set AW to 13. |
| Clock domain | HCLK |

The design always responds with OKAY and zero wait states.

The AHB to SRAM interface module assumes the SRAM read and write access timings that [Figure 3-11 on page 3-21](#) shows.

**Figure 3-11 SRAM interface timing**

If a read operation follows immediately after a write operation, the write address and write data are stored in an internal buffer, and the SRAM carries out the read operation first. The stalled write transfer is carried out when the AHB interface is idle, or when there is a write transfer.

A merging of read data between the internal buffers and the read data from SRAM is carried out automatically by the interface module, when:

- A read operation follows immediately after a write operation to the same address.
- A sequence of read operations follows immediately after a write operation with any of the read transfers using the same address.

The merging processing uses internal buffer byte valid status and ensures the read data that returns to the bus master is up-to-date. This process occurs transparently and does not result in any wait states.

3.8 AHB to flash interface modules

The AHB to flash interface modules, `cmsdk_ahb_to_flash32.v` and `cmsdk_ahb_to_flash16.v`, enable you to connect a simple 32-bit or 16-bit read-only flash memory model to an AHB system. They include parameterized wait state generation.

Figure 3-12 shows the AHB to flash interface module for 32-bit flash ROM.

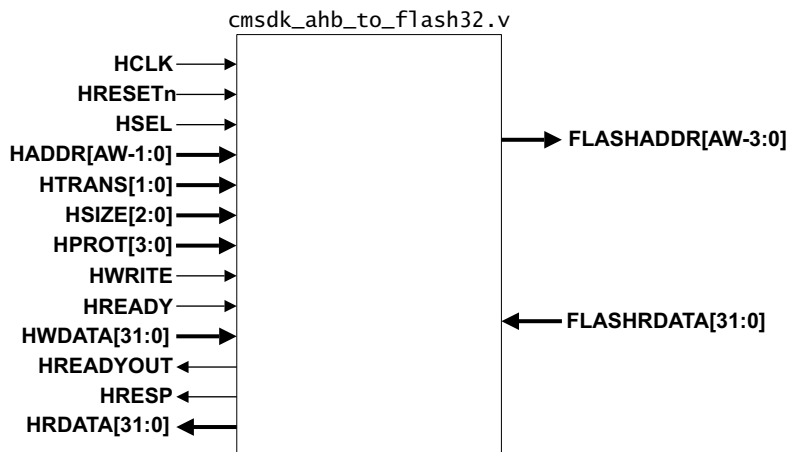


Figure 3-12 AHB to flash interface module for 32-bit flash ROM

Table 3-12 on page 3-23 shows the characteristics of the AHB to flash interface module for 32-bit flash ROM.

Table 3-11 AHB to flash interface module for 32-bit flash ROM characteristics

| Element name | Description | |
|--------------|-------------------------------------|---|
| Filename | <code>cmsdk_ahb_to_flash32.v</code> | |
| Parameters | AW | Address width. The default value is 16. |
| | WS | Wait state. The default value is 1. The valid range of wait state is 0-3. |
| Clock domain | HCLK | |

Figure 3-13 on page 3-23 shows the AHB to flash interface module for 16-bit flash ROM. This module is design to work with 16-bit AHB. A 32-bit AHB to 16-bit AHB downsizer is available in the Cortex-M0+ deliverable. Inside the integration kit, the filename is `cm0p_32to16_dnsiz.v`.

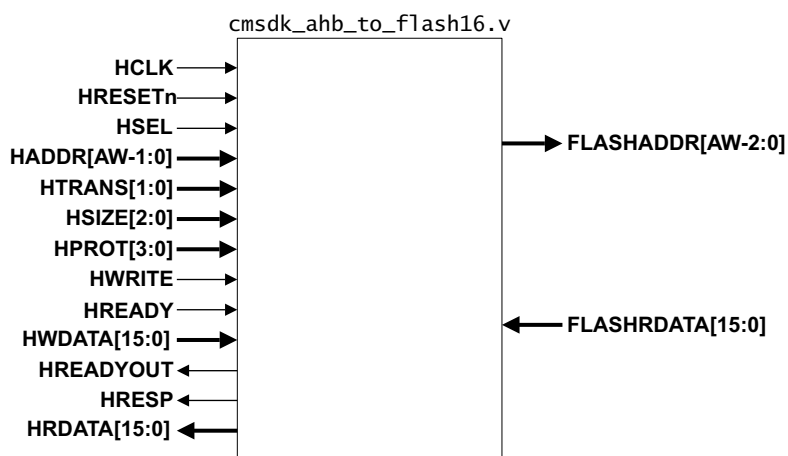


Figure 3-13 AHB to flash interface module for 16-bit flash ROM

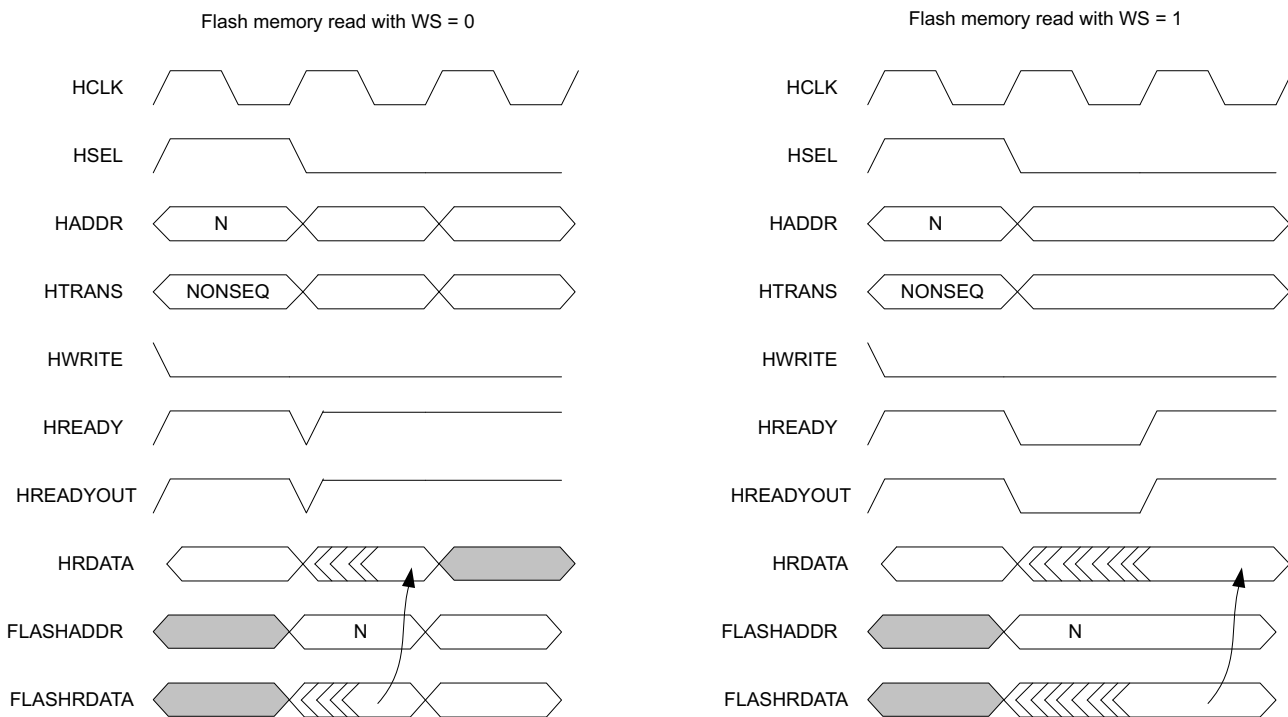
Table 3-12 shows the characteristics of the AHB to flash interface module for 16-bit flash ROM.

Table 3-12 AHB to flash interface module for 16-bit ROM characteristics

| Element name | Description | |
|--------------|------------------------|---|
| Filename | cmsdk_ahb_to_flash16.v | |
| Parameters | AW | Address width. The default value is 16. |
| | WS | Wait state. The default value is 1. The valid range of wait state is 0-3. |
| Clock domain | HCLK | |

This interface module only supports read operations.

Figure 3-14 on page 3-24 shows the flash memory read access timings with different wait states.

**Figure 3-14 AHB to flash read access timing**

3.9 AHB timeout monitor

The AHB timeout monitor, `cmsdk_ahb_timeout_mon.v`, prevents an AHB slave from locking up a system. It is placed between the AHB master and slave, and is connected directly to the AHB slave. If there is an active transfer to the slave, and the slave holds **HREADY** LOW for more than a certain number of clock cycles, the monitor generates an error response to the bus master. Figure 3-15 shows the AHB timeout monitor module.

If the bus master generates any subsequent access to this slave, the monitor returns an error response, and blocks access to the slave. The timeout monitor stops generating error responses and masking access to the slave when the slave has completed the transfer that timed out by asserting **HREADYOUT** HIGH. If a burst is in progress, the timeout monitor masks the remaining beats in the burst before becoming transparent again.

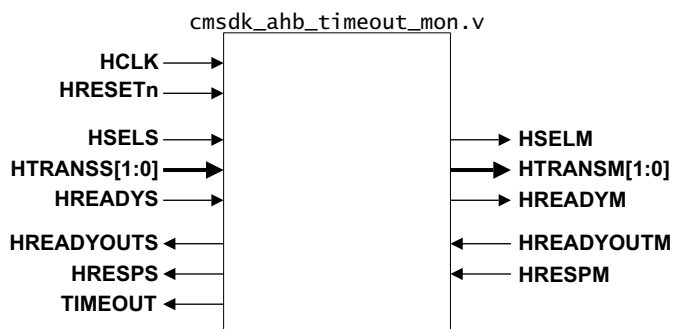


Figure 3-15 AHB timeout monitor

Figure 3-16 shows the typical usage of the AHB timeout monitor.

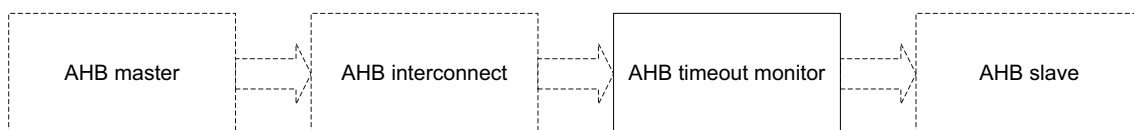


Figure 3-16 Use of AHB timeout monitor

If the monitor is directly coupled to the processor, or connected to an AHB path that is used for exception handler code access, the processor cannot execute the bus fault exception handler.

If multiple bus slaves require monitoring, ARM recommends that you use multiple monitors instead of putting one monitor at the AHB slave multiplexer connection, to prevent the monitor from blocking access to the program ROM or SRAM.

The `TIME_OUT_VALUE` Verilog parameter determines the number of wait state cycles that trigger the timeout.

You can use the **TIMEOUT** output signal to export timeout events to external logic. During timeout, the **TIMEOUT** signal is asserted continuously until the AHB slave asserts the **HREADYOUT** signal.

Table 3-13 on page 3-26 shows the characteristics of the AHB timeout monitor.

Table 3-13 AHB timeout monitor characteristics

| Element name | Description | |
|--------------|-------------------------|--|
| Filename | cmsdk_ahb_timeout_mon.v | |
| Parameters | TIME_OUT_VALUE | Number of wait cycles that trigger timeout. Permitted values for this parameter are 2-1024 inclusive. The default value is 16. |
| Clock domain | HCLK | |

3.10 AHB to external SRAM interface

The AHB to external SRAM interface module, `cmsdk_ahb_to_extmem16.v`, enables external SRAM, static memory devices or external peripherals, to connect to the Cortex-M processor design. The module supports only 16-bit and 8-bit external interfaces. Figure 3-17 shows the AHB to external SRAM interface module.

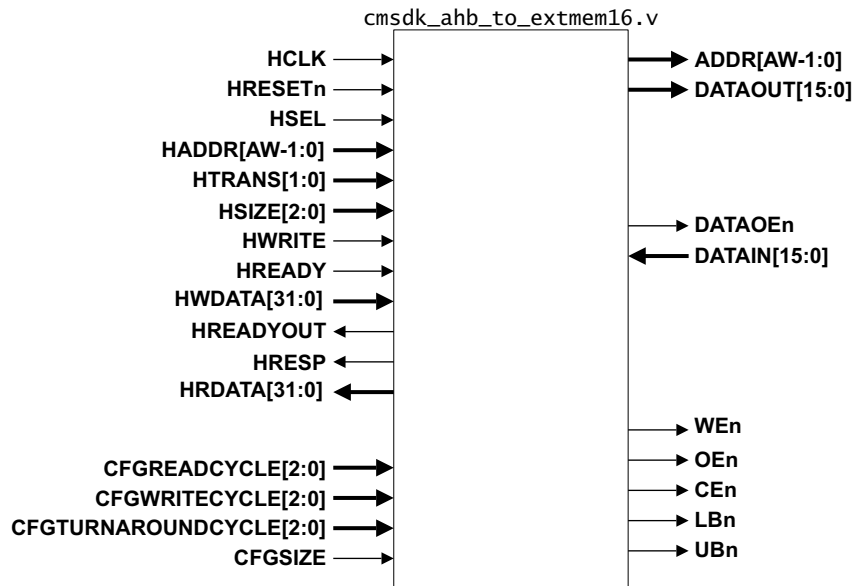


Figure 3-17 AHB to external SRAM interface

The interface module, `cmsdk_ahb_to_extmem16.v`, is designed to support an external bidirectional data bus. The **DATAOEn** signal controls the tristate buffer for data output. You must add your own tristate buffers in your system implementation. The design enables turnaround cycles to be inserted between reads and writes to prevent current spikes that could occur for a very short time when the processor system and the external device both drive the data bus. The following signals control wait states for reads, wait states for writes, and the number of turnaround cycles respectively:

- **CFGREADCYCLE**.
- **CFGWRITECYCLE**.
- **CFGTURNAROUNDCYCLE**.

You can operate the interface module in 8-bit mode, with **CFGSIZE** LOW, or 16-bit mode, with **CFGSIZE** HIGH. All the configuration control signals must remain stable during operation. The design generates an OKAY response.

Table 3-14 shows the characteristics of the AHB to external SRAM interface.

Table 3-14 AHB to external SRAM interface characteristics

| Element name | Description |
|--------------|---|
| Filename | <code>cmsdk_ahb_to_extmem16.v</code> |
| Parameters | <i>Aw</i> Address width. The default value is 16. |
| Clock domain | HCLK |

3.10.1 Signal descriptions

Table 3-15 shows the non-AMBA signals that the AHB to external SRAM interface uses.

Table 3-15 AHB to external SRAM interface signals

| Signal | Description |
|--------------------------------|--|
| CFGREADCYCLE[2:0] | Number of clock cycles for a read operation. A value of 0 indicates one read cycle. |
| CFGWRITECYCLE[2:0] | Number of clock cycles for a write operation. A value of 0 indicates one write cycle. The interface module automatically inserts one additional setup cycle before the write and one hold cycle after the write. |
| CFGTURNAROUNDCYCLE[2:0] | Number of clock cycles required to switch between a read and a write operation on the tristate bus. A value of 0 indicates one turnaround cycle. |
| CFGSIZE | Set: LOW For 8-bit memory. HIGH For 16-bit memory. |
| DATAOEn | Tristate buffer output enable for DATAOUT . Active LOW. |
| WEn | Write strobe for external memory device. Active LOW. |
| OEn | Read access output enable for external memory device. Active LOW. |
| CEn | Chip enable. Active LOW. |
| LBn | Lower byte enable. Active LOW. |
| UBn | Upper byte enable. Active LOW. |

Figure 3-18 on page 3-29 shows the external SRAM interface timing for the following signals. These are the control wait states for reads, wait states for writes, and the number of turnaround cycles respectively:

- **CFGREADCYCLE=0.**
- **CFGWRITECYCLE=0.**
- **CFGTURNAROUNDCYCLE=0.**

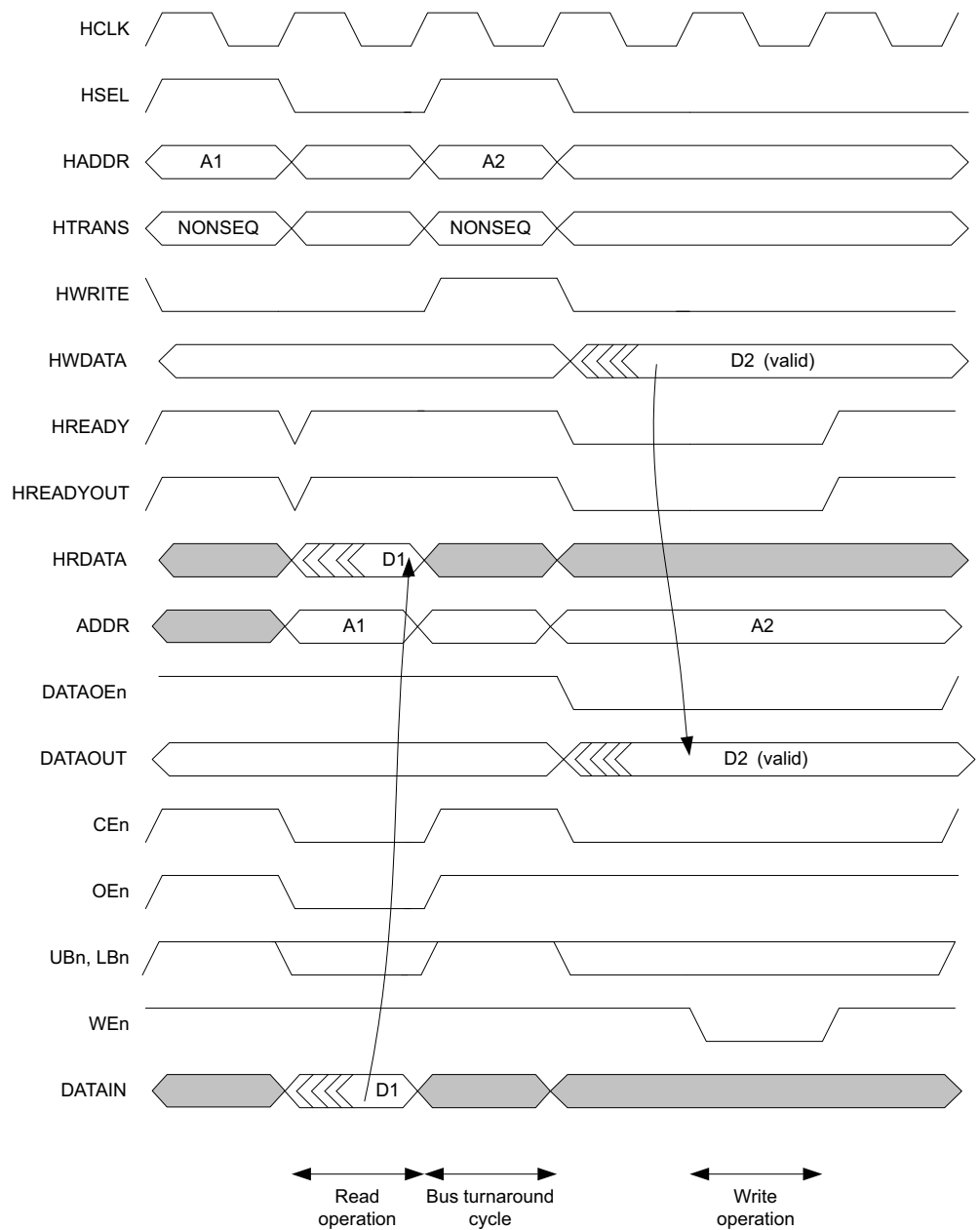


Figure 3-18 External SRAM interface timing 1

Figure 3-19 on page 3-30 shows the external SRAM interface timing for the following signals. These are the control wait states for reads, wait states for writes, and the number of turnaround cycles respectively:

- **CFGREADCYCLE**=1, that is, two cycles.
- **CFGWRITECYCLE**=1, that is, two cycles.
- **CFGTURNAROUNDCYCLE**=1, that is, two cycles.

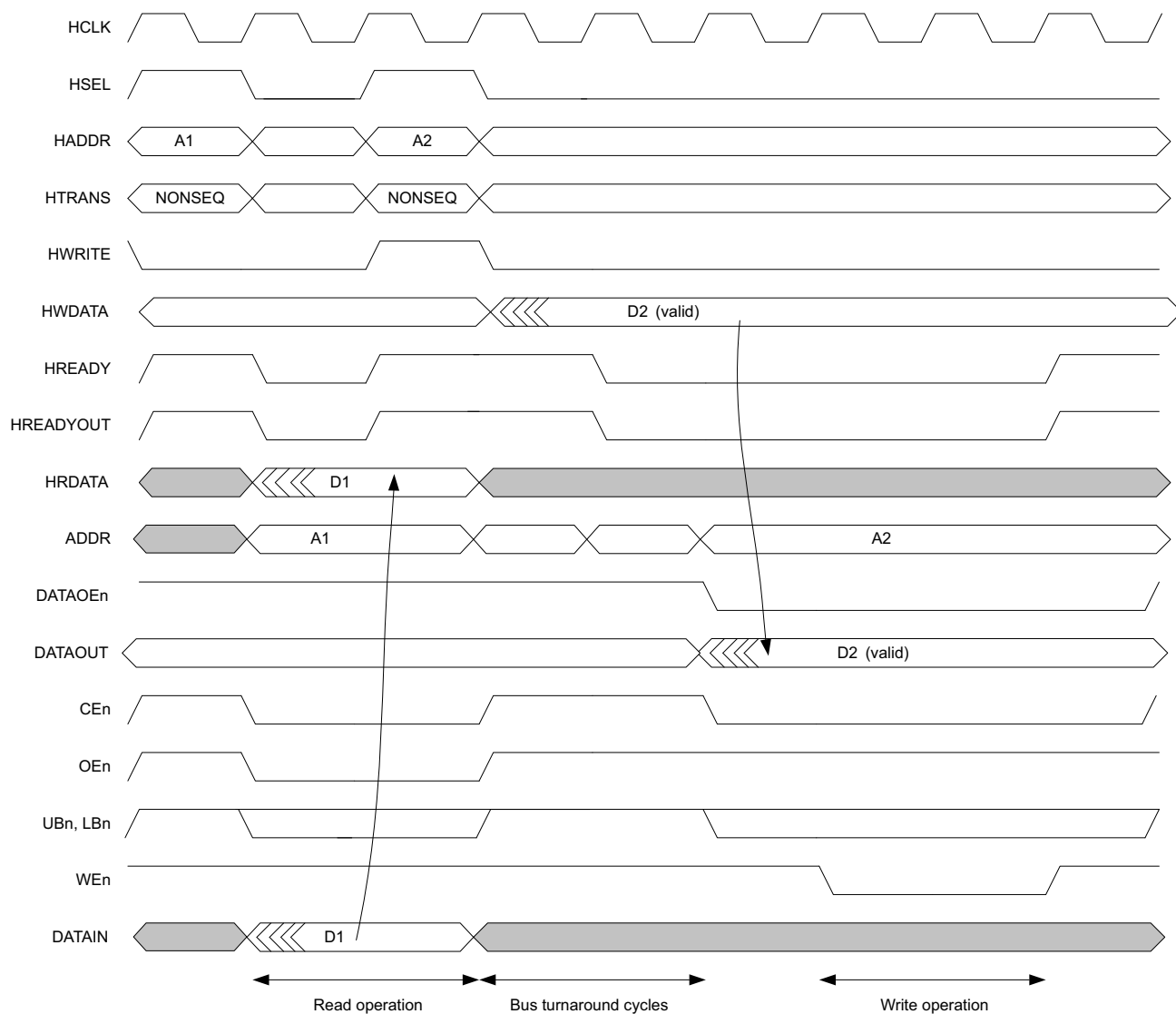


Figure 3-19 External SRAM interface timing 2

3.11 AHB bit-band wrapper

The AHB bit-band wrapper, `cmsdk_ahb_bitband.v`, provides the bit-band functionality for the Cortex-M0 and Cortex-M0+ processor.

Note

The bit-band wrapper is provided as a workaround for designers who are migrating silicon designs from a Cortex-M3 or Cortex-M4 processor to a Cortex-M0 or Cortex-M0+ processor, and require software compatibility with the Cortex-M3 and Cortex-M4 bit-band feature. Using the bit-band wrapper can result in the following:

- Longer timing paths on AHB interconnect and therefore a reduction in the maximum clock frequency.
- Higher power consumption.
- Larger design size.

ARM recommends that bit level access functionality is designed into any peripherals that can benefit from fast bit set and clear operations rather than using the bit-band wrapper.

Figure 3-20 shows the AHB bit-band wrapper module for the Cortex-M0 and Cortex-M0+ processor.

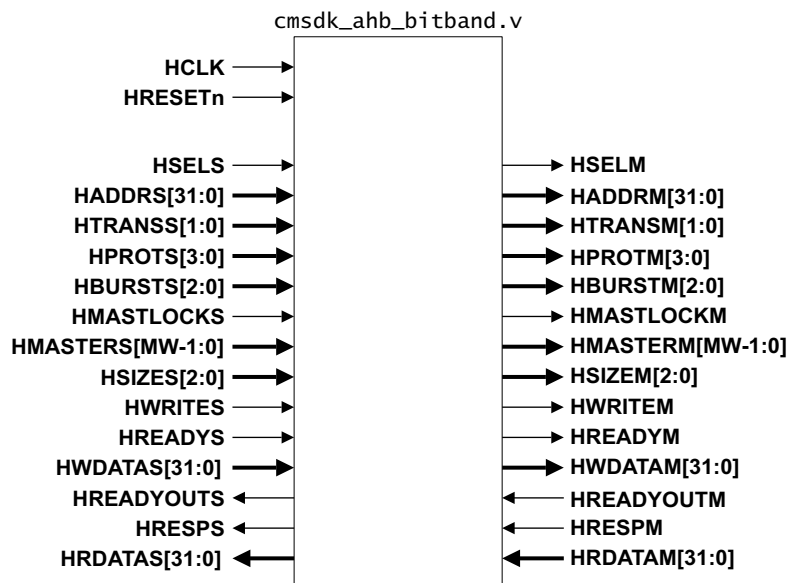


Figure 3-20 AHB bit-band wrapper for Cortex-M0 and Cortex-M0+ processor

Table 3-16 shows the characteristics of the AHB bit-band wrapper for the Cortex-M0 and Cortex-M0+ processor.

Table 3-16 AHB bit-band wrapper for Cortex-M0 processor characteristics

| Element name | Description |
|--------------|---|
| Filename | cmsdk_ahb_bitband.v |
| Parameters | MW Width of HMASTER signals. The default value is 1 because HMASTER in the Cortex-M0 and Cortex-M0+ processor is a one-bit signal. |
| | BE Big-endian. The default value is 0 for little-endian. Set the value to 1 for big-endian configuration. |
| Clock domain | HCLK |

When an AHB data transfer goes to bit-band alias regions 0x22000000 to 0x23FFFFFFC or 0x42000000 to 0x43FFFFFFC, the transfer is remapped to bit-band regions 0x20000000 to 0x200FFFFF or 0x40000000 to 0x400FFFFF.

If the transfer is a read operation, a single remapped read transfer is produced, and the *Least Significant Bit* (LSB) of the read data indicates the bit value read. If the transfer is a write operation, the transfer is converted into a locked read-modify-write sequence.

The written bit is replaced by the LSB of the write data from the bus master, for example, the Cortex-M0 or Cortex-M0+ processor. During the read-modify sequence, **HMASTLOCK** is asserted to ensure that the operation is atomic.

The value of **HADDR** must be word-aligned when accessing a bit-band alias. The transfer size is either in word, halfword, or byte. The size of the transfer to the AHB slaves matches the transfer size that the bus master uses.

For instruction transfers, indicated by 0 in **HPROT[0]** or transfers to other memory locations, the transfers are not altered.

3.11.1 Bit-banding

Bit-banding maps a complete word of memory onto a single bit in the bit-band region. For example, writing to one of the alias words sets or clears the corresponding bit in the bit-band region. This enables every individual bit in the bit-banding region to be directly accessible from a word-aligned address using a single LDR instruction. It also enables individual bits to be toggled without performing a read-modify-write sequence of instructions.

The bit-band wrapper supports two bit-band regions. These occupy the lowest 1MB of the SRAM, 0x20000000, and peripheral memory, 0x40000000, regions respectively. These bit-band regions map each word in an alias region of memory to a bit in a bit-band region of memory.

The bit-band wrapper contains logic that controls bit-band accesses as follows:

- It remaps bit-band alias addresses to the bit-band region.
- For reads, it extracts the requested bit from the read byte, and returns this in the LSB of the read data returned to the core.
- For writes, it converts the write to an atomic read-modify-write operation.

The memory map has two 32MB alias regions that map to two 1MB bit-band regions:

- Accesses to the 32MB SRAM alias region map to the 1MB SRAM bit-band region.
- Accesses to the 32MB peripheral alias region map to the 1MB peripheral bit-band region.

The following mapping formula shows how to reference each word in the alias region to a corresponding bit, or target bit, in the bit-band region:

$$\text{bit_word_offset} = (\text{byte_offset} \times 32) + (\text{bit_number} \times 4)$$

$$\text{bit_word_addr} = \text{bit_band_base} + \text{bit_word_offset}$$

where:

bit_word_offset The position of the target bit in the bit-band memory region.

bit_word_addr The address of the word in the alias memory region that maps to the targeted bit.

bit_band_base The starting address of the alias region.

byte_offset The number of the byte in the bit-band region that contains the targeted bit.

bit_number The bit position, 0-7, of the targeted bit.

Figure 3-21 shows examples of bit-band mapping between the SRAM bit-band alias region and the SRAM bit-band region:

- The alias word at 0x23FFFFE0 maps to bit [0] of the bit-band byte at 0x200FFFFF: $0x23FFFFE0 = 0x22000000 + (0xFFFF \times 32) + 0 \times 4$.
- The alias word at 0x23FFFFFC maps to bit [7] of the bit-band byte at 0x200FFFFF: $0x23FFFFFC = 0x22000000 + (0xFFFF \times 32) + 7 \times 4$.
- The alias word at 0x22000000 maps to bit [0] of the bit-band byte at 0x20000000: $0x22000000 = 0x22000000 + (0 \times 32) + 0 \times 4$.
- The alias word at 0x2200001C maps to bit [7] of the bit-band byte at 0x20000000: $0x2200001C = 0x22000000 + (0 \times 32) + 7 \times 4$.

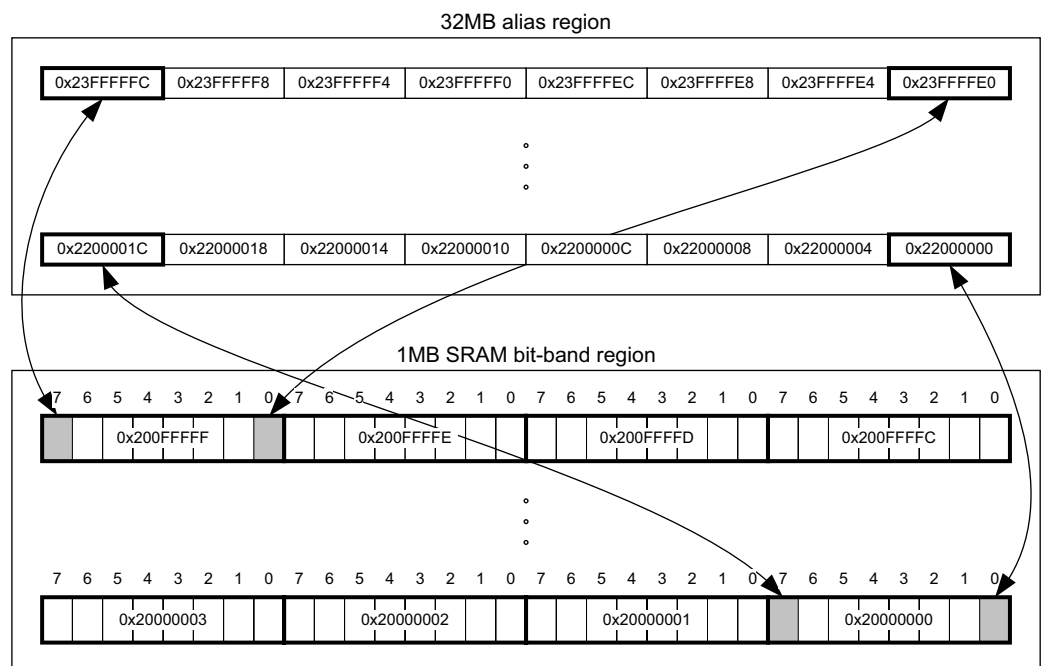


Figure 3-21 Bit-band mapping

Accessing an alias region directly

Writing to a word in the alias region has the same effect as a read-modify-write operation on the targeted bit in the bit-band region.

Bit[0] of the value written to a word in the alias region determines the value written to the targeted bit in the bit-band region:

- Writing a value with bit[0] set writes a 1 to the bit-band bit.
- Writing a value with bit[0] cleared writes a 0 to the bit-band bit.

Bits[31:1] of the alias word have no effect on the bit-band bit:

- Writing 0x01 has the same effect as writing 0xFF.
- Writing 0x00 has the same effect as writing 0x0E.

Reading a word in the alias region returns either 0x01 or 0x00:

- A value of 0x01 indicates that the targeted bit in the bit-band region is set.
- A value of 0x00 indicates that the targeted bit is clear.

Bits[31:1] are 0.

Directly accessing a bit-band region

You can directly access the bit-band region with normal reads and writes to that region.

3.11.2 Limitations

The AHB bit-band has the following limitations:

- The downstream slave must respond with **HREADYOUTM** HIGH and **HRESPM** OKAY when it is not selected.

Chapter 4

APB Components

This chapter describes the APB components that the Cortex-M System Design Kit uses. It contains the following sections:

- *APB example slaves* on page 4-2.
- *APB timer* on page 4-5.
- *APB UART* on page 4-8.
- *APB dual-input timers* on page 4-11.
- *APB watchdog* on page 4-20.
- *APB slave multiplexer* on page 4-26.
- *APB subsystem* on page 4-27.
- *APB timeout monitor* on page 4-33.

4.1 APB example slaves

The APB example slaves, `cmsdk_apb3_eg_slave.v` and `cmsdk_apb4_eg_slave.v`, demonstrate how to implement basic APB slaves. Each provides four words of hardware RW registers and additional read-only ID registers. Each APB transfer to these example slaves takes two cycles, and no additional wait states are inserted. The following example slaves are included for the Cortex-M System Design Kit:

- APB3.
- APB4.

Figure 4-1 shows an APB3 example slave module.

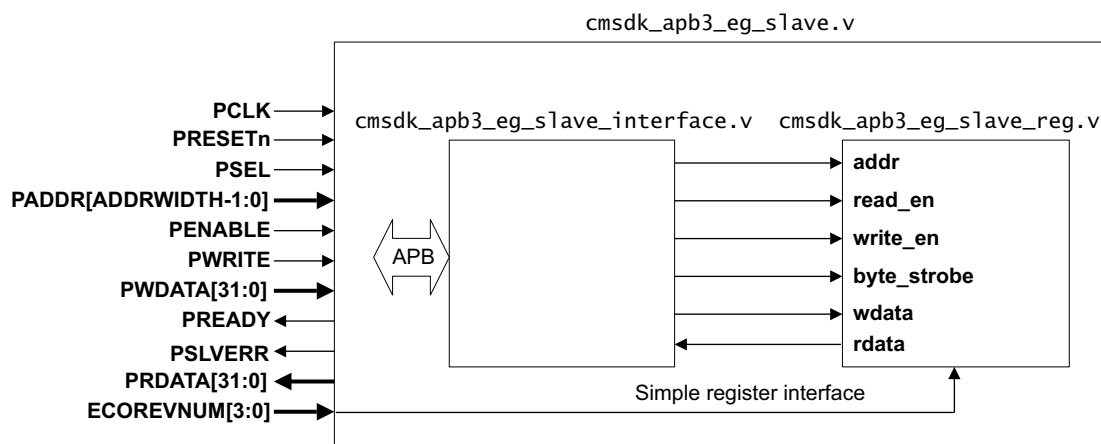


Figure 4-1 APB3 example slave

Figure 4-2 shows an APB4 example slave module.

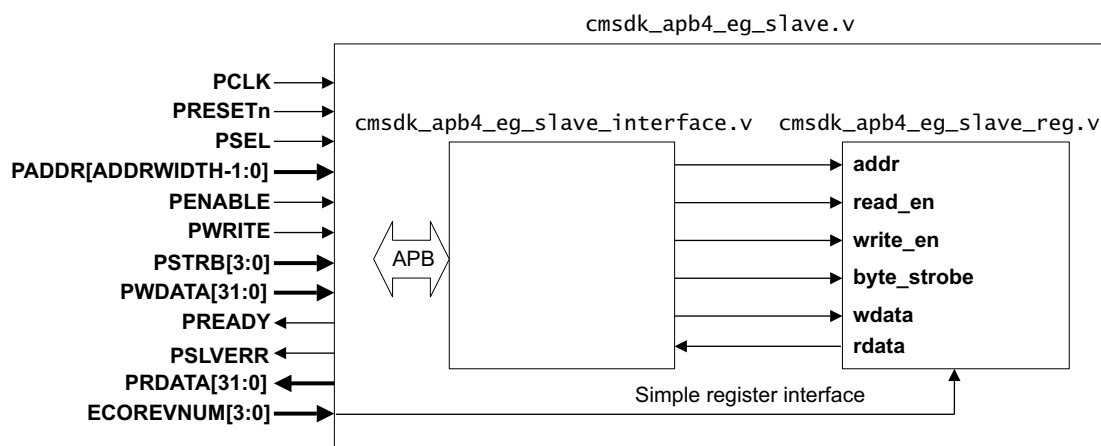


Figure 4-2 APB4 example slave

The APB example slaves include the following features:

- A simple APB slave interface.
- 32-bit data bus, endian-independent. For the APB3 example slave, the data handling is 32 bits only. For the APB4 example slave, use the **PSTRB** signal to perform the write operations on individual bytes.
- Data transfers require two clock cycles.

- Four 32-bit RW registers.

Table 4-1 shows the characteristics of the APB example slave.

Table 4-1 APB example slave characteristics

| Element name | Description |
|--------------|--|
| Filename | cmdsk_apb3_eg_slave.v, for example APB3 slave. cmdsk_apb4_eg_slave.v, for example APB4 slave. |
| Parameters | ADDRWIDTH Width of the APB address bus. The default is 12. |
| Clock domain | PCLK |

In the same way as the AHB example slave, the design of the APB example slaves are partitioned into an interface module and a register module. You can use the interface module to connect most peripheral blocks designed for traditional 8-bit or 16-bit microcontrollers to simplify migration to ARM-based systems.

4.1.1 Programmers model

Table 4-2 shows the APB example slave memory map.

Table 4-2 APB example slave memory map

| Name | Base offset | Type | Width | Reset value | Description |
|--------|-----------------|------|-------|-------------|---|
| DATA0 | 0x0000 | RW | 32 | 0x00000000 | - |
| DATA1 | 0x0004 | RW | 32 | 0x00000000 | - |
| DATA2 | 0x0008 | RW | 32 | 0x00000000 | - |
| DATA3 | 0x000C | RW | 32 | 0x00000000 | - |
| Unused | 0x0010 - 0x0FCF | - | - | - | Read as 0 and write ignored. |
| PID4 | 0xFD0 | RO | 8 | 0x04 | Peripheral ID Register 4: [7:4] Block count. [3:0] jep106_c_code. |
| PID5 | 0xFD4 | RO | 8 | 0x00 | Peripheral ID Register 5. |
| PID6 | 0xFD8 | RO | 8 | 0x00 | Peripheral ID Register 6. |
| PID7 | 0xFDC | RO | 8 | 0x00 | Peripheral ID Register 7. |
| PID0 | 0xFE0 | RO | 8 | 0x18 | APB3 example slave. Peripheral ID Register 0: [7:0] Part number[7:0]. |
| | | | | 0x19 | APB4 example slave. Peripheral ID Register 0: [7:0] Part number[7:0]. |
| PID1 | 0xFE4 | RO | 8 | 0xB8 | Peripheral ID Register 1: [7:4] jep106_id_3_0. [3:0] Part number[11:8]. |

Table 4-2 APB example slave memory map (continued)

| Name | Base offset | Type | Width | Reset value | Description |
|------|-------------|------|-------|-------------|--|
| PID2 | 0xFE8 | RO | 8 | 0x1B | Peripheral ID Register 2: [7:4] Revision. [3] jedec_used. [2:0] jep106_id_6_4. |
| PID3 | 0xFEC | RO | 8 | 0x00 | Peripheral ID Register 3: [7:4] ECO revision number. [3:0] Customer modification number. |
| CID0 | 0xFF0 | RO | 8 | 0x0D | Component ID Register 0. |
| CID1 | 0xFF4 | RO | 8 | 0xF0 | Component ID Register 1. |
| CID2 | 0xFF8 | RO | 8 | 0x05 | Component ID Register 2. |
| CID3 | 0xFFC | RO | 8 | 0xB1 | Component ID Register 3. |

Note

The APB signal **PPROT[2:0]** is not required for the operation of the APB example slaves. Therefore, these signals are not included in the design, and are not shown in [Figure 4-1 on page 4-2](#) and [Figure 4-2 on page 4-2](#).

4.2 APB timer

The APB timer, `cmsdk_apb_timer.v`, is a 32-bit down-counter with the following features:

- You can generate an interrupt request signal, **TIMERINT**, when the counter reaches 0. The interrupt request is held until it is cleared by writing to the **INTCLEAR** Register.
- You can use the zero to one transition of the external input signal, **EXTIN**, as a timer enable.
- If the APB timer count reaches 0 and, at the same time, the software clears a previous interrupt status, the interrupt status is set to 1.
- The external clock, **EXTIN**, must be slower than half of the peripheral clock because it is sampled by a double flip-flop and then goes through edge-detection logic when the external inputs act as a clock. See [Programmers model on page 4-6](#).
- A separate clock pin, **PCLKG**, for the APB register read or write logic that permits the clock to peripheral register logic to stop when there is no APB activity.
- Component ID and Peripheral ID Registers. These read-only ID registers are optional. You must modify the following in these registers:
 - Part number, 12 bits.
 - JEDEC ID value, 7 bits.
- The **ECOREVNUM** input signal is connected to the ECO revision number in Peripheral ID Register 3.

Figure 4-3 shows the APB timer module.

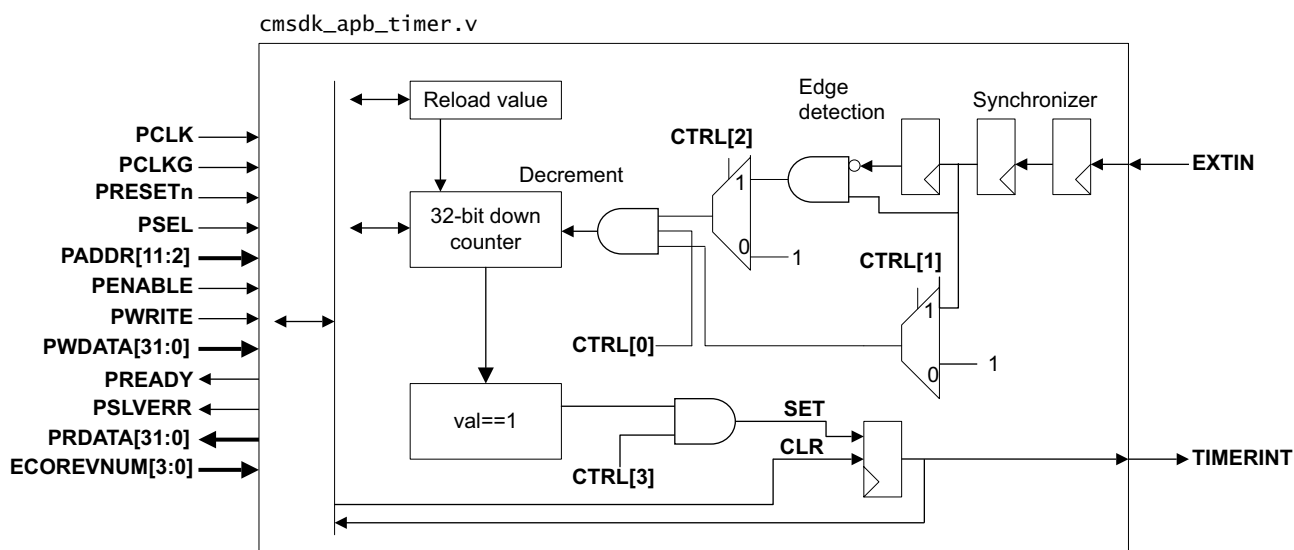


Figure 4-3 APB timer

Table 4-3 shows the characteristics of the APB timer.

Table 4-3 APB timer characteristics

| Element name | Description |
|--------------|--|
| Filename | cmsdk_apb_timer.v. |
| Parameters | None. |
| Clock domain | You can turn-off the gated peripheral bus clock for register access, PCLKG , when there is no APB access. The free running clock, PCLK , is used for timer operation. This must be the same frequency as, and synchronous to, the PCLKG signal. |

4.2.1 Programmers model

Table 4-4 shows the APB timer memory map.

Table 4-4 APB timer memory map

| Name | Base offset | Type | Width | Reset value | Description |
|-----------------------|-------------|------|-------|-------------|---|
| CTRL | 0x000 | RW | 4 | 0x0 | [3] Timer interrupt enable. [2] Select external input as clock. [1] Select external input as enable. [0] Enable. |
| VALUE | 0x004 | RW | 32 | 0x00000000 | [31:0] Current value. |
| RELOAD | 0x008 | RW | 32 | 0x00000000 | [31:0] Reload value. A write to this register sets the current value. |
| INTSTATUS INTCLEAR | 0x00C | RW | 1 | 0x0 | [0] Timer interrupt. Write one to clear. |
| PID4 | 0xFD0 | RO | 8 | 0x04 | Peripheral ID Register 4: [7:4] Block count. [3:0] jep106_c_code. |
| PID5 | 0xFD4 | RO | 8 | 0x00 | Peripheral ID Register 5 |
| PID6 | 0xFD8 | RO | 8 | 0x00 | Peripheral ID Register 6 |
| PID7 | 0xFDC | RO | 8 | 0x00 | Peripheral ID Register 7 |
| PID0 | 0xFE0 | RO | 8 | 0x22 | Peripheral ID Register 0: [7:0] Part number[7:0]. |
| PID1 | 0xFE4 | RO | 8 | 0xB8 | Peripheral ID Register 1: [7:4] jep106_id_3_0. [3:0] Part number[11:8]. |
| PID2 | 0xFE8 | RO | 8 | 0x1B | Peripheral ID Register 2: [7:4] Revision. [3] jedec_used. [2:0] jep106_id_6_4. |
| PID3 | 0xFEC | RO | 8 | 0x00 | Peripheral ID Register 3: [7:4] ECO revision number. [3:0] Customer modification number. |

Table 4-4 APB timer memory map (continued)

| Name | Base offset | Type | Width | Reset value | Description |
|------|-------------|------|-------|-------------|--------------------------|
| CID0 | 0xFF0 | RO | 8 | 0x0D | Component ID Register 0. |
| CID1 | 0xFF4 | RO | 8 | 0xF0 | Component ID Register 1. |
| CID2 | 0xFF8 | RO | 8 | 0x05 | Component ID Register 2. |
| CID3 | 0xFFC | RO | 8 | 0xB1 | Component ID Register 3. |

Note

The APB interface always responds with an OKAY, with no wait states, and is two cycles per transfer, so you can ignore the **PSLVERR** and **PREADY** outputs for APB2 applications.

4.2.2 Signal descriptions

Table 4-5 shows the APB timer signals.

Table 4-5 APB timer signals

| Signal | Direction | Description |
|-----------------|-----------|---|
| EXTIN | Input | External input. This signal is synchronized by double flip-flops before the time logic uses it. |
| TIMERINT | Output | Timer interrupt output. |

4.3 APB UART

The APB UART, `cmsdk_apb_uart.v`, is a simple design that supports 8-bit communication without parity, and is fixed at one stop bit per configuration. Figure 4-4 shows the APB UART module.

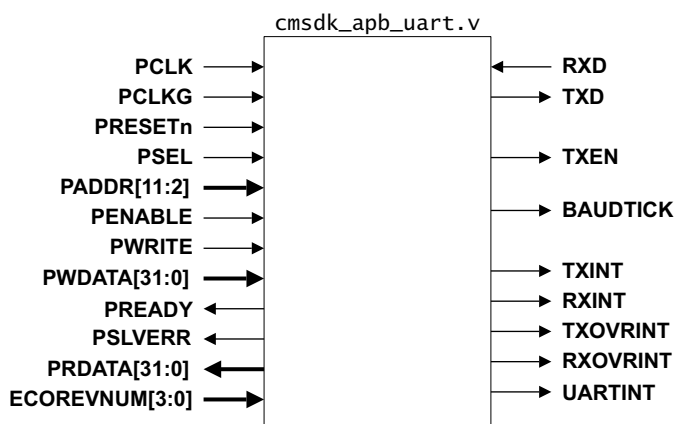


Figure 4-4 APB UART

The APB UART contains buffering. See Figure 4-5.

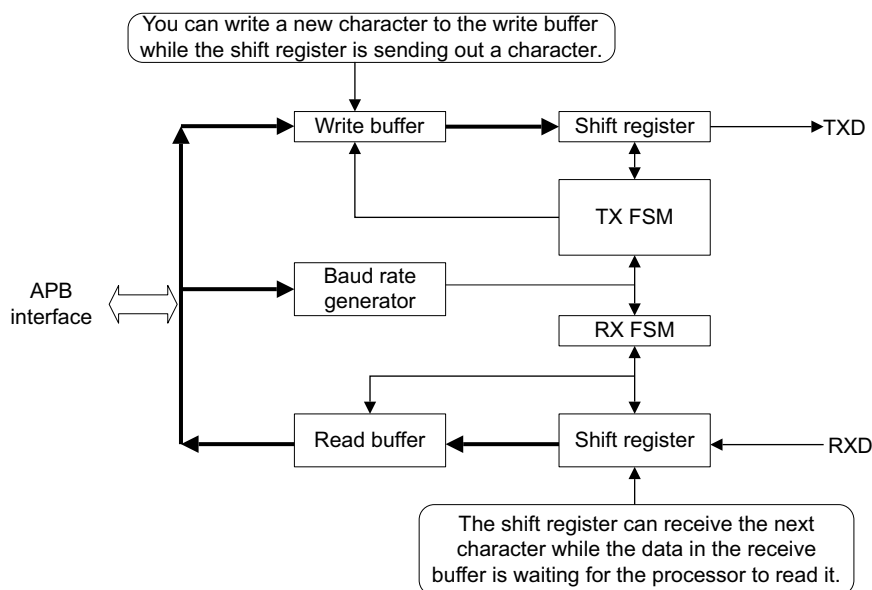


Figure 4-5 APB UART buffering

This buffer arrangement is sufficient for most simple embedded applications. For example, a processor running at 30MHz with a baud rate of 115200 means a character transfer every $30e6 \times (1+8+1)/115200 = 2604$ cycles. For duplex communication, the processor might receive an interrupt every 1300 clock cycles. Because the interrupt response time and the handler execution time are usually quite short, this leaves sufficient processing time for the thread.

The design includes a double flip-flop synchronization logic for the receive data input.

Table 4-6 shows the characteristics of the APB UART.

Table 4-6 APB UART characteristics

| Element name | Description |
|--------------|---|
| Filename | cmsdk_apb_uart.v |
| Parameters | None |
| Clock domain | PCLK for timer operation. This clock is always running. PCLKG for register access. This clock can be gated when the APB interface is idle. When PCLKG is running, it is the same as PCLK . |

4.3.1 Programmers model

Table 4-7 shows the APB UART memory map.

Table 4-7 APB UART memory map

| Name | Base offset | Type | Width | Reset value | Description |
|-----------------------|-------------|------|-------|-------------|---|
| DATA | 0x000 | RW | 8 | 0x-- | [7:0] Read Write Data value. Received data. Transmit data. |
| STATE | 0x004 | RW | 4 | 0x0 | [3] [2] [1] [0] RX buffer overrun, write 1 to clear. TX buffer overrun, write 1 to clear. RX buffer full, read-only. TX buffer full, read-only. |
| CTRL | 0x008 | RW | 7 | 0x00 | [6] [5] [4] [3] [2] [1] [0] High-speed test mode for TX only. RX overrun interrupt enable. TX overrun interrupt enable. RX interrupt enable. TX interrupt enable. RX enable. TX enable. |
| INTSTATUS INTCLEAR | 0x00C | RW | 4 | 0x0 | [3] [2] [1] [0] RX overrun interrupt. Write 1 to clear. TX overrun interrupt. Write 1 to clear. RX interrupt. Write 1 to clear. TX interrupt. Write 1 to clear. |
| BAUDDIV | 0x010 | RW | 20 | 0x00000 | [19:0] Baud rate divider. The minimum number is 16. |
| PID4 | 0xFD0 | RO | 8 | 0x04 | Peripheral ID Register 4: [7:4] [3:0] Block count. jep106_c_code. |
| PID5 ^a | 0xFD4 | RO | 8 | 0x00 | Peripheral ID Register 5. |
| PID6 ^a | 0xFD8 | RO | 8 | 0x00 | Peripheral ID Register 6. |
| PID7 ^a | 0xFDC | RO | 8 | 0x00 | Peripheral ID Register 7. |
| PID0 | 0xFE0 | RO | 8 | 0x21 | Peripheral ID Register 0: [7:0] Part number[7:0]. |

Table 4-7 APB UART memory map (continued)

| Name | Base offset | Type | Width | Reset value | Description |
|------|-------------|------|-------|-------------|--|
| PID1 | 0xFE4 | RO | 8 | 0xB8 | Peripheral ID Register 1: [7:4] jep106_id_3_0. [3:0] Part number[11:8]. |
| PID2 | 0xFE8 | RO | 8 | 0x1B | Peripheral ID Register 2: [7:4] Revision. [3] jedec_used. [2:0] jep106_id_6_4. |
| PID3 | 0xFEC | RO | 8 | 0x00 | Peripheral ID Register 3: [7:4] ECO revision number. [3:0] customer modification number. |
| CID0 | 0xFF0 | RO | 8 | 0x0D | Component ID Register 0. |
| CID1 | 0xFF4 | RO | 8 | 0xF0 | Component ID Register 1. |
| CID2 | 0xFF8 | RO | 8 | 0x05 | Component ID Register 2. |
| CID3 | 0xFFC | RO | 8 | 0xB1 | Component ID Register 3. |

a. The PID5, PID6, and PID7 registers are not used.

The APB UART supports a high-speed test mode, useful for simulation during SoC or ASIC development. When **CTRL[6]** is set to 1, the serial data is transmitted at one bit per clock cycle. This enables you to send text messages in a much shorter simulation time. If required, you can remove this feature for silicon products to reduce the gate count. You can do this by removing bit[6] of the **reg_ctrl** signal in the verilog code. The APB interface always sends an OKAY response with no wait state and is two cycles per transfer.

You must program the baud rate divider register before enabling the UART. For example, if the **PCLK** is running at 12MHz, and the required baud rate is 9600, program the baud rate divider register as $12,000,000/9600 = 1250$.

The **BAUDTICK** output pulses at a frequency of 16 times that of the programmed baud rate. You can use this signal for capturing UART data in a simulation environment.

The **TXEN** output signal indicates the status of **CTRL[0]**. You can use this signal to switch a bidirectional I/O pin in a silicon device to UART data output mode automatically when the UART transmission feature is enabled.

The buffer overrun status in the **STATE** field is used to drive the overrun interrupt signals. Therefore, clearing the buffer overrun status de-asserts the overrun interrupt, and clearing the overrun interrupt bit also clears the buffer overrun status bit in the **STATE** field.

4.4 APB dual-input timers

The APB dual-input timer module, `cmsdk_apb_dualtimers.v`, is an APB slave module consisting of two programmable 32-bit or 16-bit down-counters that can generate interrupts when they reach 0. You can program a timer to implement:

- A 32-bit or a 16-bit counter.
- One of the following timer modes:
 - Free-running.
 - Periodic.
 - One-shot.

The operation of each timer module is identical. See [Functional description on page 4-12](#) for a description of each of the timer modes.

The dual-input timer module provides access to two interrupt-generating, programmable 32-bit *Free-Running Counters* (FRCs). The FRCs operate from a common timer clock, **TIMCLK**, and each FRC has its own clock enable input, **TIMCLKEN1** and **TIMCLKEN2**. Each FRC also has a prescaler that can divide down the enabled **TIMCLK** rate by 1, 16, or 256. This enables the count rate for each FRC to be controlled independently using their individual clock enables and prescalers.

The system clock, **PCLK**, controls the programmable registers, and the timer clock, **TIMCLK**, drives the counter, enabling the counters to run from a much slower clock than the system clock.

[Figure 4-6](#) shows the dual-input timer module.

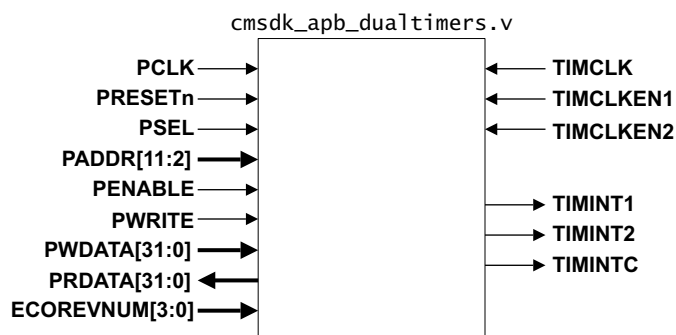


Figure 4-6 APB dual-input timers

[Table 4-8](#) shows the characteristics of the APB dual-input timer.

Table 4-8 APB dual-input timer characteristics

| Element name | Description |
|--------------|-------------------------------------|
| Filename | <code>cmsdk_apb_dualtimers.v</code> |
| Parameters | None |
| Clock domain | PCLK |

4.4.1 Functional description

Two timers are defined by default, although you can easily expand this using extra instantiations of the FRC block. The same principle of simple expansion is used for the register configuration, to enable you to use more complex counters. For each timer, the following modes of operation are available:

Free-running mode

The counter wraps after reaching its zero value, and continues to count down from the maximum value. This is the default mode.

Periodic timer mode

The counter generates an interrupt at a constant interval, reloading the original value after wrapping past zero.

One-shot timer mode

The counter generates an interrupt once. When the counter reaches 0, it halts until you reprogram it. You can do this using one of the following:

- Clearing the one-shot count bit in the control register, in which case the count proceeds according to the selection of Free-running or Periodic mode.
- Writing a new value to the Load Value register.

4.4.2 Operation

Each timer has an identical set of registers as [Table 4-9 on page 4-14](#) shows. The operation of each timer is identical. The timer is loaded by writing to the load register and, if enabled, counts down to 0. When a counter is already running, writing to the load register causes the counter to immediately restart at the new value. Writing to the background load value has no effect on the current count. The counter continues to decrement to 0, and then restarts from the new load value, if in periodic mode, and one-shot mode is not selected.

When 0 is reached, an interrupt is generated. You can clear the interrupt by writing to the clear register. If you selected one-shot mode, the counter halts when it reaches 0 until you deselect one-shot mode, or write a new load value.

Otherwise, after reaching a zero count, if the timer is operating in free-running mode, it continues to decrement from its maximum value. If you selected periodic timer mode, the timer reloads the count value from the Load Register and continues to decrement. In this mode, the counter effectively generates a periodic interrupt.

You select the mode using a bit in the Timer Control Register. See [Table 4-10 on page 4-16](#). At any point, you can read the current counter value from the Current Value Register. You can enable the counter using a bit in the Control Register.

At reset, the counter is disabled, the interrupt is cleared, and the load register is set to 0. The mode and prescale values are set to free-running, and clock divide of one respectively. [Figure 4-7 on page 4-13](#) shows a block diagram of the free-running timer module.

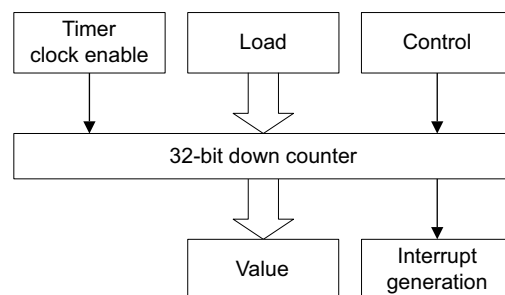


Figure 4-7 Free-running timer block

The timer clock enable is generated by a prescale unit. The counter then uses the enable to create a clock with one of the following timings:

- The system clock.
- The system clock divided by 16, generated by 4 bits of prescale.
- The system clock divided by 256, generated by 8 bits of prescale.

Figure 4-8 shows how the timer clock frequency is selected in the prescale unit. This enables you to clock the timer at different frequencies.

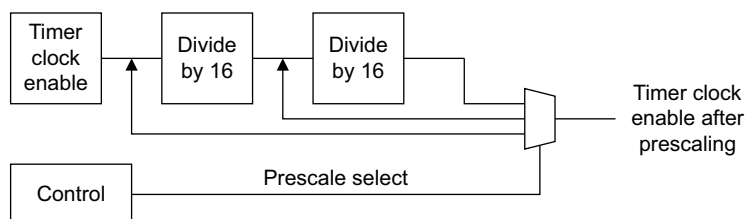


Figure 4-8 Prescale clock enable generation

———— Note ————

This selection is in addition to any similar facility already provided as part of any clock generation logic external to the timers.

Interrupt generation

An interrupt is generated when the full 32-bit counter reaches 0, and is only cleared when the `TimerXClear` location is written to. A register holds the value until the interrupt is cleared. The most significant carry bit of the counter detects the counter reaching 0.

You can mask interrupts by writing 0 to the Interrupt Enable bit in the Control Register. You can read the following from status registers:

- Raw interrupt status, before masking.
- Final interrupt status, after masking.

The interrupts from the individual counters, after masking, are logically ORed into a combined interrupt, **TIMINTC**. This provides an additional output from the timer peripheral.

4.4.3 Clocking

The timers contain the **PCLK** and **TIMCLK** clock inputs. **PCLK** is the main APB system clock, and is used by the register interface. **TIMCLK** is the input to the prescale units and the decrementing counters. You must qualify a pulse on **TIMCLK** by the appropriate **TIMCLKENx** signal being HIGH.

The design of the timers assumes that **PCLK** and **TIMCLK** are synchronous. To enable the counter to operate from a lower effective frequency than that at which **PCLK** is running, you can:

- Connect both **PCLK** and **TIMCLK** inputs to the APB **PCLK** signal, and pulse **TIMCLKENx** HIGH at the required frequency, synchronized to **PCLK**.
- Tie **TIMCLKENx** HIGH and feed an enabled version of **PCLK** into the **TIMCLK** input. This provides sparse clock pulses synchronous to **PCLK**.

This provision of two clock inputs enables the counters to continue to run while the APB system is in a sleep state when **PCLK** is disabled. External system control logic must handle the changeover periods when **PCLK** is disabled and enabled to ensure that the **PCLK** and **TIMCLK** inputs are fed with synchronous signals when any register access is to occur.

4.4.4 Programmers model

Table 4-9 shows the timer registers.

Table 4-9 Timer memory map

| Name | Base offset | Type | Width | Reset value | Description |
|-----------------------------|-------------|------|-------|-------------|---|
| TIMER1LOAD | 0x00 | RW | 32 | 0x00000000 | See <i>Load Register</i> on page 4-15. |
| TIMER1VALUE | 0x04 | RO | 32 | 0xFFFFFFFF | See <i>Current Value Register</i> on page 4-15. |
| TIMER1CONTROL | 0x08 | RW | 8 | 0x20 | See <i>Timer Control Register</i> on page 4-16. |
| TIMER1INTCLR | 0x0C | WO | - | - | See <i>Interrupt Clear Register</i> on page 4-16. |
| TIMER1RIS | 0x10 | RO | 1 | 0x0 | See <i>Raw Interrupt Status Register</i> on page 4-17. |
| TIMER1MIS | 0x14 | RO | 1 | 0x0 | See <i>Interrupt Status Register</i> on page 4-17. |
| TIMER1BGLOAD | 0x18 | RW | 32 | 0x00000000 | See <i>Background Load Register</i> on page 4-17. |
| TIMER2LOAD | 0x20 | RW | 32 | 0x00000000 | See <i>Load Register</i> on page 4-15. |
| TIMER2VALUE | 0x24 | RO | 32 | 0xFFFFFFFF | See <i>Current Value Register</i> on page 4-15. |
| TIMER2CONTROL | 0x28 | RW | 8 | 0x20 | See <i>Timer Control Register</i> on page 4-16. |
| TIMER2INTCLR | 0x2C | WO | - | - | See <i>Interrupt Clear Register</i> on page 4-16. |
| TIMER2RIS | 0x30 | RO | 1 | 0x0 | See <i>Raw Interrupt Status Register</i> on page 4-17. |
| TIMER2MIS | 0x34 | RO | 1 | 0x0 | See <i>Interrupt Status Register</i> on page 4-17. |
| TIMER2BGLOAD | 0x38 | RW | 32 | 0x00000000 | See <i>Background Load Register</i> on page 4-17. |
| TIMERITCR | 0xF00 | RW | 1 | 0x0 | See <i>Integration Test Control Register</i> on page 4-18. |
| TIMERITOP | 0xF04 | WO | 2 | 0x0 | See <i>Integration Test Output Set Register</i> on page 4-18. |
| TIMERPERIPHID4 | 0xFD0 | RO | 8 | 0x04 | Peripheral ID Register 4: [7:4] Block count. [3:0] jep106_c_code. |
| TIMERPERIPHID5 ^a | 0xFD4 | RO | 8 | 0x00 | Peripheral ID Register 5. |
| TIMERPERIPHID6 ^a | 0xFD8 | RO | 8 | 0x00 | Peripheral ID Register 6. |
| TIMERPERIPHID7 ^a | 0xFDC | RO | 8 | 0x00 | Peripheral ID Register 7. |

Table 4-9 Timer memory map (continued)

| Name | Base offset | Type | Width | Reset value | Description |
|----------------|-------------|------|-------|-------------|--|
| TIMERPERIPHID0 | 0xFE0 | RO | 8 | 0x23 | Peripheral ID Register 0: [7:0] Part number[7:0]. |
| TIMERPERIPHID1 | 0xFE4 | RO | 8 | 0xB8 | Peripheral ID Register 1: [7:4] jep106_id_3_0. [3:0] Part number[11:8]. |
| TIMERPERIPHID2 | 0xFE8 | RO | 8 | 0x1B | Peripheral ID Register 2: [7:4] Revision. [3] jedec_used. [2:0] jep106_id_6_4. |
| TIMERPERIPHID3 | 0xFEC | RO | 8 | 0x00 | Peripheral ID Register 3: [7:4] ECO revision number. [3:0] customer modification number. |
| TIMERPCELLID0 | 0xFF0 | RO | 8 | 0x0D | Component ID Register 0. |
| TIMERPCELLID1 | 0xFF4 | RO | 8 | 0xF0 | Component ID Register 1. |
| TIMERPCELLID2 | 0xFF8 | RO | 8 | 0x05 | Component ID Register 2. |
| TIMERPCELLID3 | 0xFFC | RO | 8 | 0xB1 | Component ID Register 3. |

a. The TIMERPERIPHID5, TIMERPERIPHID6, and TIMERPERIPHID7 registers are not used.

Load Register

The TIMERXLOAD Register contains the value from which the counter is to decrement. This is the value used to reload the counter when Periodic mode is enabled, and the current count reaches 0.

When this register is written to directly, the current count is immediately reset to the new value at the next rising edge of **TIMCLK** that **TIMCLKEN** enables.

The value in this register is also overwritten if the TIMERXBGLOAD Register is written to, but the current count is not immediately affected.

If values are written to both the TIMERXLOAD and TIMERXBGLOAD registers before an enabled rising edge on **TIMCLK**, the following occurs:

1. On the next enabled **TIMCLK** edge, the value written to the TIMERXLOAD value replaces the current count value.
2. Every time the counter reaches 0, the current count value is reset to the value written to TIMERXBGLOAD.

Reading from the TIMERXLOAD Register at any time after the two writes have occurred retrieves the value written to TIMERXBGLOAD. That is, the value read from TIMERXLOAD is always the value that takes effect for Periodic mode after the next time the counter reaches 0.

Current Value Register

The TIMERXVALUE Register provides the current value of the decrementing counter.

Timer Control Register

The TIMERXCONTROL Register is a read or write register. Figure 4-9 shows the register bit assignments.

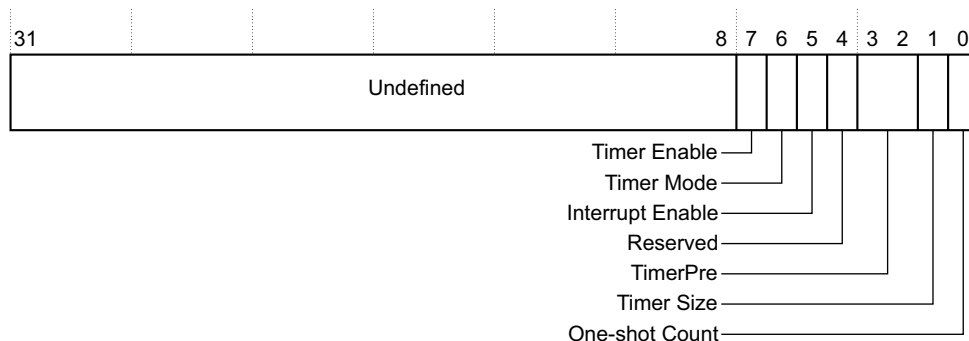


Figure 4-9 TIMERXCONTROL Register bit assignments

Table 4-10 shows the register bit assignments.

Table 4-10 TIMERXCONTROL Register bit assignments

| Bits | Name | Function |
|--------|------------------|--|
| [31:8] | - | Reserved, read undefined, must read as 0s. |
| [7] | Timer Enable | Enable bit: 0 Timer disabled, default. 1 Timer enabled. |
| [6] | Timer Mode | Mode bit: 0 Timer is in free-running mode, default. 1 Timer is in periodic mode. |
| [5] | Interrupt Enable | Interrupt Enable bit: 0 Timer Interrupt disabled. 1 Timer Interrupt enabled, default. |
| [4] | Reserved | Reserved bit, do not modify, and ignore on read |
| [3:2] | TimerPre | Prescale bits: 00 0 stages of prescale, clock is divided by 1, default. 01 4 stages of prescale, clock is divided by 16. 10 8 stages of prescale, clock is divided by 256. 11 Undefined, do not use. |
| [1] | Timer Size | Selects 16-bit or 32-bit counter operation: 0 16-bit counter, default. 1 32-bit counter. |
| [0] | One-shot Count | Selects one-shot or wrapping counter mode: 0 Wrapping mode, default. 1 One-shot mode. |

Interrupt Clear Register

Any write to the TIMERXINTCLR Register clears the interrupt output from the counter.

Raw Interrupt Status Register

The TIMERXRIS Register indicates the raw interrupt status from the counter. This value is ANDed with the timer interrupt enable bit from the Timer Control Register to create the masked interrupt, that is passed to the interrupt output pin. Figure 4-10 shows the register bit assignments.

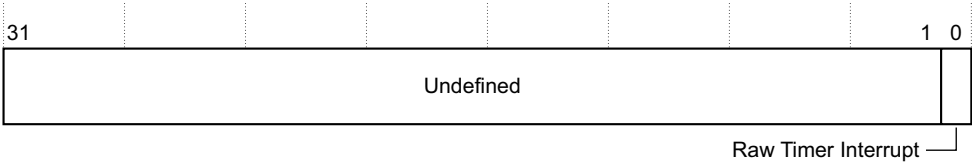


Figure 4-10 TIMERXRIS Register bit assignments

Table 4-11 shows the register bit assignments.

Table 4-11 TIMERXRIS Register bit assignments

| Bits | Name | Function |
|--------|---------------------|---|
| [31:1] | - | Reserved, read undefined, must read as 0s |
| [0] | Raw Timer Interrupt | Raw interrupt status from the counter |

Interrupt Status Register

The TIMERXMIS Register indicates the masked interrupt status from the counter. This value is the logical AND of the raw interrupt status with the timer interrupt enable bit from the Timer Control Register, and is the same value that is passed to the interrupt output pin. Figure 4-11 shows the register bit assignments.

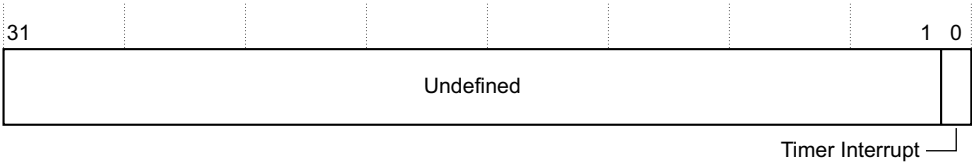


Figure 4-11 TIMERXMIS Register bit assignments

Table 4-12 shows the register bit assignments.

Table 4-12 TIMERXMIS Register bit assignments

| Bits | Name | Function |
|--------|-----------------|---|
| [31:1] | - | Reserved, read as 0 |
| [0] | Timer Interrupt | Enabled interrupt status from the counter |

Background Load Register

The TIMERXBGLOAD Register contains the value from which the counter is to decrement. This is the value used to reload the counter when Periodic mode is enabled, and the current count reaches 0.

This register provides an alternative method of accessing the **TIMERXLOAD** Register. The difference is that writes to **TIMERXBGLOAD** do not cause the counter to immediately restart from the new value.

Reading from this register returns the same value returned from **TIMERXLOAD**. See [Load Register on page 4-15](#).

Integration Test Control Register

The **TIMERITCR** Register enables integration test mode. When in this mode, the Integration Test Output Set Register directly controls the masked interrupt outputs. The combined interrupt output, **TIMINTC**, then becomes the logical OR of the bits set in the Integration Test Output Set Register. [Figure 4-12](#) shows the register bit assignments.

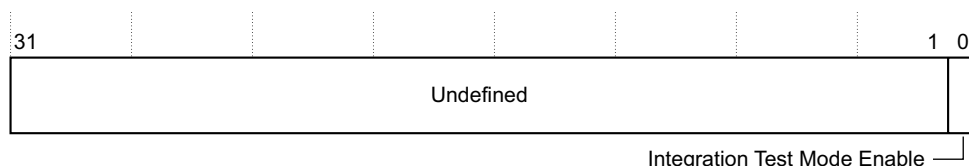


Figure 4-12 **TIMERITCR** Register bit assignments

[Table 4-13](#) shows the register bit assignments.

Table 4-13 **TIMERITCR** Register bit assignments

| Bits | Name | Function |
|--------|------------------------------|---|
| [31:1] | - | Reserved, read as 0 |
| [0] | Integration Test Mode Enable | When set HIGH, places the timers into integration test mode |

Integration Test Output Set Register

When in integration test mode, the values in this write-only register, **TIMERITOP**, directly drive the enabled interrupt outputs. [Figure 4-13](#) shows the register bit assignments.

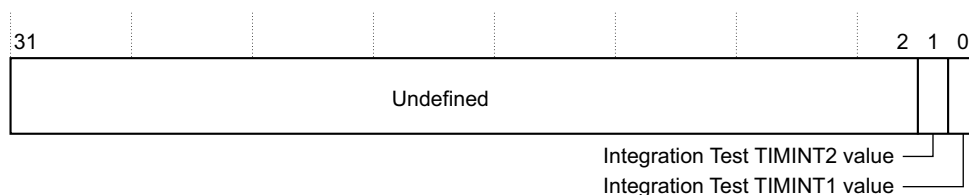


Figure 4-13 **TIMERITOP** Register bit assignments

[Table 4-14](#) shows the register bit assignments.

Table 4-14 **TIMERITOP** Register bit assignments

| Bits | Name | Function |
|--------|---------------------------------------|--|
| [31:2] | - | Reserved, read undefined, must read as 0s |
| [1] | Integration Test TIMINT2 value | Value output on TIMINT2 when in Integration Test Mode |
| [0] | Integration Test TIMINT1 value | Value output on TIMINT1 when in Integration Test Mode |

4.4.5 Signal descriptions

Table 4-15 shows the non-AMBA signals that the timer uses.

Table 4-15 Timer signals

| Signal | Type | Direction | Description |
|------------------|----------------------------|-----------|--|
| TIMCLK | Timer clock | Input | Timer clock input. This must be synchronous to PCLK for normal operation. |
| TIMCLKEN1 | Timer 1 clock enable | Input | Enable for timer 1 clock input. The counter only decrements on a rising edge of TIMCLK when TIMCLKEN1 is HIGH. |
| TIMCLKEN2 | Timer 2 clock enable | Input | Enable for timer 2 clock input. The counter only decrements on a rising edge of TIMCLK when TIMCLKEN2 is HIGH. |
| TIMINT1 | Counter 1 interrupt | Output | Active HIGH interrupt signal to the interrupt controller module. Indicates that counter 1 generated an interrupt after being decremented to 0. |
| TIMINT2 | Counter 2 interrupt | Output | Active HIGH interrupt signal to the interrupt controller module. Indicates that counter 2 generated an interrupt after being decremented to 0. |
| TIMINTC | Combined counter interrupt | Output | Active HIGH interrupt signal to the interrupt controller module. This signal indicates that one of the counters generated an interrupt having been decremented to 0, and is the logical OR of TIMINT1 and TIMINT2 . |
| ECOREVNUM | ECO revision information | Input | This input is connected to the ECO revision number in the Peripheral ID Register 3 to show revision changes during ECO of the chip design process. You can tie this signal LOW, or connect it to special tie-off cells so that you can change the ECO revision number at silicon netlists, or at a lower-level such as silicon mask. |

4.5 APB watchdog

The APB watchdog module, `cmsdk_apb_watchdog.v`, is based on a 32-bit down-counter that is initialized from the Reload Register, `WDOGLOAD`. The watchdog module generates a regular interrupt, **WDOGINT**, depending on a programmed value. The counter decrements by one on each positive clock edge of **WDOGCLK** when the clock enable, **WDOGCLKEN**, is HIGH.

The watchdog monitors the interrupt and asserts a reset request **WDOGRES** signal when the counter reaches 0, and the counter is stopped. On the next enabled **WDOGCLK** clock edge, the counter is reloaded from the `WDOGLOAD` Register and the countdown sequence continues. If the interrupt is not cleared by the time the counter next reaches 0, the watchdog module reasserts the reset signal.

The watchdog module applies a reset to a system in the event of a software failure, providing a way to recover from software crashes. You can enable or disable the watchdog unit as required. Figure 4-14 shows the APB watchdog module.

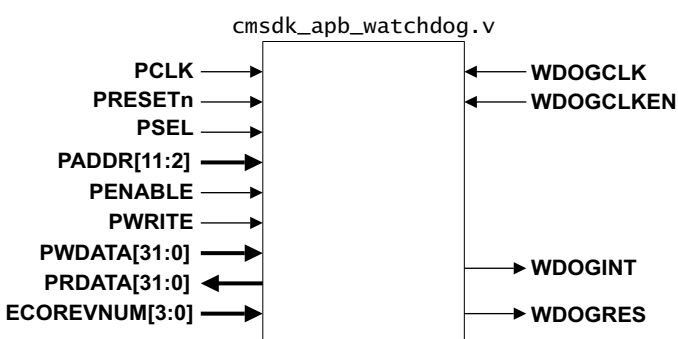


Figure 4-14 APB watchdog

Table 4-16 shows the characteristics of the APB watchdog.

Table 4-16 APB watchdog characteristics

| Element name | Description |
|--------------|-----------------------------------|
| Filename | <code>cmsdk_apb_watchdog.v</code> |
| Parameters | None |
| Clock domain | PCLK |

Figure 4-15 shows the flow diagram for the watchdog operation.

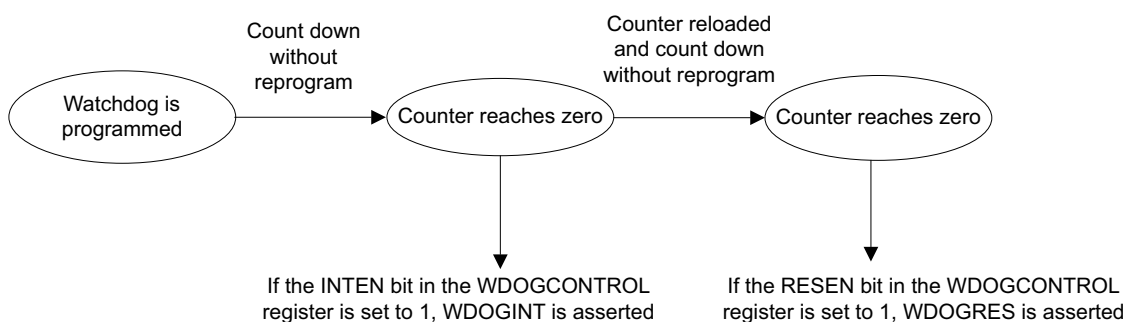


Figure 4-15 Watchdog operation flow diagram

4.5.1 Programmers model

Table 4-17 shows the watchdog registers.

Table 4-17 Watchdog memory map

| Name | Base offset | Type | Width | Reset value | Description |
|----------------------------|-------------|------|-------|-------------|--|
| WDOGLOAD | 0x00 | RW | 32 | 0xFFFFFFFF | See Watchdog Load Register on page 4-22. |
| WDOGVALUE | 0x04 | RO | 32 | 0xFFFFFFFF | See Watchdog Value Register on page 4-22. |
| WDOGCONTROL | 0x08 | RW | 2 | 0x0 | See Watchdog Control Register on page 4-22. |
| WDOGINTCLR | 0x0C | WO | - | - | See Watchdog Clear Interrupt Register on page 4-22. |
| WDOGRIS | 0x10 | RO | 1 | 0x0 | See Watchdog Raw Interrupt Status Register on page 4-22. |
| WDOGMIS | 0x14 | RO | 1 | 0x0 | See Watchdog Interrupt Status Register on page 4-23. |
| WDOGLOCK | 0xC00 | RW | 32 | 0x0 | See Watchdog Lock Register on page 4-23. |
| WDOGITCR | 0xF00 | RW | 1 | 0x0 | See Watchdog Integration Test Control Register on page 4-24. |
| WDOGITOP | 0xF04 | WO | 2 | 0x0 | See Watchdog Integration Test Output Set Register on page 4-24. |
| WDOGPERIPHID4 | 0xFD0 | RO | 8 | 0x04 | Peripheral ID Register 4: [7:4] Block count. [3:0] jep106_c_code. |
| WDOGPERIPHID5 ^a | 0xFD4 | RO | 8 | 0x00 | Peripheral ID Register 5. |
| WDOGPERIPHID6 ^a | 0xFD8 | RO | 8 | 0x00 | Peripheral ID Register 6. |
| WDOGPERIPHID7 ^a | 0xFDC | RO | 8 | 0x00 | Peripheral ID Register 7. |
| WDOGPERIPHID0 | 0xFE0 | RO | 8 | 0x24 | Peripheral ID Register 0: [7:0] Part number[7:0]. |
| WDOGPERIPHID1 | 0xFE4 | RO | 8 | 0xB8 | Peripheral ID Register 1: [7:4] jep106_id_3_0. [3:0] Part number[11:8]. |
| WDOGPERIPHID2 | 0xFE8 | RO | 8 | 0x1B | Peripheral ID Register 2: [7:4] Revision. [3] jedec_used. [2:0] jep106_id_6_4. |
| WDOGPERIPHID3 | 0xFEC | RO | 8 | 0x00 | Peripheral ID Register 3: [7:4] ECO revision number. [3:0] Customer modification number. |
| WDOGPCELLID0 | 0xFF0 | RO | 8 | 0x0D | Component ID Register 0. |
| WDOGPCELLID1 | 0xFF4 | RO | 8 | 0xF0 | Component ID Register 1. |
| WDOGPCELLID2 | 0xFF8 | RO | 8 | 0x05 | Component ID Register 2. |
| WDOGPCELLID3 | 0xFFC | RO | 8 | 0xB1 | Component ID Register 3. |

a. The WDOGPERIPHID5, WDOGPERIPHID6, and WDOGPERIPHID7 registers are not used.

Watchdog Load Register

The WDOGLOAD Register contains the value from which the counter is to decrement. When this register is written to, the count is immediately restarted from the new value. The minimum valid value for WDOGLOAD is 1.

Watchdog Value Register

The WDOGVALUE Register gives the current value of the decrementing counter.

Watchdog Control Register

The WDOGCONTROL Register enables the software to control the watchdog unit. [Figure 4-16](#) shows the register bit assignments.

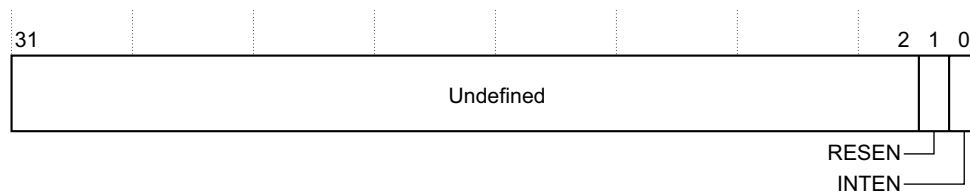


Figure 4-16 WDOGCONTROL Register bit assignments

[Table 4-18](#) shows the register bit assignments.

Table 4-18 WDOGCONTROL Register bit assignments

| Bits | Name | Function |
|--------|-------|--|
| [31:2] | - | Reserved, read undefined, must read as 0s. |
| [1] | RESEN | Enable watchdog reset output, WDOGRES . Acts as a mask for the reset output. Set HIGH to enable the reset, or LOW to disable the reset. |
| [0] | INTEN | Enable the interrupt event, WDOGINT . Set HIGH to enable the counter and the interrupt, or LOW to disable the counter and interrupt. Reloads the counter from the value in WDOGLOAD when the interrupt is enabled, after previously being disabled. |

Watchdog Clear Interrupt Register

A write of any value to the WDOGINTCLR Register clears the watchdog interrupt, and reloads the counter from the value in WDOGLOAD.

Watchdog Raw Interrupt Status Register

The WDOGRIS Register indicates the raw interrupt status from the counter. This value is ANDed with the interrupt enable bit from the control register to create the masked interrupt, that is passed to the interrupt output pin. [Figure 4-17](#) shows the register bit assignments.

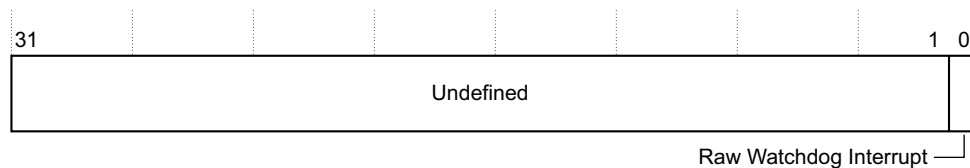


Figure 4-17 WDOGRIS Register bit assignments

Table 4-19 shows the register bit assignments.

Table 4-19 WDOGRIS Register bit assignments

| Bits | Name | Function |
|--------|------------------------|---|
| [31:1] | - | Reserved, read undefined, must read as 0s |
| [0] | Raw Watchdog Interrupt | Raw interrupt status from the counter |

Watchdog Interrupt Status Register

The WDOGMIS Register indicates the masked interrupt status from the counter. This value is the logical AND of the raw interrupt status with the INTEN bit from the control register, and is the same value that is passed to the interrupt output pin. Figure 4-18 shows the register bit assignments.

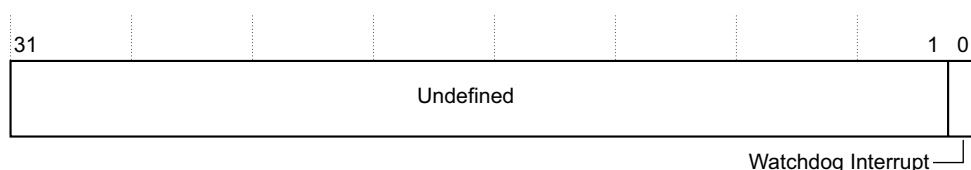


Figure 4-18 WDOGMIS Register bit assignments

Table 4-20 shows the register bit assignments.

Table 4-20 WDOGMIS Register bit assignments

| Bits | Name | Function |
|--------|--------------------|---|
| [31:1] | - | Reserved, read undefined, must read as 0s |
| [0] | Watchdog Interrupt | Enabled interrupt status from the counter |

Watchdog Lock Register

The WDOGLOCK Register disables write accesses to all other registers. This is to prevent rogue software from disabling the watchdog functionality. Writing a value of 0x1ACCE551 enables write access to all other registers. Writing any other value disables write accesses. A read from this register returns only the bottom bit:

- 0** Indicates that write access is enabled, not locked.
- 1** Indicates that write access is disabled, locked.

Figure 4-19 shows the register bit assignments.

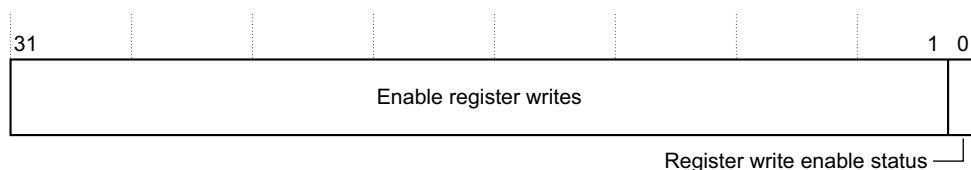


Figure 4-19 WDOGLOCK Register bit assignments

Table 4-21 shows the register bit assignments.

Table 4-21 WDOGLOCK Register bit assignments

| Bits | Name | Function |
|--------|------------------------------|--|
| [31:1] | Enable register writes | Enable write access to all other registers by writing 0x1ACCE551. Disable write access by writing any other value. |
| [0] | Register write enable status | 0 Write access to all other registers is enabled. This is the default. |
| | | 1 Write access to all other registers is disabled. |

Watchdog Integration Test Control Register

The WDOGITCR Register enables integration test mode. When in this mode, the test output register directly controls the masked interrupt output, **WDOGINT**, and reset output, **WDOGRES**. Figure 4-20 shows the register bit assignments.

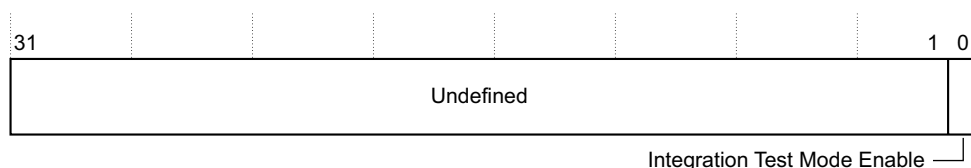


Figure 4-20 WDOGITCR Register bit assignments

Table 4-22 shows the register bit assignments.

Table 4-22 WDOGITCR Register bit assignments

| Bits | Name | Function |
|--------|------------------------------|---|
| [31:1] | - | Reserved, read undefined, must read as 0s |
| [0] | Integration Test Mode Enable | When set HIGH, places the watchdog into integration test mode |

Watchdog Integration Test Output Set Register

When the WDOGITOP Register is in integration test mode, the values in this register directly drive the enabled interrupt output and reset output. Figure 4-21 shows the register bit assignments.

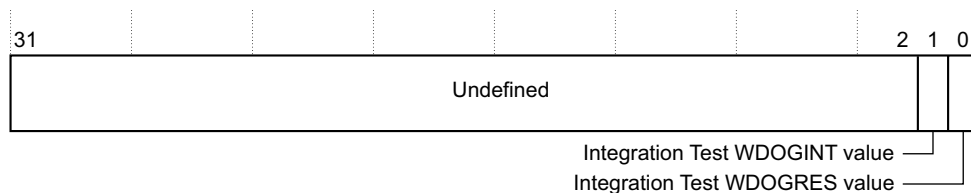


Figure 4-21 WDOGITOP Register bit assignments

Table 4-23 shows the register bit assignments.

Table 4-23 WDOGITOP Register bit assignments

| Bits | Name | Function |
|--------|--|---|
| [31:2] | - | Reserved, read undefined, must read as 0s |
| [1] | Integration Test WDOGIN T value | Value output on WDOGIN T when in Integration Test Mode |
| [0] | Integration Test WDOGRES value | Value output on WDOGRES when in Integration Test Mode |

4.5.2 Signal descriptions

Table 4-24 shows the non-AMBA signals that the watchdog unit uses.

Table 4-24 Watchdog unit signals

| Signal | Type | Direction | Description |
|------------------|--------------------------|-----------|---|
| WDOGCLK | Watchdog clock | Input | The watchdog clock must be synchronous to the APB clock PCLK . |
| WDOGCLKEN | Watchdog clock enable | Input | The enable for the watchdog clock input. The counters only decrement on a rising edge of WDOGCLK when WDOGCLKEN is HIGH. |
| WDOGRESn | Watchdog reset | Input | The watchdog clock domain reset input. |
| WDOGIN T | Watchdog interrupt | Output | The watchdog interrupt. |
| WDOGRES | Watchdog reset | Output | The watchdog timeout reset. |
| ECOREVNUM | ECO revision information | Input | This input is connected to the ECO revision number in the Peripheral ID Register 3 to show revision changes during ECO of the chip design process. You can tie this signal LOW, or connect it to special tie-off cells so that you can change the ECO revision number at silicon netlists, or a lower-level such as the silicon mask. |

4.6 APB slave multiplexer

The APB slave multiplexer, `cmsdk_apb_slave_mux.v`, supports up to 16 APB slaves. It uses four bits of **PADDR** to generate the **PSEL** and handle the data and response multiplexing. You can configure the four bits of **PADDR** that perform decoding, and drive them into **DECODE4BIT[3:0]**. The APB slave multiplexer supports various APB device footprints. Figure 4-22 shows the APB slave multiplexer module.

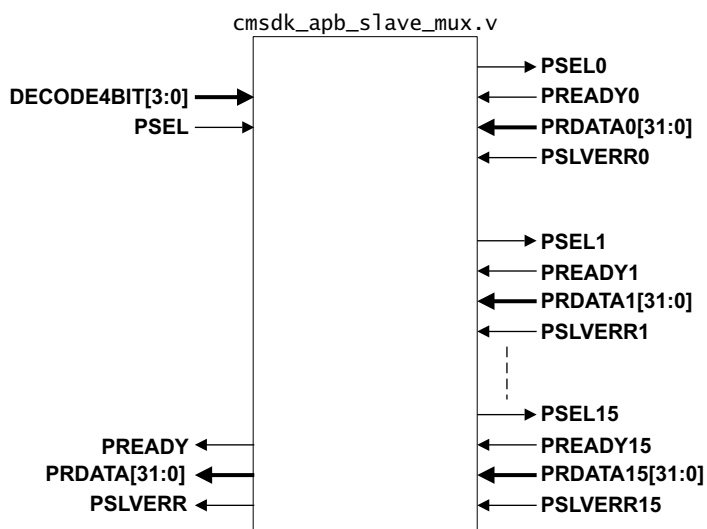


Figure 4-22 APB slave multiplexer

Table 4-25 shows the characteristics of the APB slave multiplexer.

Table 4-25 APB slave multiplexer characteristics

| Element name | Description |
|--------------|--|
| Filename | <code>cmsdk_apb_slave_mux.v</code> |
| Parameters | <div> <div>PORT0_ENABLE</div> <div>The supported parameter values are:</div> <div> <div>0</div> <div>Disable port 0.</div> </div> <div> <div>1</div> <div>Enable port 0.</div> </div> </div> <div> <div>PORT1_ENABLE</div> <div>The supported parameter values are:</div> <div> <div>0</div> <div>Disable port 1.</div> </div> <div> <div>1</div> <div>Enable port 1.</div> </div> </div> <div> <div>PORT2_ENABLE</div> <div>The supported parameter values are:</div> <div> <div>0</div> <div>Disable port 2.</div> </div> <div> <div>1</div> <div>Enable port 2.</div> </div> </div> <div> <div>.</div> <div>.</div> <div>.</div> </div> <div> <div>PORT15_ENABLE</div> <div>The supported parameter values are:</div> <div> <div>0</div> <div>Disable port 15.</div> </div> <div> <div>1</div> <div>Enable port 15.</div> </div> </div> <div>By default, all port enable parameters are set to 1, that is, enabled.</div> |
| Clock domain | No clock input. Combinational logic only. Runs in the PCLK domain. |

4.7 APB subsystem

The APB subsystem, `cmsdk_apb_subsystem.v`, is a common platform for all example systems in the Cortex-M System Design Kit. It contains the following:

- APB timers.
- APB UART.
- Dual-input timer, watchdog.
- AHB to APB bridge.
- Test slave.
- IRQ synchronizers.

Figure 4-23 shows the APB subsystem module.

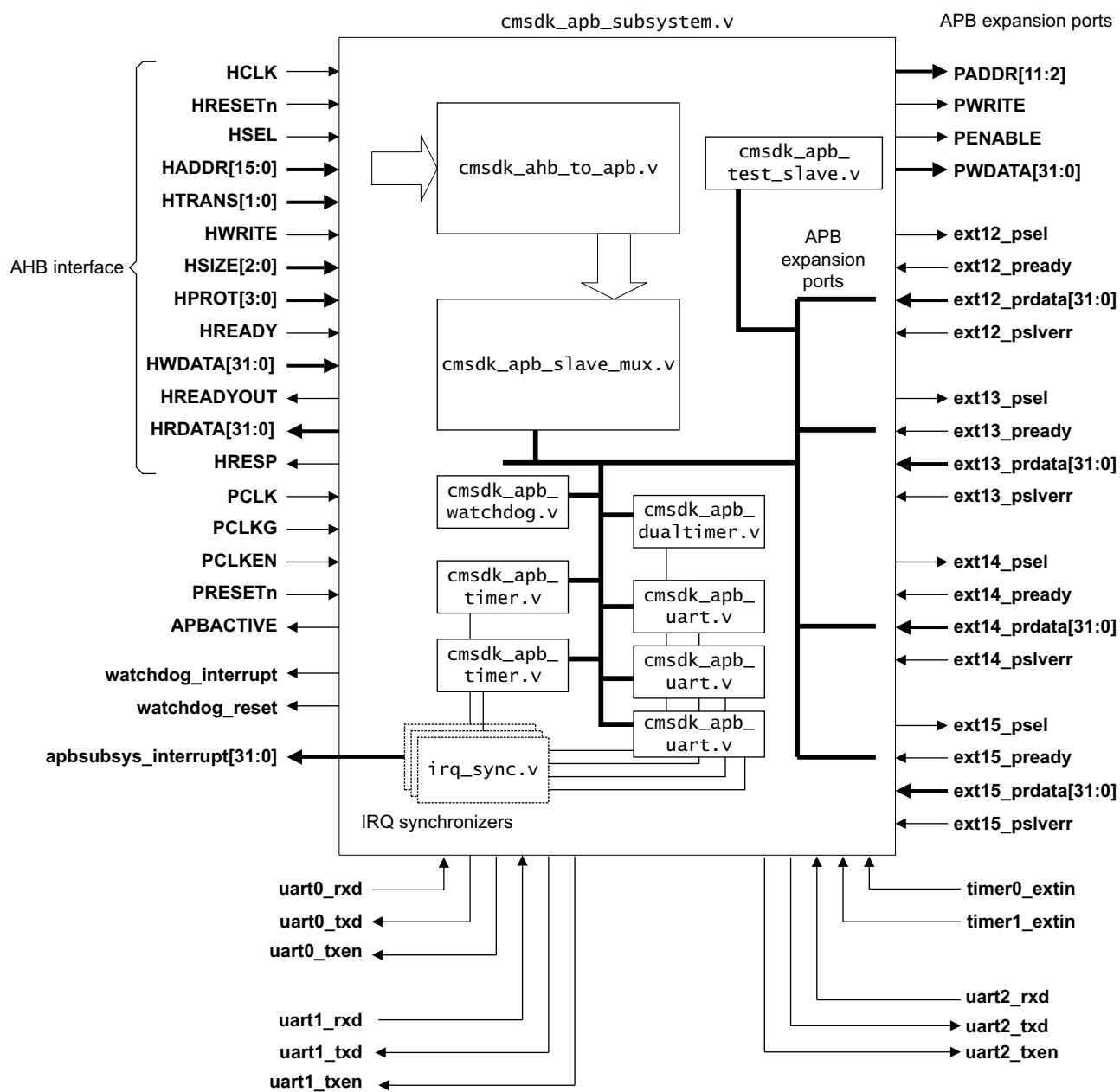


Figure 4-23 APB subsystem

Table 4-26 shows the characteristics of the APB subsystem.

Table 4-26 APB subsystem characteristics

| Element name | Description |
|--------------|--|
| Filename | cmsdk_apb_subsystem.v |
| Parameter | <p>APB_EXT_PORT12_ENABLE Enable APB expansion port 12.</p> <p>APB_EXT_PORT13_ENABLE Enable APB expansion port 13.</p> <p>APB_EXT_PORT14_ENABLE Enable APB expansion port 14.</p> <p>APB_EXT_PORT15_ENABLE Enable APB expansion port 15.</p> <hr/> <p>Note</p> <ul style="list-style-type: none"> Ports 12-15 are to connect APB3 slaves. Ports 0-11 are peripherals inside the APB subsystem. <hr/> <p>INCLUDE_IRQ_SYNCHRONIZER Set to 1 to include the IRQ synchronizer. Set to 0 to connect IRQ signals directly to the NVIC of the processor. The default value is 0.</p> <p>INCLUDE_APB_TEST_SLAVE Set to 1 to include apb_test_slave. Set to 0 to remove. The default value is 1.</p> <p>INCLUDE_APB_TIMER0 Set to 1 to include timer #0. Set to 0 to remove. The default value is 1.</p> <p>INCLUDE_APB_TIMER1 Set to 1 to include timer #1. Set to 0 to remove. The default value is 1.</p> <p>INCLUDE_APB_DUALTIMER0 Set to 1 to include dual timer. Set to 0 to remove. The default value is 1.</p> <p>INCLUDE_APB_UART0 Set to 1 to include uart #0. Set to 0 to remove. The default value is 1.</p> <p>INCLUDE_APB_UART1 Set to 1 to include uart #1. Set to 0 to remove. The default value is 1.</p> <p>INCLUDE_APB_UART2 Set to 1 to include uart #2. Set to 0 to remove. The default value is 1.</p> <p>INCLUDE_APB_WATCHDOG Set to 1 to include watchdog. Set to 0 to remove. The default value is 1.</p> <p>BE Big-endian. The default value is 0, little-endian. Set the value to 1 for big-endian configuration.</p> <hr/> <p>Note</p> <p>By default, all port enable parameters are set to 0.</p> <hr/> |
| Clock domain | <p>HCLK For the AHB to APB bridge.</p> <p>PCLK For peripheral operation.</p> <p>PCLKG Gated PCLK for APB bus interface logic. It must be the same frequency and same phase as PCLK. Tie LOW, when there are no APB activities. The APBACTIVE output signal controls the gating.</p> |

The example APB subsystem is designed primarily for little-endian configuration. The peripherals are designed with the little-endian programmers model. The big-endian parameter is provided to enable ARM to perform system-level tests to verify the bus component behavior in the big-endian configuration, and to enable system designers to evaluate the processor in the

big-endian configuration. Use of the big-endian parameter is not recommended for product development, because it adds extra hardware. Ideally, you must modify the peripherals for the big-endian systems to use the big-endian programmers model.

4.7.1 Programmers model

Table 4-27 shows the APB subsystem memory map.

Table 4-27 APB subsystem memory map

| Offset value | Device |
|---------------|---------------------------------------|
| 0x0000 | Timer 0 |
| 0x1000 | Timer 1 |
| 0x2000 | Dual Timer |
| 0x3000 | Not used ^a |
| 0x4000 | UART 0 |
| 0x5000 | UART 1 |
| 0x6000 | UART 2 |
| 0x7000 | Not used ^a |
| 0x8000 | Watchdog |
| 0x9000-0xA000 | Not used ^a |
| 0xB000 | APB test slave for validation purpose |
| 0xC000 | APB expansion port 12 |
| 0xD000 | APB expansion port 13 |
| 0xE000 | APB expansion port 14 |
| 0xF000 | APB expansion port 15 |

a. The corresponding port of the APB slave multiplexer is disabled by a Verilog parameter.

Table 4-28 shows the APB subsystem IRQ assignments.

Table 4-28 APB subsystem IRQ assignments

| IRQ | Device |
|------|--|
| 0 | UART 0 receive interrupt |
| 1 | UART 0 transmit interrupt |
| 2 | UART 1 receive interrupt |
| 3 | UART 1 transmit interrupt |
| 4 | UART 2 receive interrupt |
| 5 | UART 2 transmit interrupt |
| 6, 7 | Not used in the APB subsystem ^a |

Table 4-28 APB subsystem IRQ assignments (continued)

| IRQ | Device |
|-------|--|
| 8 | Timer 0 |
| 9 | Timer 1 |
| 10 | Dual-input timer |
| 11 | Not used |
| 12 | UART 0 overflow interrupt |
| 13 | UART 1 overflow interrupt |
| 14 | UART 2 overflow interrupt |
| 15 | Not used in APB subsystem ^b |
| 16-31 | Not used in APB subsystem ^a |

a. Reserved for GPIO in AHB.

b. Reserved for DMA.

4.7.2 Signal descriptions

The APB subsystem contains the following non-AMBA signals in its interface:

- *Clock and reset signals.*
- *UART signals on page 4-31.*
- *Timer signals on page 4-31.*
- *Watchdog signals on page 4-31.*
- *Interrupt signals on page 4-31.*
- *APB expansion port signals on page 4-32.*

Clock and reset signals

Table 4-29 shows the APB subsystem clock and reset signals.

Table 4-29 APB subsystem clock and reset signals

| Signal | Description |
|------------------|---|
| PCLKEN | <p>Clock enable for APB interface.</p> <p>The AHB to APB bridge uses this signal to enable you to run the APB operation at a lower speed than the AHB. The APB peripherals in the example system use the divided clock, PCLK and PCLKG, and therefore these peripherals ignore this signal.</p> <p>PCLK must be the gated version of HCLK using PCLKEN. PCLKG might be a gated version of PCLK, for example, using APBACTIVE.</p> |
| APBACTIVE | <p>The AHB to APB bridge generates this signal. It enables you to handle clock gating for the gated APB bus clock, PCLKG, in the example system.</p> <p>When there is no APB transfer, you can stop the gated APB bus clock to reduce power.</p> |
| HRESETn | Clock reset for AHB side. |
| PRESETn | Clock reset for APB side. |

UART signals

The subsystem includes UART 0, UART 1, and UART 2. These signals are connected to a pin multiplexer of the example system to interface with external I/O. [Table 4-30](#) shows the APB subsystem UART signals.

Table 4-30 APB subsystem UART signals

| Signal | Description |
|-------------------|-----------------|
| uartn_rxd | Receive data |
| uartn_txd | Transmit data |
| uartn_txen | Transmit enable |

Timer signals

The timer signals are connected to a multiplexer pin of the example system to interface with external I/O. [Table 4-31](#) shows the APB subsystem timer signals.

Table 4-31 APB subsystem timer signals

| Signal | Description |
|---------------------|------------------------|
| timer0_extin | Timer 0 external input |
| timer1_extin | Timer 1 external input |

Watchdog signals

In the example system, the watchdog interrupt is connected to the *Non-Maskable Interrupt* (NMI) signal of the processor, and the watchdog reset signal is connected to the reset generator of the system. [Table 4-32](#) shows the APB subsystem watchdog signals.

Table 4-32 APB subsystem watchdog signals

| Signal | Description |
|---------------------------|--------------------|
| watchdog_interrupt | Watchdog interrupt |
| watchdog_reset | Watchdog reset |

Interrupt signals

In the example system design, this signal is merged with the other interrupt signals, and connected to the *Nested Vectored Interrupt Controller* (NVIC) of the Cortex-M processor. [Table 4-33](#) shows the APB subsystem interrupt signal

Table 4-33 APB subsystem interrupt signal

| Signal | Description |
|----------------------------|-------------------------|
| apbsubsys_interrupt | APB subsystem interrupt |

APB expansion port signals

APB expansion ports 12-15 enable you to connect additional APB slaves, if required. If you do not use these ports, you can disable them using a Verilog parameter to avoid unused logic. [Table 4-34](#) shows the APB expansion port signals.

Table 4-34 APB subsystem APB expansion port signals

| Signal | Description |
|--------------------------|---|
| extn_psel | APB expansion port <i>n</i> , select |
| extn_pready | APB expansion port <i>n</i> , ready |
| extn_prdata[31:0] | APB expansion port <i>n</i> , read data |
| extn_pslverr | APB expansion port <i>n</i> , slave error |

4.7.3 APB test slave

A simple APB test slave is included for verification purposes. ARM uses the test slave to verify the handling of wait states and error responses in the AHB to APB bridge. The APB test slave contains a software-programmable register that supports word, halfword, and byte size accesses.

Programmers model

[Table 4-35](#) shows the APB test slave memory map.

Table 4-35 APB test slave memory map

| Base offset | Type | Reset value | Description |
|-------------|------|-------------|--|
| 0x000 | RW | 0x00000000 | Data Register, 32-bit. Zero wait state, two PCLK cycles per access. |
| 0x004 | RW | 0x00000000 | Alias of Data Register, with one wait state. |
| 0x008 | RW | 0x00000000 | Alias of Data Register, with two wait states. |
| 0x00C | RW | 0x00000000 | Alias of Data Register, with three wait states. |
| 0x010–0x0EF | RW | - | Reserved. RAZ/WI. |
| 0xF0 | RW | 0x00000000 | Alias of Data Register, with zero wait states and error responses. |
| 0xF4 | RW | 0x00000000 | Alias of Data Register, with one wait state and error response. |
| 0xF8 | RW | 0x00000000 | Alias of Data Register, with two wait states and error responses. |
| 0xFC | RW | 0x00000000 | Alias of Data Register, with three wait states and error responses. |
| 0x100–0xFFF | RW | - | Reserved. RAZ/WI. |

———— Note ————

The AHB interface on the APB subsystem can generate non-word sized writes which is not supported by the current APB peripherals in the Cortex-M System Design Kit. The current APB peripherals do not support non-word sized writes because **PSTRB** is not present. Therefore, restrict all write accesses to be word sized writes.

4.8 APB timeout monitor

The APB timeout monitor, `cmsdk_apb_timeout_mon.v`, prevents an APB slave from locking up a system. It is placed between an APB master and an APB slave, and is connected directly to the APB slave. If there is an active transfer to the slave, and the slave holds the **PREADY** signal LOW for more than a certain number of clock cycles, the monitor generates an error response to the bus master.

If the bus master generates any subsequent access to this slave, the monitor returns an error response and blocks the access to the slave until the slave has asserted its **PREADY HIGH**.

Figure 4-24 shows the APB timeout monitor module.

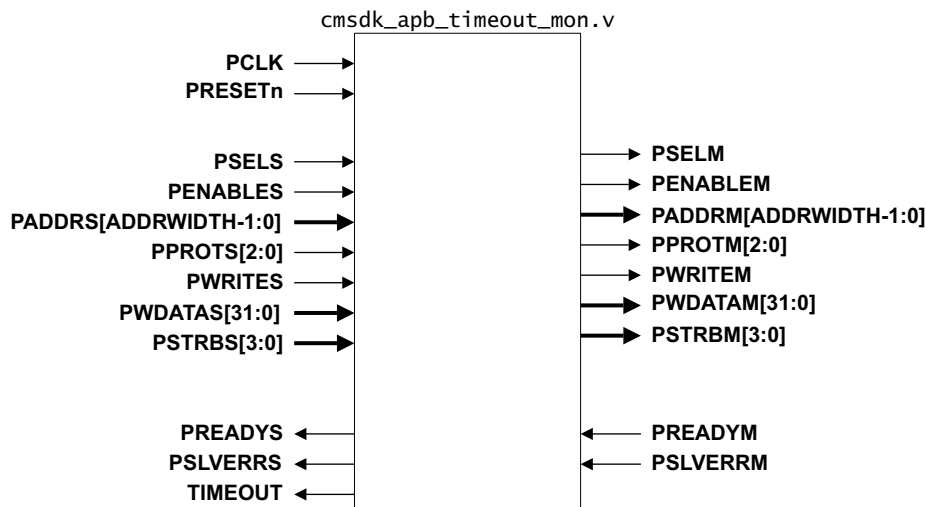


Figure 4-24 APB timeout monitor

If multiple APB slaves require monitoring, each of them might require its own APB timeout monitor, unless the bus fault handler does not require access to the blocked APB bus segment and therefore can work during a blocking state. See Figure 4-25.

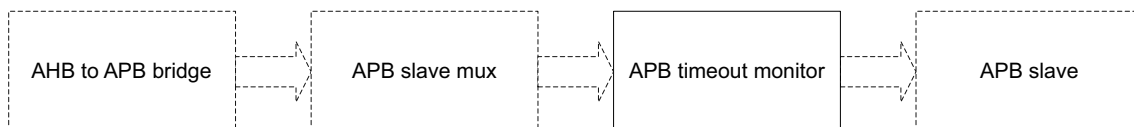


Figure 4-25 Use of APB timeout monitor

Table 4-36 shows the characteristics of the APB timeout monitor.

Table 4-36 APB timeout monitor characteristics

| Element name | Description | |
|--------------|--------------------------------------|--|
| Filename | <code>cmsdk_apb_timeout_mon.v</code> | |
| Parameter | ADDRWIDTH | Width of APB address bus. The default value is 12. |
| | TIME_OUT_VALUE | Number of wait state cycles that trigger timeout. The value can vary from 3-1023, and the default value is 16. |
| Clock domain | PCLK for APB operation. | |

Chapter 5

Advanced AHB-Lite Components

This chapter describes the advanced AHB-Lite components that the Cortex-M System Design Kit uses. It contains the following sections:

- [AHB bus matrix on page 5-2.](#)
- [AHB upsizer on page 5-14.](#)
- [AHB downsizer on page 5-17.](#)
- [AHB to APB asynchronous bridge on page 5-25.](#)
- [AHB to AHB and APB asynchronous bridge on page 5-27.](#)
- [AHB to AHB synchronous bridge on page 5-30.](#)
- [AHB to AHB sync-down bridge on page 5-32.](#)
- [AHB to AHB sync-up bridge on page 5-37.](#)

Note

The advanced AHB-Lite components are available only with the Cortex-M System Design Kit, full version. They are not included in the Cortex-M0 and Cortex-M0+ System Design Kit.

5.1 AHB bus matrix

In the Cortex-M System Design Kit, the bus matrix component provides a low-latency solution. The following subsections describe the bus matrix configurable features and operation:

- [Key features.](#)
- [Bus matrix configurability on page 5-3.](#)
- [Bus matrix module on page 5-3.](#)
- [Operation on page 5-5.](#)
- [Programmers model on page 5-5.](#)
- [Block functionality on page 5-6.](#)
- [Arbitration and locked transfers on page 5-7.](#)
- [Address map on page 5-8.](#)
- [Signal descriptions on page 5-11.](#)

5.1.1 Key features

The bus matrix has the following key features:

- Supports the AMBA 3 AHB-Lite protocol.
- Number of slave ports, from 1-16.
- Number of master ports, from 1-16.
- Routing data width, either 32 bits or 64 bits.
- Routing address width, from 32-64 bits.
- Arbiter type, that can be round-robin, fixed, or burst.
- Default slave included with each slave port.
- Optional **xUSER** signals, from 0-32 bits, with zero meaning excluded.
- Sparse connectivity:
 - The sparse connectivity feature removes any unnecessary connections, and reduces area and multiplexer delays.
 - Separate instances of the output stage and output arbiter are generated for each master port.
 - For input-output stages with only one sparse connection, the choice of arbiter is overridden with single arbiter and output stage modules. These single modules also permit 1xn interconnects.
- Design entry by command line where the address map is calculated, excluding REMAP support.
- Design entry by XML configuration file that enables you to specify an address map, including REMAP support.
- User-specified module names or automatically derived top-level name.
- User-specified source and target directories.
- Optional ``timescale` Verilog directives.

5.1.2 Bus matrix configurability

The bus matrix is a configurable component that enables you to connect multiple AHB masters to multiple AHB slaves.

The bus matrix RTL is generated automatically using the `BuildBusMatrix.pl` script. The script takes different configuration parameters, for example, the number of masters, number of slaves, and data-width, and generates the corresponding Verilog RTL.

Alternatively, you can create an XML file to describe the bus matrix design, and pass it on to the `BuildBusMatrix.pl` script to create the bus matrix.

Note

You must execute the `BuildBusMatrix.pl` script from the `logical/ahb_bus_matrix` directory. To get help, enter:

```
bin/BuildBusMatrix.pl -help
```

5.1.3 Bus matrix module

The bus matrix module, `cmsdk_ahb_busmatrix_<num_slaves>x<num_masters>_lite.v`, enables multiple AHB masters from different AHB buses to be connected to multiple AHB slaves on multiple AHB slave buses. It enables parallel access to a number of shared AHB slaves from a number of different AHB masters. The bus matrix determines the master that gains access to each slave, and routes the control signals and data signals between them. This block is required in multi-layer AHB systems.

[Figure 5-1 on page 5-4](#) shows the bus matrix module.

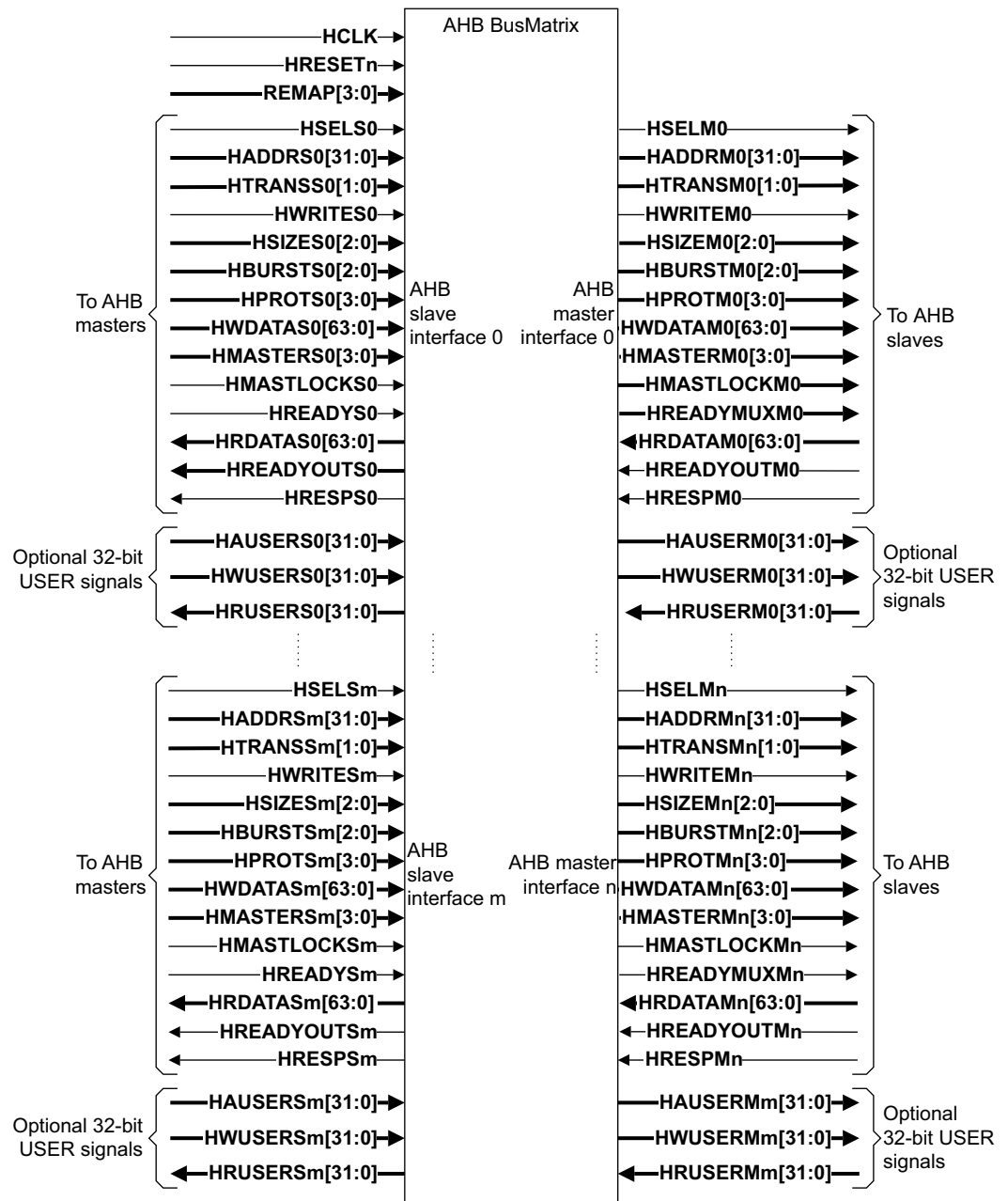


Figure 5-1 Bus matrix module components

Figure 5-1 shows a bus matrix module component block diagram for $(m+1)$ input ports, $(n+1)$ output ports, 64-bit routing data width, 32-bit routing address width, and 32-bit **USER** signal widths.

The bus matrix signal names have the following suffixes for port and pin naming:

- Signals on the AHB slave interface from AHB masters have the suffix **S**.
- Signals on the AHB master interface to AHB slaves have the suffix **M**.

The bus matrix connects to the masters and slaves using this naming scheme, with an additional integer to identify the correct master and slave. For example, connect **HWDATAS0[63:0]** to the 64-bit AHB Master 0 write data port, and **HWDATAM0[63:0]** to the AHB Slave 0 write data port.

5.1.4 Operation

The following sections describe the operation of the bus matrix:

- [Integrating the bus matrix](#).
- [Locked sequences](#).

Integrating the bus matrix

When integrating the bus matrix component:

- The input ports, with signal suffix **S**, are AHB slave ports, and you must connect them accordingly.
- The output ports, with signal suffix **M**, are AHB slave gasket ports. That is, they attach directly to an AHB slave port, that they mirror. The AHB slave gasket port is not treated in the same way as a full AHB master port.
- If you want to use the output from a bus matrix as a bus master on an additional AHB layer, and if you require a high operational frequency, ARM recommends that you use a component such as an AHB bridge, for example, an AHB to AHB synchronous bridge or the Ahb2AhbPass component in the ARM *AMBA Design Kit* (ADK).

Note

When connecting to an output port on the bus matrix, you must connect the **HSEL** pin to the attached slave even if there is only one slave present. If you do not do this, the slave might see spurious transfers.

Locked sequences

The bus matrix is only designed to support locked sequences that target a single output port. Because of this, you do not require a snooping bus across all input ports. This provides arbitration for locked transfers on all layers simultaneously.

5.1.5 Programmers model

The design of the bus matrix consists of an input stage, a decode stage, and an output stage that [Block functionality on page 5-6](#) describes. [Figure 5-2 on page 5-6](#) shows a bus matrix design with:

- Four slave ports, for connection to bus masters.
- Three master ports, for connection to slaves.

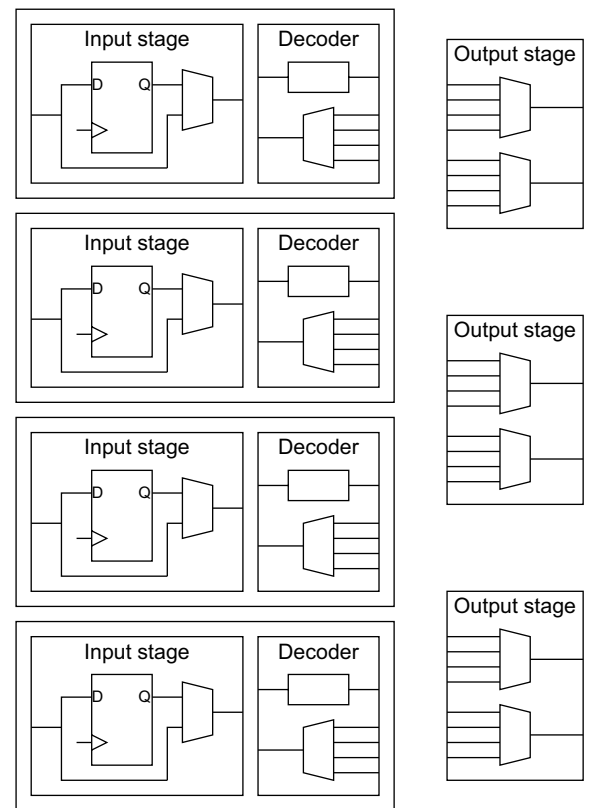


Figure 5-2 Example bus matrix design configuration

5.1.6 Block functionality

The following sections describe the functionality of the bus matrix module:

- [Input stage](#).
- [Decode stage](#).
- [Output stage on page 5-7](#).

Input stage

The input stage provides the following functions:

- It registers and holds an incoming transfer if the receiving slave is not able to accept the transfer immediately.
- If the bus matrix switches between input ports while in the middle of an undefined length burst, the input stage modifies the **HTRANS** and **HBURST** signals for the interrupted input port, so that when it is reinstated, the remaining transfers in the burst meet the AHB specification.

Decode stage

The decode stage generates the select signal for individual slaves. It also handles the multiplexing of response signals and read data. During the address phase of a transfer, the decoder asserts the slave-select signal for the appropriate output stage corresponding to the address of the transfer. In addition, the decoder routes an active signal from the output stage

back to the input stage. This signal indicates to the input that its address is currently being driven onto the chosen slave. During the data phase of a transfer, the decoder routes the response signals and read data back to the input port.

Each slave port, connected to an AHB master, is associated with a separate decoder. This enables the AHB masters to have independent address maps, that is, a shared slave is not required to appear in the same address location for all masters. This is useful for multi-processor systems.

Any gaps in the memory map are redirected to a default slave that returns an OKAY or ERROR response, depending on the type of access. An instance of a default slave is associated with each decoder.

The decoder stage also supports the system address Remap function. A 4-bit Remap control signal connects to the decoder. You can use remapping to change the address of physical memory or a device after the application has started executing. This is typically done to permit RAM to replace ROM when the initialization has been completed.

In multi-layer AHB systems that have local slaves on some of the AHB layers, the address decoding is performed in two stages. The first address decoder selects between local slaves and the shared slaves available through the AHB bus matrix module. To support this, the decode stage in the bus matrix includes an **HSEL** input that indicates whether the address from an input port is destined for a shared slave.

Output stage

The output stage has the following functions:

- Selects the address and control signals from the input stages.
- Selects the corresponding write data from the input stage.
- Determines when to switch between input ports in the input stage.

The output stage only selects an input source when that input has a transfer in the holding register.

The output stage generates an active signal for each input port when the address from that input port is being driven onto the slave. This signal enables the input stage to determine when to hold up transfers from other masters because the slave is not currently available.

When a sequence of transfers to a shared slave has finished, and there are no more transfers to the slave required by any of the input ports, the output stage switches the address and control signals to an idle state.

5.1.7 Arbitration and locked transfers

This section describes:

- [Arbitration](#).
- [Locked transfers on page 5-8](#).

Arbitration

The arbitration in the bus matrix module determines the input port that has access to the shared slave, and each shared slave has its own arbitration. Different arbitration schemes provide different system characteristics in terms of access latency and overall system performance.

The slave switch supports the following arbitration schemes:

Fixed arbitration

One port always has the highest priority and the order of priority for all other ports is fixed.

You can break up burst transfers, if a higher-priority master requests the same slave, except where the burst transfer is a locked transfer.

Fixed (burst) arbitration

This is similar to fixed arbitration but it does not break defined length burst transfers and it is the default arbitration for the bus matrix.

Round-robin arbitration

Arbitration is performed during every active clock cycle, indicated by **HREADYM**. Priority initially goes to the lowest-numbered requestor, that is, input port 0. When multiple requests are active, priority goes to the next lowest-numbered requestor compared to the currently active one. Fixed-length bursts are not broken. The arbitration waits for the end of the burst before passing control to the next requestor, if one exists. INCR bursts are treated as four-beat bursts, to optimize memory accesses, with guard logic to ensure that a sequence of short INCR bursts does not freeze the arbitration scheme.

Locked transfers

Using a multi-layer AHB system requires certain restrictions to be placed on the use of locked transfers to prevent a system deadlock. A sequence of locked transfers must be performed to the same slave in the system. Because the minimum address space that you can allocate to a single slave is 1KB, a bus master can ensure that this restriction is met by ensuring that it does not perform a locked sequence of transfers over a 1KB boundary. This ensures that it never crosses an address decode boundary.

Therefore, if a bus master is to perform two locked transfer sequences to different address regions, the bus master must not start the second locked transfer sequence until the final data phase of the first locked transfer sequence has completed.

5.1.8 Address map

If you do not use the XML configuration method, you can use command line parameters instead. In this case, the address map is calculated automatically as follows:

- [Figure 5-3](#) shows the equations that enable the address map to be divided into a number of regions depending on the number of master ports:

$$\text{regions} = \text{round_to_highest_2toN}(\text{total_master_ports})$$

$$\text{region_size} = \frac{2^{\text{routing_address_width}}}{\text{regions}}$$

$$\text{region_base} = \text{region_size} \times \text{master_port_instance}$$

$$\text{region_top} = \text{region_base} + \text{region_size} - 1$$

Figure 5-3 Region equations

- Each slave port has the same decoder instance.

- There is no Remap support.

If you are using XML configuration method, the decode stage can have a fully-customized address map, and each slave port can have an independent view of the address space.

[Example 5-1](#) shows a slave port address map description.

Example 5-1 Slave port address map description

```
<slave_interface name="SI1">
  <address_region interface="MI0" mem_lo="40000000" mem_hi="4fffffff" remapping="move"/>
  <address_region interface="MI0" mem_lo="70000000" mem_hi="7fffffff" remapping="alias"/>
  <address_region interface="MI1" mem_lo="80000000" mem_hi="9fffffff" remapping="none"/>
  <address_region interface="MI2" mem_lo="a0000000" mem_hi="bfffffff" remapping="move"/>
  <address_region interface="MI3" mem_lo="00000000" mem_hi="1fffffff" remapping="move"/>

  <remap_region interface="MI0" mem_lo="00000000" mem_hi="1fffffff" bit="0"/>
  <remap_region interface="MI1" mem_lo="50000000" mem_hi="5fffffff" bit="0"/>
  <remap_region interface="MI2" mem_lo="60000000" mem_hi="6fffffff" bit="1"/>
  <remap_region interface="MI3" mem_lo="c0000000" mem_hi="dfffffff" bit="0"/>
</slave_interface>
```

Address region

The address region parameters determine the routing of transactions to the master interfaces. Each master interface can have multiple non-contiguous address regions, when multiple sets of address region parameters are defined. However, the address regions of different master interfaces must not overlap. The `mem_lo` parameter defines the lower bound address and the `mem_hi` parameter defines the upper bound address for the master interface.

The remapping configuration parameter defines the behavior of master interfaces that support address remapping. It becomes active when the relevant REMAP bit is set. There are two types of address remapping behavior:

- If you set the remapping parameter to `alias` or `none`, the remapping creates an alias of the defined region in the new address space.
- If you set the remapping parameter to `move`, the address region is removed from the original address space and the master interface appears at the location defined by the remap region in the new address space.

Remap region

These regions are activated when using the remap facility, and each remap region is associated with a bit of the **REMAP** signal.

When the relevant remap bit is set, the remap regions take higher priority than normal address regions for the same master interface. In addition, any normal regions that have the remapping parameter set to `move`, are removed from the address space. If more than one bit is asserted for the same master interface, the least significant bit takes priority.

[Figure 5-4 on page 5-10](#) shows the address map of the slave interface, defined in [Example 5-1](#), at different remap states.

The address map is explained at the remap state $REMAP = 0001$ as follows:

- Normally, the address map, MI0, appears at two non-contiguous regions, $0x40000000$ and $0x70000000$. When you set remap bit to 0, the region at $0x40000000$ is removed because the remapping parameter is set to move and MI0 appears at the new remap region $0x00000000$ to $0x1FFFFFFF$. The MI0 region at $0x70000000$ is not removed because its remapping parameter is declared as alias.
- When you set remap bit to 0, MI1 appears at the new remap region, $0x50000000$ to $0x5FFFFFFF$. The region at $0x80000000$ did not change because its remapping is set to none.
- MI2 did not change even though its remapping is declared as move, because it is associated with remap bit one. Remap bit one is not set at the current remap state.
- MI3 is moved to a new base address $0xC0000000$.

———— **Note** ————

You can consider MI0 as ROM and MI3 as RAM. At boot time, the **REMAP** signal is set to 0001 and you can observe ROM at base address $0x00000000$. After booting, the **REMAP** signal is set to 0000 , the RAM now appears at $0x00000000$, and the ROM is moved up in the address space to $0x40000000$.

Figure 5-4 shows the address map at different remap states.

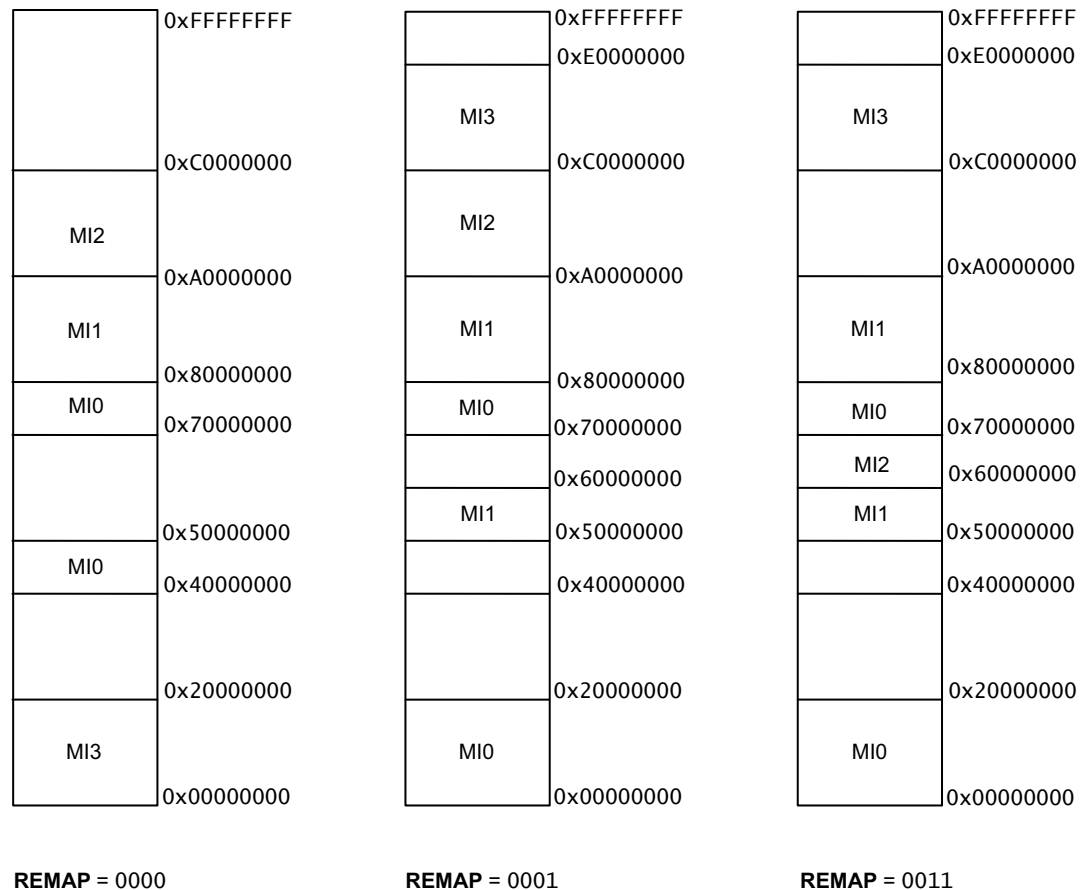


Figure 5-4 Address map at different remap states

5.1.9 Signal descriptions

Table 5-1 shows the bus matrix signals.

Table 5-1 Bus matrix signals

| Signal | Direction | Description |
|----------------------------------|-----------|--|
| HCLK | Input | System bus clock. Logic is triggered on the rising edge of the clock. |
| HRESETn | Input | Activate LOW asynchronous reset. |
| System address control | | |
| REMAP[3:0] | Input | System address remap control. |
| Interface to masters, AHB slave | | |
| HADDRSx[N-1:0] | Input | N-bit address bus from AHB master. The value of N can vary from 32 to 64. |
| HBURSTSx[2:0] | Input | Burst size information. |
| HMASTERSx[3:0] | Input | Current active master. |
| HMASTLOCKSx | Input | Indicates that the transfer on the master AHB is a locked transfer. |
| HPROTSx[3:0] | Input | Protection information. |
| HRDATASx[63:0 or 31:0] | Output | Read data to bus master. You can configure the width to either 64 bits or 32 bits. |
| HREADYOUTSx | Output | HREADY signal feedback to the master bus, indicating whether the AHB bus matrix module is ready for the next operation. |
| HREADYSx | Input | HREADY signal on the master AHB bus, indicating the start or end of a transfer. |
| HRESPSx | Output | Response from the AHB bus matrix module to the AHB master. |
| HSELSx | Input | Active HIGH select signal to indicate that a shared slave connected to the AHB bus matrix module is selected. |
| HSIZESx[2:0] | Input | Size of the data. |
| HWDATASx[63:0 or 31:0] | Input | Write data from AHB masters. You can configure the width to be either 64 or 32 bits. |
| HWRITESx | Input | Indicates a read or write operation. |
| Interface to slaves (AHB master) | | |
| HADDRMx[N-1:0] | Output | N-bit address bus for the AHB slave. The value of N can vary from 32 to 64. |
| HBURSTMx[2:0] | Output | Burst size information. |
| HMASTERMx[3:0] | Output | Currently active master. |
| HMASTLOCKMx | Output | Indicates that the transfer on the AHB slave is a locked transfer. |
| HPROTMx[3:0] | Output | Protection information. |
| HRDATAMx[63:0 or 31:0] | Input | Data read back from AHB slaves. You can configure the width to be either 64 bits or 32 bits. |
| HREADYOUTMx | Input | HREADYOUT from the AHB slave or slave multiplexer. |
| HREADYMUXMx | Output | HREADY feedback to all slaves. |
| HRESPMx | Input | HRESP from the AHB slave or slave multiplexer. |

Table 5-1 Bus matrix signals (continued)

| Signal | Direction | Description |
|--|-----------|--|
| HSELM_x | Output | Active HIGH select signal to indicate that the slave bus is accessed. You can use this signal to drive a single AHB slave directly, or drive a secondary AHB decoder if you use multiple AHB slaves. |
| HSIZEM_x[2:0] | Output | Size of the data. |
| HWDATAM_x[63:0 or 31:0] | Output | Write data to AHB slaves. You can configure the width to be either 64 bits or 32 bits. |
| HWRITEM_x | Output | Indicates a read or write operation. |
| USER signals | | |
| HAUSERS_x | Input | Additional sideband bus that has the same timing as the slave interface address phase signals. |
| HWUSERS_x | Input | Additional sideband bus that has the same timing as the slave interface write data phase signals. |
| HRUSERS_x | Output | Additional sideband bus that has the same timing as the slave interface read data phase signals. |
| HAUSERM_x | Output | Additional sideband bus that has the same timing as the master interface address phase signals. |
| HWUSERM_x | Output | Additional sideband bus that has the same timing as the master interface write data phase signals. |
| HRUSERM_x | Input | Additional sideband bus that has the same timing as the master interface read data phase signals. |

USER signals

The bus matrix supports **USER** signals on master and slave interfaces. These signals are optional, and a value of 0 on the `user_signal_width` parameter removes them from the generated Verilog. If the `user_signal_width` parameter or the `--userwidth` command line switch is set to a non-zero value, that value defines the width of those **USER** signals. The **USER** signals have the same timing as the payload signals for that channel. For example, the **HAUSER** signals have the same timing as the address payload signals.

The **USER** signals are:

- **HAUSER.**
- **HRUSER.**
- **HWUSER.**

N-bit addressing

The bus matrix supports N-bit addressing, that is, you can configure the address bus to be 32-64 bits. By default, the address width is set to 32 bits, but you can change this by setting the `--addrwidth` command line switch or by changing the `routing_address_width` global parameter in the XML file. Setting the address width affects both the address buses for the slave ports and master ports.

Note

The presence of **USER** signals and the support of N-bit addressing enables the bus matrix to fully connect to a network interconnect product such as the PrimeCell High Performance Matrix PL301. The bus matrix **USER** signals are fully mapped to their AXI counterparts. This is typically useful in mixed protocol designs. See the *CoreLink Network Interconnect NIC-301 Technical Reference Manual* for more information.

5.2 AHB upsizer

The following subsections describe the AHB upsizer features and its operation:

- [Overview](#).
- [Method of using AHB upsizer](#).

5.2.1 Overview

The AHB upsizer, `cmsdk_ahb_upsizer64.v`, enables a 32-bit AHB bus master to connect to a 64-bit AHB slave. [Figure 5-5](#) shows the AHB upsizer module.

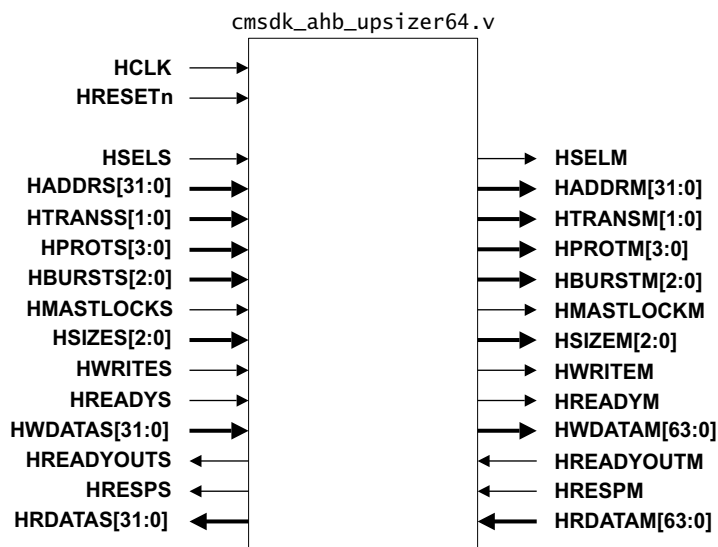


Figure 5-5 AHB upsizer

[Table 5-2](#) shows the characteristics of the AHB upsizer.

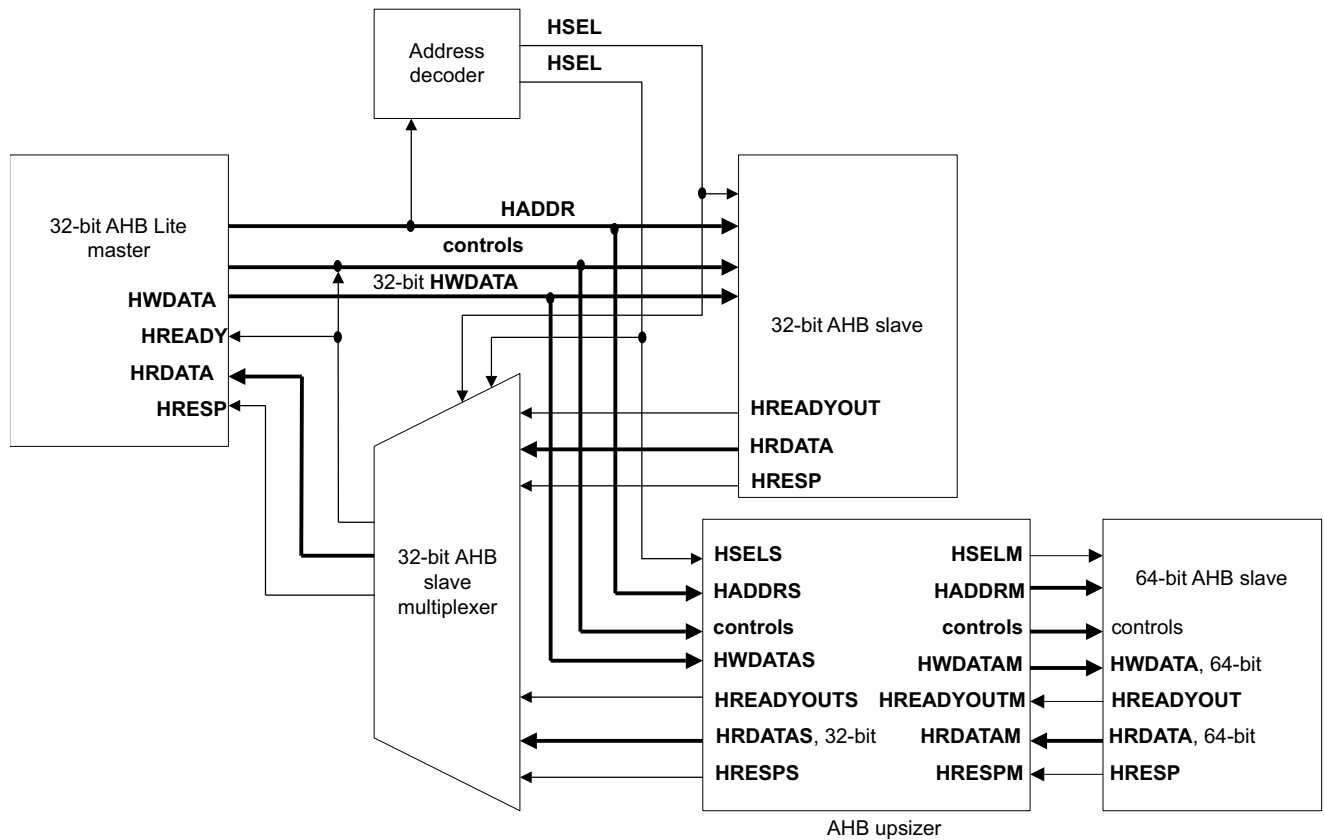
Table 5-2 AHB upsizer characteristics

| Element name | Description |
|--------------|------------------------------------|
| Filename | <code>cmsdk_ahb_upsizer64.v</code> |
| Parameters | None |
| Clock domain | HCLK |

The AHB upsizer handles the routing of data only and does not modify the transfer type or response information.

5.2.2 Method of using AHB upsizer

You can connect the AHB upsizer in a number of ways with various combinations of 32-bit or 64-bit AHB Lite systems. For example, you can couple the AHB upsizer directly with a 64-bit slave as [Figure 5-6 on page 5-15](#) shows.



Note: The controls included are **HTRANS**, **HWRITE**, **HSIZE**, **HPROT**, and **HBURST**.

Figure 5-6 Using AHB upsizer, type one

If there is more than one 64-bit AHB slave in the AHB segment, it is possible to share an AHB upsizer as [Figure 5-7 on page 5-16](#) shows.

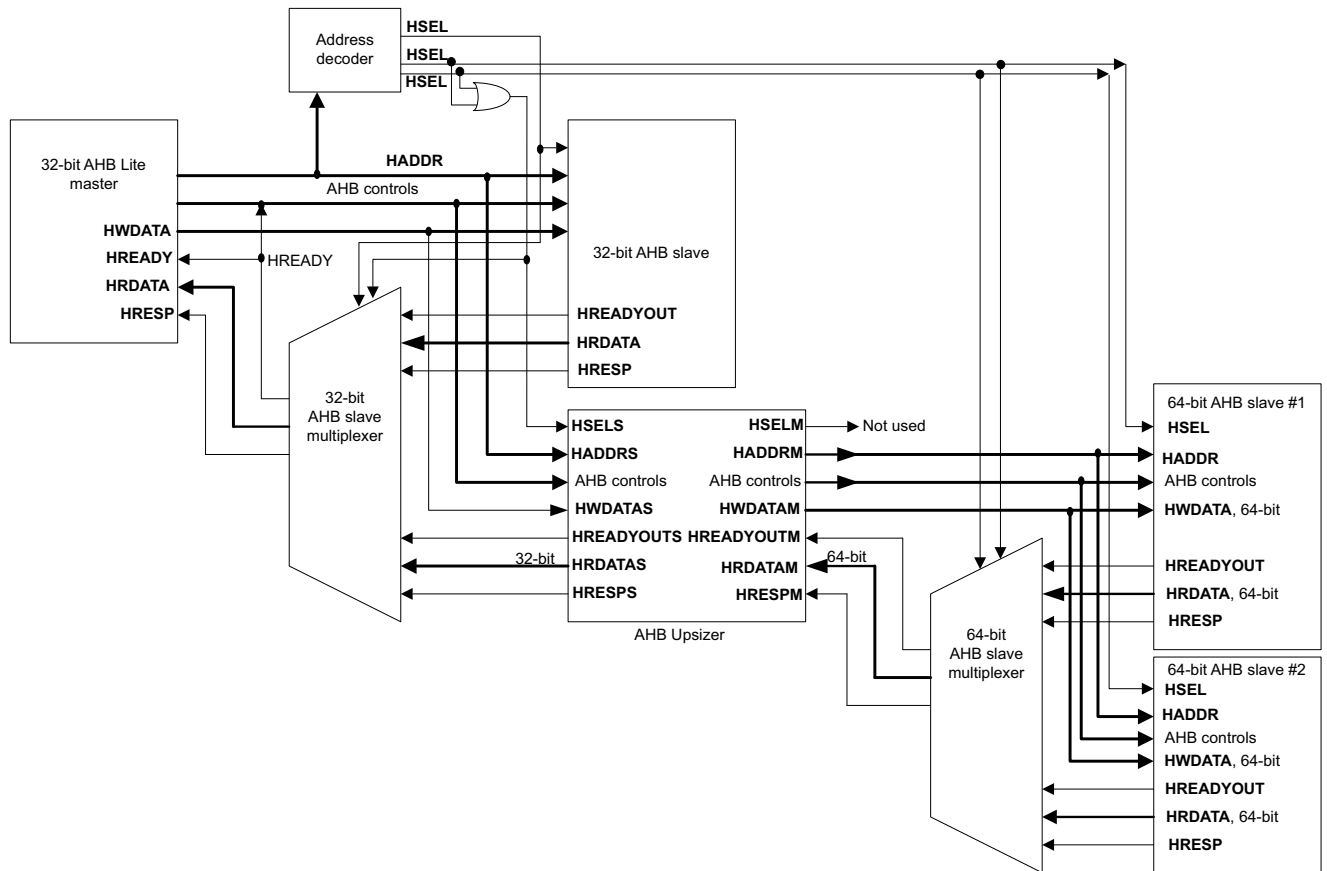


Figure 5-7 Using AHB upsizer, type two

You can use the **HSEL** signals generated from the 32-bit bus directly on the 64-bit AHB slaves because the AHB upsizer does not change the behavior of the **HSEL** signals. This arrangement avoids the requirement to have two AHB decoders in the system.

5.3 AHB downsizer

The following subsections describe the AHB downsizer features and its operation:

- [About the AHB downsizer.](#)
- [Programmers model on page 5-18.](#)
- [Signal descriptions on page 5-21.](#)
- [Using AHB downsizer on page 5-22.](#)

5.3.1 About the AHB downsizer

The AHB downsizer module, `cmsdk_ahb_downsizer64.v`, enables a 64-bit AHB bus master to connect to a 32-bit AHB slave. The downsizer module reduces the width of the data bus by half from an AHB master to an AHB slave. You can use full-width master transfer, involving modification of the transfer type, burst, and size, and latching half of the master read data. In addition, you might require multiple slave writes or reads to transfer data to and from the narrow slave.

Figure 5-8 shows the AHB downsizer module.

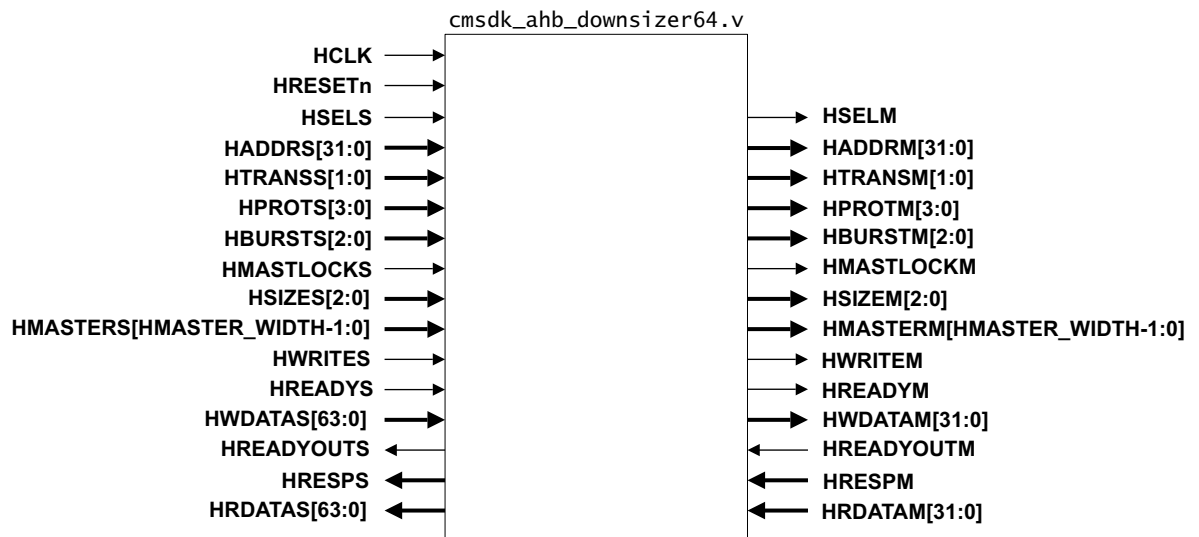


Figure 5-8 AHB downsizer

Table 5-3 shows the characteristics of the AHB downsizer.

Table 5-3 AHB downsizer characteristics

| Element name | Description | |
|--------------|-------------------------|---|
| Filename | cmsdk_ahb_downsizer64.v | |
| Parameter | HMASTER_WIDTH | Width of HMASTER . This is set to 1 by default. |
| | ERR_BURST_BLOCK_ALL | When this is set to 1, if an error response is received during a burst sequence, the remaining transfers in the burst sequence are blocked by the AHB downsizer and do not reach the downstream 32-bit AHB. The AHB downsizer returns an error response to each of the remaining transfers remains in the burst sequence. This behavior applies to 64-bit, 32-bit, 16-bit, and 8-bit transfers. When this is set to 0, the same blocking behavior applies to 64-bit burst transfers only. Burst of other transfer sizes do not have this blocking behavior. See Burst blocking after error on page 5-20 . This is set to 1 by default. |
| Clock domain | HCLK | |

5.3.2 Programmers model

The following sections describe the programming details for the downsizer:

- [Downsizer transfers](#).
- [Burst blocking after error on page 5-20](#).
- [Modification of control signals on page 5-20](#).

Downsizer transfers

You can use the following options for downsizer transfers:

Downsizer not selected

When the **HSELS** signal of the downsizer module is LOW, the transfer size is passed to the 32-bit AHB, and **HSELM** is driven LOW. The 32-bit slaves must ignore the transfers by monitoring **HSELM** and **HADDRM**. **HREADY**s from the 64-bit bus is output to **HREADYM**. All 32-bit devices connected to the 32-bit AHB must monitor this **HREADYM** signal to determine the end of the current transfer, and the start of the next transfer.

Narrow transfers, downsizer selected

If the downsizer module is selected, and the transfer is 32 bits or less, the downsizer module passes the transfer through. All the control signals and responses from the slave are left unmodified. In this case, the only function of the downsizer is to route the appropriate half of the wide master write data bus to the narrow slave data bus for write transfers.

Read transfers require even less control, and the narrow slave read data is replicated across the wide master bus.

Table 5-4 shows the handling of narrow transfers.

Table 5-4 Narrow transfer handling

| Transfer on 64-bit AHB | Transfer on 32-bit AHB | Address |
|---------------------------|---------------------------|--|
| 32, 16, or 8-bit transfer | 32, 16, or 8-bit transfer | HADDR passes through. If HADDR[2] = 0 then the HWDATAS[31:0] signals pass through. Otherwise, the HWDATAS[63:32] signals pass through. HRDATAS = HRDATAM , HRDATAM . |

For 32, 16, and 8-bit transfers, **HWDATA** is selected by bit[2] of the transfer address. If this bit is set LOW, **HWDATA[31:0]** is routed to the 32-bit AHB. If this bit is set HIGH, bits[63:32] are routed.

If an ERROR response is received from the 32-bit slave, the downsizer module automatically terminates the current transfer by passing the response to the 64-bit bus. If the current transfer request on the 64-bit bus is a valid transfer, **NON_SEQ** or **SEQ**, it is captured by the registers in the downsizer module and is applied to the 32-bit AHB one cycle later. The downsizer module inserts a wait state on the 64-bit bus to ensure that the next transfer is not missed.

If the transfer is a burst, and an ERROR response is received from a 32-bit slave, the rest of the burst is blocked. You can disable the burst blocking behavior for 32, 16, and 8-bit burst transfers by setting the **ERR_BURST_BLOCK_ALL** verilog parameter to 0. See [Burst blocking after error on page 5-20](#).

Wide transfers, downsizer selected

The role of the downsizer module is more complicated for 64-bit transfers. For both read and write transfers, the wide master transfers are broken down into two narrow slave cycles. The address to the slave is modified, to ensure that the two slave accesses go to different address locations. Table 5-5 shows the address line modification and data routing.

Table 5-5 Address line modification and data routing

| Transfer on 64-bit AHB | Transfer on 32-bit AHB | Address |
|------------------------|------------------------|--|
| 64-bit transfer | Cycle 1 | HADDR passes through. HWDATAS[31:0] passes through. HRDATAM is stored in the downsizer module. HADDRS[2:0] must equal 000. |
| | Cycle 2 | HADDRM[2] is set to 1. HWDATAS[63:32] passes through. HRDATAM passes through to HRDATAS[63:32] . Previously stored data is output to HRDATAS[31:0] . |

64-bit write transfers are split into two 32-bit transfers on two successive addresses. Table 5-5 shows the generation of **HADDRM[2]** and the routing of write data. Because **HWDATAS** is stable during the two AHB transfers on the 32-bit AHB, no register is required to hold **HWDATA**.

During 64-bit read accesses, the construction of a full-width word for the master to read two slave accesses is required. The data from the first read is latched, and the data from the second read flows straight through the block. Bits[31:0] are

always transferred in the first cycle, and bits[63:32] are transferred in the second cycle using the next word address. This transfer characteristic occurs independently of target system endianness.

If an ERROR response is received from the 32-bit slave, the response is passed to the 64-bit bus immediately. If this occurs on the first half of the 64-bit transfer, the second half of the transfer is not carried out.

If a two-cycle response is received, the downsizer module automatically aborts the current transfer by inserting an IDLE cycle on the 32-bit bus. If the current transfer request on the 64-bit bus is a valid transfer, NON_SEQ, the registers in the downsizer module capture it and apply it to the 32-bit AHB one cycle later. The downsizer module inserts a wait state on the 64-bit bus to ensure that the next transfer is not missed.

If the transfer is a burst, and an ERROR response is received from 32-bit slave, the rest of the burst is blocked.

Burst blocking after error

If an ERROR response is received from a 32-bit slave during a 64-bit burst, and if the 64-bit master continues the burst, the rest of the burst is blocked. During blocking, the ERROR response is fed back to the 64-bit AHB and an IDLE transfer is issued to the 32-bit AHB. The blocking ends when a nonsequential transfer request is detected, or if **HSELS** on the downsizer module is LOW. This feature ensures that there is no discontinuity in **HADDR** and **HTRANS**.

The blocking does not apply to 32, 16, or 8-bit transfers. In these cases, the rest of the transfer requests pass through as normal. If a busy cycle is detected during burst blocking, the downsizer module replies with an OKAY response. However, the subsequent SEQUENTIAL transfers are still blocked.

If the ERROR response occurs in the last cycle of the burst, no blocking is generated because the next transfer is an IDLE or NONSEQUENTIAL access. In this case, if the next access is nonsequential, the downsizer module issues an IDLE cycle on the 32-bit AHB in the second cycle of the ERROR response, stores the transfer control information, and applies it to the 32-bit AHB in the next cycle.

A wait state is inserted on the 64-bit bus to enable the 32-bit bus to catch up with the transfer.

Modification of control signals

Table 5-6 shows that the control signals are modified in the same way for both read and write transfers.

Table 5-6 Signal mapping when downsizer module is activated

| Control signals | Master cycle type | | Replaced by slave cycles | Comments |
|-----------------|-------------------|----|---------------------------|--|
| HTRANS | IDLE | to | IDLE | - |
| | BUSY | to | BUSY | - |
| | NONSEQ | to | NONSEQ, followed by a SEQ | No change if transfer is 8, 16, or 32 bits. |
| | SEQ | to | SEQ, followed by a SEQ | No change if transfer is 8, 16, or 32 bits. Exception for WRAP16 boundary, WRAP16 is mapped to INCR and NONSEQ is issued at 32-word boundary. |

Table 5-6 Signal mapping when downsizer module is activated (continued)

| Control signals | Master cycle type | | Replaced by slave cycles | Comments |
|-----------------|-------------------|----|-----------------------------|--|
| HADDR[2] | = 0 | to | 0 then 1 | No change if transfer is 8, 16, or 32 bits. |
| | = 1 | - | - | Not permitted for 64-bit transfer. |
| HSIZE | 8, 16, or 32 bits | to | 8, 16, or 32 bits | No conversion required. |
| | 64 bits | to | 32 bits | Conversion process activated. |
| | 128 or 256 bits | to | 32 bits | Not supported. |
| HBURST | SINGLE | to | INCR | No change if transfer is 8, 16, or 32 bits. |
| | INCR | to | INCR | No change if transfer is 8, 16, or 32 bits. |
| | INCR4 | to | INCR8 | No change if transfer is 8, 16, or 32 bits. |
| | WRAP4 | to | WRAP8 | No change if transfer is 8, 16, or 32 bits. |
| | INCR8 | to | INCR16 | No change if transfer is 8, 16, or 32 bits. |
| | WRAP8 | to | WRAP16 | No change if transfer is 8, 16, or 32 bits. |
| | INCR16 | to | INCR | No change if transfer is 8, 16, or 32 bits. |
| | WRAP16 | to | INCR | No change if transfer is 8, 16, or 32 bits. NONSEQ broadcast if WRAP boundary is reached. |

5.3.3 Signal descriptions

Table 5-7 shows the downsizer module signals.

Table 5-7 Downsizer module signals

| Signal | Direction | Description |
|---|-----------|---|
| HCLK | Input | System bus clock. Logic is triggered on the rising edge of the clock. |
| HRESETn | Input | Activate LOW asynchronous reset. |
| 64-bit AHB interface signals, AHB slave | | |
| HADDRS[31:0] | Input | Address from the 64-bit AHB. |
| HBURSTS[2:0] | Input | Burst size information on the 64-bit AHB. |
| HMASTLOCKS | Input | Indicates that the transfer on the 64-bit AHB is locked. |
| HPROTS[3:0] | Input | Protection information on the 64-bit AHB. |
| HRDATAS[63:0] | Output | Read data to the 64-bit bus. |
| HREADYOUTS | Output | HREADY signal feedback to the 64-bit bus, indicating that the downsizer is ready for the next operation. |
| HREADYS | Input | HREADY signal on the 64-bit AHB bus, indicating the start and end of a transfer on the 64-bit bus. |
| HRESPS | Output | Response from the downsizer module to the 64-bit bus. |
| HSELS | Input | Active HIGH select signal to indicate 32-bit memory range is accessed on the 64-bit AHB. |

Table 5-7 Downsizer module signals (continued)

| Signal | Direction | Description |
|--|-----------|--|
| HSIZES[2:0] | Input | Size of the data on the 64-bit AHB. |
| HWDATAS[63:0] | Input | Write data from the 64-bit bus. |
| HWRITES | Input | Indication of a read or write operation on the 64-bit AHB. |
| 32-bit AHB interface signals, AHB master | | |
| HADDRM[31:0] | Output | Address for the 32-bit AHB. |
| HBURSTM[2:0] | Output | Burst size information on the 32-bit AHB. |
| HMASTLOCKM | Output | Indicates that the transfer on the 32-bit AHB is locked. |
| HPROTM[3:0] | Output | Protection information on the 32-bit AHB. |
| HRDATAM[31:0] | Input | Data read back from AHB slaves. |
| HREADYM | Output | HREADY feedback to all slaves on the 32-bit AHB. |
| HREADYOUTM | Input | HREADY from the 32-bit AHB slaves or slave multiplexer. |
| HRESPM | Input | HRESP from the 32-bit AHB slaves or slave multiplexer. |
| HSELM | Output | Active HIGH select signal to indicate that a 32-bit bus is accessed. Use this signal to drive a single AHB slave directly, or drive a secondary AHB decoder if you use a multiple 32-bit AHB slaves. |
| HSIZEM[2:0] | Output | Size of the data on the 32-bit AHB. |
| HWDATAM[31:0] | Output | Write data to the 32-bit AHB slaves. |
| HWRITEM | Output | Indication of a read or write operation on the 32-bit AHB. |

Instead of reading **HREADY** from the 64-bit bus, or **HREADYOUT** from the 32-bit slave multiplexer, the AHB slaves on the 32-bit bus must read the **HREADYM** generated from the downsizer module. This signal is multiplexed between **HREADYOUTM**, when a slave attached to the M port of the downsizer is selected, including during 64-bit to 32-bit conversion, and **HREADYS**, when the downsizer is not selected.

Note

For this bridge, ARM recommends that when a slave is not selected with **HSELM** LOW, it must respond with **HREADYOUTM** HIGH and **HRESPM** LOW.

During a conversion, the 64-bit transfer is split into two 32-bit transfers, and all the AHB slaves on the 32-bit AHB bus are able to read the **HREADY** signal that the activated 32-bit slave generates. However, this **HREADY** signal must not be passed onto **HREADY** in the 64-bit bus system because this requires the insertion of wait states for the second 32-bit AHB transfer. Because of this, an additional **HREADYM** signal enables the AHB slave to determine when the end of an AHB transfer has occurred.

5.3.4 Using AHB downsizer

You can place the AHB downsizer in a number of ways with varied combination of 32-bit or 64-bit AHB system. For example, you can couple the AHB downsizer directly with 32-bit slave in a 64-bit AHB system as [Figure 5-9 on page 5-23](#) shows.

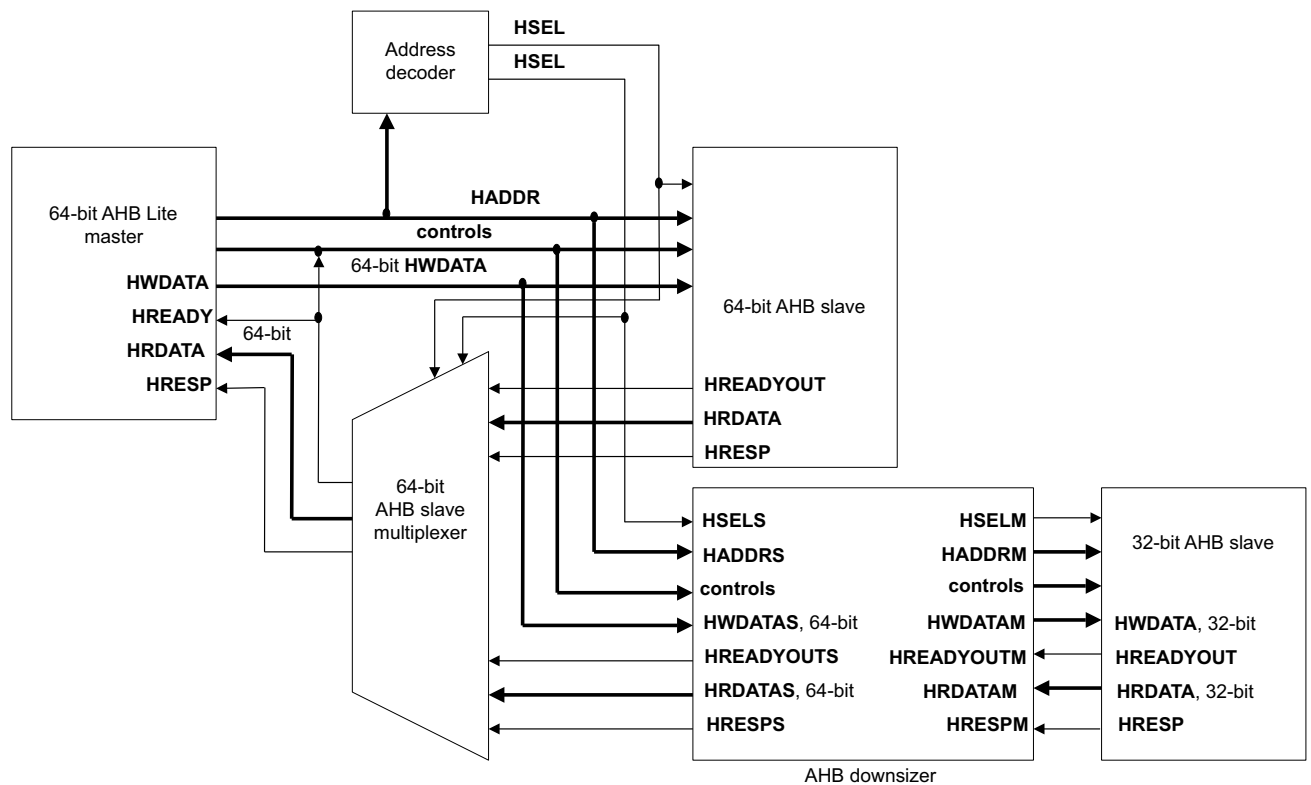


Figure 5-9 Using AHB downsizer, direct connection

You can also have one AHB downsizer to be shared between multiple 32-bit AHB slaves. The **HSEL** for the 32-bit AHB slaves must be derived from the **HSEL** output from the downsizer as [Figure 5-10 on page 5-24](#) shows.

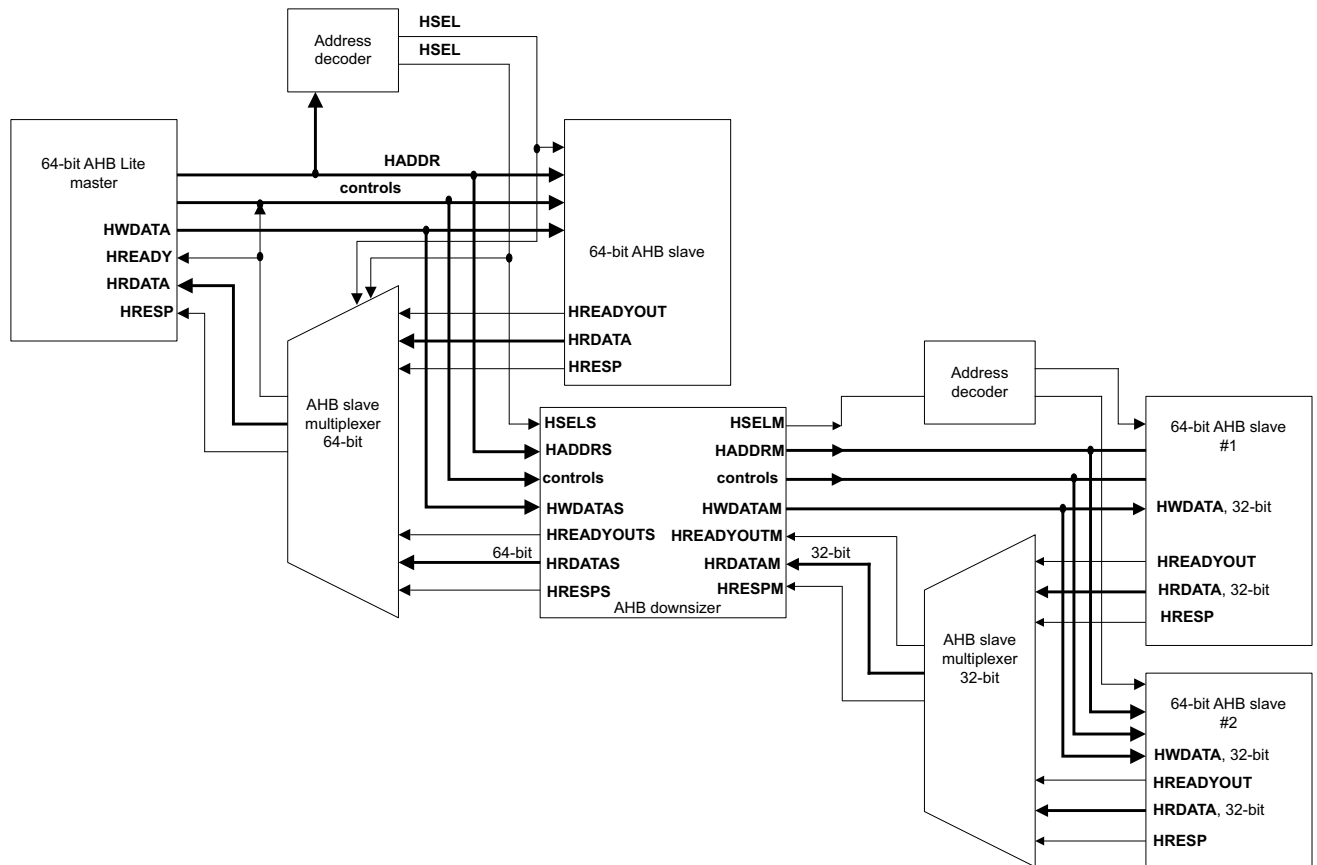


Figure 5-10 Using AHB downsizer, multiple connection

5.4 AHB to APB asynchronous bridge

The following sections describe the AHB to APB asynchronous bridge features and its operation:

- [About the AHB to APB asynchronous bridge.](#)
- [Cross-clock domain handling in AHB to APB asynchronous bridge on page 5-26.](#)

5.4.1 About the AHB to APB asynchronous bridge

The AHB to APB asynchronous bridge, `cmsdk_ahb_to_apb_async.v`, supports APB2, APB3, and APB4. [Figure 5-11](#) shows the AHB to APB asynchronous bridge module.

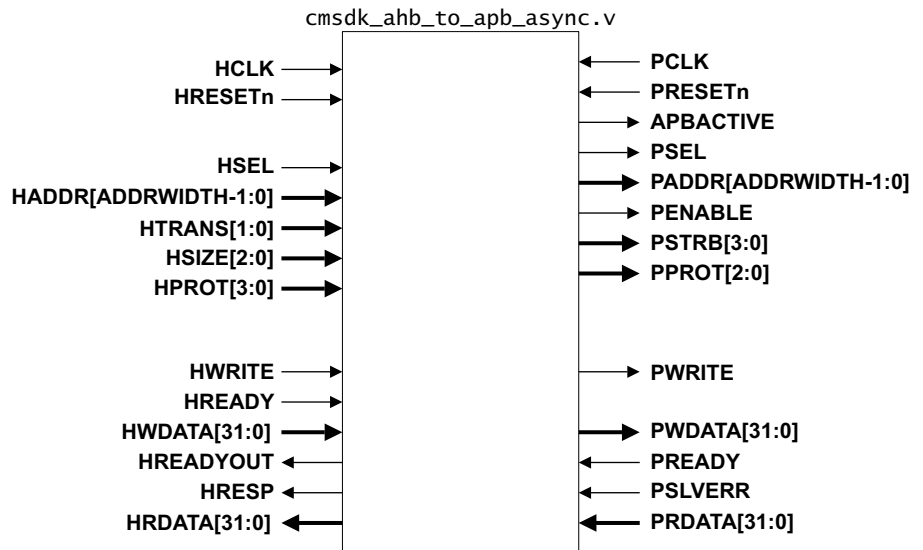


Figure 5-11 AHB to APB asynchronous bridge

[Table 5-8](#) shows the characteristics of the AHB to APB asynchronous bridge.

Table 5-8 AHB to APB asynchronous bridge characteristics

| Element name | Description | |
|--------------|---------------------------------------|--|
| Filename | <code>cmsdk_ahb_to_apb_async.v</code> | |
| Parameter | ADDRWIDTH | Width of the AHB or APB address bus. The default value is 16 (64Kbyte AHB or APB address space). |
| Clock domain | HCLK PCLK | |

The AHB to APB bridge has an output called **APBACTIVE** which is used to control clock gating cell for generation of a gated **PCLK**.

The gated **PCLK** is called as **PCLKG** in the example system.

When there is no APB transfer, this signal is LOW and stops the **PCLKG**. The peripherals designed with separate clock pins for bus logic and peripheral operation can take advantage of the gated **PCLK** to reduce power consumption.

The **APBACTIVE** signal is generated in the APB clock domain.

For more information, see [AHB to APB sync-down bridge on page 3-18](#).

5.4.2 Cross-clock domain handling in AHB to APB asynchronous bridge

Figure 5-12 shows the structure of the AHB to APB asynchronous bridge. The AHB to APB asynchronous bridge is divided into AHB clock domain and APB clock domain.

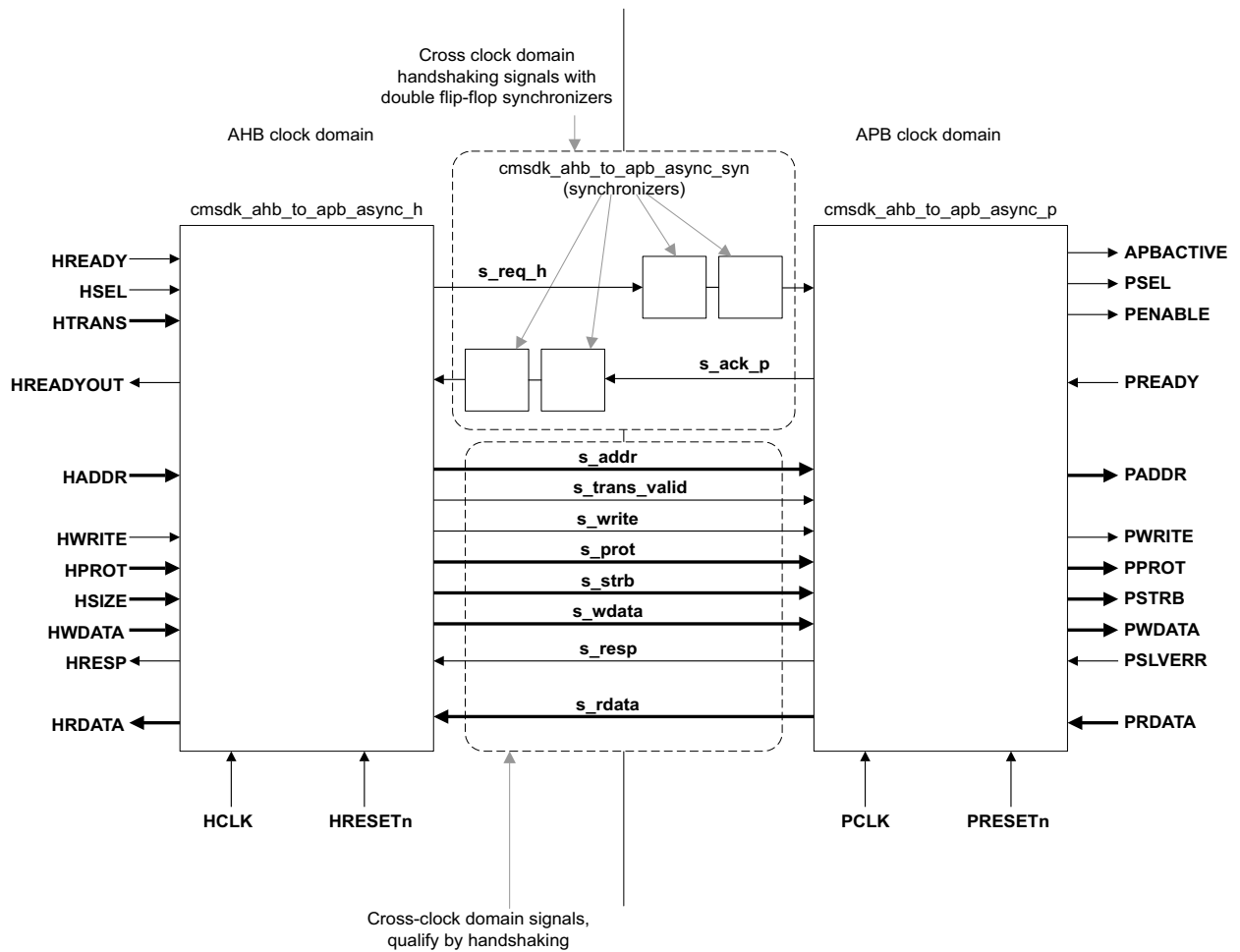


Figure 5-12 Structure of AHB to APB asynchronous bridge

If *Static Timing Analysis* (STA) is carried out, you can set the cross-clock domain signal path as false paths to avoid from being reported as timing violated paths.

5.5 AHB to AHB and APB asynchronous bridge

The following subsections describe the AHB to AHB and APB asynchronous bridge features and its operation:

- [About the AHB to AHB and APB asynchronous bridge.](#)
- [Handling of transfers initiated while master side is still in reset on page 5-28.](#)
- [Bursts on page 5-28.](#)
- [Reset requirements on page 5-28.](#)
- [External clock gating using the active signals on page 5-29.](#)
- [Clock domain crossing on page 5-29.](#)

5.5.1 About the AHB to AHB and APB asynchronous bridge

The AHB to AHB and APB asynchronous bridge, `cmsdk_ahb_to_ahb_apb_async.v`, supports AHB-Lite and APB4. [Figure 5-13](#) shows the AHB to AHB and APB asynchronous bridge module.

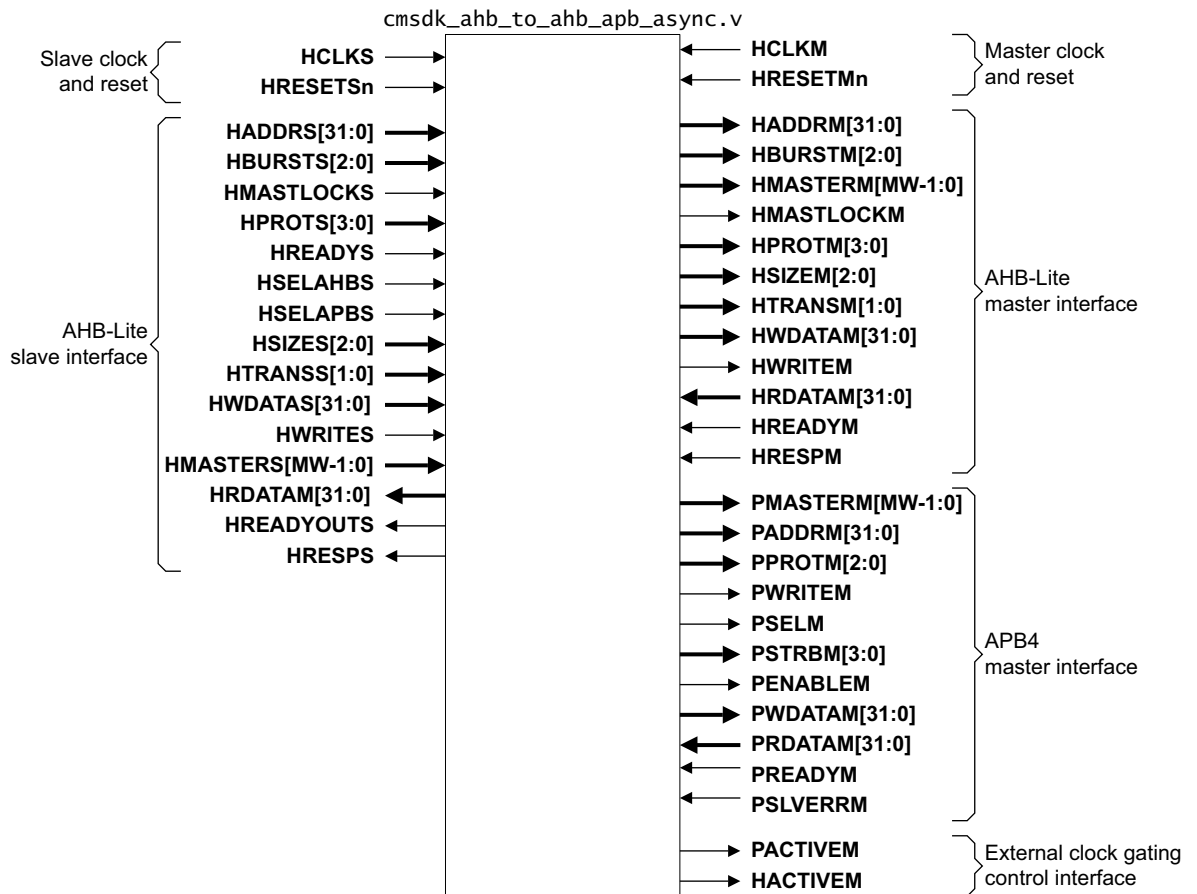


Figure 5-13 AHB to AHB and APB asynchronous bridge

Table 5-9 shows the characteristics of the AHB to AHB and APB asynchronous bridge.

Table 5-9 AHB to AHB and APB asynchronous bridge characteristics

| Element name | Description | |
|--------------|------------------------------|--|
| Filename | cmsdk_ahb_to_ahb_apb_async.v | |
| Parameter | MW | Width of the HMASTERS , HMASTERM , and PMMASTERM sideband signals. These address signals are intended to identify the master that initiated the transfer. |
| Clock domain | HCLKS HCLKM | |

5.5.2 Handling of transfers initiated while master side is still in reset

When a transfer is initiated on the slave side while the master side is in reset or coming out of reset, the bridge inserts wait states until the master is ready to process the transfer.

5.5.3 Bursts

The bridge handles beats in a burst as single beat transfers. It:

- Ignores **HTRANS[0]** and **HBURSTS**.
- Always drives **HTRANS[0]** to 1'b0, causing **HTRANS** to indicate either an IDLE or NONSEQ transfer.
- Always drives **HBURSTM** to 3'b000, indicating a SINGLE burst transfer.

5.5.4 Reset requirements

The reset requirements for the AHB to AHB and APB asynchronous bridge are:

- The slave and master resets must become asserted together, from not asserted.
- The slave reset must be asserted for at least two slave clock cycles. The slave reset must be deasserted synchronously to the slave clock.
- The master reset must be asserted for at least two master clock cycles. The master reset must be deasserted synchronously to the master clock.

Figure 5-14 shows an example reset synchronizer that fits these requirements.

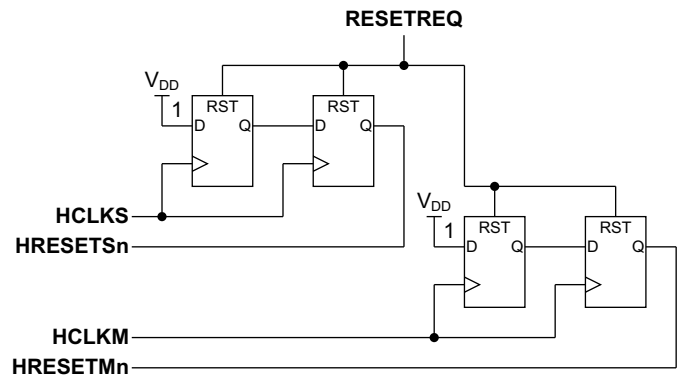


Figure 5-14 Example reset synchronizer

5.5.5 External clock gating using the active signals

The AHB to AHB and APB asynchronous bridge has two outputs, **HACTIVEM** and **PACTIVEM**, that respectively indicate whether the AHB-Lite master interface and APB4 master interface are active.

Use these outputs to control clock gating to the system connected to the bridge AHB-Lite and APB4 master port respectively.

The bridge master clock **HCLKM** must always be driven, regardless of the value of **HACTIVEM** and **PACTIVEM**.

5.5.6 Clock domain crossing

Set clock domain crossing paths through the following signals as false paths to prevent your timing analysis tool from reporting them as violations.

Slave clock domain to master clock domain

- **s_tx_sema_q.**
- **s_hmastlock_q.**
- **m_haddr_q.**
- **m_hmaster_q.**
- **m_hmastlock_q.**
- **m_hprot_q.**
- **m_hselapb_q.**
- **m_hsize_1to0_q.**
- **m_hwdata_q.**
- **m_hwrite_q.**

Master clock domain to slave clock domain

- **m_tx_sema_q.**
- **s_hrdata_q.**
- **s_hresp_q.**

The AHB-Lite to AHB-Lite and APB asynchronous bridge instantiates special modules for clock domain crossing:

- `cmsdk_ahb_to_ahb_apb_async_launch.v.`
- `cmsdk_ahb_to_ahb_apb_async_sample_and_hold.v.`
- `cmsdk_ahb_to_ahb_apb_async_synchronizer.v.`

You must modify these modules to use library-specific cells before you implement your design to ensure correct clock domain crossing operation.

5.6 AHB to AHB synchronous bridge

The following sections describe the AHB to AHB synchronous bridge features and its operation:

- [About the AHB to AHB synchronous bridge.](#)
- [Using AHB to AHB synchronous bridge on page 5-31.](#)
- [Component dependency on page 5-31.](#)

5.6.1 About the AHB to AHB synchronous bridge

The AHB to AHB synchronous bridge, `cmsdk_ahb_to_ahb_sync.v`, provides timing isolation in a large AHB system.

Figure 5-15 shows the AHB to AHB synchronous bridge.

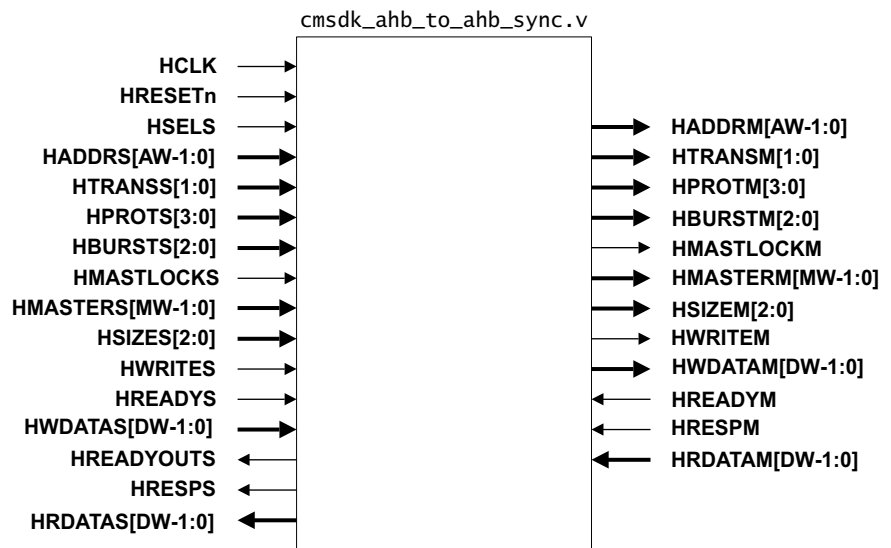


Figure 5-15 AHB to AHB synchronous bridge

Table 5-10 shows the characteristics of the AHB to AHB synchronous bridge.

Table 5-10 AHB to AHB synchronous bridge characteristics

| Element name | Description |
|--------------|---|
| Filename | <code>cmsdk_ahb_to_ahb_sync.v</code> |
| Parameter | The parameters are as follows: |
| AW | Address width of the AHB address bus. This is set to 32 by default. |
| MW | Width of the HMASTER signal. This is set to 4 by default. |
| BURST | Set to: |
| | 0 Does not support burst by default. Burst transfers are converted to single transfers. |
| | 1 Burst support present. |
| DW | Data width. You can configure the width to either 64 bits or 32 bits. This is set to 32 by default. |
| Clock domain | HCLK |

5.6.2 Using AHB to AHB synchronous bridge

Figure 5-16 shows the method to use the AHB to AHB synchronous bridge to isolate timing paths between two AHB segments.

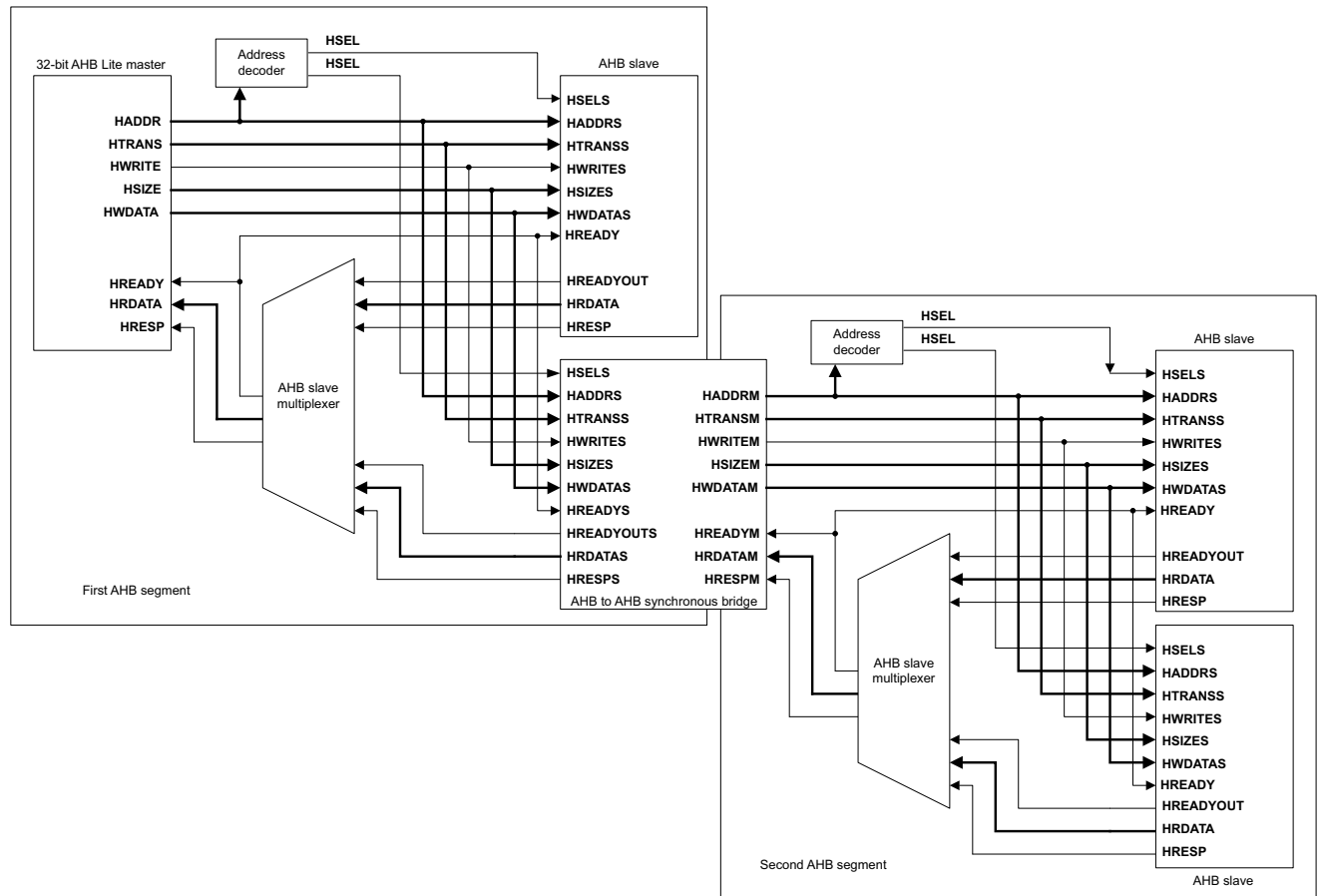


Figure 5-16 Using AHB to AHB synchronous bridge

The AHB to AHB synchronous bridge is useful for designs with multiple FPGAs when the AHB segment is divided into multiple devices. It registers both slave interface signals and master interface signals. When single direction registering is required, you can use either the AHB to AHB sync-up bridge or the AHB to AHB sync-down bridge.

5.6.3 Component dependency

The AHB to AHB synchronous bridge contains one component that is shared with the AHB to AHB sync-down bridge:

`cmsdk_ahb_to_ahb_sync_down/verilog/cmsdk_ahb_to_ahb_sync_error_canc.v.`

When using AHB to AHB synchronous bridge, ensure that either:

- The RTL search path includes `logical/cmsdk_ahb_to_ahb_sync_down/verilog`.
- The component listed above is explicitly included in the project.

5.7 AHB to AHB sync-down bridge

The following subsections describe the AHB to AHB sync-down bridge features and its operation:

- [About the AHB to AHB sync-down bridge.](#)
- [Using the AHB to AHB sync-down bridge on page 5-33.](#)
- [Optional write buffer on page 5-34.](#)
- [Synthesizing the AHB to AHB sync-down bridge on page 5-35.](#)
- [Component dependency on page 5-36.](#)

5.7.1 About the AHB to AHB sync-down bridge

The AHB to AHB sync-down bridge, `cmsdk_ahb_to_ahb_sync_down.v`, syncs-down the AHB domain from a higher frequency to a lower frequency if required. The supported features are as follows:

- Optional write buffer that you can enable or disable using **HPROT**[2].
- Registered address and control path.
- Unregistered feedback path, except error response case.
- **BWERR** for Buffered Write Error, returned to the processor as an interrupt pulse.
- Optional burst support.

Figure 5-17 shows the AHB to AHB sync-down bridge.

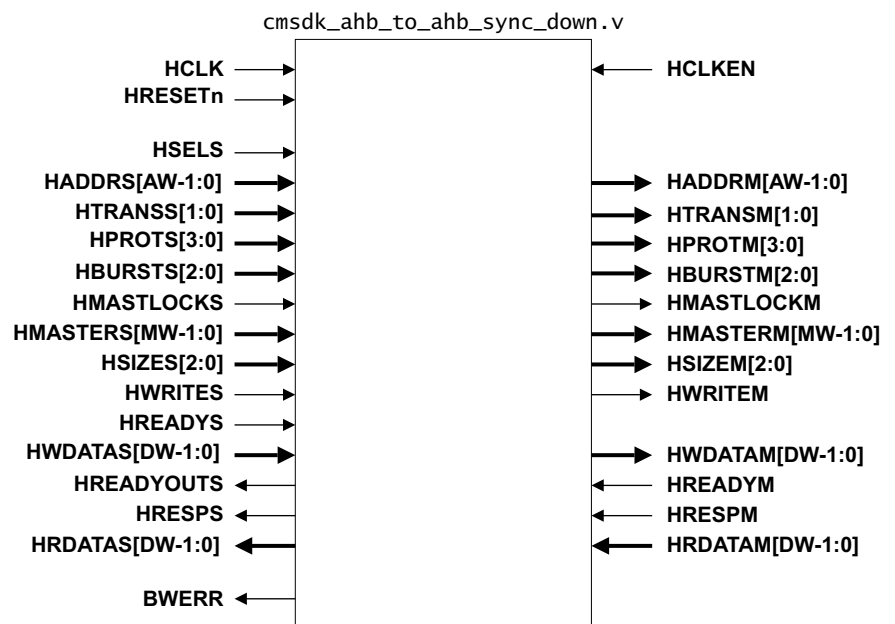


Figure 5-17 AHB to AHB sync-down bridge

You can configure this design as follows:

- If you set the **BURST** parameter to 0, burst transfers are converted into single transfers.
- The **HCLK** signal of the slower AHB domain is divided from the **HCLK** of the faster AHB domain using **HCLKEN**.

Table 5-11 shows the characteristics of the AHB to AHB sync-down bridge.

Table 5-11 AHB to AHB sync-down bridge characteristics

| Element name | Description |
|--------------|--|
| Filename | cmsdk_ahb_to_ahb_sync_down.v |
| Parameter | <p>The parameters are as follows:</p> <p>AW Address width of the AHB address bus. This is set to 32 by default.</p> <p>MW Width of the HMASTER signal. This is set to 1 by default.</p> <p>DW Data width. You can configure the width to either 64 bits or 32 bits. This is set to 32 by default.</p> <p>BURST When set to 0, burst transactions are converted into single transactions. When set to 1, burst transactions are supported. Removing burst support reduces the gate count. This is set to 0 by default.</p> <p>WB Write buffer support. Set to 1 to include a single-level write buffer. Set to 0 to remove write buffer. This is set to 0 by default.</p> |
| Clock domain | HCLK |

If a buffered write error occurs, it generates a single-cycle **BWERR** pulse in the fast **HCLK** domain.

5.7.2 Using the AHB to AHB sync-down bridge

The AHB to AHB sync-down bridge has an **HCLKEN** signal that is used to indicate the clock division between the fast AHB and the slow AHB. Figure 5-18 shows the clock divide operation for a 3 to 1 clock divide ratio.

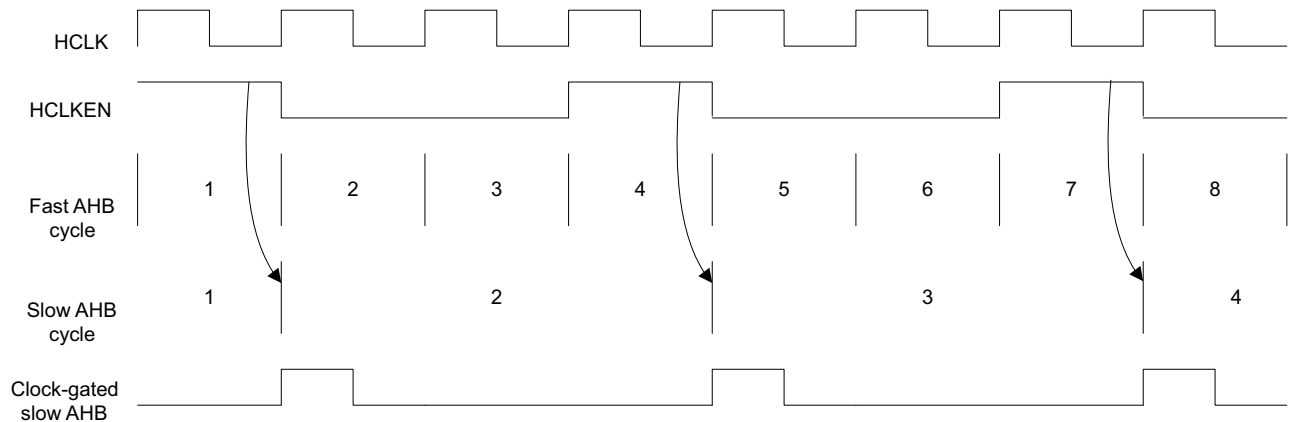


Figure 5-18 Clock divide operation

The **HCLKEN** must be synchronous to **HCLK**. It can be generated by a programmable counter. The slow AHB bus can use **HCLKEN** as a clock gating control to generate a clock-gated slow AHB which is used by the components connected to the slower AHB. See Figure 5-19 on page 5-34.

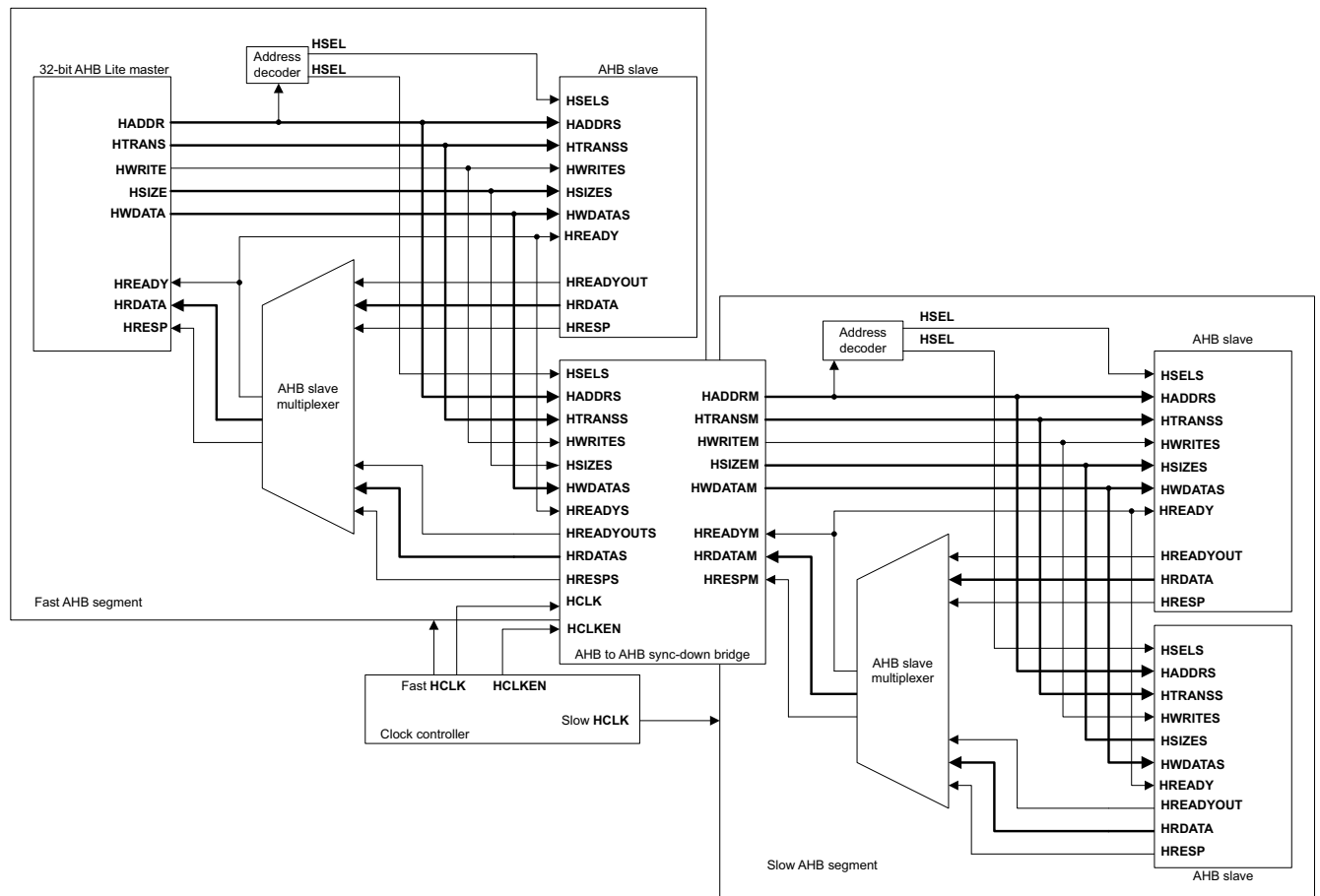


Figure 5-19 Using AHB to AHB sync-down bridge

If **HCLKEN** is tied HIGH, the two AHB segments have the same operation speed. However, the signals flowing from master to slave direction are registered so there is an additional latency cycle.

5.7.3 Optional write buffer

The optional write buffer of the AHB to AHB sync-down bridge can reduce the wait state, when writing a data to a peripheral or to a memory location in the slower clock domain. Without the write buffer, the bus master must wait till the transfer gets complete in the slower AHB. If the write buffer is present and the transfer is a bufferable write, then the write buffer returns an OKAY response to the bus master, and the actual write operation is performed later on the downstream AHB.

Write transfers with **HPROT[2]** set HIGH are considered bufferable, and read transfers are considered non-bufferable.

If the bus bridge is used for connecting peripherals, with some of the registers being non-bufferable, you can edit the top-level file of the bus bridge to include an address decoding logic to disable write buffering of these registers. For example, the peripheral registers for clearing interrupt request must be non-bufferable.

If the bus bridge is used in a multiprocessor design with a bus level exclusive access monitor, exclusive stores must be handled as non-bufferable.

The optional write buffer is one-level deep. If the bus master issues another transfer to the same AHB segment while the previous buffer write is on-going, then the transfer request is reserved on hold in the write buffer, and wait states are inserted. When the downstream AHB completes the previous transfer, the held transfer is sent to the downstream AHB. If the held transfer is a bufferable write, then the write buffer returns OKAY response to the AHB master issuing the transfer.

5.7.4 Synthesizing the AHB to AHB sync-down bridge

The AHB to AHB sync-down bridge contains combinational paths from the slow AHB domain to the fast AHB domain because of the unregistered feedback path feature. These signals include **HRDATA**, and **HREADYOUT**. The unregistered feedback path permits lower access latency, but few precautions are required when synthesizing the system.

The following precautions are required during the system synthesis:

- The synthesis of the AHB to AHB sync-down bridge is targeted at the frequency of the faster AHB. The design has only one **HCLK** input, so you must connect the input to the fast AHB clock signal.
- If the synthesis of the slow AHB and the fast AHB system are carried out separately, the timing constraints of the signals from the slow AHB to the bridge are controlled properly to prevent timing violation.

For example, when synthesizing the slow clock domain logic, if the synthesis process is partitioned into fast clock domain and slow clock domain, then you must setup the timing constraints of the read data signal, **HRDATA** based on the overall timing path. See [Figure 5-20](#).

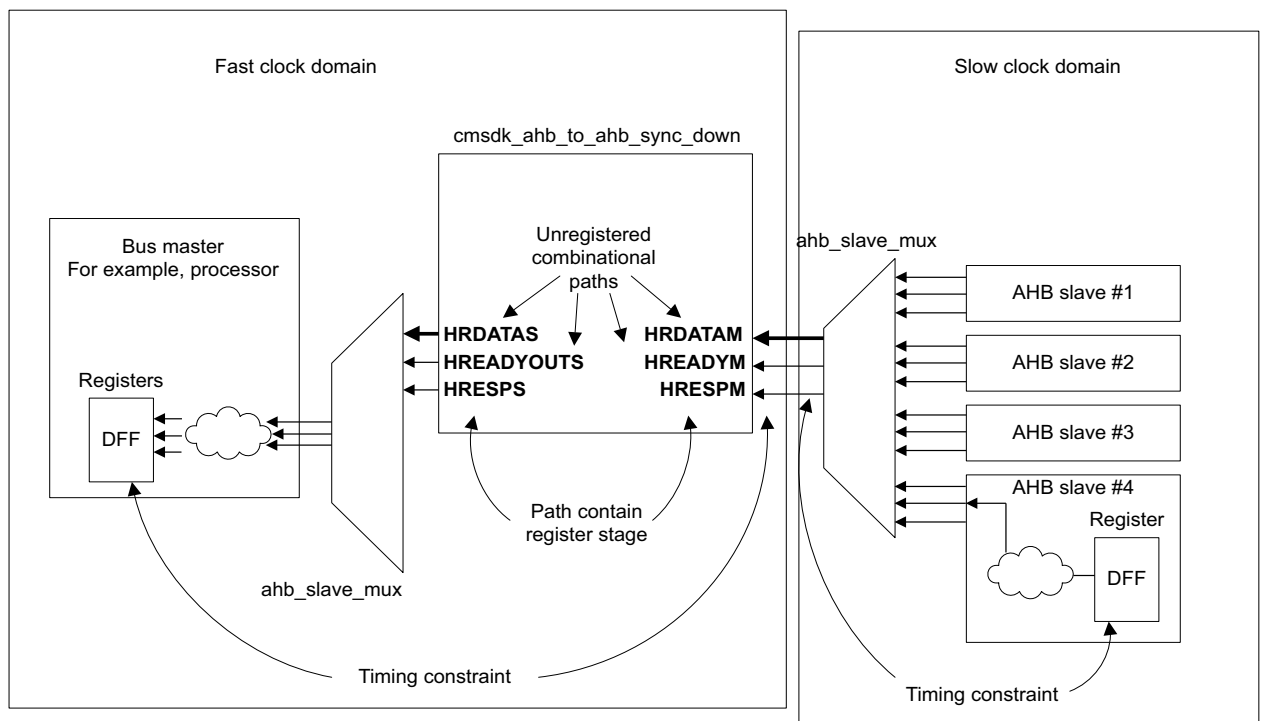


Figure 5-20 Synthesizing the AHB to AHB sync-down bridge

The simplest arrangement is to restrict the whole path to just one fast **HCLK** cycle. If this is not possible, you can use a multi-cycle path. You might have to mask the **HRDATAS** output from the AHB to AHB sync-down bridge to 0, when **HREADYOUTS** is LOW, to prevent metastability issues.

Similar handling is required for the **HREADYOUT** and **HRESP** from the slow clock domain. There is an unregistered combinational path for the **HREADYOUT** connection in the AHB to AHB sync-down bridge. The **HRESPS** output from bridge is generated from registered logic within the bridge, but timing constraints are still required to ensure that there is no timing violation from **HRESP** source in the slow clock domain to the registers within the AHB to AHB sync-down bridge.

5.7.5 Component dependency

The AHB to AHB sync-down bridge contains two subcomponents that are shared with other components:

- `cmsdk_ahb_to_ahb_sync_down/verilog/cmsdk_ahb_to_ahb_sync_error_canc.v`, shared with the AHB to AHB sync-up bridge and the AHB to AHB synchronous bridge.
- `cmsdk_ahb_to_ahb_sync_down/verilog/cmsdk_ahb_to_ahb_sync_wb.v`, shared with the AHB to AHB sync-up bridge.

As a result, if you modify these subcomponents, it can affect the other AHB components.

5.8 AHB to AHB sync-up bridge

The following subsections describe the AHB to AHB sync-down bridge features and its operation:

- [Overview of the AHB to AHB sync-up bridge.](#)
- [Using the AHB to AHB sync-up bridge on page 5-38.](#)
- [Synthesizing the AHB to AHB sync-up bridge on page 5-39.](#)
- [Component dependency on page 5-40.](#)

5.8.1 Overview of the AHB to AHB sync-up bridge

The AHB to AHB sync-up bridge, `cmsdk_ahb_to_ahb_sync_up.v`, enables you to sync-up the AHB from a lower frequency to a higher frequency if required. The supported features are as follows:

- Optional write buffer, you can enable or disable the buffering using **HPROT**[2].
- Unregistered address and control path.
- Registered feedback path.
- **BWERR** for Buffered Write Error, return to processor as interrupt pulse.
- Optional burst support.

Figure 5-21 shows the AHB to AHB sync-up bridge module.

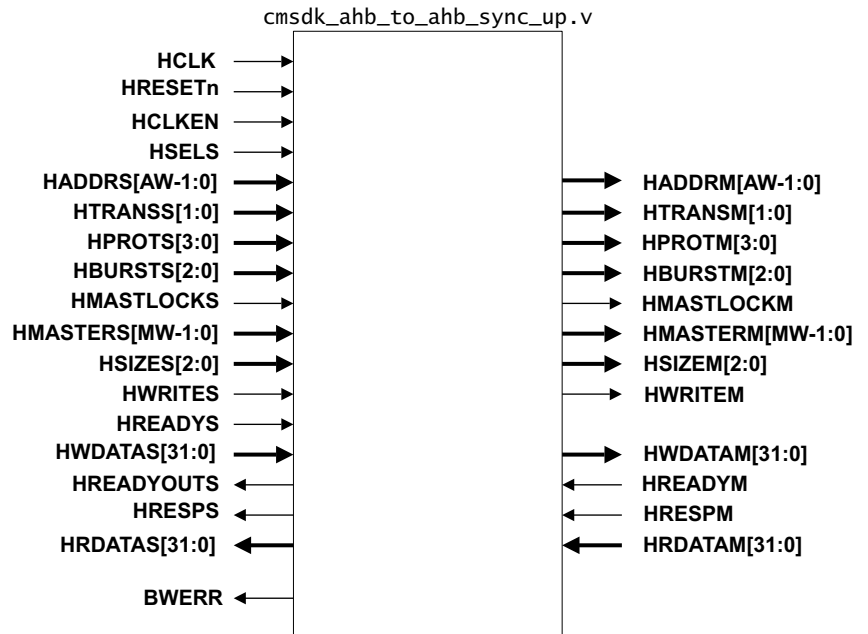


Figure 5-21 AHB to AHB sync-up bridge

You can configure this design as follows:

- If you set the **BURST** parameter to 0, the burst transfers are converted into single transfers.
- The **HCLK** signal of the slower AHB domain is divided from the **HCLK** of the faster AHB domain using **HCLKEN**.

Table 5-12 shows the characteristics of the AHB to AHB sync-up bridge.

Table 5-12 AHB to AHB sync-up bridge characteristics

| Element name | Description |
|--------------|---|
| Filename | cmsdk_ahb_to_ahb_sync_up.v |
| Parameter | <p>The parameters are as follows:</p> <p>AW Address width of the AHB address bus. This is set to 32 by default.</p> <p>MW Width of the HMASTER signal. This is set to 1 by default.</p> <p>DW Data width. You can configure the width to either 64 bits or 32 bits. This is set to 32 by default.</p> <p>BURST When set to 0, burst transactions are converted into single transactions. When set to 1, burst transactions are supported. Removing burst support reduces the gate count. This is set to 0 by default.</p> <p>WB Write buffer support. Set to 1 to include a single-level write buffer. Set to 0 to remove write buffer. This is set to 0 by default.</p> |
| Clock domain | HCLK |

If a buffered write error occurs, it generates a single-cycle pulse **BWERR** in the fast **HCLK** domain. For more information on the optional write buffer, see [Optional write buffer on page 5-34](#).

5.8.2 Using the AHB to AHB sync-up bridge

The AHB to AHB sync-up bridge has a **HCLKEN** signal which is used to indicate the clock division between the fast AHB and the slow AHB. [Figure 5-22](#) shows the clock divide operation for a 3-1 clock divide ratio.

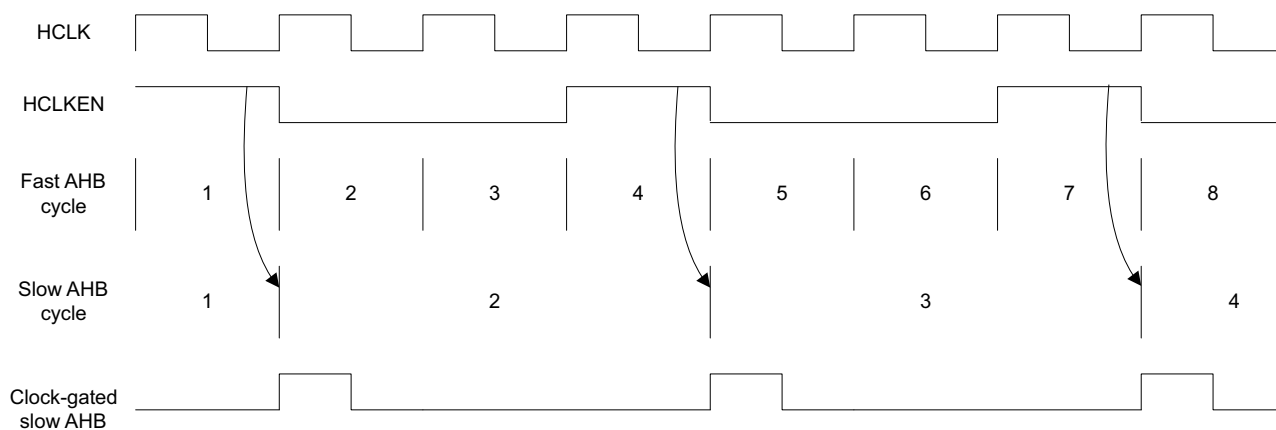


Figure 5-22 Clock divide operation

The **HCLKEN** must be synchronous to **HCLK**. It can be generated by a programmable counter. The slow AHB bus can use **HCLKEN** as a clock gating control to generate a clock-gated slow AHB which is used by the components connected to the slower AHB.

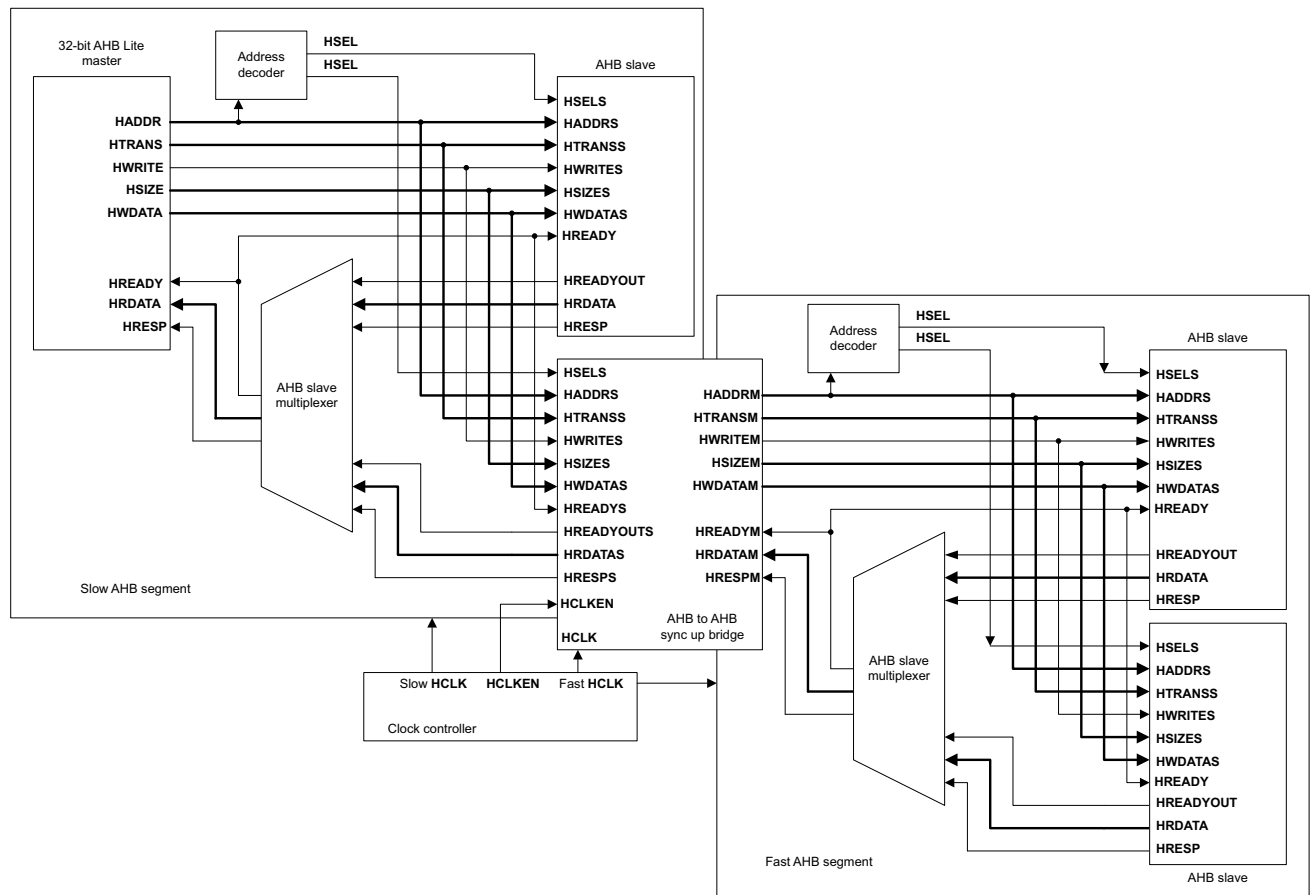


Figure 5-23 Using AHB to AHB sync-up bridge

If **HCLKEN** is tied HIGH, the two AHB segments have the same operation speed. However, the signals flowing from slaves to master direction are registered so there is an additional latency cycle.

For more information on write buffer option, see [Optional write buffer on page 5-34](#).

5.8.3 Synthesizing the AHB to AHB sync-up bridge

The AHB to AHB sync-up bridge contains combinational address, and control paths from the slow AHB domain to the fast AHB domain. These signals include **HADDRM**, **HTRANS**, **HSIZEM**, **HWRITEM**, **HPROTM**, **HMASTERM**, **HMASTLOCKM**, and **HWDATAM**. The unregistered paths permits lower access latency, but few precautions are required when synthesizing the system. See [Synthesizing the AHB to AHB sync-down bridge on page 5-35](#) for precautions.

For example, if the synthesis process is partitioned into fast clock domain and slow clock domain, when synthesizing the slow clock domain logic, you must setup the timing constraints of these signals based on the overall timing path so that they will not cause timing violation in the fast clock domain as [Figure 5-24 on page 5-40](#) shows.

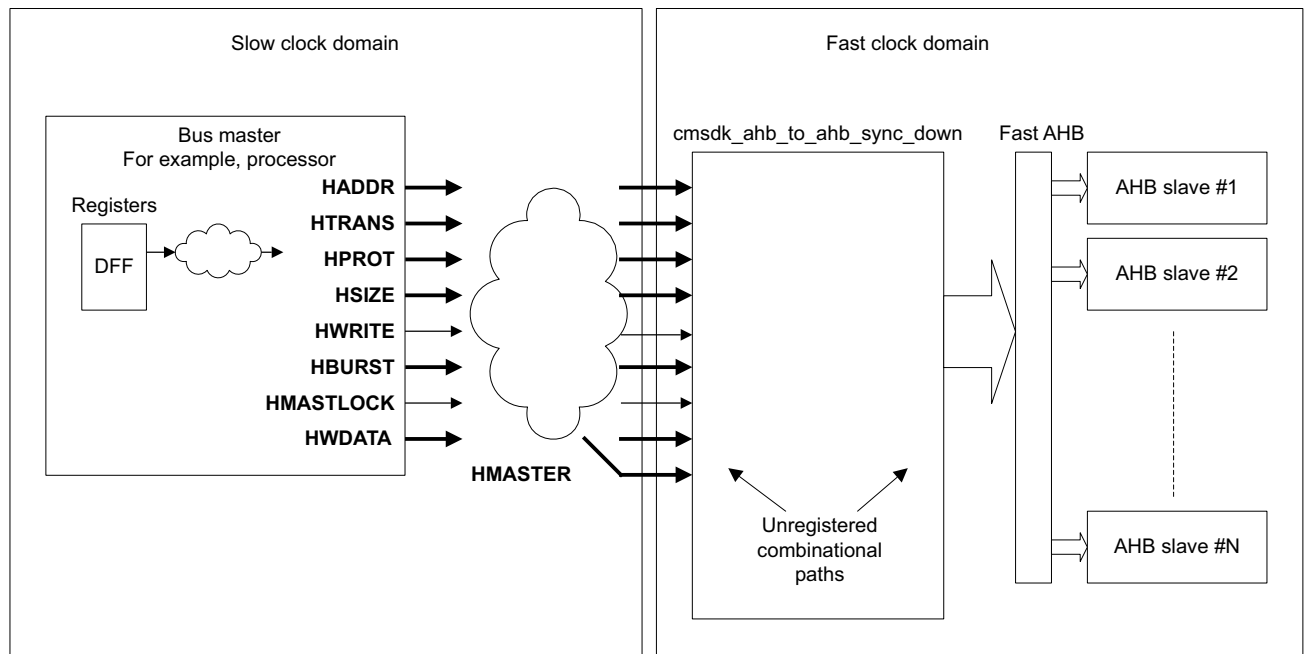


Figure 5-24 Combinational paths from slow AHB to fast AHB

5.8.4 Component dependency

The AHB to AHB sync-up bridge contains two components that are shared with the AHB to AHB sync-down bridge:

- cmsdk_ahb_to_ahb_sync_down/verilog/cmsdk_ahb_to_ahb_sync_error_canc.v.
- cmsdk_ahb_to_ahb_sync_down/verilog/cmsdk_ahb_to_ahb_sync_wb.v.

When using AHB to AHB sync up bridge, ensure that either:

- The RTL search path includes logical/cmsdk_ahb_to_ahb_sync_down/verilog.
- The two components listed above are explicitly included in the project.

Chapter 6

Behavioral Memory Models

This chapter describes the behavioral models in the Cortex-M System Design Kit. It contains the following sections:

- *ROM model wrapper* on page 6-2.
- *RAM model wrapper* on page 6-6.
- *Behavioral SRAM model with AHB interface* on page 6-10.
- *32-bit flash ROM behavioral model* on page 6-11.
- *16-bit flash ROM behavioral model* on page 6-12.
- *FPGA SRAM synthesizable model* on page 6-13.
- *FPGA ROM* on page 6-14.
- *External asynchronous 8-bit SRAM* on page 6-15.
- *External asynchronous 16-bit SRAM* on page 6-16.

6.1 ROM model wrapper

The AHB ROM wrapper model, `cmsdk_ahb_rom.v`, permits easy switching between the following implementations of ROM:

- None.
- Behavioral ROM model, using behavioral SRAM with write disabled.
- SRAM model with an AHB SRAM interface module, suitable for FPGA flow, and permitting read and write operations
- Flash wrapper with simple 32-bit flash memory.
- Flash wrapper with simple 16-bit flash memory.

[Table 6-1](#) shows the characteristics of the ROM model wrapper.

Table 6-1 ROM model wrapper characteristics

| Element name | Description |
|--------------|--|
| Filename | <code>cmsdk_ahb_rom.v</code> |
| Parameters | <p>The parameters are as follows:</p> <p><code>MEM_TYPE</code> The <code>MEM_TYPE</code> value ranges from 0 to 4. See Table 6-2.</p> <p><code>AW</code> Address width. The default value is 16.</p> <p><code>filename</code> File name for memory image.</p> <p><code>WS_N</code> First access or NONSEQUENTIAL access wait state. The default value is 0.</p> <p><code>WS_S</code> Sequential access wait state. The default value is 0.</p> <p><code>BE</code> Big-endian. The default value is 0, little-endian. Set the value to 1 for big-endian configuration.</p> <p>Note</p> <p>This parameter is a placeholder to permit the possibility of propagating endian configuration to a specific memory implementation. Currently, it is not used by existing memory implementations that this design kit provides.</p> |
| Clock domain | HCLK |

[Table 6-2](#) shows the configuration of the `cmsdk_ahb_rom.v`, that a `MEM_TYPE` Verilog parameter defines.

Table 6-2 Configuration of `cmsdk_ahb_rom.v`

| <code>MEM_TYPE</code> | Design of <code>cmsdk_ahb_rom.v</code> | Configurability | |
|---|--|----------------------------|--|
| 0, <code>AHB_ROM_NONE</code> | See Figure 6-1 on page 6-3 | None. | |
| 1, <code>AHB_ROM_BEH_MODEL</code> | See Figure 6-2 on page 6-3 | <code>AW</code> | Address width. |
| | | <code>filename</code> | File name for memory image. |
| | | <code>WS_N</code> | First access or NONSEQUENTIAL access wait state. |
| | | <code>WS_S</code> | Sequential access wait state. |
| 2, <code>AHB_ROM_FPGA_SRAM_MODEL</code> | See Figure 6-3 on page 6-4 | <code>AW</code> | Address width. |
| | | <code>filename</code> | File name for memory image. |
| | | Always in zero wait state. | |

Table 6-2 Configuration of cmsdk_ahb_rom.v (continued)

| MEM_TYPE | Design of cmsdk_ahb_rom.v | Configurability | |
|--------------------------|--|------------------------|---|
| 3, AHB_ROM_FLASH32_MODEL | See Figure 6-4 on page 6-4 | AW filename WS_N | Address width. File name for memory image. First access or NONSEQUENTIAL access wait state. |
| 4, AHB_ROM_FLASH16_MODEL | See Figure 6-5 on page 6-5 | AW filename WS_N | Address width. File name for memory image. First access or NONSEQUENTIAL access wait state. |

[Figure 6-1](#) shows the design of cmsdk_ahb_rom.v for AHB_ROM_NONE.

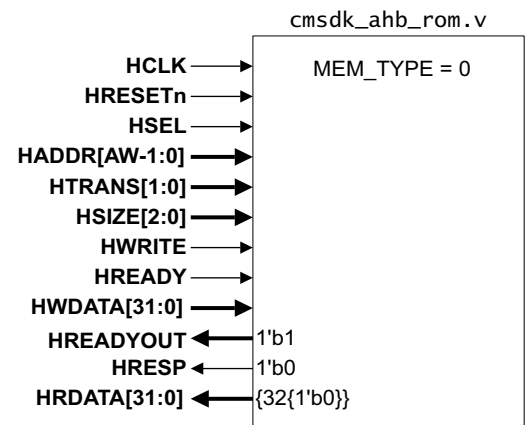


Figure 6-1 Design of cmsdk_ahb_rom.v for AHB_ROM_NONE

[Figure 6-2](#) shows the design of cmsdk_ahb_rom.v for AHB_ROM_BEH_MODEL.

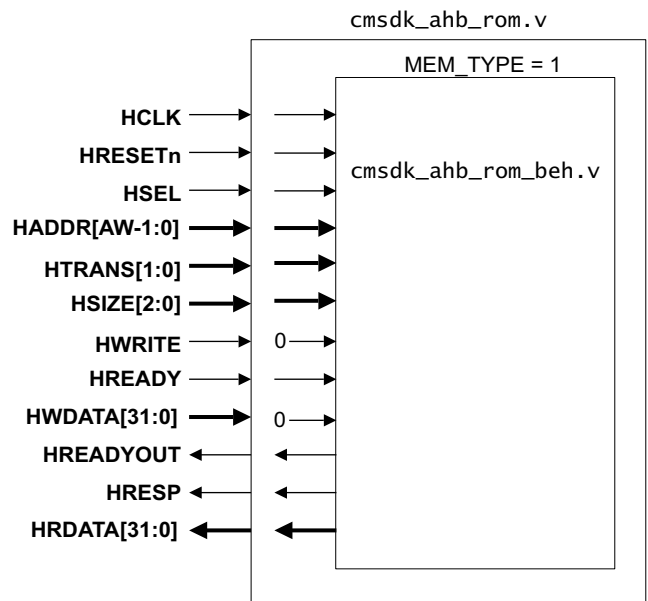


Figure 6-2 Design of cmsdk_ahb_rom.v for AHB_ROM_BEH_MODEL

[Figure 6-3 on page 6-4](#) shows the design of cmsdk_ahb_rom.v for AHB_ROM_FPGA_SRAM_MODEL.

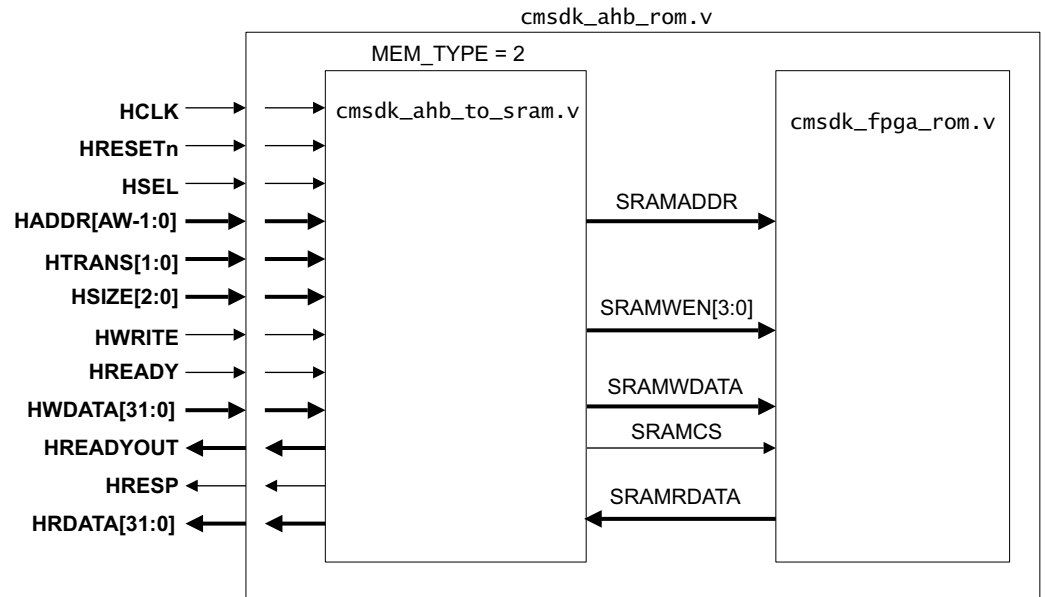


Figure 6-3 Design of cmsdk_ahb_rom.v for AHB_ROM_FPGA_SRAM_MODEL

Figure 6-4 shows the design of cmsdk_ahb_rom.v for AHB_ROM_FLASH32_MODEL.

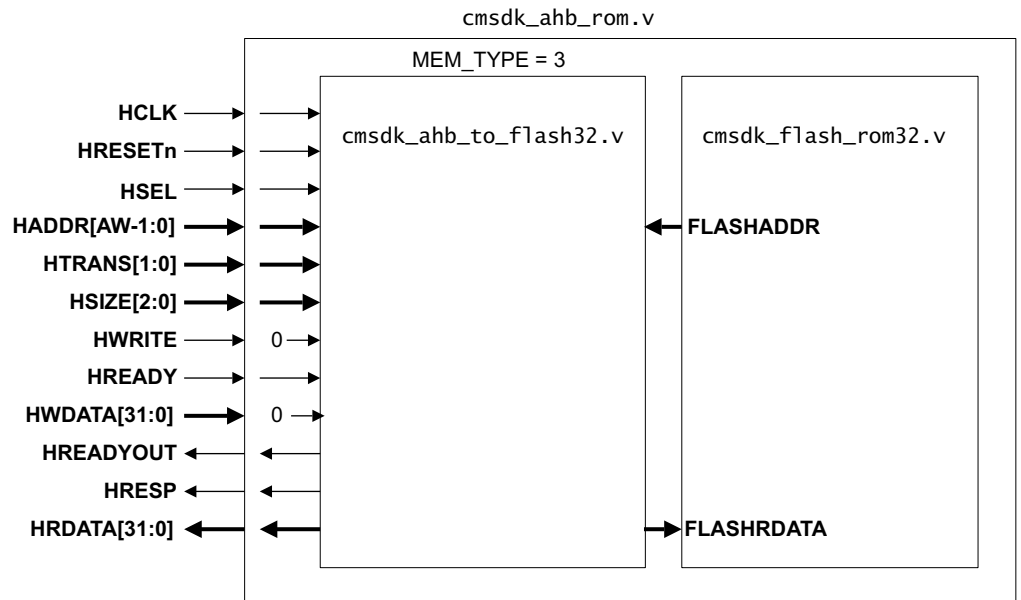


Figure 6-4 Design of cmsdk_ahb_rom.v for AHB_ROM_FLASH32_MODEL

Figure 6-5 on page 6-5 shows the design of cmsdk_ahb_rom.v for AHB_ROM_FLASH16_MODEL. This design requires a downsizer module, cm0p_32to16_dnsizer.v, which is supplied with the integration kit of the Cortex-M0+ processor.

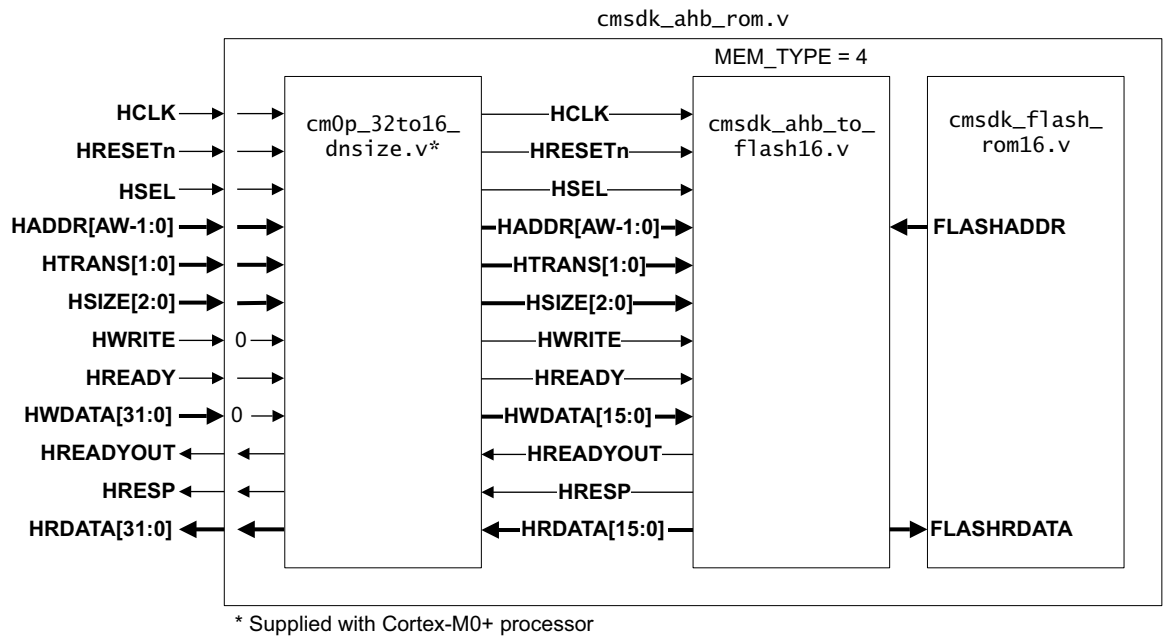


Figure 6-5 Design of cmsdk_ahb_rom.v for AHB_ROM_FLASH16_MODEL

6.2 RAM model wrapper

The AHB RAM wrapper model, `cmsdk_ahb_ram.v`, permits easy switching between the following implementations of RAM:

- Behavioral RAM model.
- SRAM model with an AHB SRAM interface module, suitable for ASIC or FPGA flow.

Table 6-3 shows the characteristics of the RAM model wrapper.

Table 6-3 RAM model wrapper characteristics

| Element name | Description | | | | | | | | | | | | |
|-----------------------|--|-----------------------|--|-----------------|--|-----------------------|-----------------------------|-------------------|--|-------------------|---|-----------------|---|
| Filename | <code>cmsdk_ahb_ram.v</code> | | | | | | | | | | | | |
| Parameter | The parameters are as follows: <table> <tr> <td><code>MEM_TYPE</code></td><td>The <code>MEM_TYPE</code> value ranges from 0 to 4. See Table 6-4.</td></tr> <tr> <td><code>AW</code></td><td>Width of the address bus. The default value is 16.</td></tr> <tr> <td><code>filename</code></td><td>File name for memory image.</td></tr> <tr> <td><code>WS_N</code></td><td>First access or NONSEQUENTIAL access wait state. The default value is 0.</td></tr> <tr> <td><code>WS_S</code></td><td>Sequential access wait state. The default value is 0.</td></tr> <tr> <td><code>BE</code></td><td>Big-endian. The default value is 0, little-endian. Set the value to 1 for big-endian configuration.</td></tr> </table> | <code>MEM_TYPE</code> | The <code>MEM_TYPE</code> value ranges from 0 to 4. See Table 6-4. | <code>AW</code> | Width of the address bus. The default value is 16. | <code>filename</code> | File name for memory image. | <code>WS_N</code> | First access or NONSEQUENTIAL access wait state. The default value is 0. | <code>WS_S</code> | Sequential access wait state. The default value is 0. | <code>BE</code> | Big-endian. The default value is 0, little-endian. Set the value to 1 for big-endian configuration. |
| <code>MEM_TYPE</code> | The <code>MEM_TYPE</code> value ranges from 0 to 4. See Table 6-4. | | | | | | | | | | | | |
| <code>AW</code> | Width of the address bus. The default value is 16. | | | | | | | | | | | | |
| <code>filename</code> | File name for memory image. | | | | | | | | | | | | |
| <code>WS_N</code> | First access or NONSEQUENTIAL access wait state. The default value is 0. | | | | | | | | | | | | |
| <code>WS_S</code> | Sequential access wait state. The default value is 0. | | | | | | | | | | | | |
| <code>BE</code> | Big-endian. The default value is 0, little-endian. Set the value to 1 for big-endian configuration. | | | | | | | | | | | | |
| Clock domain | HCLK | | | | | | | | | | | | |

Table 6-4 shows the configuration of `cmsdk_ahb_ram.v`, that a `MEM_TYPE` Verilog parameter defines.

Table 6-4 Configuration of `cmsdk_ahb_ram.v`

| MEM_TYPE | Design of cmsdk_ahb_ram.v | Configurability | |
|-----------------------------|---|--|--|
| 0, AHB_RAM_NONE | See Figure 6-6 on page 6-7 | None | |
| 1, AHB_RAM_BEH_MODEL | See Figure 6-7 on page 6-7 | AW filename WS_N WS_S | Address width. File name for memory image. First access or NONSEQUENTIAL access wait state. Sequential access wait state. |
| 2, AHB_RAM_FPGA_SRAM_MODEL | See Figure 6-8 on page 6-8 | AW filename Always in zero wait state. | Address width. File name for memory image. |
| 3, AHB_RAM_EXT_SRAM16_MODEL | See Figure 6-9 on page 6-8 | AW WS_N | Address width. Additional wait state for read, write, and turnaround cycle. |
| 4, AHB_RAM_EXT_SRAM8_MODEL | See Figure 6-10 on page 6-9 | AW WS_N | Address width. Additional wait state for read, write, and turnaround cycle. |

Figure 6-6 on page 6-7 shows the design of `cmsdk_ahb_ram.v` for `AHB_RAM_NONE`.

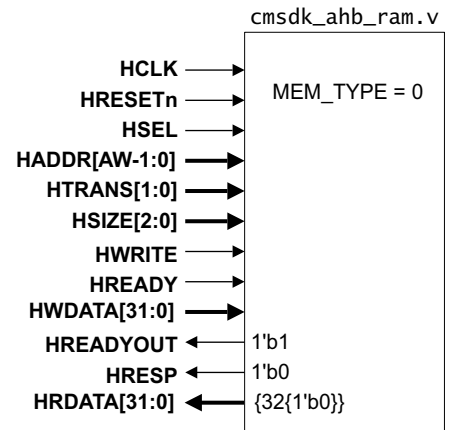


Figure 6-6 Design of cmsdk_ahb_ram.v for AHB_RAM_NONE

Figure 6-7 shows the design of cmsdk_ahb_ram.v for AHB_RAM_BEH_MODEL.

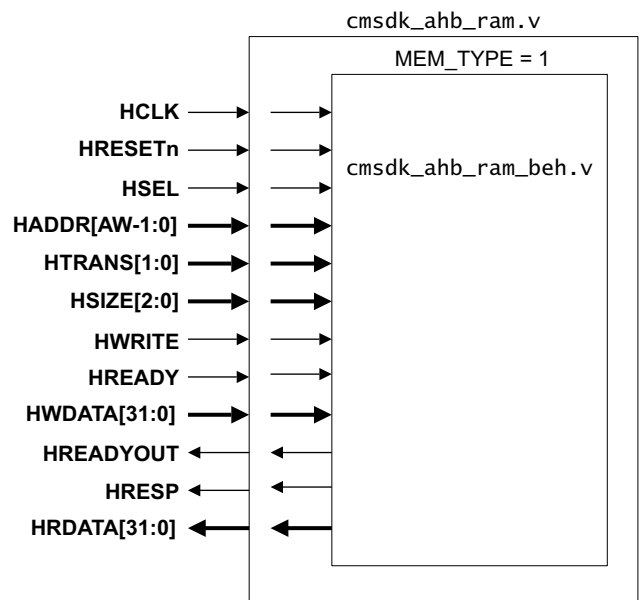


Figure 6-7 Design of cmsdk_ahb_ram.v for AHB_RAM_BEH_MODEL

Figure 6-8 on page 6-8 shows the design of cmsdk_ahb_ram.v for AHB_FPGA_SRAM_MODEL.

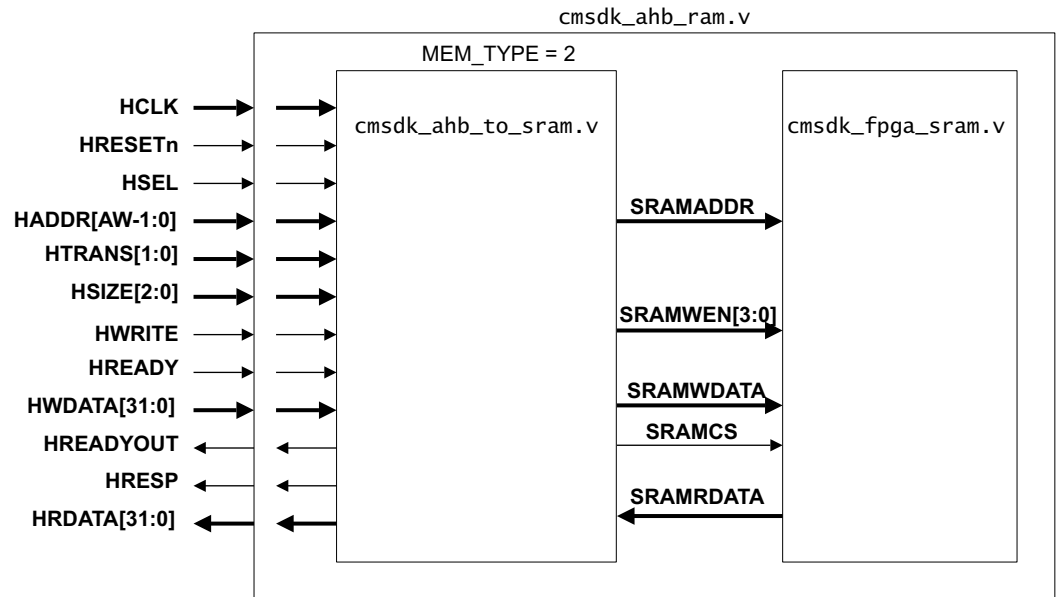


Figure 6-8 Design of cmsdk_ahb_ram.v for AHB_RAM_FPGA_SRAM_MODEL

Figure 6-9 shows the design of cmsdk_ahb_ram.v for AHB_RAM_EXT_SRAM16_MODEL.

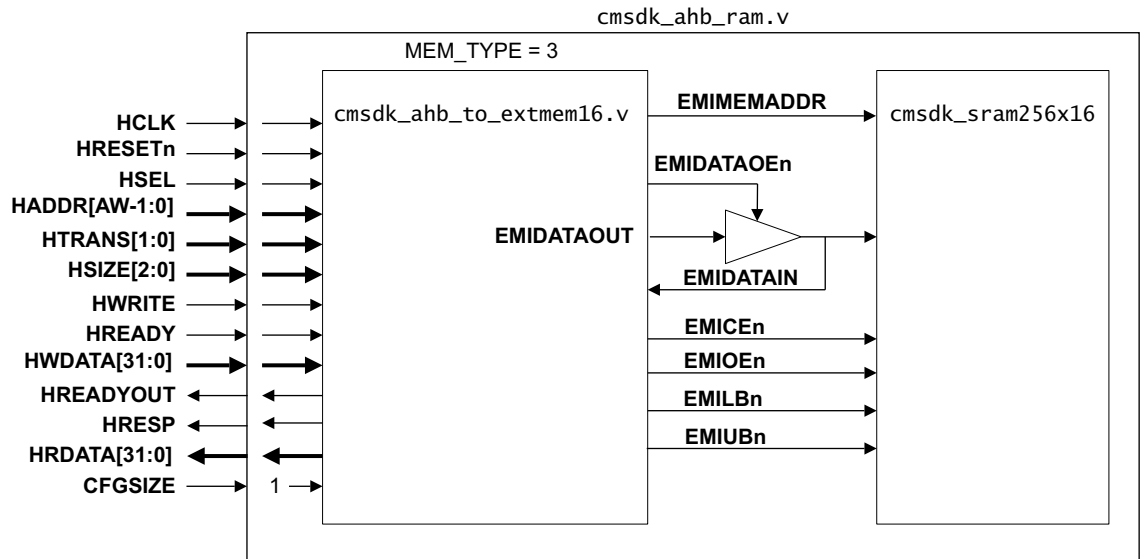


Figure 6-9 Design of cmsdk_ahb_ram.v for AHB_RAM_EXT_SRAM16_MODEL

Figure 6-10 on page 6-9 shows the design of cmsdk_ahb_ram.v for AHB_RAM_EXT_SRAM8_MODEL.

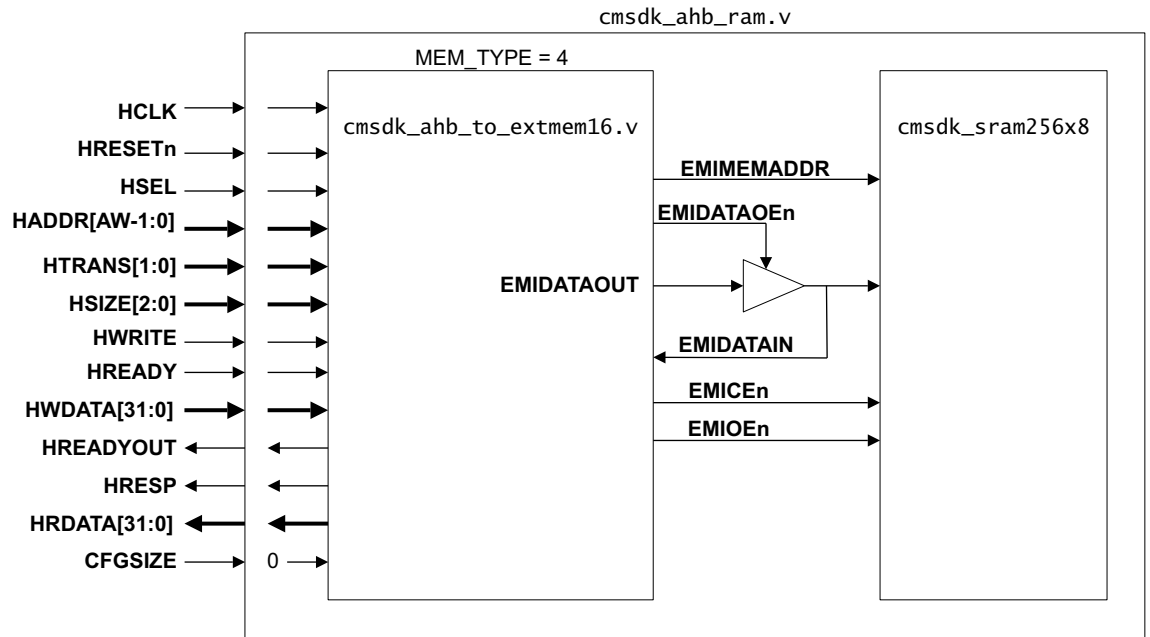


Figure 6-10 Design of `cmsdk_ahb_ram.v` for `AHB_RAM_EXT_SRAM8_MODEL`

The `AHB_RAM_EXT_SRAM16_MODEL` and `AHB_RAM_EXT_SRAM8_MODEL` options are not illustrations of real system designs. The external memory interface module, `cmsdk_ahb_to_extmem16`, normally connects to off-chip memory. In the case of `cmsdk_ahb_ram.v`, these configuration options permit you to perform benchmarking with your own external memory model.

6.3 Behavioral SRAM model with AHB interface

The behavioral AHB SRAM model, `cmsdk_ahb_ram_beh.v`, enables you to define an initial memory image and generate wait states for both NONSEQUENTIAL and SEQUENTIAL transfers. The model always replies with an OKAY response. Figure 6-11 shows the behavioral SRAM model with AHB interface.

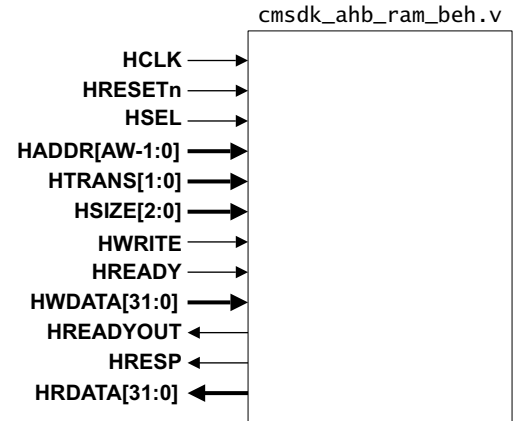


Figure 6-11 Behavioral SRAM model with AHB interface

Table 6-5 shows the characteristics of the behavioral SRAM model with an AHB interface.

Table 6-5 Behavioral SRAM model with an AHB interface characteristics

| Element name | Description | | | | | | | | |
|--------------|---|----|---|----------|-----------------------------|------|--|------|---|
| Filename | <code>cmsdk_ahb_ram_beh.v</code> | | | | | | | | |
| Parameter | <p>The parameters are as follows:</p> <table> <tr> <td>AW</td><td>Address width. The default value is 16.</td></tr> <tr> <td>filename</td><td>File name for memory image.</td></tr> <tr> <td>WS_N</td><td>First access or NONSEQUENTIAL access wait state. The default value is 0.</td></tr> <tr> <td>WS_S</td><td>Sequential access wait state. The default value is 0.</td></tr> </table> | AW | Address width. The default value is 16. | filename | File name for memory image. | WS_N | First access or NONSEQUENTIAL access wait state. The default value is 0. | WS_S | Sequential access wait state. The default value is 0. |
| AW | Address width. The default value is 16. | | | | | | | | |
| filename | File name for memory image. | | | | | | | | |
| WS_N | First access or NONSEQUENTIAL access wait state. The default value is 0. | | | | | | | | |
| WS_S | Sequential access wait state. The default value is 0. | | | | | | | | |
| Clock domain | HCLK | | | | | | | | |

6.4 32-bit flash ROM behavioral model

The 32-bit flash ROM behavioral model, `cmsdk_flash_rom32.v`, is a simple behavioral model for 32-bit flash memory. Reset and clock signals perform wait state behavioral modeling.

Figure 6-12 shows the 32-bit flash ROM behavioral model.

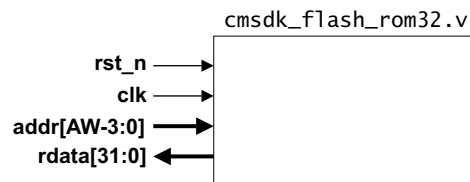


Figure 6-12 32-bit flash ROM behavioral model

Table 6-6 shows the characteristics of the 32-bit flash ROM behavioral model.

Table 6-6 32-bit flash ROM behavioral model characteristics

| Element name | Description | | | | | | |
|--------------|--|----|--|----------|-----------------------------|----|--|
| Filename | <code>cmsdk_flash_rom32.v</code> | | | | | | |
| Parameter | <p>The parameters are as follows:</p> <table> <tr> <td>AW</td><td>Width of address bus. The default value is 16.</td></tr> <tr> <td>filename</td><td>File name for memory image.</td></tr> <tr> <td>WS</td><td>Wait state or required read cycle. The default value is 0.</td></tr> </table> | AW | Width of address bus. The default value is 16. | filename | File name for memory image. | WS | Wait state or required read cycle. The default value is 0. |
| AW | Width of address bus. The default value is 16. | | | | | | |
| filename | File name for memory image. | | | | | | |
| WS | Wait state or required read cycle. The default value is 0. | | | | | | |
| Clock domain | <p>CLK. In the example system provided, CLK is the same as HCLK.</p> <hr/> <p>Note</p> <p>The connections for clocks and resets are for the wait state modeling.</p> | | | | | | |

6.4.1 Signal descriptions

Table 6-7 shows the signal descriptions of the 32-bit flash ROM behavioral model.

Table 6-7 32-bit flash ROM behavioral model signals

| Signal | Type | Description |
|---------------------------|--------|-------------|
| <code>addr[AW-3:0]</code> | Input | Address |
| <code>rdata[31:0]</code> | Output | Read data |

6.5 16-bit flash ROM behavioral model

The 16-bit flash ROM behavioral model, `cmsdk_flash_rom16.v`, is a simple behavioral model for 16-bit flash memory. Reset and clock signals perform wait state behavioral modeling.

Figure 6-13 shows the 16-bit flash ROM behavioral model.

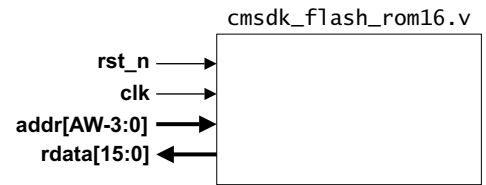


Figure 6-13 16-bit flash ROM behavioral model

Table 6-8 shows the characteristics of the 16-bit flash ROM behavioral model.

Table 6-8 16-bit flash ROM behavioral model characteristics

| Element name | Description | | | | | | |
|--------------|--|----|--|----------|-----------------------------|----|--|
| Filename | <code>cmsdk_flash_rom16.v</code> | | | | | | |
| Parameter | <p>The parameters are as follows:</p> <table> <tr> <td>AW</td><td>Width of address bus. The default value is 16.</td></tr> <tr> <td>filename</td><td>File name for memory image.</td></tr> <tr> <td>WS</td><td>Wait state or required read cycle. The default value is 0.</td></tr> </table> | AW | Width of address bus. The default value is 16. | filename | File name for memory image. | WS | Wait state or required read cycle. The default value is 0. |
| AW | Width of address bus. The default value is 16. | | | | | | |
| filename | File name for memory image. | | | | | | |
| WS | Wait state or required read cycle. The default value is 0. | | | | | | |
| Clock domain | <p>CLK. In the example system provided, CLK is the same as HCLK.</p> <hr/> <p>Note</p> <p>The connections for clocks and resets are for the wait state modeling.</p> | | | | | | |

6.5.1 Signal descriptions

Table 6-9 shows the signal descriptions of the 16-bit flash ROM behavioral model.

Table 6-9 16-bit flash ROM behavioral model signals

| Signal | Type | Description |
|---------------------|--------|-------------|
| ADDR[AW-3:0] | Input | Address |
| RDATA[15:0] | Output | Read data |

6.6 FPGA SRAM synthesizable model

The FPGA SRAM synthesizable model, `cmsdk_fpga_sram.v`, is a model for SRAM that behaves like simple synchronous RAM in ASIC or FPGA. This memory does not have an AHB interface, and the AHB to SRAM interface module is required to connect this memory to AHB. It demonstrates how to use the AHB to SRAM interface module. This model is suitable for FPGA synthesis. Figure 6-14 shows the SRAM synthesizable model.

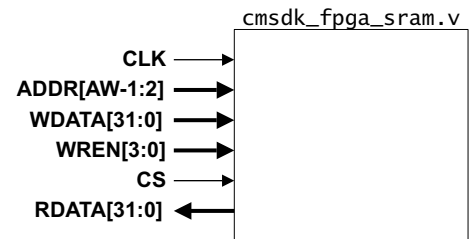


Figure 6-14 FPGA SRAM

Table 6-10 shows the characteristics of the FPGA SRAM model.

Table 6-10 FPGA SRAM characteristics

| Element name | Description |
|--------------|---|
| Filename | <code>cmsdk_fpga_sram.v</code> |
| Parameter | <code>AW</code> Address width. The default value is 16. |
| Clock domain | <code>CLK</code> |

6.6.1 Signal descriptions

Table 6-11 shows the FPGA SRAM signals.

Table 6-11 FPGA SRAM signals

| Signal | Type | Description |
|---------------------------|--------|--------------|
| <code>ADDR[AW-1:2]</code> | Input | Address |
| <code>WDATA[31:0]</code> | Input | Write data |
| <code>WREN[3:0]</code> | Input | Write enable |
| <code>CS</code> | Input | Chip select |
| <code>RDATA[31:0]</code> | Output | Read data |

6.7 FPGA ROM

The FPGA ROM, `cmsdk_fpga_rom.v`, is a model that behaves like a simple synchronous RAM in FPGA. This memory does not have an AHB interface, and the AHB to SRAM interface module is required to connect this memory to AHB. It demonstrates how to use the AHB to SRAM interface module. This module is suitable for FPGA synthesis. Unlike the FPGA SRAM model, this model contains initialization of memory. [Figure 6-15](#) shows the FPGA ROM model.

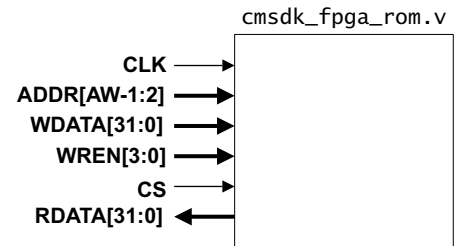


Figure 6-15 FPGA ROM

[Table 6-12](#) shows the characteristics of the FPGA ROM.

Table 6-12 FPGA ROM characteristics

| Element name | Description |
|--------------|---|
| Filename | <code>cmsdk_fpga_rom.v</code> |
| Parameter | The parameters are as follows: <i>AW</i> Address width. <i>filename</i> File name for memory image. |
| Clock domain | CLK . In the example system provided, CLK is same as HCLK . |

6.7.1 Signal descriptions

[Table 6-13](#) shows the FPGA ROM signals.

Table 6-13 FPGA ROM signals

| Signal | Type | Description |
|---------------------|--------|--------------|
| ADDR[AW-1:2] | Input | Address |
| WDATA[31:0] | Input | Write data |
| WREN[3:0] | Input | Write enable |
| CS | Input | Chip select |
| RDATA[31:0] | Output | Read data |

6.8 External asynchronous 8-bit SRAM

The external asynchronous 8-bit SRAM, `cmsdk_sram256x8.v`, is a behavioral model for external 8-bit SRAM that behaves like a typical asynchronous SRAM component. This memory model is compatible with the AHB to external memory interface. Figure 6-16 shows the external asynchronous 8-bit SRAM.

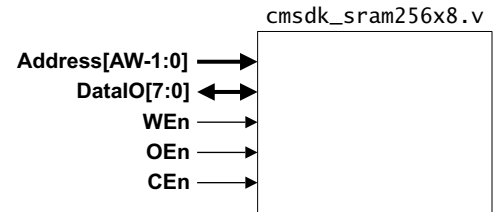


Figure 6-16 External asynchronous 8-bit SRAM

Table 6-14 shows the external asynchronous 8-bit SRAM characteristics.

Table 6-14 External asynchronous 8-bit SRAM characteristics

| Element name | Description |
|--------------|---|
| Filename | <code>cmsdk_sram256x8.v</code> |
| Parameter | The parameters are as follows: <div> <div>AW</div> <div>Address width. The default value is 18.</div> </div> <div> <div>filename</div> <div>File name for initial SRAM content. Default to NULL that means no initial memory image.</div> </div> |
| Clock domain | Not applicable. |

6.8.1 Signal descriptions

Table 6-15 shows the external asynchronous 8-bit SRAM signals.

Table 6-15 External asynchronous 8-bit SRAM signals

| Signal | Type | Description |
|---------------------|---------------|---|
| ADDR[AW-1:0] | Input | Address. |
| DATAIO[7:0] | Bidirectional | Data. |
| WEn | Input | Write enable. Active LOW for write operation. |
| OEn | Input | Output enable. Active LOW for read operations. |
| CEn | Input | Chip enable. Active LOW for both read and write operations. |

6.9 External asynchronous 16-bit SRAM

The external asynchronous 16-bit SRAM, `cmsdk_sram256x16.v`, is a behavioral model for external 16-bit SRAM that behaves like a typical asynchronous SRAM component. This memory model is compatible with the AHB to external memory interface. Figure 6-17 shows the external asynchronous 16-bit SRAM.

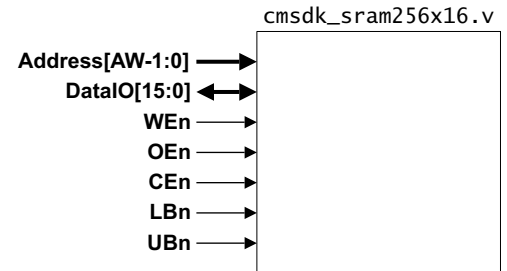


Figure 6-17 External asynchronous 16-bit SRAM

Table 6-16 shows the external asynchronous 16-bit SRAM characteristics.

Table 6-16 External asynchronous 16-bit SRAM characteristics

| Element name | Description | | | | |
|-----------------------|--|-----------------|---|-----------------------|---|
| Filename | <code>cmsdk_sram256x16.v</code> | | | | |
| Parameter | The parameters are as follows: <table border="0"> <tr> <td><code>AW</code></td><td>Address width. The default value is 18.</td></tr> <tr> <td><code>filename</code></td><td>File name for initial SRAM content. The default is NULL and this means no initial memory image.</td></tr> </table> | <code>AW</code> | Address width. The default value is 18. | <code>filename</code> | File name for initial SRAM content. The default is NULL and this means no initial memory image. |
| <code>AW</code> | Address width. The default value is 18. | | | | |
| <code>filename</code> | File name for initial SRAM content. The default is NULL and this means no initial memory image. | | | | |
| Clock domain | Not applicable. | | | | |

6.9.1 Signal descriptions

Table 6-17 shows the external asynchronous 16-bit SRAM signals.

Table 6-17 External asynchronous 16-bit SRAM signals

| Signal | Type | Description |
|---------------------|---------------|---|
| ADDR[AW-1:0] | Input | Address. |
| DATAIO[15:0] | Bidirectional | Data. |
| WEn | Input | Write enable. Active LOW for write operation. |
| OEn | Input | Output enable. Active LOW for read operations. |
| CEn | Input | Chip enable. Active LOW for both read and write operations. |
| LBn | Input | Lower byte enable. Active LOW. |
| UBn | Input | Upper byte enable. Active LOW. |

Chapter 7

Verification Components

This chapter describes the verification components in the Cortex-M System Design Kit. It contains the following sections:

- [*AHB-Lite protocol checker on page 7-2.*](#)
- [*APB protocol checker on page 7-5.*](#)
- [*AHB FRBM on page 7-7.*](#)

7.1 AHB-Lite protocol checker

The AHB-Lite protocol checker, `AhbLitePC.v`, is a simulation model that you can use to detect bus protocol violations during simulation. The AHB-Lite protocol checker is located in `logical/models/protocol_checkers/AhbLitePC/verilog`. The design is based on OVL. To use the AHB-Lite protocol checker, you must download the OVL Verilog library from Accellera, and add the OVL library path in the include and search paths of your simulator setup. [Figure 7-1](#) shows the AHB-Lite protocol checker.

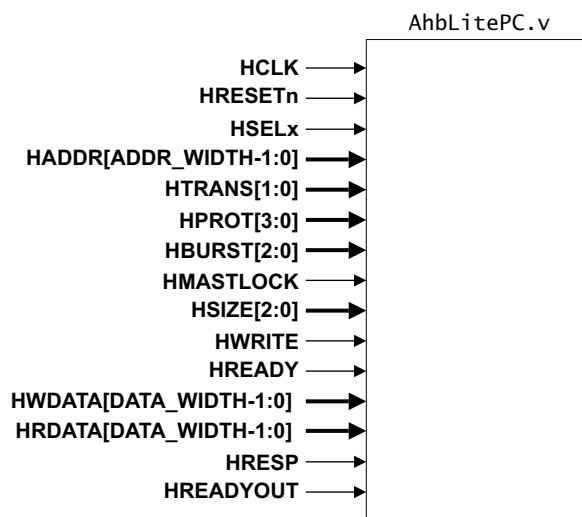


Figure 7-1 AHB-Lite protocol checker

The example systems and a number of components already contain usage examples of the AHB-Lite protocol checker. The instantiation of AHB-Lite protocol checker is conditional in the examples, only when you set the `ARM_AHB_ASSERT_ON` compile directive. When using the AHB-Lite protocol checker in your design, you can use the compiler directive of your choice. Use of conditional compilation is required because the AHB-Lite protocol checker is not a synthesizable component.

When an AHB-Lite bus protocol violation is detected, error or warning messages are shown in the console or transcript window of the simulator.

[Table 7-1](#) shows the characteristics of the AHB-Lite protocol checker.

Table 7-1 AHB-Lite protocol checker characteristics

| Element name | Description |
|--------------|---|
| Filename | <code>AhbLitePC.v</code> |
| Parameters | See Table 7-2 on page 7-3 |
| Clock domain | HCLK |

Table 7-2 shows the Verilog parameters and their descriptions.

Table 7-2 AHB-Lite Verilog parameter descriptions

| Parameter | Description |
|--------------------------------------|--|
| ADDR_WIDTH | Width of the address bus, from 1 to 32 bits. The default value is 32 bits. |
| DATA_WIDTH | Width of the data bus, from 1 to 32 bits. The default value is 32 bits. |
| BIG_ENDIAN | Data endianness: 0 Little-endian. This is the default. 1 Big-endian. |
| MASTER_TO_INTERCONNECT | Where in the system the protocol checker is instantiated. Mainly specifies the existence of functional HSEL and HREADYOUT : 0 Between interconnect master interface and AHB-Lite slave. HSEL and HREADYOUT functional. This is the default. 1 Between AHB-Lite master and interconnect slave interface. HSEL and HREADYOUT non-functional. |
| EARLY_BURST_TERMINATION | Interconnect master interface capable of early burst termination: 0 No. This is the default. 1 Yes. |
| MASTER_REQUIREMENT_PROPTYPE | Property types: 0 Assert. This is the default. 1 Assume. 2 Ignore. |
| MASTER_RECOMMENDATION_PROPTYPE | |
| MASTER_XCHECK_PROPTYPE | |
| SLAVE_REQUIREMENT_PROPTYPE | |
| SLAVE_RECOMMENDATION_PROPTYPE | |
| SLAVE_XCHECK_PROPTYPE | |
| INTERCONNECT_REQUIREMENT_PROPTYPE | |
| INTERCONNECT_RECOMMENDATION_PROPTYPE | |
| INTERCONNECT_XCHECK_PROPTYPE | |

Table 7-3 shows the use of the property type parameters.

Table 7-3 Use of property type parameters

| Type of verification environment | MASTER_*_PROPTYPE | SLAVE_*_PROPTYPE | INTERCONNECT_*_PROPTYPE |
|----------------------------------|--|---|--|
| Simulation-based verification | 0 (assert) to check master-generated signals | 0 (assert) to check slave-generated signals | 0 (assert) to check interconnect-generated signals |

Table 7-3 Use of property type parameters (continued)

| Type of verification environment | MASTER_*_PROPTYPE | SLAVE_*_PROPTYPE | INTERCONNECT_*_PROPTYPE |
|--|--|---|--|
| Formal verification of an AHB-Lite master | 0 (assert) to check master-generated output signals | 1 (assume) to constrain slave-generated input signals | 1 (assume) to constrain interconnect-generated input signals |
| Formal verification of an AHB-Lite interconnect master interface that generates HSEL and HREADY output signals | 0 (assert) to check master-generated output signals | 1 (assume) to constrain slave-generated input signals | 0 (assert) to check interconnect-generated output signals |
| Formal verification of an AHB-Lite slave | 1 (assume) to constrain master-generated input signals | 0 (assert) to check slave-generated output signals | 1 (assume) to constrain interconnect-generated input signals |

———— **Note** —————

The AHB-Lite protocol checker is intended to be used to assist the detection of common AHB-Lite operation issues. It does not provide definitive checking of all bus protocol violation scenarios and does not provide all the constraints that formal verification requires.

7.2 APB protocol checker

The APB protocol checker, `ApbPC.v`, is a simulation model that you can use to detect bus protocol violations during simulation. The `ApbPC.v` supports APB2, APB3, and APB4. The APB protocol checker is located in `logical/models/protocol_checkers/ApbPC/verilog`.

To use the APB protocol checker, you must download the OVL Verilog library from Accellera, and add the OVL library path in the include and search paths of your simulator setup.

The example systems, and a number of components, already contain usage examples of the APB protocol checker in the AHB to APB bridge. The instantiation of the APB protocol checker is conditional in the examples, only when you set the `ARM_APB_ASSERT_ON` compile directive. When using the APB protocol checker in your design, you can use a compile directive of your choice. Use of conditional compilation is required because the APB protocol checker is not a synthesizable component.

Figure 7-2 shows the APB protocol checker.

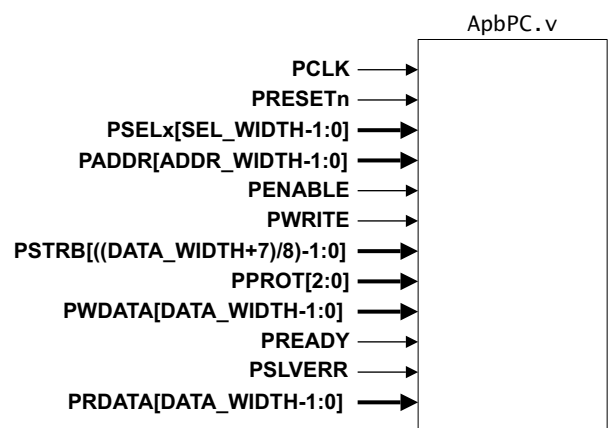


Figure 7-2 APB protocol checker

When an APB bus protocol violation is detected, the transcript window of the simulator shows error or warning messages.

Table 7-4 shows the characteristics of the APB protocol checker.

Table 7-4 APB protocol checker characteristics

| Element name | Description |
|--------------|---|
| Filename | <code>ApbPC.v</code> |
| Parameters | See Table 7-5 on page 7-6 |
| Clock domain | PCLK |

Table 7-5 shows the Verilog parameters and their descriptions.

Table 7-5 APB Verilog parameter descriptions

| Parameter | Description |
|-----------------------------|--|
| ADDR_WIDTH | Width of the PADDR bus, from 1 to 32 bits. The default value is 32 bits. |
| DATA_WIDTH | Width of the PxDATA bus, from 1 to 32 bits. The default value is 32 bits. |
| SEL_WIDTH | Width of the PSELx bus, 1 bit or more. The default value is 1 bit. |
| MASTER_REQUIREMENT_PROPTYPE | Property types: |
| SLAVE_REQUIREMENT_PROPTYPE | 0 Assert. This is the default. 1 Assume. 2 Ignore. |
| PREADY_FUNCTIONAL | Optional signals: |
| PSLVERR_FUNCTIONAL | 0 Not functional. 1 Functional. This is the default. |
| PPROT_FUNCTIONAL | |
| PSTRB_FUNCTIONAL | |

Table 7-6 shows the use of the property type parameters.

Table 7-6 Use of property type parameters

| Type of verification environment | MASTER_REQUIREMENT_PROPTYPE | SLAVE_REQUIREMENT_PROPTYPE |
|--------------------------------------|--|---|
| Simulation-based verification | 0 (assert) to check master-generated signals | 0 (assert) to check slave-generated signals |
| Formal verification of an APB master | 0 (assert) to check master-generated output signals | 1 (assume) to constrain slave-generated input signals |
| Formal verification of an APB slave | 1 (assume) to constrain master-generated input signals | 0 (assert) to check slave-generated output signals |

Note

The APB protocol checker is intended to be used to assist the detection of common APB operation issues. It does not provide definitive checking of all bus protocol violation scenarios and does not provide all the constraints that formal verification requires.

7.3 AHB FRBM

The AHB FRBM, `cmsdk_ahb_fileread_master32.v` and `cmsdk_ahb_fileread_master64.v`, enable designers to simulate AHB systems quickly and efficiently by using them to generate explicit bus transfers. The FRBM can operate in the Cortex-M System Design Kit with or without an ARM core present. The FRBM and its perl script are compatible with the version in the FRBM in ADK.

The FRBM is a generic AHB *Bus Functional Model* (BFM) that directly controls bus activity by interpreting a stimulus file. The FRBM facilitates the efficient validation of blocks or systems.

The 64-bit FRBM, `cmsdk_ahb_fileread_master64.v`, is split into the following parts:

- AHB-Lite file reader.
- AHB-Lite to AHB wrapper.

The 32-bit FRBM, `cmsdk_ahb_fileread_master32.v`, has an additional part, the funnel, that converts 32-bit transfers on a 64-bit bus to 32-bit transfers on a 32-bit bus.

The AHB-Lite file reader can:

- Perform all AHB burst types at data widths of 8, 16, 32, and 64 bits.
- Insert BUSY states during bursts.
- Perform idle transfers.
- Compare received data with the expected data and report the differences during simulations.

The stimulus file controls the AHB-Lite file reader at simulation run time. It does not have a slave interface, and therefore other AHB masters cannot address it.

You must transform the human-readable input stimulus file to a data file in Verilog hexadecimal format using the `fm2conv.pl` preprocessor script.

The FRBM is designed so that, wherever possible, RTL code is used to describe its logic. All RTL code is written for synthesis using pragmas where necessary, to enable the block to pass through synthesis tools.

Figure 7-3 shows the 32-bit AHB FRBM.

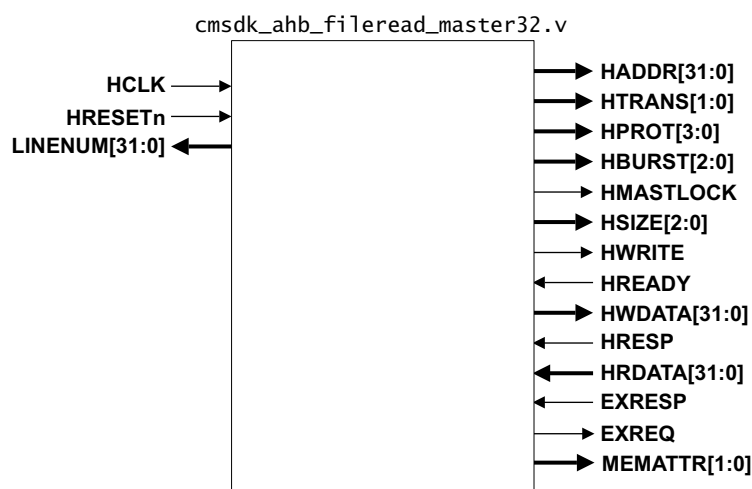


Figure 7-3 32-bit AHB FRBM

Figure 7-4 shows the 64-bit AHB FRBM.

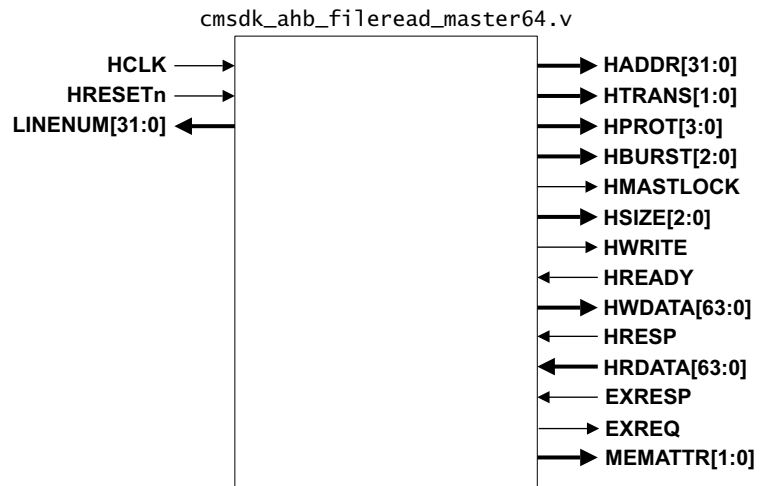


Figure 7-4 64-bit AHB FRBM

Table 7-7 shows the characteristics of the FRBM.

Table 7-7 FRBM characteristics

| Element | Description | | | | | | |
|---------------|--|---------------|---|------------|--|---------------|--|
| Filenames | cmsdk_ahb_fileread_master32.v and cmsdk_ahb_fileread_master64.v | | | | | | |
| Parameters | <p>The parameters are as follows:</p> <table> <tr> <td>InputFileName</td><td>Stimulus file. The default value is filestim.m2d.</td></tr> <tr> <td>MessageTag</td><td>Tag on each FileReader message. The default value is FileReader.</td></tr> <tr> <td>StimArraySize</td><td>Stimulus data array size. The default value is 5000.</td></tr> </table> | InputFileName | Stimulus file. The default value is filestim.m2d. | MessageTag | Tag on each FileReader message. The default value is FileReader. | StimArraySize | Stimulus data array size. The default value is 5000. |
| InputFileName | Stimulus file. The default value is filestim.m2d. | | | | | | |
| MessageTag | Tag on each FileReader message. The default value is FileReader. | | | | | | |
| StimArraySize | Stimulus data array size. The default value is 5000. | | | | | | |
| Clock domain | HCLK | | | | | | |

7.3.1 Programmers model

The cmsdk_ahb_fileread_master uses the following Verilog parameters:

| | |
|---------------|--|
| InputFileName | This is the name of the stimulus data file to be read. If the file is not found, simulation is aborted. The default file name is filestim.m2d. |
| MessageTag | A string that is prepended to all stimulation messages from this cmsdk_ahb_fileread_master. You can use this tag to differentiate between messages from multiple FRBMs in a system. The default message tag is FileReader. |
| StimArraySize | The size, in words, of the internal array used to store the stimulus data. This value has a direct effect on the simulation startup time and memory requirement. This value is large enough to store the whole data file. If the data file is larger than the array, simulation is aborted. The default value is 5000. |

The following sections describe the AHB-Lite FRBM functions:

- [Write command on page 7-9.](#)
- [Read command on page 7-9.](#)
- [Sequential command on page 7-10.](#)

- [Busy command on page 7-11.](#)
- [Idle command on page 7-12.](#)
- [Poll command on page 7-13.](#)
- [Loop command on page 7-14.](#)
- [Resp field on page 7-14.](#)
- [Clock and reset on page 7-14.](#)
- [Error reporting at runtime on page 7-15.](#)
- [End of stimulus on page 7-15.](#)

Write command

The write command, **W**, starts a write burst and one or more **S** vectors can follow it. For bursts of fixed length, the Burst field determines the number of **S** vectors. For bursts of undefined length, there can be any number of **S** vectors as long as they do not cause the address to cross a 1KB boundary.

Figure 7-5 shows the write command timing.

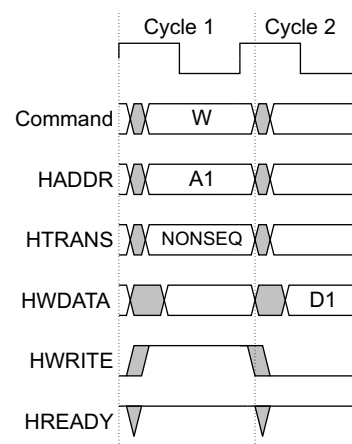


Figure 7-5 Write command timing

The write command operates as follows:

- Cycle 1** The file reader sets up the control signals from the command and asserts **HWRITE**. **HTRANS** is **NONSEQ** to indicate the first transfer of the burst. The Data field is stored and ready to be driven during the data phase. If **HREADY** is asserted, the file reader proceeds to the second control phase.
- Cycle 2** This is the first data phase in which the data is driven for the previous cycle. Unless the Burst field specifies a single transfer, the file reader calculates the next address based on the Size and Burst values.

Read command

The read command, **R**, starts a read burst and one or more **S** vectors can follow it. For bursts of fixed length, the Burst field determines the number of **S** vectors. For bursts of undefined length, there can be any number of **S** vectors as long as they do not cause the address to cross a 1KB boundary.

Figure 7-6 on page 7-10 shows the read command timing.

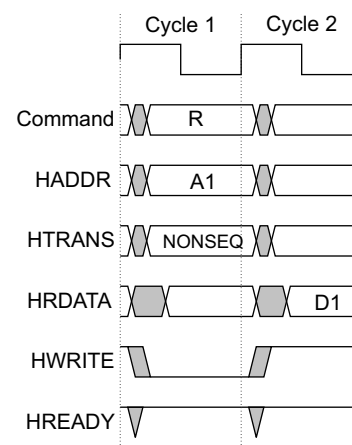


Figure 7-6 Read command timing

The read command operates as follows:

- Cycle 1** The file reader sets up the control signals from the command and deasserts **HWRITE**. **HTRANS** is **NONSEQ** to indicate the first transfer of a burst. If **HREADY** is asserted, the file reader proceeds to the second control phase.
- Cycle 2** The data read for the previous cycle is compared with the Data field after applying the mask and byte lane strobes. Any differences are reported to the simulation environment. Unless the Burst field indicates a single transfer, the file reader calculates the next address based on the Size and Burst values.

Sequential command

The sequential command, **S**, is a vector that provides data for a single beat within the burst. The file reader calculates the required address. A sequential command is valid when a read or write command starts a burst transfer.

Figure 7-7 shows the sequential command timing.

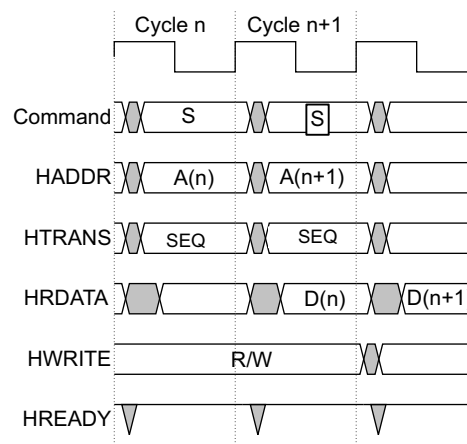


Figure 7-7 Sequential command timing

A sequential command is valid when a read or write command starts a burst transfer and operates as follows:

Cycle n The file reader drives the calculated address, and **HTRANS** is SEQ to indicate the remaining transfers of the burst.

If **HREADY** is asserted, the file reader proceeds to the second control phase.

Cycle n + 1 In a write burst, the file reader drives the Data field data and ignores the Mask field.

In a read burst, the file reader applies the Mask and Bstrb fields to the input data and then compares the Data field with the input data. The file reader reports differences between the expected data and the read data to the simulation environment.

Busy command

The busy command, B, inserts either a BUSY transfer or a BUSY cycle, depending on the Wait field. A busy command is valid when a read or write command starts a burst transfer.

During a burst with the Wait field not specified, the busy command inserts a single **HCLK** BUSY transfer on the AHB. A burst can have a busy command after its last transfer while the master determines whether it has another transfer to complete.

Figure 7-8 shows the busy command timing.

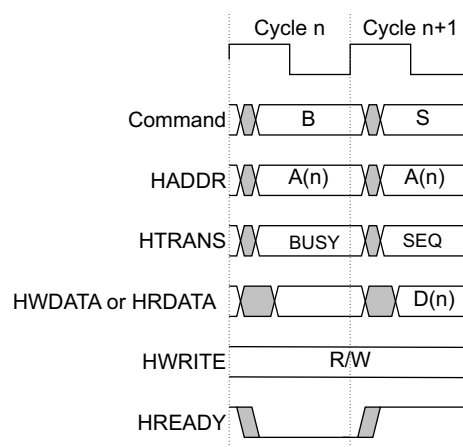


Figure 7-8 Busy transfer timing

Cycle n The file reader drives the calculated address and **HTRANS** is BUSY.

Cycle n + 1 The file reader proceeds to the next control phase. Data is ignored.

During a burst with the Wait field specified, the busy command inserts a complete AHB transfer. See [Figure 7-9 on page 7-12](#).

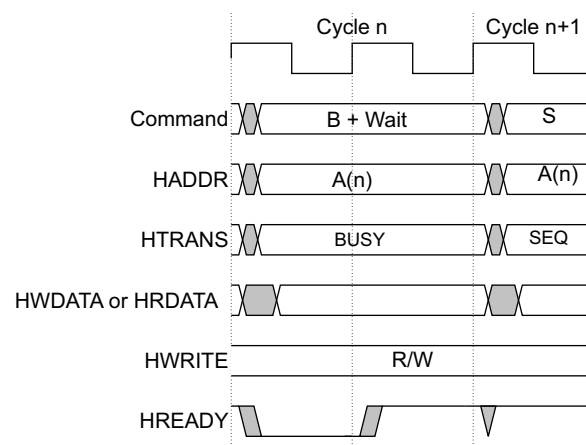


Figure 7-9 Busy cycle timing

The address phase is extended by wait states because of the data phase of a previous transfer, if present.

Idle command

The idle command, I, performs either an IDLE transfer or an IDLE cycle, depending on the Wait field. The options enable you to set up the control information during the IDLE transfer, and to specify whether the transfer is locked or unlocked.

If the Wait field is not specified, the idle command inserts a single **HCLK** cycle IDLE transfer on the AHB. See [Figure 7-10](#).

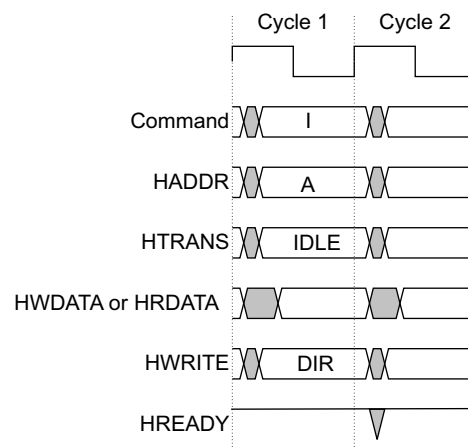


Figure 7-10 Idle transfer timing

Cycle 1 **HTRANS** is IDLE and the control signals take the default values, except for those specified in the command.

Cycle 2 The file reader proceeds to the next control phase. Data is ignored.

If the Wait field is specified, the idle command inserts a complete AHB transfer. See [Figure 7-11 on page 7-13](#). The address phase is extended by wait states because of the data phase of a previous transfer, if present.

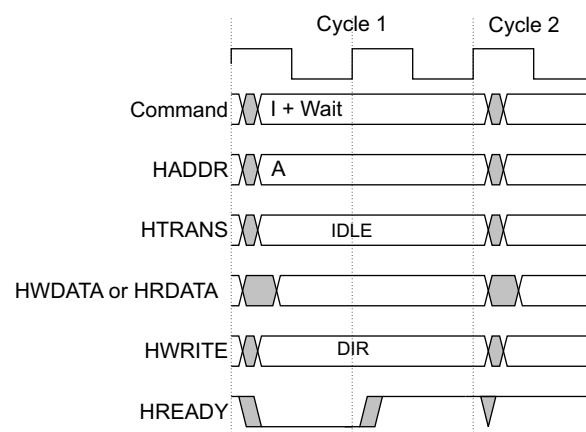


Figure 7-11 Idle cycle timing

- Cycle 1** **HTRANS** is set to IDLE and the control signals are set to the default values, except for those specified in the command.
- Cycle 2** If the Wait field is not specified, or the Wait field is specified and **HREADY** is asserted, then the file reader proceeds to the next control phase. Data is ignored.

Poll command

The poll command, P, continually reads the input data until it matches the value in the Data field or until the number of reads equals the number in the Timeout field. If the input data does not match after the Timeout number, the file reader reports an error. Not specifying a Timeout value or specifying a Timeout value of 0, causes the poll command to read continually until the data matches the required value. The poll command is valid only for INCR or SINGLE burst types and for aligned addresses.

Figure 7-12 shows the poll command timing.

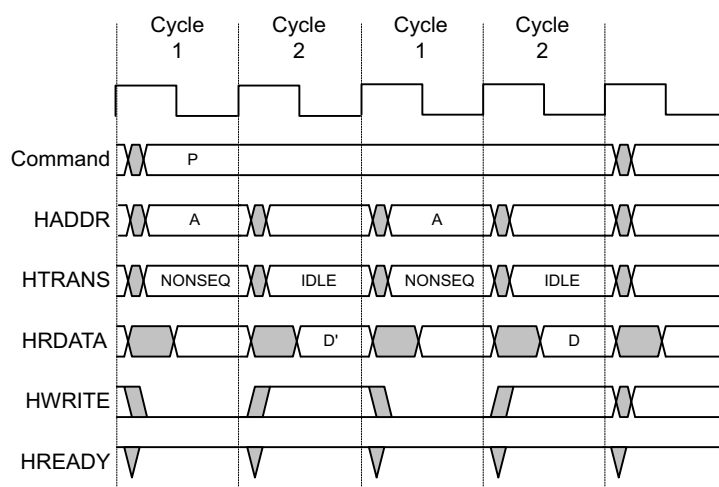


Figure 7-12 Poll command timing

The poll command operates as follows:

- Cycle 1** The file reader sets up a read of the single address in the Address field. If **HREADY** is asserted, the file reader proceeds to the second control phase.

Cycle 2 The file reader issues an IDLE transfer and reads the data for the previous address value.

Loop command

The loop command, L, repeats the last command a number of times. When the burst type is INCR or SINGLE, a loop command must follow only a write or read. Because the file reader has a 32-bit counter, the maximum number of loops is $2^{32} - 1$.

———— Note ————

Commands that do not directly represent bus transactions, for example, the simulation comment command C, are not looped. Consecutive loops are cumulative and not multiplicative. For example:

```
I 0x4000
C Commencing IDLES
L 1000
L 1000
```

This sequence performs an idle command to address 0x4000, generates the comment, and then performs 2000 more IDLE accesses to address 0x4000.

Comment command

The comment command, C, sends a message to the simulation window.

Quit command

The quit command, Q, causes the simulation to terminate immediately. Additionally, the quit command provides a summary of the number of commands and errors.

Resp field

The Resp field tests for the expected response. You must present the Resp field on a command that is expected to receive an ERROR response from a slave.

If the Resp field is set to Errorcont or Errorcanc, and an ERROR response is received, no warning is issued. If the Resp field is set to Errorcont or Errorcanc, and an ERROR response is not received, a simulation warning is generated.

If an error occurs during a burst transfer, and the Resp field is set to Errorcont, the burst continues.

If an error occurs during a burst transfer and the Resp field is set to Errorcanc, the burst is cancelled. No attempt is made to retransmit the erroneous transfer. It is not necessary for the stimulus file to contain the remaining transfers in the burst. An IDLE transfer is always inserted during the ultimate cycle of the ERROR response if the burst is to be cancelled.

Clock and reset

The file reader is synchronous to the AHB bus clock signal **HCLK** and the AHB reset signal **HRESETn** resets it.

Error reporting at runtime

An error can occur in the following circumstances:

- A read transfer where the expected data does not match the actual data.
- A transfer that receives an AHB ERROR response and the Error field is not set.
- A transfer where the Error field is set and the transfer does not receive an AHB ERROR response.
- A Poll command where the expected data is not received within the timeout number of attempts.
- The stimulus file is longer than the array size allocated.

When an error is reported, the line number of the corresponding command on the input file is reported to the simulation window.

End of stimulus

A summary of the number of commands and errors is given when any of the following is reached:

- A quit command.
- End of input file.
- End of the internal command array.

Simulation is terminated when a Q command is encountered.

When the stimulus ends, all AHB signals are set LOW. This implies IDLE read transfers to address 0x00.

If the end of the internal command array is reached, and the end of the stimulus file has not been reached, a warning is issued, and all AHB signals are set LOW. This implies IDLE read transfers to address 0x00.

7.3.2 Command syntax

The filename of the stimulus data file is specified using a Verilog parameter, at the point of instantiation in the HDL code.

The syntax uses a single letter for each command followed by a number of fields.

Command syntax

Table 7-8 shows the stimulus command syntax.

Table 7-8 Stimulus command syntax

| Cmd | Fields | | | | | | | | |
|-----|-----------|-------|--------|--------|---------|--------|--------|--------|-----------|
| W | Address | Data | | [Size] | [Burst] | [Prot] | [Lock] | [Resp] | [Comment] |
| R | Address | Data | [Mask] | [Size] | [Burst] | [Prot] | [Lock] | [Resp] | [Comment] |
| S | - | Data | [Mask] | - | - | - | - | [Resp] | [Comment] |
| B | - | - | - | - | - | - | - | [Wait] | [Comment] |
| I | [Address] | [Dir] | - | [Size] | [Burst] | [Prot] | [Lock] | [Wait] | [Comment] |

Table 7-8 Stimulus command syntax (continued)

| Cmd | Fields | | | | | | | | |
|-----|---------|------|--------|--------|---------|--------|-----------|---|-----------|
| P | Address | Data | [Mask] | [Size] | [Burst] | [Prot] | [Timeout] | - | [Comment] |
| L | Number | - | - | - | - | - | - | - | [Comment] |
| C | Message | - | - | - | - | - | - | - | [Comment] |
| Q | - | - | - | - | - | - | - | - | [Comment] |

Note

Items in square brackets [] are optional. See [Table 7-9 on page 7-17](#) for default values.

The commands are:

- W** The write command starts a write burst and one or more S vectors can follow it. The number of S vectors is set by the size and burst fields for fixed length bursts. There is no limit to the number of S vectors for undefined length bursts, as long as it does not cause the address to cross a 1KB boundary.
- R** The read command starts a read burst and one or more S vectors can follow it. The number of S vectors is set by the size and burst fields for fixed length bursts. There is no limit to the number of S vectors for undefined length bursts, as long as it does not cause the address to cross a 1KB boundary.
- S** The sequential vector provides data for a single beat in the burst. The file reader calculates the address required.
- B** The busy command inserts either a BUSY cycle or a BUSY transfer mid burst, depending on the value of the Wait field. An INCR burst can have a busy after its last transfer, while the master determines whether it has another transfer to complete. It is not valid to have a busy command when a burst is not in progress.
- I** The idle command performs either an IDLE cycle or an IDLE transfer, depending on the value of the Wait field. The options enables you to set up the control information during the idle transfer, and to specify whether the transfer is locked or unlocked.
- P** The poll command performs a read transfer that repeats until the data matches the required value. If it repeats this Number times, and the value is not read, then an error is reported. Either omitting Timeout or setting it to 0 causes the Poll to repeat continually until the data matches the required value. You can use the poll vector for INCR or SINGLE burst types and for aligned addresses.
- L** The loop command repeats the last command a number of times. An L command must only follow a W or R when the burst type is INCR or SINGLE.
- C** The comment command, C, sends a message to the simulation window.
- Q** The quit command, Q, causes the simulation to terminate immediately. Additionally, the quit command gives a summary of the number of commands and errors.

Table 7-9 shows the stimulus command fields.

Table 7-9 Command fields

| Field | Default | Values | Prefix | Description |
|---------|---|--|--------------|--|
| Address | 0x00000000 | 32-bit hex value | 0x, optional | First address of burst. |
| Data | - | 8, 16, 32, or 64-bit hex value | 0x, optional | Data field for read, write, sequential, and poll commands. The width of the Data field must match either the specified transfer size or the bus width of the FRBM. |
| Mask | 0xFF for each active byte lane as determined by Address and Size, and 0x00 for inactive byte lanes, or 0xFFFFFFFF if adk1 switch is set | 8, 16, 32, or 64-bit hex value | 0x, optional | Bit mask. Enables masking of read data when testing against required data. You must write the Mask and Data fields as the same size. They must match either the specified transfer size or the bus width of the FRBM. |
| Size | word or doubleword depending on user-defined -buswidth switch | b byte size8 h hword size16 w word size32 d dword size64 | - | Data size for read, write, sequential, and poll commands. |
| Burst | incr | sing single incr incr4 wrap4 incr8 wrap8 incr16 wrap16 | - | Burst type for read, write, and idle transfer commands. For the poll command, the only permitted values for Burst are sing, single, or incr. |
| Prot | 0000 | 4-bit binary | p P | Indicates the HPROT value for the transfer. |
| Lock | noLock | noLock lock | - | Transfers lock. |
| Resp | okay | okay ok errcanc errcanc | - | <p>When errcont is specified:^a</p> <ul style="list-style-type: none"> If an ERROR response occurs, no warning is generated. A burst in progress continues. If no ERROR response occurs, a warning is generated. A burst in progress continues. <p>When errcanc is specified:</p> <ul style="list-style-type: none"> If an ERROR response occurs, no warning is generated. A burst in progress is cancelled.^b If no ERROR response occurs, a warning is generated. A burst in progress continues. |
| Dir | read | read write | - | Controls the value of HWRITE during an idle command. |
| Number | - | Decimal value from 1-(2 ³² -1) | - | Loops repeat value. |
| Timeout | 0 | Decimal value from 0-(2 ³² -1) | t T | Number of times the poll command repeats the data check before generating an error when data does not match the expected value. Specifying 0 repeats continuously. |

Table 7-9 Command fields (continued)

| Field | Default | Values | Prefix | Description |
|-------------------|---------|--|--|---|
| Wait | nowait | wait nowait | - | Waits for HREADY before continuing. Makes an IDLE or BUSY cycle. |
| Message | - | 1 to 80 characters and symbols. See Table 7-10 for supported characters. | Comment contained within double quotes | Sends a user-defined comment to the simulation window. |
| Comment Delimiter | - | ; # // -- | - | All common comment delimiters are valid. |

- You can use the value err or error as a synonym for errcont for compatibility with legacy BFM versions, but it is not recommended for use in new development.
- An IDLE transfer is always inserted in the last cycle of the ERROR response if the burst is cancelled. No attempt is made to retransmit the erroneous transfer. It is not necessary for the stimulus file to contain the remaining transfers of the burst.

[Table 7-10](#) shows the keyboard characters that are supported by the comment command. For other characters replace with a dash, -, character by the script.

Table 7-10 Characters supported by comment command

| Character | Symbol | | | | | | | | | |
|-----------------|--------|----|---|---|---|---|---|---|---|--|
| a-z, lower case | ! | \$ | % | ^ | & | * | (|) | | |
| A-Z, upper case | _ | - | + | = | { | } | [|] | | |
| 0-9 | : | ; | @ | | ' | ~ | # | < | > | |
| White space | , | . | ? | / | | | | | | |

7.3.3 File preprocessing

The stimulus file is converted into a format that can feed directly into the HDL code using the `fm2conv.pl` script. This script verifies that the syntax of the input file is correct. This script provides useful error messages when the syntax is incorrect. [Figure 7-13](#) shows the process of stimulus file conversion.

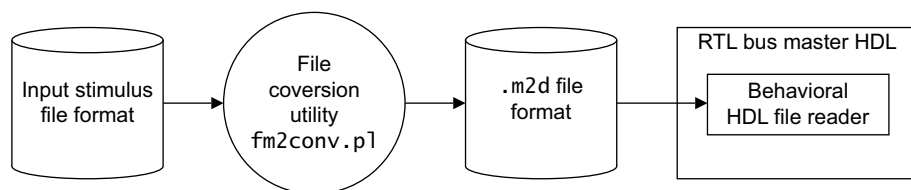


Figure 7-13 Stimulus file conversion

Loops

The `fm2conv.pl` script unfolds loops of S vectors, but relies on the FRBM functionality for other commands.

———— Note ————

Large loops of S vectors can create large stimulus data files.

Data and mask representations

You can specify data and mask values as any of the following:

- The bus width.
- With the `-buswidth` switch to `fm2conv.pl`.
- The same length as the transfer size with or without a `0x` prefix.

The byte lanes are driven according to both the least significant address bits, and the specified endian organization. The default is little-endian.

If the data or mask are represented as fewer bits than the data bus, then the transfer size is implicitly set to be that width. If this value conflicts with an explicit Size field, then an error is generated. The following examples show data and mask representations of fewer bits than the data bus:

R 00000002 DD

Read transfer with burst type INCR and implied size BYTE. The Data mask is `0x0000000000FF0000` in default little-endian mode.

R 0000ABCD 0x0123456789ABCDEF AB

The data field is 64 bits, that is, the bus width, and the mask field is BYTE, so the transfer size is BYTE.

R 00000004 EEEEEEE FF

Invalid stimulus on a 64-bit system. The data field implies a word transfer whereas the mask field implies BYTE.

W 000000C0 AB WORD

Invalid stimulus. The Data field implies a byte transfer whereas the Size field specifies word transfer.

FRBM versions

Because the FRBM and `fm2conv.pl` utilities are closely coupled through the stimulus data file, you must take care to ensure that you use the correct versions. [Table 7-11](#) shows the compatibility between versions.

Table 7-11 Compatibility between versions of FRBM and `fm2conv.pl`

| fm2conv.pl version | |
|-----------------------------|-----------------------------|
| ADK 1.0 | ADK 2.0 |
| File reader version ADK 1.0 | - |
| - | File reader version ADK 2.0 |

Because of enhancements in FRBM functionality and stimulus extensions, the stimulus files and data files for the AHB file reader are incompatible with previous versions of the file reader. The file preprocessor can translate ADK 1.0 stimulus files using the corresponding command-line switch. You can identify the versions by their ADK version keyword as follows:

ADK_REL1v For previous versions.

ADK2v For the FRBM that this document describes.

Table 7-12 shows the compatibility between versions.

Table 7-12 Compatibility between versions of stimulus file and fm2conv.pl

| fm2conv.pl version | |
|-------------------------------|--|
| ADK 1.0 | ADK 2.0 |
| Stimulus file version ADK 1.0 | Stimulus file version ADK 1.0 ^a |
| - | Stimulus file version ADK 2.0 |

a. Using -adk 1.0 command-line switch.

Endianness

By default, the preprocessor script assumes little-endian data organization. Therefore, if only a single byte of data is specified for a byte access, it is placed on the byte lane that the little-endian addressing determines.

Big-endian mode is supported for AMBA 2.0. The type of big-endian is legacy big-endian, also called ARM big-endian or BE-32. You must specify the data and mask in the same way as for little-endian mode. The preprocessor script places the data and mask bytes in the correct lanes.

Stimulus file size

When the file reader simulation begins, the entire stimulus file is read into an array. Ensure that the array size is large enough to store the entire stimulus file. The fm2conv.pl utility reports the array size required and the total number of vectors in a summary of the stimulus file conversion. A warning is generated if the array size is too small for the resulting stimulus file.

If the array size in the RTL file reader bus master is changed from the default value, you can set the array size using a generic parameter in the FRBM HDL by using the -stimarraysize command line switch with the fm2conv.pl utility.

File preprocessor usage

Table 7-13 shows the command-line switches that the fm2conv.pl preprocessor accepts.

Table 7-13 Preprocessor command-line options

| Switch | Options | Default | Description |
|------------------------|---------------|--------------|--|
| -help | - | - | Displays the usage messages. |
| -quiet | - | - | Suppresses warning messages. |
| -adk1 | - | - | Translates an ADK 1.0 stimulus file. You can also specify this option within the stimulus file. |
| -endian = <endianness> | little or big | little | The endianness determines the byte lanes that are driven for sparsely declared Data and Mask fields. This is not supported for v6 stimulus. Instead, you must specify the full bus width for big-endian transfers. The big-endian option is implemented as ARM big-endian. |
| -infile = <filename> | - | filestim.m2i | Input file name. |
| -outfile = <filename> | - | filestim.m2d | Output file name. This name must match the definition specified in the file reader bus master HDL. |

Table 7-13 Preprocessor command-line options (continued)

| Switch | Options | Default | Description |
|-------------------------|------------|---------|--|
| -buswidth = <width> | 32 or 64 | 64 | Specifies the data bus width of the target FRBM. |
| -arch = <arch> | ahb2 or V6 | ahb2 | Specifies the ARM processor architecture version of the target FRBM. This version of the FRBM does not support V6. |
| -StimArraySize = <size> | <size> | 5000 | The size of the file reader bus master file array. This size must match the value set in the FRBM HDL. |

Error reporting during file preprocessing

The script performs additional checks to ensure correct FRBM operation. [Table 7-14](#) shows the error checks. File conversion is aborted if an error with the command-line options is found. File conversion continues if any other error is found, so that you can generate non-AMBA compliant stimulus for test purposes, if required.

Table 7-14 fm2conv.pl error messages

| Error number | Description |
|--------------|--|
| 17 | Input file is unreadable, does not exist or has incorrect permissions. |
| 20 | Input file has the same name as the output file. |
| 21 | Cannot create the output file. |
| 32 | Unrecognized commands within the file. |
| 36 | Required fields are missing or in the wrong format. |
| 37 | Loop command has the Number field missing. |
| 38 | Comment command requires a string within double quotes. |
| 40 | Size value exceeds the data bus width. The maximum value is dword size64 for the ADK 2.0 64-bit version FRBM, and word size32 for the ADK 2.0 32-bit version FRBM. |
| 43 | Loop Number field is out of range. |
| 44 | Poll TimeOut field is out of range. |
| 48 | Data field length exceeds the FRBM data bus width. |
| 49 | Data field has an invalid length. |
| 52 | Mask field length exceeds the FRBM data bus width. |
| 53 | Mask field has an invalid length. |
| 56 | Mismatch between transfer size, whether specified or implicitly set by the Data or the Mask width, and the Data or Mask field. |
| 64 | Address is not aligned with the size of the transfer. |
| 80 | For Poll commands, burst types are not the valid INCR or SINGLE. |
| 84 | S or B vectors before a defined-length or undefined-length burst has started. |
| 88 | Burst exceeds the 1KB address boundary, for both defined and undefined-length bursts. |
| 89 | Loop number exceeds the number of remaining transfers, for each defined-length burst type. |

The most common AMBA protocol violations are detected by the file preprocessor script, but the absence of errors and warnings does not guarantee that the stimulus are compliant with the AMBA protocol.

Table 7-15 shows the warnings. File conversion continues when a warning is issued.

Table 7-15 fm2conv.pl warnings

| Warning number | Description |
|----------------|---|
| 128 | Perl version is older than 5.005. Command line switches are not supported. |
| 132 | Invalid data bus width is selected. |
| 133 | Invalid architecture is selected. |
| 134 | ADK 1.0 architecture is selected and the data bus width is not specified as 32 bits. |
| 136 | Output file length exceeds the specified size of the stimarraysize. |
| 144 | EOF found during a burst, and additional transfers are expected. |
| 164 | An optional field has an invalid value. |
| 165 | Invalid character exists in the comment string. |
| 168 | Comment command has a string of length greater than 80 characters. |
| 169 | Consecutive blank or commented lines exceed 63 for the line number reporting to work. |
| 216 | Number of S vectors following a W R command is incorrect for a fixed length burst, and a burst is terminated early. This enables the simulation of early-terminated bursts. |
| 240 | Unsupported memory command is encountered. |
| 241 | Unsupported AltMaster field is encountered and the entire line is ignored. |
| 242 | Unsupported DeGrant field is encountered and is ignored. |
| 248 | A feature in development status. |
| 254 | Currently unsupported value in a field, for example, size > 64. |
| 255 | Internal or debug error. Not expected to occur in normal usage. |

Errors and warnings have the following numbering scheme:

| | |
|-------|---------------------------|
| [7] | Severity. |
| [6:4] | Error or warning type. |
| [3:2] | Error or warning subtype. |
| [1:0] | Enumerator. |

Table 7-16 shows the numbering scheme for bit[7].

Table 7-16 Numbering scheme for bit 7

| Value | Meaning |
|-------|---------|
| 0 | Error |
| 1 | Warning |

Table 7-17 shows the numbering scheme for bits[6:4] and bits[3:2].

Table 7-17 Numbering scheme for bits [6:4] and bits[3:2]

| Value bits[6:4] | Meaning | Value bits[3:2] | Meaning |
|-----------------|----------------------|-----------------|---------------|
| 000 | Command line | 00 | Environment |
| | | 01 | Options |
| | | 10 | - |
| | | 11 | - |
| 001 | File input or output | 00 | Input file |
| | | 01 | Output file |
| | | 10 | - |
| | | 11 | - |
| 010 | Syntax | 00 | Command |
| | | 01 | Field |
| | | 10 | Range |
| | | 11 | - |
| 011 | Transfer size | 00 | Data |
| | | 01 | Mask |
| | | 10 | Mismatch |
| | | 11 | - |
| 100 | Alignment | 00 | Address |
| | | 01 | - |
| | | 10 | - |
| | | 11 | - |
| 101 | Burst | 00 | Within burst |
| | | 01 | Outside burst |
| | | 10 | Length |
| | | 11 | - |
| 110 | Reserved | - | - |
| 111 | Reserved | - | - |

Table 7-18 shows the numbering scheme for bits[1:0].

Table 7-18 Numbering scheme for bits[1:0]

| Value | Meaning |
|-------|---|
| (any) | Creates a unique identifier in conjunction with bits[7:2] |

Appendix A

Revisions

This appendix describes the technical changes between released issues of this book.

Table A-1 Differences between issue A and issue B

| Change | Location | Affects |
|---|---|---------|
| Added the following section in <i>AHB upsizer</i> on page 5-14: <ul style="list-style-type: none"><i>Method of using AHB upsizer</i> on page 5-14 | Chapter 5 <i>Advanced AHB-Lite Components</i> | r0p0 |
| Added the following section in <i>AHB downsizer</i> on page 5-17: <ul style="list-style-type: none"><i>Using AHB downsizer</i> on page 5-22 | Chapter 5 <i>Advanced AHB-Lite Components</i> | r0p0 |
| Added the following section in <i>AHB to APB asynchronous bridge</i> on page 5-25: <ul style="list-style-type: none"><i>Cross-clock domain handling in AHB to APB asynchronous bridge</i> on page 5-26 | Chapter 5 <i>Advanced AHB-Lite Components</i> | r0p0 |
| Added the following section in <i>AHB to AHB synchronous bridge</i> on page 5-30: <ul style="list-style-type: none"><i>Using AHB to AHB synchronous bridge</i> on page 5-31 | Chapter 5 <i>Advanced AHB-Lite Components</i> | r0p0 |
| Added the following section in <i>AHB to AHB sync-down bridge</i> on page 5-32: <ul style="list-style-type: none"><i>Using the AHB to AHB sync-down bridge</i> on page 5-33<i>Optional write buffer</i> on page 5-34<i>Synthesizing the AHB to AHB sync-down bridge</i> on page 5-35. | Chapter 5 <i>Advanced AHB-Lite Components</i> | r0p0 |
| Added the following section in <i>AHB to AHB sync-up bridge</i> on page 5-37: <ul style="list-style-type: none"><i>Using the AHB to AHB sync-up bridge</i> on page 5-38<i>Synthesizing the AHB to AHB sync-up bridge</i> on page 5-39. | Chapter 5 <i>Advanced AHB-Lite Components</i> | r0p0 |

Table A-2 Differences between issue B and issue C

| Change | Location | Affects |
|--|---|---------|
| Updated references to Cortex-M0 to include Cortex-M0+ | Throughout document | r1p0 |
| All module names prefixed with <code>cmsdk_</code> | Throughout document | r1p0 |
| Changed <i>Real View Development Suite</i> (RVDS) to <i>Development Suite</i> (DS-5) and CodeSourcery g++ to ARM GCC | Throughout document | r1p0 |
| AHB downsizer HRESP signal width changed to 1-bit. | Throughout document | r1p0 |
| HREADYOUTM signal name changed to HREADYM . | Throughout document | r1p0 |
| Updated Peripheral ID Register 2 reset value to 0x1B | Throughout document | r1p0 |
| Added I/O port GPIO for Cortex-M0+ processor | Figure 1-2 on page 1-3 AHB GPIO on page 3-11 | r1p0 |
| Extensively revised chapter to reduce duplication | Chapter 2 Functional Description | r1p0 |
| Updated OVL assertions | Use of OVL on page 2-6 | r1p0 |
| Updated signal names | Figure 3-2 on page 3-3 Figure 3-3 on page 3-6 Figure 3-5 on page 3-9. | r1p0 |
| AHB slave multiplexer changed from eight ports to ten ports | AHB slave multiplexer on page 3-6 | r1p0 |
| Added AHB master multiplexer limitations | Limitations on page 3-10 | r1p0 |
| Updated AHB to APB sync-down bridge description | AHB to APB sync-down bridge on page 3-18 | r1p0 |
| Added AHB to Flash16 interface model | AHB to flash interface modules on page 3-22 | r1p0 |
| Updated timeout monitor description | AHB timeout monitor on page 3-25 | r1p0 |
| Updated bit-band description | AHB bit-band wrapper on page 3-31 | r1p0 |
| Added bit-band limitations | Limitations on page 3-34 | r1p0 |
| Updated APB slave module diagrams | APB example slaves on page 4-2 | r1p0 |
| Updated APB UART characteristics | Table 4-6 on page 4-9 | r1p0 |
| Added APB dual-input timer characteristics | Table 4-8 on page 4-11 | r1p0 |
| Added APB watchdog characteristics | Table 4-16 on page 4-20 | r1p0 |
| Updated APB subsystem diagram | Figure 4-23 on page 4-27 | r1p0 |
| Updated clock and reset signal descriptions | Clock and reset signals on page 4-30 | r1p0 |
| Updated bus matrix description | AHB bus matrix on page 5-2 | r1p0 |
| Added AHB to AHB and APB asynchronous bridge | AHB to AHB and APB asynchronous bridge on page 5-27 | r1p0 |
| Added component dependency descriptions | AHB GPIO on page 3-11 AHB to AHB synchronous bridge on page 5-30 AHB to AHB sync-down bridge on page 5-32 AHB to AHB sync-up bridge on page 5-37 | r1p0 |
| Added <code>ahb_rom.v</code> configuration for 16-bit flash ROM | Table 6-2 on page 6-2 and Figure 6-5 on page 6-5 | r1p0 |

Table A-2 Differences between issue B and issue C (continued)

| Change | Location | Affects |
|---|---|----------------|
| Added 16-bit flash ROM behavioral model | <i>16-bit flash ROM behavioral model on page 6-12</i> | r1p0 |
| Updated FPGA SRAM description | <i>FPGA SRAM synthesizable model on page 6-13</i> | r1p0 |
| Removed AHB protocol checker | <i>Chapter 7 Verification Components</i> | r1p0 |
| Updated AHB-Lite protocol checker description | <i>AHB-Lite protocol checker on page 7-2</i> | r1p0 |
| Updated APB protocol checker description | <i>APB protocol checker on page 7-5</i> | r1p0 |