SoC Designer

Version 9.5.0

AHBv2 Protocol Bundle User Guide

Non-Confidential

Copyright© 2017 ARM Limited. All Rights Reserved

ARM 101027_0905_00

SoC Designer
AHBv2 Protocol Bundle User Guide

Copyright © 2017 Arm Limited (or its affiliates). All rights reserved.

Release Information

The following changes have been made to this document.

Issue	Date	Confidentiality	Change
0905-00	November 2017	Non-Confidential	Release with 9.5
0902-00	May 2017	Non-Confidential	Release with 9.2.

Non-Confidential Proprietary Notice

This document is protected by copyright and other related rights and the practice or implementation of the information contained in this document may be protected by one or more patents or pending patent applications. No part of this document may be reproduced in any form by any means without the express prior written permission of Arm. **No license, express or implied, by estoppel or otherwise to any intellectual property rights is granted by this document unless specifically stated**.

Your access to the information in this document is conditional upon your acceptance that you will not use or permit others to use the information for the purposes of determining whether implementations infringe any patents.

THIS DOCUMENT IS PROVIDED “AS IS”. ARM PROVIDES NO REPRESENTATIONS AND NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY, SATISFACTORY QUALITY, NON-INFRINGEMENT OR FITNESS FOR A PARTICULAR PURPOSE WITH RESPECT TO THE DOCUMENT. For the avoidance of doubt, Arm makes no representation with respect to, and has undertaken no analysis to identify or understand the scope and content of, third party patents, copyrights, trade secrets, or other rights.

This document may include technical inaccuracies or typographical errors.

TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL ARM BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF ARM HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

This document consists solely of commercial items. You shall be responsible for ensuring that any use, duplication or disclosure of this document complies fully with any relevant export laws and regulations to assure that this document or any portion thereof is not exported, directly or indirectly, in violation of such export laws. Use of the word “partner” in reference to Arm’s customers is not intended to create or refer to any partnership relationship with any other company. Arm may make changes to this document at any time and without notice.

If any of the provisions contained in these terms conflict with any of the provisions of any click through or signed written agreement covering this document with Arm, then the click through or signed written agreement prevails over and supersedes the conflicting provisions of these terms. This document may be translated into other languages for convenience, and you agree that if there is any conflict between the English version of this document and any translation, the terms of the English version of the Agreement shall prevail.

The Arm corporate logo and words marked with ® or ™ are registered trademarks or trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. All rights reserved. Other brands and names mentioned in this document may be the trademarks of their respective owners. Please follow Arm’s trademark usage guidelines at http://www.arm.com/company/policies/trademarks.

Copyright © Arm. All rights reserved.
Arm Limited. Company 02557590 registered in England.
110 Fulbourn Road, Cambridge, England CB1 9NJ.

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

Copyright© 2017 Arm. All Rights Reserved.

Table of Contents

1 Introduction 6

2 Requirements 6

3 Package Contents 6

3.1 AHBv2 Models and Examples .... **Error! Bookmark not defined.**

3.2 AHBv2 Probes .... **Error! Bookmark not defined.**

3.3 AHBv2 Transactors .... **Error! Bookmark not defined.**

3.4 AHBv2 Ports .... **Error! Bookmark not defined.**

3.5 AHBv2 Component Wizard Templates .... **Error! Bookmark not defined.**

4 Models 7

4.1 AHBv2_Master 8

4.2 AHBv2_Slave 9

4.3 AHBv2_Mem and AHBv2_LiteMem 10

4.4 AHBv2_Stub 12

4.4.1 Stub Macros 12

4.4.2 Simulation Features 13

4.4.2.1 Register 13

4.4.2.2 Memory 13

4.4.2.3 Disassembly 13

4.5 MxAHBv2 and MxAHBv2_Lite 14

4.5.1 Arbitration 14

4.6 AHBv2ToAPB and AHBv2LiteToAPB 16

4.7 AHBv2ToAHBv2LiteSS and AHBv2LiteToAHBv2SS 17

4.8 AHBv2ToAHBv2LiteMS 18

4.9 AHBv2ToMx and AHBv2LiteToMx 19

Copyright© 2017 Arm. All Rights Reserved.

4.10 MxToAHBv2 and MxToAHBv2Lite 20

4.11 AHBv2Mux and AHBv2Mux_Lite 21

5 Probes 22

5.1 Tracer 22

5.2 Breakpoint 23

5.3 Profiling 23

6 Component Wizard 24

6.1 Generating AHBv2 Ports 24

7 Transactors 25

7.1 AHBv2 Transactors and Cycle Model Studio 26

8 AHBv2 Port Interfaces 26

8.1 AMBA AHB Interfaces AHBv2 Transaction Ports 26
8.1.1 AHB_Master_Port 28
8.1.1.1 AHB_Master_PortBase 28
8.1.1.2 AHB_Master_Port Template Specialization 29
8.1.2 AHB_Slave_Port 32
8.1.2.1 AHB_Slave_PortBase 32
8.1.2.2 AHB_Slave_Port Template Specialization 35
8.1.3 AHBPortIF 38
8.1.4 Transaction Phases 39
8.1.5 Examples 39

8.2 AHB Extensions 40
8.2.1 Cortex-M3 Sideband Signals 40

 Copyright© 2017 Arm. All Rights Reserved.

1 Introduction

This is the user guide for SoC Designer AHBv2 Protocol Bundle. This protocol bundle contains SoC Designer components, probes, and transactors for the Arm AMBA AHB transaction protocol.

2 Requirements

AHBv2 protocol bundle requires the following:

- SoC Designer v9.2.0
- Compilation tools as set forth in the *SoC Designer Installation Guide* (Arm 100975).

3 Package Contents

AHBv2 protocol bundle contains:

- AHBv2 transaction port definition header files and libraries. These are required during runtime of any components with AHBv2 ports and when creating components with AHBv2 ports.
- AHBv2 transactors, which bridge the signal and transaction-level communication. These can be used by Cycle Model Studio to generate SoC Designer components with AHBv2 transaction ports.
- Generic components such as configurable bus and memory.
- AHBv2-specific tracer and profiling probes, which provide visibility into transactions between components. probes are included in this protocol bundle.
- Example source code to help you develop custom AHB components.
- Template files needed by the SoC Designer Component Wizard for generation of components with AHBv2 ports.

Copyright© 2017 Arm. All Rights Reserved.

Non-Confidential

4 Models

The following table lists the AHBv2 components included in this bundle.

Component	*Description*
AHBv2_Master	Example AHBv2 master component. This component is available in source code format.
AHBv2_Slave	Example AHBv2 slave component. This component is available in source code format.
AHBv2_Mem	Generic AHB memory model with an AHBv2 transaction slave port.
AHBv2_LiteMem	AHB-Lite version of AHBv2_Mem.
AHBv2_Stub	Scriptable (reads in `*.mxscr`) AHB master component.
MxAHBv2	Generic AHB bus component which can be configured for up to 16 masters and 16 slaves.
MxAHBv2_Lite	AHB-Lite version of MxAHBv2. Supports up to 16 slaves.
AHBv2ToAPB	Bridge component which translates AHB to APB transactions.
AHBv2LiteToAPB	Bridge component which translates AHB-Lite slave to APB transactions.
AHBv2ToAHBv2LiteSS	Bridge component that allows AHB-Lite slaves to be hooked up to an AHB bus.
AHBv2LiteToAHBv2SS	Bridge component that allows AHB slaves to be hooked up to an AHB-Lite bus.
AHBv2ToAHBv2LiteMS	Bridge component that allows an AHB master to be hooked up to an AHB-Lite system.
AHBv2ToMx	AHBv2 to MX protocol conversion bridge.
AHBv2LiteToMx	AHBv2-Lite to MX protocol conversion bridge.
MxToAHBv2	MX to AHBv2 protocol conversion bridge.
MxToAHBv2Lite	MX to AHBv2-Lite protocol conversion bridge.
AHBv2Mux	AHB slave multiplexor for multi-layer AHB designs.

Table 4-1 AHBv2 Components

Copyright© 2017 Arm. All Rights Reserved.

4.1 AHBv2_Master

AHBv2_Master is an example AHBv2 master component which is available in source code format. This component initiates a configurable number of AHB NONSEQ transfers over incrementing addresses.

The source code is located in `$MAXSIM_PROTOCOLS/AHBv2/src/AHBv2_Master`. There is also an example system which uses AHBv2_Master, which is located in `$MAXSIM_PROTOCOLS/AHBv2/examples/MasterSlave_2x2`.

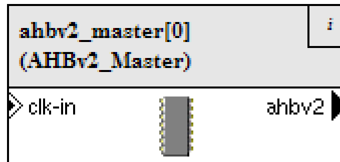

Figure 4-1 AHBv2_Master

The table below lists the component parameters.

Parameter Name	*Description*
Data Width	Data bus width in bits. Supported values are 32, 64, and 128.
Enable Debug Messages	Boolean flag to enable/disable debug messages.
Enable Verbosity	Boolean flag to enable/disable messages related to transactions generated by this component.
Num Transactions	Total number of transactions to generate.
Start Address	The starting address of transactions. The address is incremented by the Data Width/8 for each transaction.

Table 4-2 AHBv2_Master parameters

Copyright© 2017 Arm. All Rights Reserved.

Non-Confidential

4.2 AHBv2_Slave

AHBv2_Slave is an example AHBv2 slave component which is available in source code format. This component behaves as a simple memory device.

The source code is located in `$MAXSIM_PROTOCOLS/AHBv2/src/AHBv2_Slave`. There is also an example system which uses AHBv2_Slave, which is located in `$MAXSIM_PROTOCOLS/AHBv2/examples/MasterSlave_2x2`.

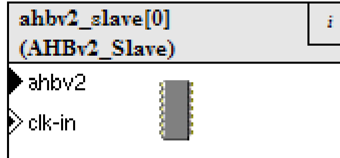

Figure 4-2 AHBv2_Slave

The table below lists the component parameters.

Name	*Description*
Enable Debug Messages	Boolean flag to enable/disable debug messages.
Enable Verbosity	Boolean flag to enable/disable messages related to transactions received by this component.
region name	Name of the memory region occupied by this AHB slave.
region size	Size of the region occupied by this AHB slave.
region start	The base address of the memory region occupied by this AHB slave.

Table 4-3 AHBv2_Slave parameters

Copyright© 2017 Arm. All Rights Reserved.

Non-Confidential

4.3 AHBv2_Mem and AHBv2_LiteMem

AHBv2_Mem is a generic AHB memory model with an AHB slave interface. AHBv2_LiteMem is the same as AHBv2_Mem, except that it includes an AHB-Lite slave interface.

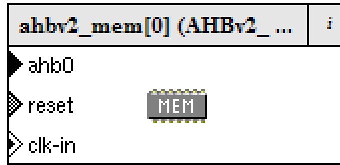

Figure 4-3 AHBv2_Mem

The table below lists the component parameters.

Name	*Description*
32bit aligned	• True - Accesses are aligned to 32-bit addresses. The lower 2 bits of the byte addresses are ignored, and the parameter **byte addressable** must be True. • False - Accesses can use byte addresses without any particular alignment.
AHB extra warnings	Additional access checks specific to AHB. Enable for debugging purposes only as it decreases simulation speed.
ahb_name*X*	Name of the memory region *X*.
ahb_size*X*	Size of the memory region *X*.
ahb_start*X*	The base address of the memory region *X*
big endian	• True - Byte order is big endian. • False - Byte order is little endian.
byte addressable	• True - Addresses refer to byte addresses. • False - Addresses refer to halfword addresses.
casLatency(r)	Number of latency cycles when accessing a new column during reads.
casLatency(w)	Number of latency cycles when accessing a new column during writes.
colBits	Number of bits in an address describing the column (counted from LSB). The sum of **pageBits** + **colBits** + **rowBits** must be 32.
data width	Data bus width. Allowed values are 32, 64, and 128.
enable debug messages	Enable extra debug messages.

Copyright© 2017 Arm. All Rights Reserved.

Non-Confidential

init value	Value assigned to memory during initialization.
pageBits	Number of bits in an address describing the page (counted from MSB). Ths sum of **pageBits** + **colBits** + **rowBits** must be 32.
pageLatency(r)	Number of latency cycles when accessing a new page during reads.
pageLatency(w)	Number of latency cycles when accessing a new page during writes.
rasLatency(r)	Number of latency cycles when accessing a new row during reads.
rasLatency(w)	Number of latency cycles when accessing a new row during writes.
read only	• True - Memory is Read Only except through the debug interfaces. • False – Memory is not Read Only.
refresh ratio	Number of cycles after which a refresh cycle is modeled.
rowBits	Number of bits in an address describing the row (counted from MSB-**pageBits**). The sum of **pageBits + colBits** + **rowBits** must be 32.
use input file	• True - Model can be initialized with a binary file containing a memory image. The binary file consists only of values. The addresses are assumed to start at the base address and are incremented with each value. • False = No input file in use.

Table 4-4 AHBv2_Mem parameters

Copyright© 2017 Arm. All Rights Reserved.

4.4 AHBv2_Stub

AHBv2_Stub is an AHBv2 master component which can be controlled with a SoC Designer `.mxscr` script. AHB transactions are generated on the ahb2_m transaction master port. The ahb2_s port can be used to drive AHB transactions.

AHBv2_Stub has an internal memory that you can use as a place holder for an AHB slave. Like other stub components, AHBv2_Stub has a number of signal masters and slaves that can be controlled from an `.mxscr` script.

An example system is located in `$MAXSIM_PROTOCOLS/AHBv2/examples/AHBv2_Stub`.

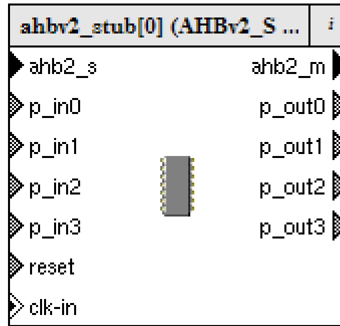

Figure 4-4 AHBv2_Stub

Note: *On stub components, accessing transaction slave ports using MxScript is not supported. Use a memory component if scripting is required.*

The table below lists the component parameters.

Name	*Description*
ahb_start*X*	Start address of the AHB memory regions.
CPP include path	Additional include path for header files to be used by script preprocessor.
Data Width	Data bus width. Supported values are 32, 64, and 128.
Enable Debug Messages	Enable extra debug messages.
Memory Init Byte Value	Initialization value for the internal memory (when acting as AHB slave)

Table 4-5 AHBv2_Stub parameters

4.4.1 Stub Macros

Convenience macros for AHB traffic generation are available in `AHBv2_Stub_Macros.h` located in `$MAXSIM_PROTOCOLS/AHBv2/include`. For backward compatibility to AHBv1, all macros defined in the original AHBv1 stub macros have been redefined, which means that you can reuse the old AHBv1 stub scripts by replacing the `#include` line to include `AHBv2_Stub_Macros.h` (instead of `AHB_Stub_Macros.h`).

 Copyright© 2017 Arm. All Rights Reserved.

4.4.2 Simulation Features

4.4.2.1 Register

The AHBv2_Stub register window shows information about the currently-loaded `.mxscr` script as well as stub internal data which may be useful in debugging user-defined stub macros.

4.4.2.2 Memory

The AHBv2_Stub memory window has three address spaces:

- MxStub - Internal use only (ignore)
- Memory - AHBv2_Stub internal memory
- AHB-Master - External memory seen by the ahb2_m port

4.4.2.3 Disassembly

AHBv2_Stub supports the disassembly view which shows the `.mxscr` script file and the commands being executed on the stub. As with other disassembly windows, simulation control buttons as well as instruction (or a script command) cycle counts are available from this window.

Copyright© 2017 Arm. All Rights Reserved.

Non-Confidential

4.5 MxAHBv2 and MxAHBv2_Lite

MxAHBv2 is an AHB bus component which can be configured up to 16 masters and 16 slaves. MxAHBv2_Lite is an AHB-Lite version of the bus which can support up to 16 slaves.

You can disable and hide unused ports in SoC Designer Canvas. To do this:

1. Make the port connections for the ports in use.
2. Right-click on the component in SoC Designer Canvas.
3. From the context menu, select **Disable All Unconnected Ports**.
4. Select **Hide All Disabled Ports**.

4.5.1 Arbitration

Arbitration policy can be configured to either a round-robin or a fixed-priority scheme. Fixed priority uses the master 0 port as the highest priority master (master 15 is the lowest priority).

For arbitration to work correctly, slave ports must be used consecutively starting with the ahb_s00 port. For example, connecting a master to ahb_s01 and leaving ahb_s00 unconnected is not allowed (an error message is issued at the beginning of simulation).

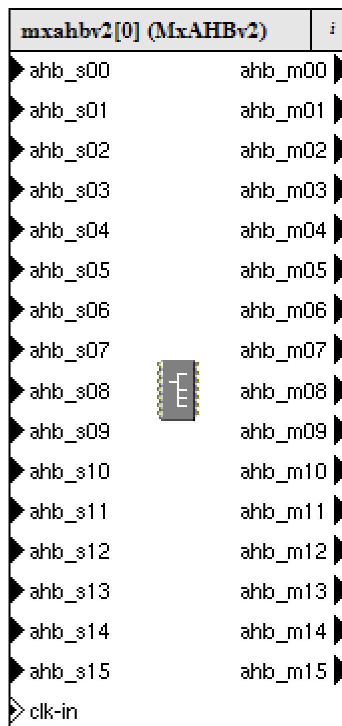

Figure 4-5 MxAHBv2

Copyright© 2017 Arm. All Rights Reserved.

Non-Confidential

The table below lists the component parameters.

Name	*Description*
Arbitration Policy	Round-Robin or Priority. The default is Round-Robin. Priority-based arbitration sets master 0 (master connected on ahb_s00 port) as the highest priority master, and master 15 (master connected on ahb_s15) as the lowest priority master. *Note: Available for MxAHBv2 only.*
Data Width	Data bus width. Supported values are 32, 64, and 128.
Enable Debug Messages	Enable extra debug messages.
Use MME	Use Memory Map Editor for memory map configuration. Memory regions reported by the slaves are ignored when MME is in use.
HTRANS IDLE when HSEL==0	When set to True, HTRANS is forced to IDLE on the slave side when the slave is not selected (HSEL==0).

Table 4-6 MxAHBv2 parameters

Copyright© 2017 Arm. All Rights Reserved.

4.6 AHBv2ToAPB and AHBv2LiteToAPB

These components map an AHB or AHB-Lite slave interface to APB transactions.

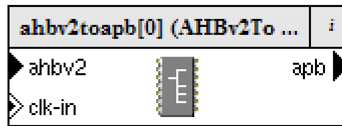

Figure 4-6 AHBv2ToAPB

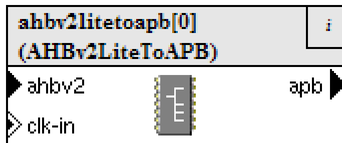

Figure 4-7 AHBv2LiteToAPB

Connect the ahbv2 port to the master port of MxAHBv2 (or MxAHBv2_Lite for AHB-Lite). The apb port is a bus master port, which means it can be hooked up to multiple APB slaves.

Both variants of the bridge support AMBA3 extensions of APB (PREADY and PSLVERR responses are supported).

Table 4-7 lists the component parameters.

Name	*Description*
Enable Debug Messages	Boolean flag to enable/disable debug messages.
APB region base	Base address of the peripheral region.
APB region size	Size of the peripheral region.
use MME	Use Memory Map Editor for setting up the memory map.

Table 4-7 AHBv2(Lite)ToAPB parameters

Copyright© 2017 Arm. All Rights Reserved.

4.7 AHBv2ToAHBv2LiteSS and AHBv2LiteToAHBv2SS

These components bridge between full AHB and AHB-Lite slave interfaces. Use AHBv2ToAHBv2LiteSS to hook up an AHB-Lite slave to an AHB bus. Conversely, use AHBv2LiteToAHBv2SS to hook up a regular AHB slave to an AHB-Lite bus. AHB signals that only exist in the full AHB interface (HRESP[1]) are assumed to be tied to zero when going through these bridges.

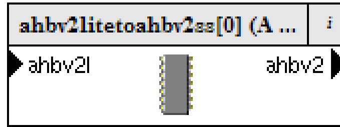

Figure 4-8 AHBv2LiteToAHBv2SS

Figure 4-9 AHBv2ToAHBv2LiteSS

Table 4-8 lists the component parameters.

Name	*Description*
Enable Debug Messages	Boolean flag to enable/disable debug messages.
Data Width	Data bus width. Supported values are 32, 64, and 128.
AHB region base	Base address of the AHB slave.
AHB region size	Size of the AHB slave address region.

Table 4-8 AHBv2ToAHBv2LiteSS and AHBv2LiteToAHBv2SS parameters

Copyright© 2017 Arm. All Rights Reserved.

Non-Confidential

4.8 AHBv2ToAHBv2LiteMS

This bridge enables a full AHB master to connect to an AHB-Lite bus system. An example system is shown in Figure 4-13.

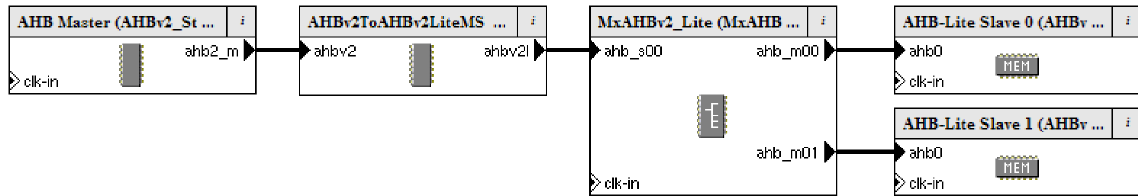

Figure 4-10 AHBv2ToAHBv2LiteMS

Table 4-9 lists the component parameters.

Name	*Description*
Enable Debug Messages	Boolean flag to enable/disable debug messages.
Data Width	Data bus width. Supported values are 32, 64, and 128.

Table 4-9 AHBv2ToAHBv2LiteMS parameters

 Copyright© 2017 Arm. All Rights Reserved.

Non-Confidential

4.9 AHBv2ToMx and AHBv2LiteToMx

These are protocol conversion bridges that enable AHBv2 and AHBv2-Lite masters to connect to MX components. The AHBv2 ports on these components implement the AHB master interface. An interconnect error is issued if these AHBv2 ports are connected to AHBv2 transaction master ports that implement the AHB slave interface (transaction master ports on MxAHBv2).

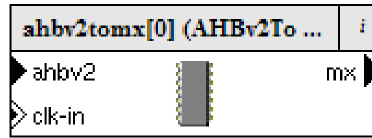

Figure 4-11 AHBv2ToMx

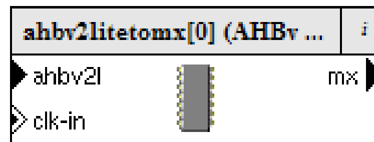

Figure 4-12 AHBv2LiteToMx

The table below lists the component parameters.

Name	*Description*
Enable Debug Messages	Boolean flag to enable/disable debug messages.
Data Width	Data bus width. Supported values are 32 and 64. Note the data width must match between AHBv2 and the MX sides.

Table 4-10 AHBv2ToMx and AHBv2LiteToMx component parameters

 Copyright© 2017 Arm. All Rights Reserved.

Non-Confidential

4.10 MxToAHBv2 and MxToAHBv2Lite

These are protocol conversion bridges that enable AHBv2 and AHBv2-Lite slaves to connect to MX components. The AHBv2 ports on these components implement the AHB slave interface. An interconnect error is issued if these AHBv2 ports are connected to AHBv2 transaction slave ports that implement the AHB master interface (transaction slave ports on MxAHBv2).

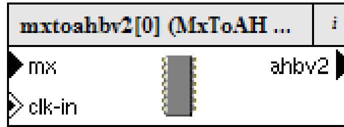

Figure 4-13 MxToAHBv2

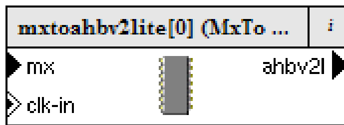

Figure 4-14 MxToAHBv2Lite

Table 4-11 lists the component parameters.

Name	*Description*
Enable Debug Messages	Boolean flag to enable or disable debug messages.
Data Width	Data bus width. Supported values are 32 and 64. The data width on the AHBv2 side must match the data width on the MX side. An error is issued for MX transactions that request data transfer greater than allowed by the configured data width.
hmaster	Value of HMASTER signal to drive onto the AHBv2 slave. Default is 0.
hprot	Value of HPROT signal to drive onto the AHBv2 slave. Default is 0.

Table 4-11 MxToAHBv2 and MxToAHBv2Lite component parameters

Copyright© 2017 Arm. All Rights Reserved.

Non-Confidential

4.11 AHBv2Mux and AHBv2Mux_Lite

These are AHB slave-side bridges that enable access to a single AHB slave from multiple AHB layers. The bridge arbitrates between the AHB master layers when there are concurrent accesses. The arbitration is priority based, where the layer connected on the lower-numbered slave port on the bridge has higher priority; in other words, the layer connected on port 0 has the highest priority. Arbitration takes place between whole transfers: burst transfers and back-to-back transactions are not interrupted even when there are higher-priority master requests for access.

Up to 16 layers are supported. Lower-numbered ports must be used first. Unused, higher-numbered ports can be disabled. An example multi-layer system using AHBv2Mux is shown in Figure 4-18.

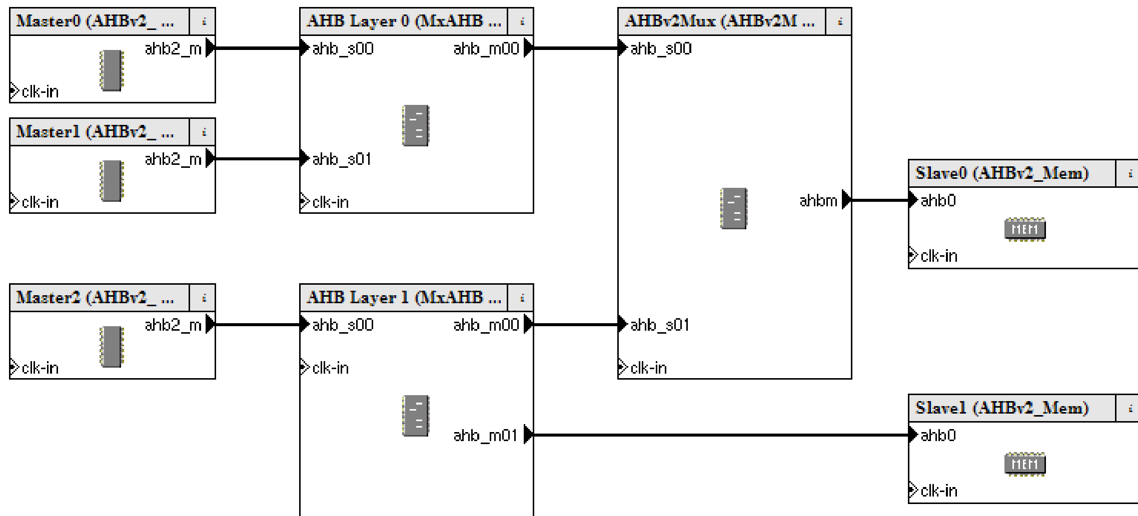

Figure 4-15 Example usage of AHBv2Mux

Table 4-12 lists the component parameters.

Name	*Description*
ahb_size*X*	Size of slave region *X*. Up to 6 regions are supported.
ahb_start*X*	Base address of slave region *X*.
Data Width	AHB data width. Supported values are 32, 64 and 128.

Table 4-12 AHBv2Mux and AHBv2Mux_Lite component parameters

Copyright© 2017 Arm. All Rights Reserved.

Non-Confidential

5 Probes

The simulation probes listed in Table 5-1 are included in the AHBv2 Protocol Bundle.

Name	*Description*
AHBv2 Tracer	Enables tracing of AHB signals on an AHBv2 connection. You can view traced signals in the SoC Designer Simulator waveform window.
AHBv2 BreakPoint	Transaction breakpoint on an AHBv2 connection.
AHBv2 Profiler	Profiles AHBv2 transactions. Profiled data can be viewed in the SoC Designer Simulator profiler window.

Table 5-1 AHBv2 probes

5.1 Tracer

This probe allows tracing of AHB signals. Traced signals can be viewed in the SoC Designer waveform window. To add a tracer probe, right-click on an AHBv2 connection and select **Enable/Disable Tracing**. This launches the **Tracer Properties** dialog (Figure 5-1).

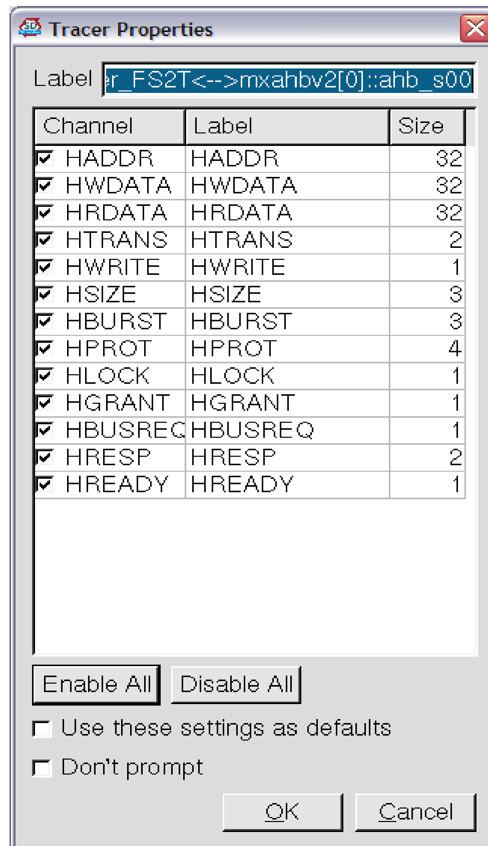

Figure 5-1 Tracer properties

By default, all signals are traced. Disable and enable tracing using the checkboxes located on the left side of the signal.

 Copyright© 2017 Arm. All Rights Reserved.

Non-Confidential

5.2 Breakpoint

To insert a breakpoint probe, either double-click on the connection or right-click on the connection and select **Insert/Remove Breakpoint**. By default, the breakpoint is activated and breaks on any active AHB transaction across the connection. To configure breakpoint conditions, bring up the breakpoint property dialog by right-clicking on the connection and selecting **Edit Breakpoint Properties**. The **Breakpoint Condition** dialog appears (Figure 5-2).

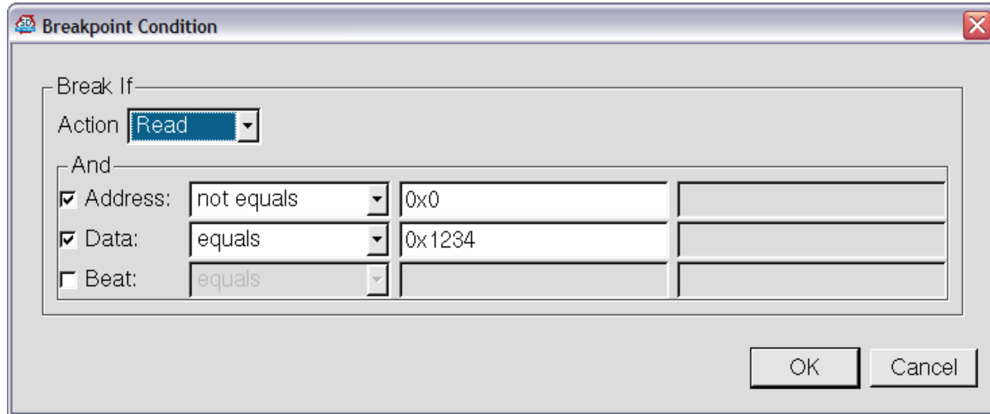

Figure 5-2 Breakpoint properties

5.3 Profiling

The Profiling Probe enables latency profiling over an AHBv2 connection. To enable this probe, right-click on a connection and select **Profiler**, then **Enable**. The **Display** option launches the **Operation vs. Cycles** profiling window.

See the *SoC Designer User Guide* (Arm 100996) for more information.

To view other available profiling streams, open the **Profiling Manager** and locate the connection to which the profiling probe was attached. There are three separate streams available for profiling an AHBv2 connection:

- *Events*: plots the transactions over cycles
- *Latency*: latency for each operation type
- *Address*: plots the accessed address location

Copyright© 2017 Arm. All Rights Reserved.

Non-Confidential

6 Component Wizard

The SoC Designer Component Wizard allows generation of AHBv2 master and slave ports. See the *SoC Designer User Guide* (Arm 100996) for general information regarding the Component Wizard.

6.1 Generating AHBv2 Ports

To generate a model with AHBv2 ports, launch the component wizard from SoC Designer Canvas and proceed to the port definition step. Click **New** to create a new port, and select the desired AHBv2 port type from the port type pulldown menu. See the *SoC Designer User Guide* (Arm 100996) for more information.

To select the correct port type, see Figure 8-2, AHBv2 port types and Figure 8-3, AHBv2 AHB-Lite port types.

The model generation process generates a `.cpp` and a `.h` file for each AHBv2 port that was selected. The port class inherits from one of the specialized template AHBv2 port classes.

Copyright© 2017 Arm. All Rights Reserved.

7 Transactors

Table 7-1 lists the SoC Designer transactors included in the AHBv2 Protocol Bundle.

Name	*Description*
AHB_Slave_FT2S	A transaction-to-signal transactor for a transaction slave port on the slave side of a full AHB bus
AHB_Slave_FS2T	A signal-to-transaction transactor for a transaction master port on the slave side of a full AHB bus
AHB_Master_FT2S	A transaction-to-signal transactor for a transaction slave port on the master side of a full AHB bus
AHB_Master_FS2T	A signal-to-transaction transactor for a transaction master port on the master side of a full AHB bus
AHB_Lite_Slave_FT2S	A transaction-to-signal transactor for a transaction slave port on the slave side of an AHB-Lite bus
AHB_Lite_Slave_FS2T	A signal-to-transaction transactor for a transaction master port on the slave side of an AHB-Lite bus
AHB_Lite_Master_FT2S	A transaction-to-signal transactor for a transaction slave port on the master side of an AHB-Lite bus
AHB_Lite_Master_FS2T	A signal-to-transaction transactor for a transaction master port on the master side of an AHB-Lite bus

Table 7-1 AHBv2 transactors

The figure below illustrates where each of the transactors should be used with Cycle Model components.

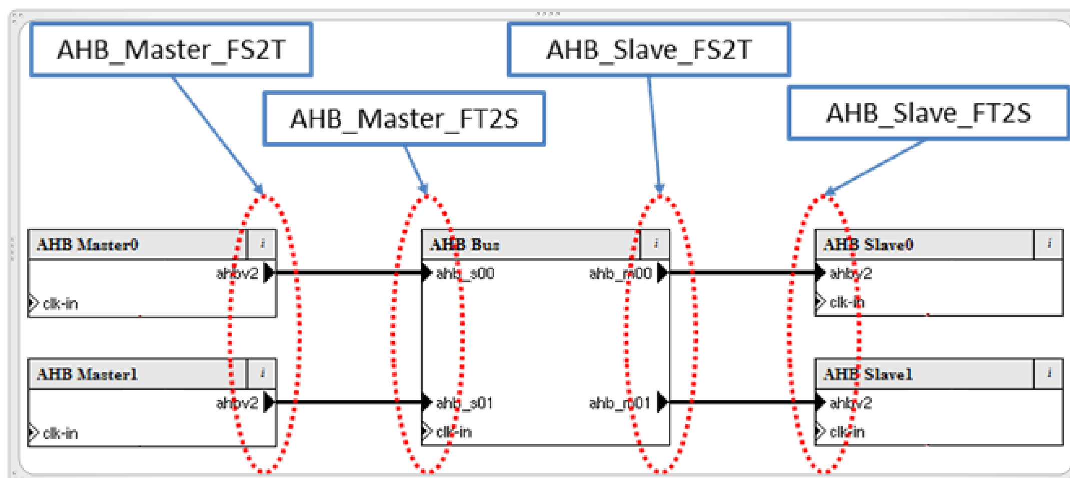

Figure 7-1 AHBv2 transactors

Copyright© 2017 Arm. All Rights Reserved.

Non-Confidential

7.1 AHBv2 Transactors and Cycle Model Studio

Cycle Model Studio (CMS) locates SoC Designer AHBv2 transactors from the path pointed to by the `$MAXSIM_PROTOCOLS` environment variable. If you are working on a CMS project that requires AHBv2 transactors, make sure the AHBv2 Protocol Bundle is installed and set up prior to launching CMS.

8 AHBv2 Port Interfaces

AHBv2 SoC Designer transaction interfaces overcome a problem with the previous version of the AHB interfaces which prohibited a transaction to go through asynchronous paths across a component. The v2 interfaces are described in this chapter.

Note: *Do not confuse v2 with the AMBA protocol version. v2 refers to the SoC Designer transaction interface version for AMBA AHB, and has no relation to the AMBA protocol specification version number.*

8.1 AMBA AHB Interfaces AHBv2 Transaction Ports

AMBA AHB has two distinct interfaces for the master and the slave side of the bus. This is depicted in the figure below.

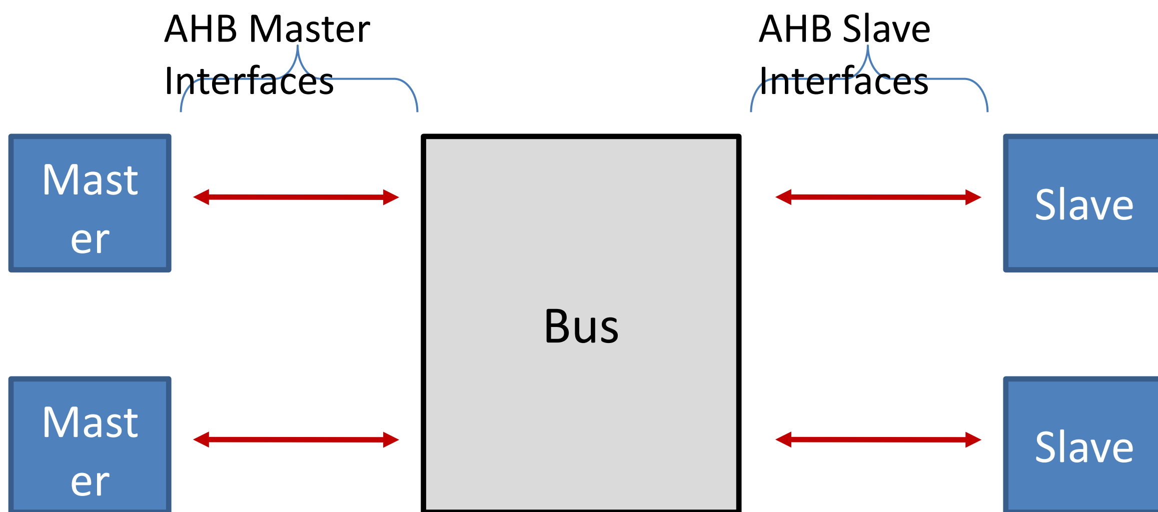

Figure 8-1 AHB Interfaces

AHBv2 ports distinguish between the two distinct AHB interfaces as well as the distinction between the full AMBA AHB and AHB-Lite protocols. The ports are also categorized into transaction master and slave interfaces. This is illustrated in the figures below.

Copyright© 2017 Arm. All Rights Reserved.

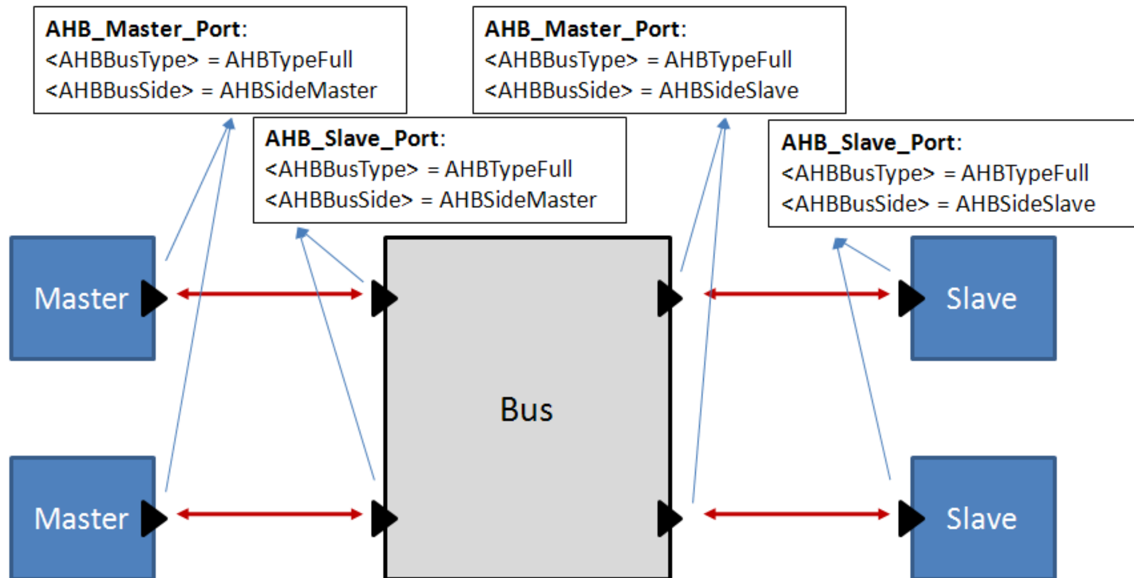

Figure 8-2 AHBv2 port types

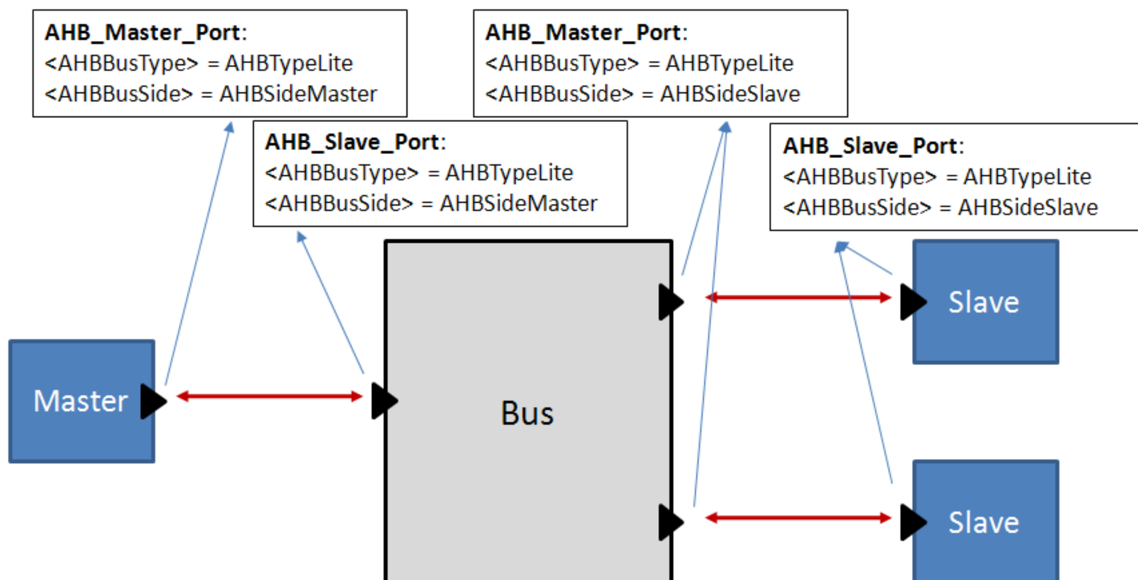

Figure 8-3 AHBv2 AHB-Lite port types

Copyright© 2017 Arm. All Rights Reserved.

Non-Confidential

8.1.1 AHB_Master_Port

The ports that face outwards from a component are categorized as AHB transaction master ports. AHB transaction master port, AHB_Master_Port, is defined in the header file `AHB_Master_Port.h`. AHB master ports are templatized based on the bus type (Full or Lite) and the AHB interface (Master side interface or Slave side interface), and inherit from the base class, `AHB_Master_PortBase`.

8.1.1.1 AHB_Master_PortBase

```
class WEXP_PORT AHB_Master_PortBase : public MxTransactionMasterPort,
public AHBPortIF
{
	friend class AHB_Slave_PortBase;
public:
	AHB_Master_PortBase(CASIModule* o, std::string n);
	virtual ~AHB_Master_PortBase();

	void setWData(uint32_t data, uint8_t idx = 0);

	uint32_t getRData(uint8_t idx);

	/////////////////////////////////////////////
	// MxTransactionMasterPortBase Interface
	/////////////////////////////////////////////
    eslapi::CASIStatus readDbg(uint64_t addr, uint32_t* value,
uint32_t* ctrl);
    eslapi::CASIStatus writeDbg(uint64_t addr, uint32_t* value,
uint32_t* ctrl);

	void connect   (CASITransactionIF* iface);
	void disconnect(CASITransactionIF* iface);
	void init(uint32_t addrWidth, uint32_t dataWidth);

private: // disabled methods
	AHB_Master_PortBase();
	AHB_Master_PortBase(const AHB_Master_PortBase&);
	AHB_Master_PortBase& operator=(const AHB_Master_PortBase&);
};
```

8.1.1.1.1 void setWData(uint32_t data, uint8_t idx = 0)

Use this method to set `HWDATA`. This should be called multiple times for data > 32bits, with incrementing `idx`.

8.1.1.1.2 uint32_t getRData(uint8_t idx);

Use this method to retrieve `HRDATA`. This should be called multiple times for data > 32bits, with incrementing `idx`.

 Copyright© 2017 Arm. All Rights Reserved.

8.1.1.1.3 readDbg/writeDbg

These methods can be used to initiate debug (zero-cycle) transactions. The desired data size should be encoded into the `ctrl` parameter. Use `AHB2_SIZE enum` defined in `AHB_TLM.h` to encode the data size.

8.1.1.1.4 init(uint32_t addrWidth, uint32_t dataWidth)

Call this method during the CASI `init` phase. The default implementation assumes 32 bits for both the address and the data bus.

8.1.1.2 AHB_Master_Port Template Specialization

There are four template specializations for AHB master ports:

AHB_Master_Port Type	*Description*
AHBTypeFull, AHBSideMaster	Transaction master port for a full AHB interface, AHB master interface.
AHBTypeLite, AHBSideMaster	Transaction master port for an AHB-Lite interface, AHB master interface.
AHBTypeFull, AHBSideSlave	Transaction master port for a full AHB interface, AHB slave interface.
AHBTypeLite, AHBSideSlave	Transaction master port for an AHB-Lite interface, AHB slave interface.

Table 8-1 AHB master port types

8.1.1.2.1 AHB_Master_Port<AHBTypeFull, AHBSideMaster>

This is the master port type to use for a port that is on the master side of a full AHB bus.

```
template<>
class WEXP_PORT AHB_Master_Port<AHBTypeFull, AHBSideMaster> : public
AHB_Master_PortBase
{
	public:
	AHB_Master_Port(CASIModule* o, std::string n);
	virtual ~AHB_Master_Port() {}

	void setBusReq(bool busreq);

	void setAddr(uint64_t addr, uint64_t trans, bool write, uint8_t
size, uint8_t burst, uint8_t prot, bool lock);

	////////////////////////
	// AHBPortIF Interface
	////////////////////////
	uint64_t getSig(AHB2_SIGNAL_IDX sigIdx);
	bool setSig(AHB2_SIGNAL_IDX sigIdx, uint64_t val);
	void clear();
};
```


Copyright© 2017 Arm. All Rights Reserved.

Non-Confidential

8.1.1.2.1.1 void setBusReq(bool busreq)

Use this method to set `HBUSREQ`.

8.1.1.2.1.2 void setAddr(…)

Use this method to set all AHB address and control-related signals.

8.1.1.2.1.3 uint64_t getSig(AHB2_SIGNAL_IDX sigIdx)

This method returns the specified signal value latched during the last CASI `communicate` phase.

8.1.1.2.1.4 bool setSig(AHB2_SIGNAL_IDX sigIdx, uint64_t val)

Use this method to set individual AHB master signals.

8.1.1.2.2 AHB_Master_Port<AHBTypeLite, AHBSideMaster>

This is the master port type to use for a port that is on the master side of an AHB-Lite bus.

```
template<>
class WEXP_PORT AHB_Master_Port<AHBTypeLite, AHBSideMaster> : public
AHB_Master_PortBase
{
	public:
	AHB_Master_Port(CASIModule* o, std::string n);
	virtual ~AHB_Master_Port() {}

	void setAddr(uint64_t addr, uint64_t trans, bool write, uint8_t
size, uint8_t burst, uint8_t prot, bool lock);

	////////////////////////
	// AHBPortIF Interface
	////////////////////////
	uint64_t getSig(AHB2_SIGNAL_IDX sigIdx);
	bool setSig(AHB2_SIGNAL_IDX sigIdx, uint64_t val);
	void clear();
};
```

8.1.1.2.2.1 void setAddr(…)

Use this method to set all AHB address and control related signals.

8.1.1.2.2.2 uint64_t getSig(AHB2_SIGNAL_IDX sigIdx)

This method returns the specified signal value latched during the last CASI `communicate` phase.

8.1.1.2.2.3 bool setSig(AHB2_SIGNAL_IDX sigIdx, uint64_t val)

Use this method to set individual AHB master signals.

8.1.1.2.3 AHB_Master_Port<AHBTypeFull, AHBSideSlave>

This is the master port type to use for a port that is on the slave side of a full AHB bus.

```
template<>
```


Copyright© 2017 Arm. All Rights Reserved.

```
class WEXP_PORT AHB_Master_Port<AHBTypeFull, AHBSideSlave> : public
AHB_Master_PortBase
{
	public:
	AHB_Master_Port(CASIModule* o, std::string n);
	virtual ~AHB_Master_Port() {}

	void setAddr(uint64_t addr, uint64_t trans, bool write, uint8_t
size, uint8_t burst, uint8_t prot, bool mastlock, uint8_t master, bool
sel, bool ready);

	////////////////////////
	// AHBPortIF Interface
	////////////////////////
	uint64_t getSig(AHB2_SIGNAL_IDX sigIdx);
	bool setSig(AHB2_SIGNAL_IDX sigIdx, uint64_t val);
	void clear();
};
```

8.1.1.2.3.1 void setAddr(…)

Use this method to set the AHB address and control related signals.

8.1.1.2.3.2 uint64_t getSig(AHB2_SIGNAL_IDX sigIdx)

This method returns the specified signal value which was latched during the last CASI `communicate` phase.

8.1.1.2.3.3 bool setSig(AHB2_SIGNAL_IDX sigIdx, uint64_t val)

Use this method to set individual signals on the port.

8.1.1.2.4 AHB_Master_Port<AHBTypeLite, AHBSideSlave>

This is the master port type to use for a port that is on the slave side of an AHB-Lite bus.

```
template<>
class WEXP_PORT AHB_Master_Port<AHBTypeLite, AHBSideSlave> : public
AHB_Master_PortBase
{
	public:
	AHB_Master_Port(CASIModule* o, std::string n);
	virtual ~AHB_Master_Port() {}

	void setAddr(uint64_t addr, uint64_t trans, bool write, uint8_t
size, uint8_t burst, uint8_t prot, bool lock, bool sel, bool ready);

	////////////////////////
	// AHBPortIF Interface
	////////////////////////
	uint64_t getSig(AHB2_SIGNAL_IDX sigIdx);
	bool setSig(AHB2_SIGNAL_IDX sigIdx, uint64_t val);
	void clear();
};
```


Copyright© 2017 Arm. All Rights Reserved.

Non-Confidential

8.1.1.2.4.1 void setAddr(…)

Use this method to set the AHB address and control related signals.

8.1.1.2.4.2 uint64_t getSig(AHB2_SIGNAL_IDX sigIdx)

This method returns the specified signal value which was latched during the last CASI `communicate` phase.

8.1.1.2.4.3 bool setSig(AHB2_SIGNAL_IDX sigIdx, uint64_t val)

Use this method to set individual signals on the port.

Note: All AHBv2 port headers are located in `$MAXSIM_PROTOCOLS/AHBv2/include`.

8.1.2 AHB_Slave_Port

The ports going into a component are categorized as AHB transaction slave ports. The AHB transaction slave port, AHB_Slave_Port, is defined in the header file `AHB_Slave_Port.h`. These slave ports inherit from the base class, `AHB_Slave_PortBase`, and are divided into special templatized classes based on the AHB bus and the interface type.

8.1.2.1 AHB_Slave_PortBase

```
class WEXP_PORT AHB_Slave_PortBase : public CASITransactionSlave,
public AHBPortIF
{
	friend class AHB_Master_PortBase;

public:
	AHB_Slave_PortBase(CASIModule* o, std::string n);
	virtual ~AHB_Slave_PortBase();

	void setRData(uint32_t data, uint8_t idx = 0);

	uint32_t getWData(uint8_t idx);

	void init(uint32_t addrWidth, uint32_t dataWidth);

private: // disabled methods
	AHB_Slave_PortBase();
	AHB_Slave_PortBase(const AHB_Slave_PortBase&);
	AHB_Slave_PortBase& operator=(const AHB_Slave_PortBase&);

	CASIMemoryMapConstraints puMemoryMapConstraints;

public:
	/* debug accesses */
    virtual eslapi::CASIStatus readDbg(uint64_t addr, uint32_t* value,
uint32_t* ctrl);
    virtual eslapi::CASIStatus writeDbg(uint64_t addr, uint32_t* value,
uint32_t* ctrl);

    /* Memory map functions */
```


Copyright© 2017 Arm. All Rights Reserved.

Non-Confidential

```
    virtual int getNumRegions();
    virtual void getAddressRegions(uint64_t* start, uint64_t* size,
string* name);
    virtual void setAddressRegions(uint64_t* start, uint64_t* size,
string* name);
    virtual CASIMemoryMapConstraints* getMappingConstraints();

      //**********************************************************
      //*********** IGNORE EVERYTHING FROM THIS LINE *************
      //**********************************************************
public:
    /* Synchronous access functions */
    virtual eslapi::CASIStatus read(uint64_t addr, uint32_t* value,
uint32_t* ctrl);
    virtual eslapi::CASIStatus write(uint64_t addr, uint32_t* value,
uint32_t* ctrl);

    /* Asynchronous access functions */
    virtual eslapi::CASIStatus readReq(uint64_t addr, uint32_t* value,
uint32_t* ctrl,
                             CASITransactionCallbackIF* callback);
    virtual eslapi::CASIStatus writeReq(uint64_t addr, uint32_t* value,
uint32_t* ctrl,
                              CASITransactionCallbackIF* callback);

    /* Arbitration functions */
    virtual eslapi::CASIGrant requestAccess(uint64_t addr);
    virtual eslapi::CASIGrant checkForGrant(uint64_t addr);

      /* CASI : new shared-memory based asynchronous transaction
functions */
      virtual void cancelTransaction(CASITransactionInfo* info);
      virtual eslapi::CASIStatus debugTransaction(CASITransactionInfo*
info);

      //***********************************************************
      //*********** IGNORE EVERYTHING TO THIS LINE ****************
      //***********************************************************
};
```

8.1.2.1.1 void setRData(uint32_t data, uint8_t idx = 0);

Use this method to set `HRDATA`. This must be called multiple times with incrementing `idx` for data greater than 32bits.

8.1.2.1.2 uint32_t getWData(uint8_t idx);

Use this method to retrieve `HWDATA`. This must be called multiple times with incrementing `idx` for data greater than 32bits.

8.1.2.1.3 void init(uint32_t addrWidth, uint32_t dataWidth);

This should be called from the CASI `init` phase to initialize the address and data bus widths. The default implementation assumed 32 bits for both address and data.

Copyright© 2017 Arm. All Rights Reserved.

Non-Confidential

Copyright© 2017 Arm. All Rights Reserved.

Non-Confidential

8.1.2.1.4 readDbg/writeDbg

These methods are used for debug (zero-cycle) accesses. These methods should be implemented by the port owner to support debug accesses from the connected masters. The **ctrl** parameter has an encoded value of the data size to be transferred. This encoding uses the *AHB2_SIZE* enum defined in `AHB_TLM.h`.

8.1.2.1.5 Memory map functions

The following methods need to be implemented for memory mapping. See the *SoC Designer ESL API Developer's Guide* (Arm 101142) for information regarding these methods.

- `virtual int getNumRegions();`
- `virtual void getAddressRegions(uint64_t* start, uint64_t* size, string* name);`
- `virtual void setAddressRegions(uint64_t* start, uint64_t* size, string* name);`
- `virtual CASIMemoryMapConstraints* getMappingConstraints();`

8.1.2.2 AHB_Slave_Port Template Specialization

There are four template specializations for AHB slave ports:

AHB_Slave_Port Type	*Description*
AHBTypeFull, AHBSideMaster	Transaction slave port for a full AHB interface, AHB master interface.
AHBTypeLite, AHBSideMaster	Transaction slave port for an AHB-Lite interface, AHB master interface.
AHBTypeFull, AHBSideSlave	Transaction slave port for a full AHB interface, AHB slave interface.
AHBTypeLite, AHBSideSlave	Transaction slave port for an AHB-Lite interface, AHB slave interface.

Table 8-2 AHB slave port types

Copyright© 2017 Arm. All Rights Reserved.

Non-Confidential

8.1.2.2.1 AHB_Slave_Port<AHBTypeFull, AHBSideMaster>

This is the slave port type to use for a port that is on the master side of a full AHB bus.

```
template<>
class WEXP_PORT AHB_Slave_Port<AHBTypeFull, AHBSideMaster> : public
AHB_Slave_PortBase
{
	public:
	AHB_Slave_Port(CASIModule* o, std::string n);
	virtual ~AHB_Slave_Port() {}

	void setGrant(bool grant);
	void setResponse(bool ready, uint8_t resp);

	////////////////////////
	// AHBPortIF Interface
	////////////////////////
	uint64_t getSig(AHB2_SIGNAL_IDX sigIdx);
	bool setSig(AHB2_SIGNAL_IDX sigIdx, uint64_t val);
	void clear();
};
```

8.1.2.2.1.1 void setGrant(bool grant)

Use this method to set `HGRANT`.

8.1.2.2.1.2 void setResponse(bool ready, uint8_t resp)

Use this method to set `HREADY` and `HRESP`.

8.1.2.2.1.3 uint64_t getSig(AHB2_SIGNAL_IDX sigIdx)

Use this method to retrieve individual AHB signals.

8.1.2.2.1.4 bool setSig(AHB2_SIGNAL_IDX sigIdx, uint64_t val)

Use this method to set individual AHB signals.

8.1.2.2.2 AHB_Slave_Port<AHBTypeLite, AHBSideMaster>

This is the slave port type to use for a port that is on the master side of an AHB-Lite bus.

```
template<>
class WEXP_PORT AHB_Slave_Port<AHBTypeLite, AHBSideMaster> : public
AHB_Slave_PortBase
{
	public:
	AHB_Slave_Port(CASIModule* o, std::string n);
	virtual ~AHB_Slave_Port() {}

	void setResponse(bool ready, uint8_t resp);

	////////////////////////
	// AHBPortIF Interface
	////////////////////////
	uint64_t getSig(AHB2_SIGNAL_IDX sigIdx);
```


Copyright© 2017 Arm. All Rights Reserved.

```
        bool setSig(AHB2_SIGNAL_IDX sigIdx, uint64_t val);
        void clear();
};
```

8.1.2.2.2.1 void setResponse(bool ready, uint8_t resp)

Use this method to set `HREADY` and `HRESP`.

8.1.2.2.2.2 uint64_t getSig(AHB2_SIGNAL_IDX sigIdx)

Use this method to retrieve individual AHB signals.

8.1.2.2.2.3 bool setSig(AHB2_SIGNAL_IDX sigIdx, uint64_t val)

Use this method to set individual AHB signals.

8.1.2.2.3 AHB_Slave_Port<AHBTypeFull, AHBSideSlave>

Use this slave port type for a port that is on the slave side of a full AHB bus.

```
template<>
class WEXP_PORT AHB_Slave_Port<AHBTypeFull, AHBSideSlave> : public
AHB_Slave_PortBase
{
        public:
        AHB_Slave_Port(CASIModule* o, std::string n);
        virtual ~AHB_Slave_Port() {}

        void setResponse(bool readyout, uint8_t resp);

        ////////////////////////
        // AHBPortIF Interface
        ////////////////////////
        uint64_t getSig(AHB2_SIGNAL_IDX sigIdx);
        bool setSig(AHB2_SIGNAL_IDX sigIdx, uint64_t val);
        void clear();
};
```

8.1.2.2.3.1 void setResponse(bool readyout, uint8_t resp)

Use this method to set `HREADYOUT` and `HRESP`.

8.1.2.2.3.2 uint64_t getSig(AHB2_SIGNAL_IDX sigIdx)

Use this method to retrieve individual AHB signals.

8.1.2.2.3.3 bool setSig(AHB2_SIGNAL_IDX sigIdx, uint64_t val)

Use this method to set individual AHB signals.

Copyright© 2017 Arm. All Rights Reserved.

Non-Confidential

8.1.2.2.4 AHB_Slave_Port<AHBTypeLite, AHBSideSlave>

Use this slave port type for a port that is on the slave side of an AHB-Lite bus.

```
template<>
class WEXP_PORT AHB_Slave_Port<AHBTypeLite, AHBSideSlave> : public
AHB_Slave_PortBase
{
	public:
	AHB_Slave_Port(CASIModule* o, std::string n);
	virtual ~AHB_Slave_Port() {}

	void setResponse(bool readyout, uint8_t resp);

	////////////////////////
	// AHBPortIF Interface
	////////////////////////
	uint64_t getSig(AHB2_SIGNAL_IDX sigIdx);
	bool setSig(AHB2_SIGNAL_IDX sigIdx, uint64_t val);
	void clear();
};
```

8.1.2.2.4.1 void setResponse(bool readyout, uint8_t resp)

Use this method to set `HREADYOUT` and `HRESP`.

8.1.2.2.4.2 uint64_t getSig(AHB2_SIGNAL_IDX sigIdx)

Use this method to retrieve individual AHB signals.

8.1.2.2.4.3 bool setSig(AHB2_SIGNAL_IDX sigIdx, uint64_t val)

Use this method to set individual AHB signals.

8.1.3 AHBPortIF

AHBPortIF is an abstract class for all AHBv2 port classes. AHBv2 port classes contain two sub-ports (AHB_Sender_Port and AHB_Receiver_Port) which are not visible in SoC Designer Canvas or Simulator. These sub-ports enable bi-directional communication between two AHB ports.

The underlying CASI communication method used behind AHBv2 is `driveTransaction`, but it is not recommended that users directly implement `driveTransaction`, as the `AHB_Master_Port` and `AHB_Slave_Port` port classes already contain methods for the user to model cycle accurate AHB transactions. The underlying `driveTransaction` is only used to transfer the AHB signals set by the port classes from one component to another.

Copyright© 2017 Arm. All Rights Reserved.

Non-Confidential

8.1.4 Transaction Phases

AHBv2 relies on a two-phased communication mechanism. The CASI `communicate` phase is used only for the transfer of data, and the `update` phase is used for latching the data and updating the internal state machine.

CASI Phase	*Description*
`communicate`	Call `sendDrive` to transfer the AHB signals set during the last update phase. `sendDrive` sends out the buffered data which are the updated signals since the last `sendDrive` call.
`update`	Use `getSig` to latch the data driven during the current `communicate` phase. `getSig` can be used to set the new data to be driven out in the next `communicate` phase.

Table 8-3 AHBv2 transaction phases

8.1.5 Examples

Source code examples are included in the protocol bundle installation. See `$MAXSIM_PROTOCOLS/AHBv2/src/AHBv2_Master` for an example implementation of an AHB master, and `$MAXSIM_PROTOCOLS/AHBv2/src/AHBv2_Slave` for an example of an AHB slave model.

Copyright© 2017 Arm. All Rights Reserved.

Non-Confidential

8.2 AHB Extensions

AHBv2 defines signals that are reserved for supporting the additional AMBA2 extensions for specific Arm cores. This section documents the mapping between these extensions and the sideband signals defined in AHBv2. This section describes the `AHB2_SIGNAL_IDX` enumerations declared in `$MAXSIM_PROTOCOLS/AHBv2/include/AHB_TLM.h`.

8.2.1 Cortex-M3 Sideband Signals

Signal	*AHB2_SIGNAL_IDX*
EXREQ	SIDEBAND0
MEMATTR	SIDEBAND1
EXRESP	SIDEBAND2

8-4 Cortex-M3 AHB Extensions

Copyright© 2017 Arm. All Rights Reserved.

Non-Confidential