HB0397

CoreAXItoAHBL v3.1

Microsemi
Power Matters.

Microsemi Corporate Headquarters
One Enterprise, Aliso Viejo,
CA 92656 USA
Within the USA: +1 (800) 713-4113
Outside the USA: +1 (949) 380-6100
Sales: +1 (949) 380-6136
Fax: +1 (949) 215-4996
E-mail: sales.support@microsemi.com
www.microsemi.com

About Microsemi

Microsemi Corporation (Nasdaq: MSCC) offers a comprehensive portfolio of semiconductor and system solutions for aerospace & defense, communications, data center and industrial markets. Products include high-performance and radiation-hardened analog mixed-signal integrated circuits, FPGAs, SoCs and ASICs; power management products; timing and synchronization devices and precise time solutions, setting the world's standard for time; voice processing devices; RF solutions; discrete components; enterprise storage and communication solutions, security technologies and scalable anti-tamper products; Ethernet solutions; Power-over-Ethernet ICs and midspans; as well as custom design capabilities and services. Microsemi is headquartered in Aliso Viejo, Calif., and has approximately 4,800 employees globally. Learn more at www.microsemi.com.

50200397-5/11.16

1 Revision History

The revision history describes the changes that were implemented in the document. The changes are listed by revision, starting with the most current publication.

1.1 Revision 5.0

Updated changes related to CoreAXItoAHBL v3.1.

1.2 Revision 4.0

Updated changes related to CoreAXItoAHBL v3.0.

1.3 Revision 3.0

Updated changes related to CoreAXItoAHBL v2.2.

1.4 Revision 2.0

Updated changes related to CoreAXItoAHBL v2.1.

1.5 Revision 1.0

Revision 1.0 was the first publication of this document. Created for CoreAXItoAHBL v2.0.

Contents

Figures

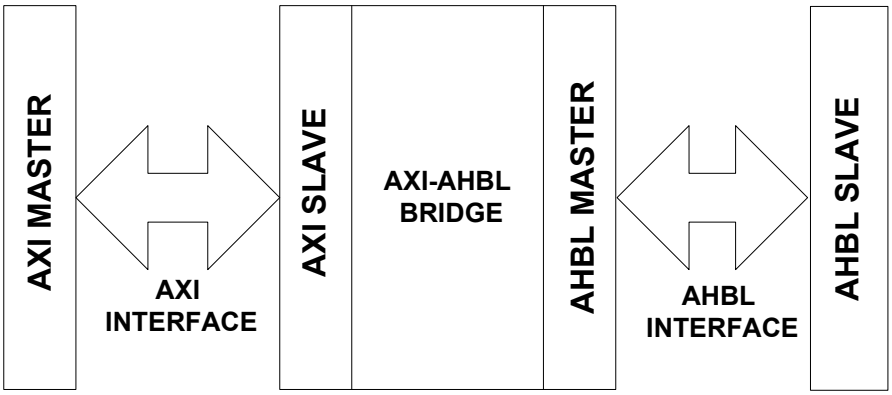
Tables

2 Introduction

2.1 Overview

The CoreAXItoAHBL IP core is an advanced extensible interface (AXI) slave and an advanced high-performance bus lite (AHB-Lite) master. This provides an interface (bridge) between the AXI domain and AHB-Lite domain. CoreAXItoAHBL allows an AXI bus system to be connected to an AHB-Lite bus, enabling an AXI master to communicate with an AHBL slave/subsystem.

Figure 1 • **CoreAXItoAHBL Bridge Block Diagram**



2.2 Features

- Provides an interface (bridge) between the advanced extensible interface (AXI) domain and advanced high-performance lite bus (AHB-Lite) domain
- Makes alternate AXI write and AXI read transactions possible
- Supports AXI data bus width of 64-bits, with transfer size of 64/32-bit
- Maximum number of AXI beats or transfers of 16
- Supports unaligned AXI write / read transactions
- Permits the AXI and AHBL clocks to be derived from asynchronous sources
- Supports narrow transfers for the last transfer in AXI write transactions using write strobes
- Provides ERROR/OKAY response for every AXI master transaction
- Supports AHB data bus width of 32-bits
- Prevents sequential AHBL transfers from crossing 1 KB boundaries

2.3 Core Version

This handbook applies to CoreAXItoAHBL version 3.1.

2.4 Supported Families

- PolarFire
- SmartFusion®2
- IGLOO®2

2.5 Device Utilization and Performance

Utilization and performance data is listed in Table 1 for the SmartFusion2 (M2S050) device family. The data listed in this table is indicative only. The overall device utilization and performance of the core is system dependent.

Table 1 • **Device Utilization and Performance**

Family	NO_BURST_TRANS	WRAP_SUPPORT	ASYNC_CLOCKS	ID_WIDTH	Logic Elements				ACLK Frequency (MHz)	HCLK Frequency (MHz)
					Sequential	Combinatorial	Total	%		
SmartFusion2	0	0	0	4	784	1426	2210	3.92	132	142
SmartFusion2	0	0	1	4	792	1433	2225	3.95	143	144
SmartFusion2	0	1	0	4	807	2229	3036	5.39	149	119
SmartFusion2	0	1	1	4	819	2242	3061	5.43	140	121
SmartFusion2	1	0	0	4	771	1358	2129	3.78	157	155
SmartFusion2	1	0	1	4	779	1394	2173	3.85	153	155
SmartFusion2	1	1	0	4	784	1939	2723	4.83	144	128
SmartFusion2	1	1	1	4	784	1985	2769	4.91	151	131

Note: The data in this table is achieved using Verilog RTL, typical synthesis and layout settings (Non timing-driven mode) speed grade was -1. The core uses 8, 64 * 18 RAM blocks for all configurations.

3 Functional Description

CoreAXItoAHBL appears as a slave on the AXI bus and operates as a master on the AHB-Lite bus. Read and write transactions on the AXI interface are converted into corresponding transfers on the AHB-Lite interface.

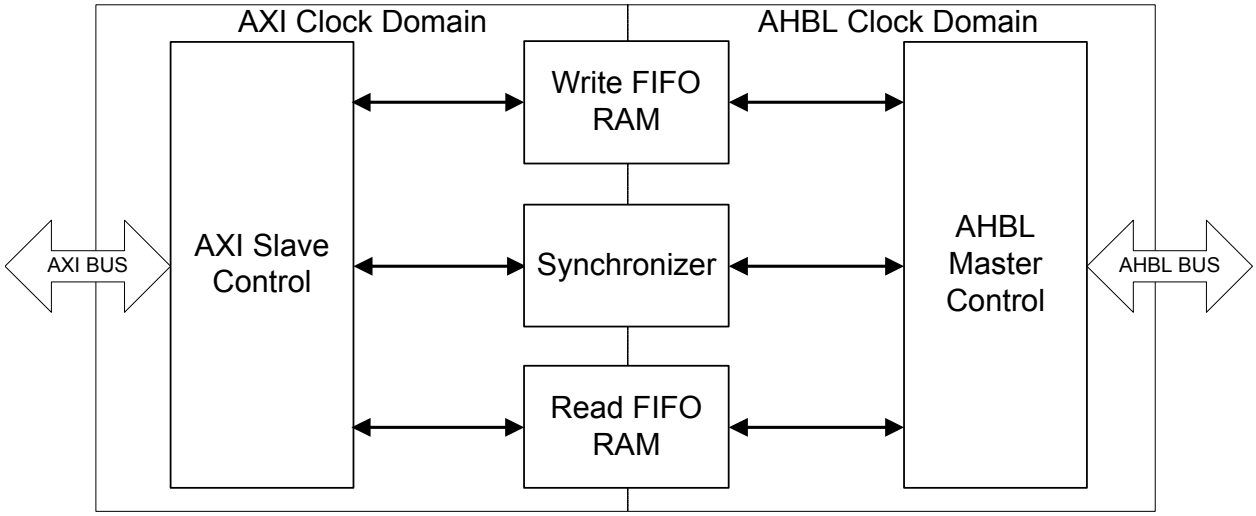
The ACLK and HCLK clocks are configurable to be synchronous or asynchronous via parameter/generic. The core implements the clock-domain-crossing (CDC) logic, where the AHB clock and AXI clock are asynchronous to each other.

CoreAXItoAHBL consists of the following four major functional blocks:

- write memory buffer
- read memory buffer
- AXI slave controller
- AHB-Lite master controller

A basic block diagram of the design for CoreAXItoAHBL is shown in Figure 2.

Figure 2 • **CoreAXItoAHBL Block Diagram**



3.1 AXI Slave Control

The AXI Slave Control block provides the AXI slave interface of the bridge. This block is responsible for storing the AXI write data in the Write FIFO RAM block and returning the read data and error responses to the AXI master from the Read FIFO RAM and AHB Master Control blocks.

Once address information has been detected (AxVALID high) and acknowledged (AxREADY high) on either the AXI write address or read address channels, the AXI Slave Control block de-asserts the AWREADY and ARREADY signals until the transaction related to that address has been completed (that is, response returned to the AXI master and acknowledgment received). Write address requests have priority over read address requests in the AXI Slave Control block. If the execution order of write and read operations is critical, it is important to ensure that a write request is not issued after a read request when the core is already processing a transaction (as the write will be allocated priority and get performed ahead of the read operation). This block supports unaligned write transactions (that is, transactions performed on addresses which are not aligned to the transfer size) through the use of write strobes and unaligned read transfers through the use of address offsets.

3.2 Write FIFO RAM

The Write FIFO RAM block is a 16 deep, 64-bit wide synchronous write, asynchronous read RAM block. It stores the AXI write data received by the AXI Slave Control block. The AHBL Master Control block generates the read enable to this FIFO when the AXI Slave Control block has stored all data from the AXI write transaction.

3.3 Read FIFO RAM

The Read FIFO RAM block is a 16 deep, 64-bit wide synchronous write, asynchronous read RAM block. It stores the AXI read data received by the AHBL Master Control block. The AXI Slave Control block generates the read enable to the FIFO when the AHBL Master Control block has stored all data from the AHBL read transfers.

3.4 AHBL Master Control

The AHBL Master Control block is the AHBL master interface of the bridge. This block generates a number of AHB write and read transactions on the AHBL bus based on the start address, burst type and number of valid bytes specified by the AXI Slave Control block. Based on the configuration parameters, the AHBL Master Control blocks ensure that transfer of the largest burst and transfer size possible are performed. The AHBL address is incremented based on the size and burst type calculated. Error responses received on the AHBL interface are forwarded to the AXI Slave Control block.

The AHBL Master Control block, controls the read enable to the Write FIFO RAM block and the write enable to the Read FIFO RAM block.

3.5 Clock Domains

The CoreAXItoAHBL bridge consists of the following two clock domains:

- AXI clock domain
- AHB clock domain

The AXI Slave Control block operates in the AXI clock domain, while the AHB Master Control block operates in the AHB clock domain. Where the two clock domains are derived from asynchronous sources, the core makes use of the Write FIFO RAM and Read FIFO RAM blocks to pass data between the two clock domains. Toggling signals are passed between the clock domains to indicate that data is valid in the corresponding FIFO for sampling.

3.6 AXI-AHBL Interface Support

3.6.1 AHBL Address (HADDR) Generation

Since, the AXI master issues only the start address for read or write transactions, HADDR is required to be generated for the subsequent read or write beats of the burst transfer. When a valid read or write request is issued by the AXI interface, the start address of the transfer is registered. For subsequent beats, the address (HADDR) is generated depending on the type (ARBURST or AWBURST), and length (ARLEN/AWLEN) of the burst.

The AHBL Master Control block ensures that sequential AHB transfers do not cross 1 KB boundaries, to support the minimum slave size defined in the AHB-Lite specification.

3.6.2 AXI Transfer Size: Translation of AXI Interface → AHBL Interface

The core supports 64-bit transfer size (ARSIZE and AWSIZE = 3'b011) and a 32-bit transfer size (ARSIZE and AWSIZE = 3'b010). A slave error response will be returned to the AXI master, if a transfer of size other than 64/32-bit is attempted. To perform a write of less than 64-bits to a slave, the master must use a transfer size of 64-bits and make use of the write strobes to write to specific byte locations within the 8 byte locations addressed by the transfer. Similarly, to perform a write of less than 32-bits to a slave, the master must use a transfer size of 32-bits and make use of the write strobes to write to specific byte locations within the 4 byte locations addressed by the transfer.

Note: Sparse assertion of the write strobes (that is, holes in the write strobes) are not supported by the core. For example, WSTRB = 8'h5F (for 64-bit) and WSTRB = 8'h05 (for 32-bit).

For read data transactions, the AXI master must use a transfer size of 64/32-bits. To avoid read sensitive locations located in the lower byte locations addressed by the transfer, the AXI master may use a transfer size of 64-bits and offset the lower 3-bits of the read address (ARADDR).

3.6.3 AXI Burst Length: Translation of AXI Interface → AHBL Interface

The core supports a maximum of 16 AXI transfers per transaction. Depending on the burst length and type of the AXI transaction, the AXI transaction is translated into multiple sequential and non-sequential AHB transfers. The AHB Master Control block supports 4-, -8-, and 16-beat incrementing burst AHB transfers. If an unaligned AXI transaction is received, which is not 32-bit aligned, the AHB Master Control block will perform a number of non-sequential transfers to move to a 32-bit aligned address before attempting AHB burst transfers.

A parameter exists to prevent the core from generating AHBL bursts when connecting to simple slaves/subsystems. Once this parameter is set, the core will only issue non-sequential transactions on the AHBL interface.

3.6.4 AXI Burst Type: Translation of AXI Interface → AHBL Interface

3.6.4.1 Fixed Address Bursts

The core provides support for AXI fixed address bursts. AXI transactions of this burst type perform repeated access to the same location, typically peripheral FIFOs, where the address remains constant for every beat of the burst. To convert this transaction type to the AHB interface, the core generates a number of non-sequential AHBL transfers, with incrementing addresses based on the AHBL transfer size. When the address reaches a 64/32-bit boundary depending on AXI transfer size, it wraps back to the initial base address specified by the AXI master. CoreAXItoAHBL supports unaligned addresses being specified by the AXI master for fixed address burst transactions.

Table 2 shows the resultant AHB transfers generated when the AXI master performs fixed address AXI transactions.

Table 2 • **Fixed Address AXI Transaction to AHB Transfer Conversion**

AWADDR/ARADDR[2:0]	AXI Transfer Size	WSTRB (in case of write transaction)	Burst Length (AxLEN + 1)	Resultant AHB Transfers		
				Non-sequential		
				8-bit	16-bit	32-bit
3’b000 (64-bit aligned)	64-bit	8’hFF	2	0	0	4
3’b100 (unaligned)	64-bit	8’hF0	2	0	0	2
3’b001 (unaligned)	64-bit	8’hFE	2	2	2	2
3’b000 (32-bit aligned)	32-bit	8’h0F	2	0	0	2
3’b100 (32-bit aligned)	32-bit	8’hF0	2	0	0	2
3’b001 (unaligned)	32-bit	8’h0E	2	2	2	0
3’b110 (unaligned)	32-bit	8’hC0	2	0	2	0

Note: CoreAXItoAHBL does not permit the use of narrow transfers for fixed address AXI write transactions. For example, AWSIZE = 3’d3 (64-bit), WSTRB = 8’h7F and AWSIZE = 3'd2 (32-bit), WSTRB = 8'h07

3.6.4.2 Incrementing Address Bursts

CoreAXItoAHBL implements support for AXI incrementing address bursts. AXI transactions of this burst type increment by the transfer size for every transfer of the transaction. To convert this transaction type to the AHB interface, the core generates a number of non-sequential and sequential AHB transfers, depending on the number of valid bytes in the AXI transaction. If an unaligned start address is specified by the AXI master, the core performs a number of non-sequential transactions to move onto a 32-bit aligned address, before generating a combination of 4-, 8-, and 16-beat, 32-bit AHB transfers. If the write

strobes implement a narrow transaction during the last beat of the AXI transaction, then the core finishes with a combination of non-sequential AHBL transfers.

Table 3 shows the resultant AHB transfers generated when the AXI master performs incrementing address AXI write transactions.

Table 3 • **Incrementing Address AXI Write Transaction to AHB Transfer Conversion**

AWADDR [2:0] (Offset optional)	AXI Transfer Size	WSTRB during First Beat	WSTRB during Last Beat	Burst Length (AWLEN + 1)	Resultant AHB Transfers (Assuming transaction doesn't cross a 1 KB boundary and NO_BURST_TRANS = 0)					
					Non-sequential			Sequential		
					8-bit	16-bit	32-bit	32-bit 4-beat	32-bit 8-beat	32-bit 16-beat
3'b000	64-bit	8'hFF (64-bit aligned)	8'hFF	8	0	0	0	0	0	1
3'b100	64-bit	8'hF0 (unaligned)	8'hFF	8	0	0	3	1	1	0
3'b001	64-bit	8'hFE (unaligned)	8'hFF	8	1	1	3	1	1	0
3'b000	64-bit	8'hFF (64-bit aligned)	8'hEF (narrow)	8	1	1	3	1	1	0
3'b100	64-bit	8'hF0 (unaligned)	8'h0F (narrow)	8	0	0	2	1	1	0
3'b001	64-bit	8'h02 (unaligned & narrow)	N/A	1	1	0	0	0	0	0
3'b000	32-bit	8'h0F (32-bit aligned)	8'hF0	16	0	0	0	0	0	1
3'b100	32-bit	8'hF0 (32-bit aligned)	8'h0F	16	0	0	0	0	0	1
3'b001	32-bit	8'h0E (unaligned)	8'hF0	16	1	1	3	1	1	0
3'b110	32-bit	8'hC0 (unaligned)	8'h0F	16	0	1	3	1	1	0
3'b000	32 bit	8'h0F (32-bit aligned)	8'hE0 (narrow)	16	1	1	3	1	1	0
3'b101	32-bit	8'hE0 (unaligned)	8'h0C (narrow)	16	1	2	2	1	1	0

Note: For AWSIZE = 3'b011 (64-bit) CoreAXItoAHBL expects all byte lanes to contain valid data (WSTRB = 8'hFF) for all transfers other than the first and last, in transactions greater than two transfers in length.

Note: For AWSIZE = 3'b010 (32-bit) CoreAXItoAHBL expects all byte lanes to contain valid data (WSTRB = 8'h0F or WSTRB = 8'hF0) for all transfers other than the first and last, in transactions greater than two transfers in length.

Table 4 shows the resultant AHB transfers generated when the AXI master performs incrementing address AXI read

transactions.

Table 4 • **Incrementing Address AXI Read Transaction to AHB Transfer Conversion**

ARADDR [2:0]	AXI Transfer Size	Burst Length (ARLEN + 1)	Resultant AHB Transfers					
			Non-sequential			Sequential		
			8-bit	16-bit	32-bit	32-bit 4-beat	32-bit 8-beat	32-bit 16-beat
3'b000 (64-bit aligned)	64-bit	8	0	0	0	0	0	1
3'b100 (unaligned)	64-bit	8	0	0	3	1	1	0
3'b001 (unaligned)	64-bit	8	1	1	3	1	1	0
3'b000 (32-bit aligned	32-bit	16	0	0	0	0	0	1
3'b101 (unaligned)	32-bit	16	1	1	3	1	1	0
3'b011 (unaligned)	32-bit	16	1	0	3	1	1	0

3.6.4.3 Wrapping Address Bursts

The address of AXI transactions of this burst type increments by the transfer size for every transfer of the transaction until the wrap boundary is reached, at which point it returns to the lower wrap address. The wrap boundary is determined by the number of transfers in the transaction times the transfer size. To convert this transaction type to the AHB interface, the core generates a number of non-sequential and sequential AHB transfers, depending on the number of valid bytes in the AXI transaction and the current address location in relation to the wrap boundary.

Note: The core ensures that sequential AHB transfers do not cross the wrap boundary or 1 KB boundaries.

Support for wrapping AXI transactions is not instantiated by default in CoreAXItoAHBL. Instead, a generic/parameter (WRAP_SUPPORT) exists to allow the logic to be instantiated if required, at the cost of extra logic consumption and lower operating frequency.

3.6.5 AXI Write Strobe: Translation of AXI Interface → AHB Interface

The AXI write data channel contains write strobes providing AXI masters with a means to indicate byte lanes which contain valid write data. CoreAXItoAHBL poses the following limitations to the use of the write strobes by the AXI master:

- For AXI write transactions consisting of a single beat, the core permits the transfer to be both unaligned and narrow using the write strobes. For example, WSTRB = 8'h7E, 8'h06, 8'h08 (for 64-bit AXI transactions) and WSTRB = 8'h70, 8'h08 (for 32-bit AXI transactions).
- For AXI write transactions that consist of multiple transfers, the core permits the transfer to be unaligned during the first data beat (for 64-bit AXI transactions WSTRB = 8'hFE, 8'F8, 8'h80 and for 32-bit AXI transactions WSTRB = 8'h0E, 8'hC0, 8'h08), and narrow during the last data beat (for 64-bit AXI transactions WSTRB = 8'h7F, 8'h03, 8'h01 and for 32-bit AXI transactions WSTRB = 8'h30, 8'h07).
- The core permits write strobes to be unaligned for fixed address burst write transactions, where the transfer is unaligned for every transfer in the transaction.
- For 64-bit AXI transactions, the core expects the AXI master to assert all 8 write strobes (WSTRB = 8'hFF) for all other circumstances. For 32-bit AXI transactions, the core expects the AXI master to assert 4 write strobes (WSTRB = 8'h0F or WSTRB = 8'hF0) for all other circumstances.

Note: CoreAXItoAHBL does not support sparse assertion of the write strobes. Example WSTRB = 8'h55 (for 64-bit AXI transactions), WSTRB = 8'h05 (for 32-bit AXI transactions).

3.6.6 AHBL Slave Size (32-Bit)

The core supports only 32-bit AHBL slaves. When a transfer transaction consisting of a single transfer of size 64-bits is initiated by the AXI master, the transaction is split into at least two AHBL transfers of size

32-bit (transaction may result in up to six AHBL transfers of size 8-, 16- & 32-bit being generated depending on the alignment of the AXI transaction).

3.6.7 Error Response

CoreAXItoAHBL returns a slave error response to the AXI master under the following circumstances:

- AXI master attempts a write or read transaction with a transfer size other than 64-bit/ 32-bit (AWSIZE/ARSIZE = 3'd3 or AWSIZE/ARSIZE = 3'd2)
- AXI master attempts a wrapping burst transaction without the wrapping burst logic instantiated (WRAP_SUPPORT = 0 && AWBURST/ARBURST = 2)
- AXI master attempts a wrapping burst when the burst length is something other than 2, 4, 8, or 16 (AWLEN/ARLEN = 1, 3, 7, 15)
- AXI master attempts either a write or read transaction with the burst type (AWBURST/ARBURST) set to 2'b11. This burst type is defined as being 'reserved' in the AXI specification.
- Premature assertion of the WLAST signal
- Late assertion of the WLAST signal
- Error returned by the AHB slave during an AHB transfer

3.6.8 Unaligned Address Support

AXI transactions can be unaligned in two ways:

- The AXI master may choose to offset the lower n bits of the address. However, the lower n bits of the address must match the write strobes in this case, where the transfer size is 2^n. For example, for 64-bit AXI transactions: AWADDR = 0x000000009, WSTRB = 0xFE and for 32-bit AXI transaction: AWADDR = 0x0000000A, WSTRB = 0x0C.
- The AXI master can use an address aligned to the transfer size but configure the write strobes to only write to the upper byte locations. For example, for 64-bit AXI transactions: AWADDR = 0x000000000, WSTRB = 0xC0, and for 32-bit AXI transactions: AWADDR = 0x00000000, WSTRB = 0x0E.

In both the cases, the write strobes during the first transfer in the transaction need to reflect that the transaction is unaligned.

4 Interface

4.1 Configuration Parameters

There are a number of configurable options which are applied to CoreAXItoAHBL (as shown in Table 5). If a configuration other than the default is required, the configuration dialog box in the SmartDesign should be used to select appropriate values for the configurable options.

Table 5 • **CoreAXItoAHBL Configuration Options**

Name	Valid Range	Default	Description
ID_WIDTH	-	4	Sets the width of the ID field supported. The ID width should be sufficient to support the AXI master transfer ID width and the unique master ID identifier appended by the AXI interconnect when the core is instantiated in multi-master AHBL systems.
NO_BURST_TRANS	0 or 1	0	Prevents AHB-Lite burst transfers being generated when set. AHBL burst transfers are enabled by default.
WRAP_SUPPORT	0 or 1	0	Adds support for AXI wrapping burst transactions. Wrapping burst transactions are disabled by default. **Note:** This option should only be enable if required as it has a significant impact on logic resource consumption and maximum operating frequency.
ASYNC_CLOCKS	0 or 1	0	Parameter should be set if the ACLK and HCLK clock domains are asynchronous. Instantiates CDC synchronizers in the design.

4.2 I/O Signals

Signal descriptions for CoreAXItoAHBL are defined in Table 6.

Table 6 • **CoreAXItoAHBL I/O Signals**

Port Name	Width	Direction	Description
AHBL Slave Interface Ports			
HCLK	1	In	AHBL clock. All registers within the AHB Master Control block are clocked on the rising edge of HCLK.
HRESETN	1	In	AHBL Reset. Active low AHBL reset signal. Asynchronous assertion and synchronous de-assertion. This is used to reset all registers in the AHB Master Control block.
HADDR	32	Out	AHBL address – 32 bit address on the AHB-Lite interface
HWRITE	1	Out	AHBL write – When high, indicates that the current transfer is a write transfer. When low, indicates that the current transfer is a read transfer.

Table 6 • **CoreAXItoAHBL I/O Signals**

HTRANS	2	Out	AHBL transfer type – Indicates the transfer type of the current transaction: b00: IDLE b01: BUSY b10: NON-SEQUENTIAL b11: SEQUENTIAL
HSIZE	3	Out	AHBL transfer size – Indicates the size of the AHBL transfer Supported transfer sizes: b000: 8-bit (byte) transaction b001: 16-bit (half word) transaction b010: 32-bit (word) transaction
HWDATA	32	Out	AHBL write data – Write data from the AHB-Lite master to the AHB-Lite slave
HBURST	3	Out	Type of burst generated by the AHBL master Supported burst types: b000: Single burst b011: 4-beat incrementing burst b101: 8-beat incrementing burst b111: 16-beat incrementing burst
HREADYIN	1	In	AHBL ready input – Indicates that the previous bus transfer has completed.
HRESP	1	In	AHBL response status – Indicates that an error has occurred during the transfer when driven high whilst HREADY is low. HREADY must return high before the error response can be considered complete (two cycle error response). An ‘OKAY’ response can be returned in a single cycle when HRESP is low whilst HREADY is high.
HRDATA	32	In	AHBL read data – Read data from the AHBL slave to the AHBL master
AXI Master Interface Ports			
Global Signal Ports (Clocks)			
ACLK	1	In	AXI clock – All registers within the AXI Slave Control block are clocked on the rising edge of ACLK.
ARESETN	1	In	AXI reset signal – Active low reset signal. The signal is asynchronously asserted and synchronously de-asserted.
AXI Write Address Channel			
AWID	ID_WIDTH	In	Write Address ID – Details the transaction identification tag. The upper bits of this signal represent the unique master identifier appended by the interconnect, when the core is instantiated in multi-master AHB-Lite systems.
AWADDR	32	In	Write address – Gives the address of the first transfer in a write transaction The associated control signals are used to determine the addresses of the remaining transfers in the burst.
AWLEN	4	In	Burst length – Denotes the number of transfers in a transaction.

Table 6 • **CoreAXItoAHBL I/O Signals**

AWSIZE	3	In	Burst size – Indicates the size of each transfer in the transaction Supported burst sizes: 3'b010: 32-bit (word) transactions 3'b011: 64-bit (double word) transactions
AWBURST	2	In	Burst type – Signals the type of burst transfer performed. Supported AXI burst types: 2'b00: Fixed address burst 2'b01: Incrementing address burst 2'b10: Wrapping address burst 2'b11: Reserved
AWVALID	1	In	Write address valid – Indicates that valid write address and control information are available: 1: address and control available 0: address and control not available
AWREADY	1	Out	Write address ready – Indicates that the slave is ready to accept an address and associated control signals: 1: slave ready 0: slave busy
AXI Write Data Channel			
WID	ID_WIDTH	In	Write Data ID tag – The Identification tag for the write data transaction. The WID must match the AWID value of the write transaction.
WDATA	64	In	Write data bus is 64 bits wide.
WSTRB	8	In	Write strobes. – Indicates the byte lanes of the WDATA signal that contain valid write data. There is one write strobe for each 8 bits of the write data bus. WSTRB[n] corresponds to WDATA [(8 × n) + 7 :(8 × n)].
WLAST	1	In	Write last – Indicates that the current transfer is the last transfer in the write transaction.
WVALID	1	In	Write valid – Indicates that valid write data and strobes are available: 1: write data and strobes available 0: write data and strobes unavailable.
WREADY	1	Out	Write ready – Indicates that the slave will register the write data and strobes on the next ACLK rising edge, at which point the write data can be updated/removed. 1: slave ready 0: slave not ready
AXI Write Response Channel			
BREADY	1	In	Response ready – Indicates that the AXI master will register the AXI slave write response on the next ACLK rising edge, at which point the slave write response can be removed. 1: master ready 0: master not ready

Table 6 • **CoreAXItoAHBL I/O Signals**

BID	ID_WIDTH	Out	Response ID – The Identification tag for the write response The BID must match the AWID value of the write transaction to which the slave is responding.
BRESP	2	Out	Write response – Indicates the status of the write transaction. Responses provided by CoreAXItoAHBL: 00: OKAY 10: SLVERR Refer to the Error Response section of this document for details of error conditions which are reported to the AXI master by CoreAXItoAHBL.
BVALID	1	Out	Write response valid – Indicates to the AXI master that CoreAXItoAHBL is presenting valid write response. 1: write response available. 0: write response not available.
AXI Read Address Channel			
ARID	ID_WIDTH	In	Read Address ID – Details the transaction identification tag. The upper bits of this signal represent the unique master identifier appended by the interconnect, when the core is instantiated in multi-master AHB-Lite systems.
ARADDR	32	In	Read address – Gives the address of the first transfer in a read transaction The associated control signals are used to determine the addresses of the remaining transfers in the burst.
ARLEN	4	In	Burst length – Denotes the number of transfers in a transaction.
ARSIZE	3	In	Burst size – Indicates the size of each transfer in the transaction Supported burst sizes: 3’b010: 32-bit (word) transactions 3’b011: 64-bit (double word) transactions
ARBURST	2	In	Burst type – Signals the type of burst transfer performed. Supported AXI burst types: 2’b00: Fixed address burst 2’b01: Incrementing address burst 2’b10: Wrapping address burst 2’b11: Reserved
ARVALID	1	In	Read address valid – Indicates that valid read address and control information are available: 1: address and control available 0: address and control not available
ARREADY	1	Out	Read address ready – Indicates that the slave is ready to accept an address and associated control signals: 1: slave ready 0: slave busy
AXI Read Data Channel			
RREADY	1	In	Read ready – Indicates that the AXI master will register the read data on the next ACLK rising edge, at which point the read data can be updated/removed. 1: slave ready 0: slave not ready

Table 6 • CoreAXItoAHBL I/O Signals

RID	ID_WIDTH	Out	Read Data ID tag – The Identification tag for the read data transaction. The slaves generates RID, which must match the ARID value of the read transaction.
RDATA	64	Out	Read data – Read data bus is 64 bits wide
RRESP	2	Out	Read response – Indicates the status of the read transaction. Responses provided by CoreAXItoAHBL: 00: OKAY 10: SLVERR Refer to the Error Response section of this document for details of error conditions which are reported to the AXI master by CoreAXItoAHBL.
RLAST	1	Out	Read Last – Indicates that the current transfer is the last transfer in the read transaction.
RVALID	1	Out	Read Valid - Indicates to the AXI master that CoreAXItoAHBL is presenting valid read data. 1: read data available 0: read data not available

5 Tool Flow

5.1 License

CoreAXItoAHBL is license free.

5.1.1 RTL

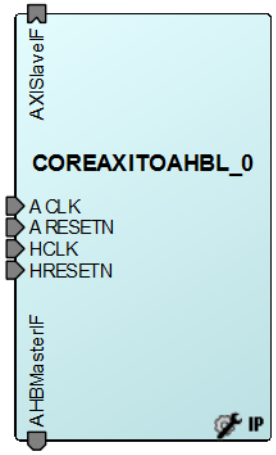
Complete register transfer level (RTL) source code is provided for the core and testbenches.

5.2 SmartDesign

CoreAXItoAHBL is pre-installed in the SmartDesign IP deployment design environment.

The core should be configured using the configuration GUI within the SmartDesign tool, as shown in Figure 4. For information on using SmartDesign to instantiate and generate cores, refer to Libero SoC online help.

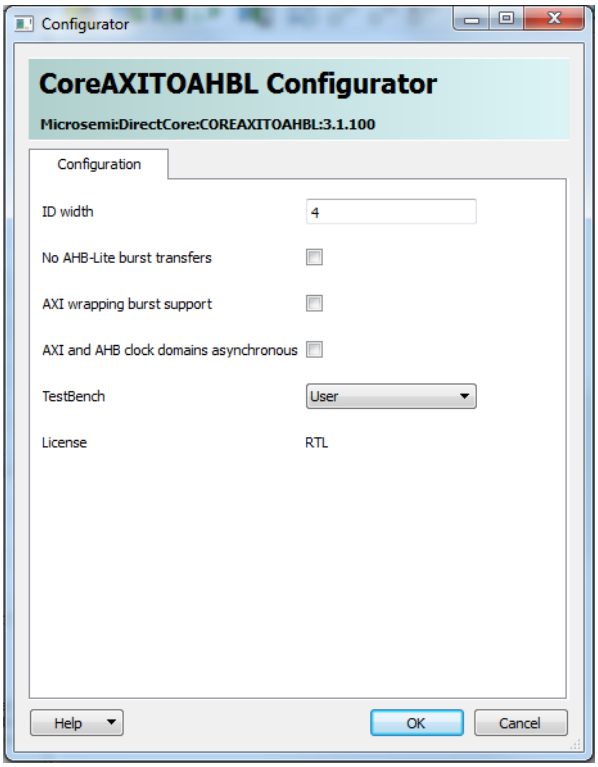
Figure 3 • **SmartDesign CoreAXItoAHBL Instance View**



5.3 Configuring CoreAXItoAHBL in SmartDesign

The core can be configured using the configuration GUI within SmartDesign. An example of the GUI for the SmartFusion2 family is shown in Figure 4.

Figure 4 • **SmartDesign CoreAXItoAHBL Configuration Dialog Box**



5.4 Simulation Flows

The User Testbench for CoreAXItoAHBL is included in all releases.

To run simulations, select the User Testbench flow within the SmartDesign CoreAXItoAHBL configuration GUI, right-click the canvas, and select **Generate Design**.

When SmartDesign generates the design files, it installs the user testbench files.

To run the user testbench, set the design root to the CoreAXItoAHBL instantiation in the Libero SoC design hierarchy pane and click **Simulation** in the **Libero SoC Design Flow** window. This invokes ModelSim® and automatically runs the simulation.

5.5 Synthesis in Libero

To run synthesis on the CoreAXItoAHBL, set the design root to the IP component instance and run the synthesis tool from the Libero design flow pane.

5.6 Place-and-Route in Libero

After the design is synthesized, run the compilation and then place-and-route the tools. CoreAXItoAHBL requires no special place-and-route settings.

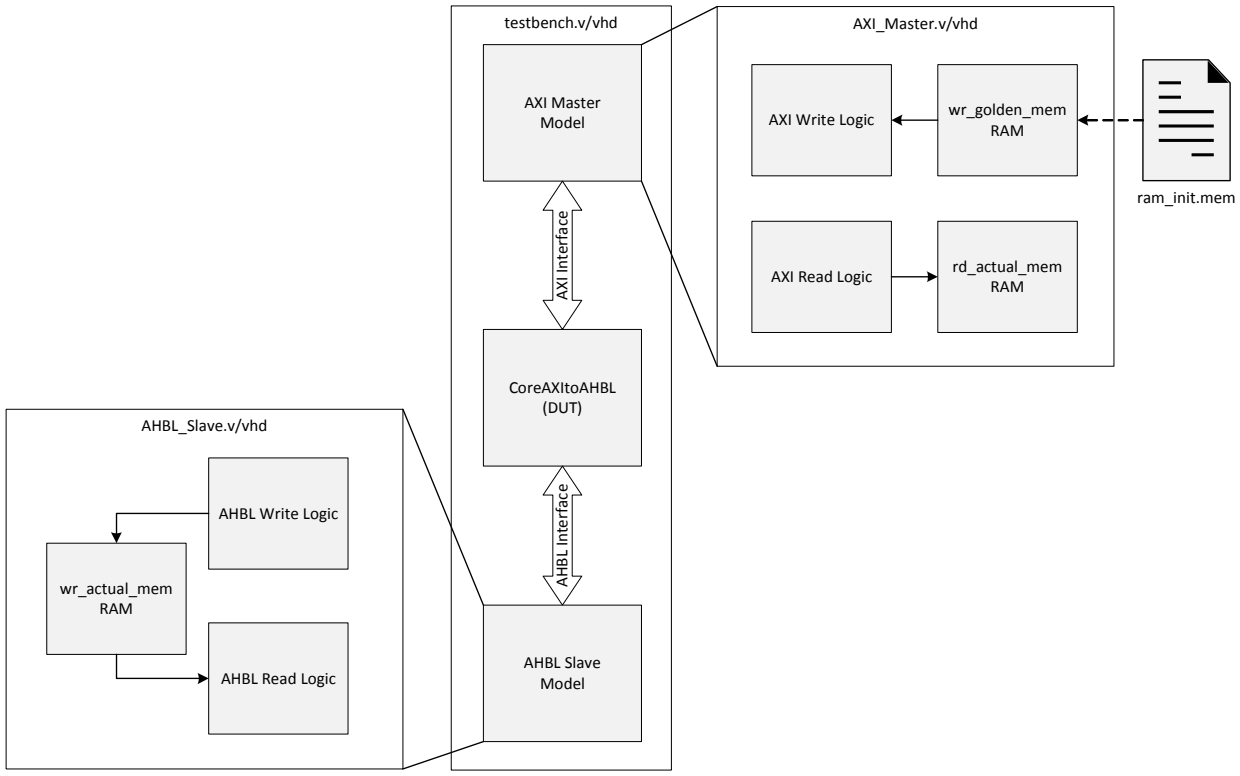
6 Testbench

This testbench integrates the CoreAXItoAHBL macro into a system and performs a basic loopback test consisting of incrementing address burst AXI transactions of varying transaction lengths.

6.1 User Testbench

An example user testbench is included with CoreAXItoAHBL.

Figure 5 • **User Testbench**



As shown in Figure 5 the user testbench instantiates CoreAXItoAHBL design under test (DUT). The CoreAXItoAHBL testbench environment consists of the following components:

- **AXI master model:** The AXI master model drives write and read AXI transactions to the DUT. The AXI master model implements a set of functions which allow AXI transactions to be generated. For write transactions, write data is taken from the wr_golden_mem RAM block, which gets initialized with the contents of the ram_init.mem file. For read transactions, read data is stored in the rd_actual_mem RAM block. A set of function calls are included in the AXI master model to perform a basic loopback test to drive the user testbench. Users can create modified calls of these tasks and replace the contents of the ram_init.mem file to simulate custom cases. An alternative .mem file and RAM size can be specified by the RAM_INIT_FILE and RAM_ADDR_WIDTH parameters respectively.
- **AHB slave model:** The AHBL slave model stores write data in the wr_actual_mem RAM block during an AHBL write transfer. Data from the corresponding address locations of the wr_actual_mem RAM block is returned during an AHBL read transfer. An alternative RAM size can be specified via the RAM_ADDR_WIDTH parameter.

7 Ordering Information

7.1 Ordering Codes

Order CoreAXItoAHBL through your local Microsemi sales representative. Use the following number convention when ordering: CoreAXItoAHBL. XX is listed in Table 7.

Table 7 • **Ordering Codes**

XX	Description
RM	RTL for RTL source — multi-use license.

8 Product Support

Microsemi SoC Products Group backs its products with various support services, including Customer Service, Customer Technical Support Center, a website, electronic mail, and worldwide sales offices. This appendix contains information about contacting Microsemi SoC Products Group and using these support services.

8.1 Customer Service

Contact Customer Service for non-technical product support, such as product pricing, product upgrades, update information, order status, and authorization.

From North America, call **800.262.1060**
From the rest of the world, call **650.318.4460**
Fax, from anywhere in the world, **408.643.6913**

8.2 Customer Technical Support Center

Microsemi SoC Products Group staffs its Customer Technical Support Center with highly skilled engineers who can help answer your hardware, software, and design questions about Microsemi SoC Products. The Customer Technical Support Center spends a great deal of time creating application notes, answers to common design cycle questions, documentation of known issues, and various FAQs. So, before you contact us, please visit our online resources. It is very likely we have already answered your questions.

8.3 Technical Support

For Microsemi SoC Products Support, visit
http://www.microsemi.com/products/fpga-soc/design-support/fpga-soc-support.

8.4 Website

You can browse a variety of technical and non-technical information on the Microsemi SoC Products Group home page, at http://www.microsemi.com/products/fpga-soc/fpga-and-soc.

8.5 Contacting the Customer Technical Support Center

Highly skilled engineers staff the Technical Support Center. The Technical Support Center can be contacted by email or through the Microsemi SoC Products Group website.

8.5.1 Email

You can communicate your technical questions to our email address and receive answers back by email, fax, or phone. Also, if you have design problems, you can email your design files to receive assistance. We constantly monitor the email account throughout the day. When sending your request to us, please be sure to include your full name, company name, and your contact information for efficient processing of your request.

The technical support email address is soc_tech@microsemi.com.

8.5.2 My Cases

Microsemi SoC Products Group customers may submit and track technical cases online by going to My Cases.

8.5.3 Outside the U.S.

Customers needing assistance outside the US time zones can either contact technical support via email (soc_tech@microsemi.com) or contact a local sales office. Visit About Us for sales office listings and corporate contacts.

8.6 ITAR Technical Support

For technical support on RH and RT FPGAs that are regulated by International Traffic in Arms Regulations (ITAR), contact us via soc_tech@microsemi.com. Alternatively, within My Cases, select **Yes** in the ITAR drop-down list. For a complete list of ITAR-regulated Microsemi FPGAs, visit the ITAR web page.