HB0397

Handbook
CoreAXItoAHBL v3.5

Microsemi
a Microchip company

Microsemi 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
Email: sales.support@microsemi.com
www.microsemi.com

About Microsemi

Microsemi, a wholly owned subsidiary of Microchip Technology Inc. (Nasdaq: MCHP), 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. Learn more at www.microsemi.com.

Contents

Figures

Tables

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 9.0

Updated changes related to CoreAXItoAHBL v3.5.

1.2 Revision 8.0

Updated changes related to CoreAXItoAHBL v3.4.

1.3 Revision 7.0

Updated changes related to CoreAXItoAHBL v3.3.

1.4 Revision 6.0

Updated changes related to CoreAXItoAHBL v3.2.

1.5 Revision 5.0

Updated changes related to CoreAXItoAHBL v3.1.

1.6 Revision 4.0

Updated changes related to CoreAXItoAHBL v3.0.

1.7 Revision 3.0

Updated changes related to CoreAXItoAHBL v2.2.

1.8 Revision 2.0

Updated changes related to CoreAXItoAHBL v2.1.

1.9 Revision 1.0

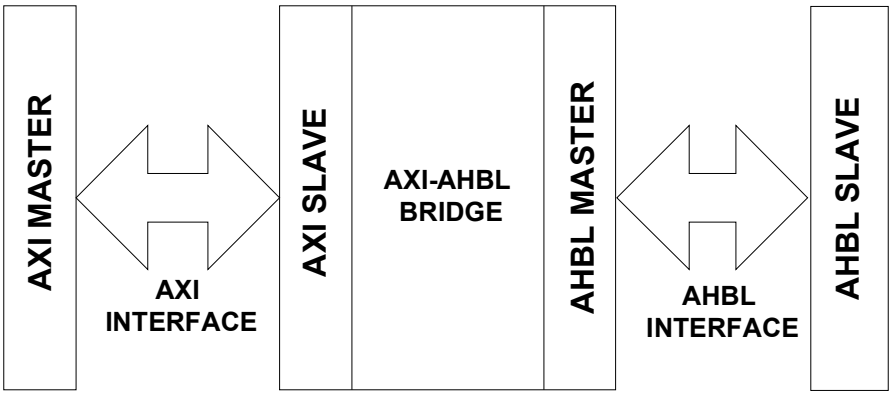
Revision 1.0 was the first publication of this document. Created for CoreAXItoAHBL v2.0.

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 Bus Lite (AHB-Lite) domain
- Makes alternate AXI write and AXI read transactions possible
- Supports AXI data bus width of 32-bits, with transfer size of 32/16/8 bit
- Supports AXI data bus width of 64-bits, with transfer size of 64/32/16/8 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 3.5.

2.4 Supported Families

CoreAXItoAHBL is a generic core and supports all the device families.

2.5 Device Utilization and Performance

Utilization and performance data is listed in Table 1, page 3 for some of the device families. 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	Parameters				Utilization				Performance	
	AXI_DWIDTH	NO_BURST_TRANS	WRAP_SUPPORT	ASYNC_CLOCKS	Sequential (DFF)	Combinational (4LUT)	Total	Percentage	ACLK Frequency (in MHz)	HCLK Frequency (in MHz)
SmartFusion (M2S050) /IGLOO2 (M2GL050)	32	0	0	0	429	1196	1625	2.88	157	111
	32	0	0	1	437	1206	1643	2.92	129	111
	32	0	1	0	434	1713	2147	3.81	162	106
	32	0	1	1	442	1747	2189	3.89	140	107
	32	1	0	0	416	812	1228	2.18	157	155
	32	1	0	1	424	787	1211	2.15	151	179
	32	1	1	0	421	1242	1663	2.95	157	131
	32	1	1	1	429	1230	1659	2.94	133	113
	64	0	0	0	725	1769	2494	4.43	131	118
	64	0	0	1	733	1770	2503	4.44	129	118
	64	0	1	0	728	2536	3264	5.79	133	112
	64	0	1	1	736	2523	3259	5.78	132	109
	64	1	0	0	712	1454	2166	3.84	135	132
	64	1	0	1	720	1464	2184	3.88	132	129
	64	1	1	0	715	1791	2506	4.45	138	109
	64	1	1	1	723	1771	2494	4.43	137	113

Table 1 • **Device Utilization and Performance *(continued)***

FAMILY	Parameters				Utilization				Performance	
	AXI_DWIDTH	NO_BURST_TRANS	WRAP_SUPPORT	ASYNC_CLOCKS	Sequential (DFF)	Combinational (4LUT)	Total	Percentage	ACLK Frequency (in MHz)	HCLK Frequency (in MHz)
RTG4 (RT4G150)	32	0	0	0	429	1167	1596	1.05	137	106
	32	0	0	1	437	1164	1601	1.05	137	101
	32	0	1	0	438	1738	2176	1.43	148	93
	32	0	1	1	442	1736	2178	1.43	143	88
	32	1	0	0	416	777	1193	0.79	130	151
	32	1	0	1	424	807	1231	0.81	140	138
	32	1	1	0	421	1228	1649	1.09	126	119
	32	1	1	1	429	1239	1668	1.10	147	113
	64	0	0	0	725	1765	2490	1.64	115	106
	64	0	0	1	733	1772	2505	1.65	115	105
	64	0	1	0	728	2536	3264	2.15	109	99
	64	0	1	1	736	2603	3339	2.20	104	99
	64	1	0	0	712	1358	2070	1.36	114	122
	64	1	0	1	720	1362	2082	1.37	114	125
	64	1	1	0	715	1845	2560	1.69	106	96
	64	1	1	1	723	1837	2560	1.69	106	102

Table 1 • **Device Utilization and Performance *(continued)***

FAMILY	Parameters				Utilization				Performance	
	AXI_DWIDTH	NO_BURST_TRANS	WRAP_SUPPORT	ASYNC_CLOCKS	Sequential (DFF)	Combinational (4LUT)	Total	Percentage	ACLK Frequency (in MHz)	HCLK Frequency (in MHz)
PolarFire (MPF300T)	32	0	0	0	357	1020	1377	0.46	181	140
	32	0	0	1	365	1002	1367	0.46	157	147
	32	0	1	0	362	1647	2009	0.67	197	131
	32	0	1	1	370	1633	2003	0.67	213	136
	32	1	0	0	344	714	1058	0.35	195	233
	32	1	0	1	352	681	1033	0.34	192	228
	32	1	1	0	349	1154	1503	0.50	201	159
	32	1	1	1	357	1122	1479	0.49	199	157
	64	0	0	0	581	1640	2221	0.74	147	151
	64	0	0	1	589	1584	2173	0.73	142	145
	64	0	1	0	584	2418	3002	1.00	144	131
	64	0	1	1	592	2415	3007	1.00	144	121
	64	1	0	0	568	1237	1805	0.60	141	188
	64	1	0	1	576	1244	1820	0.61	149	174
	64	1	1	0	571	1637	2208	0.74	146	143
	64	1	1	1	579	1633	2212	0.74	147	143

Note:

1. The data in the Table 1, page 3 is achieved using Verilog RTL, typical synthesis and layout settings. Frequency (in MHz) was set to 100 and speed grade was -1. The parameters ID_WIDTH is set to 4, AXI_SEL_MM_S is set to 0 and EXPOSE_WID is set to 0.
2. In case of SmartFusion2, IGLOO2, and RTG4 device families, when AXI_DWIDTH is 64, the core uses eight, 64 * 18 RAM blocks and when AXI_DWIDTH is 32, the core uses four, 64 * 18 RAM blocks.
3. In case of PolarFire device family, when AXI_DWIDTH is 64, the core uses twelve, 64 * 12 RAM blocks and when AXI_DWIDTH is 32, the core uses six, 64 * 12 RAM blocks.

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.

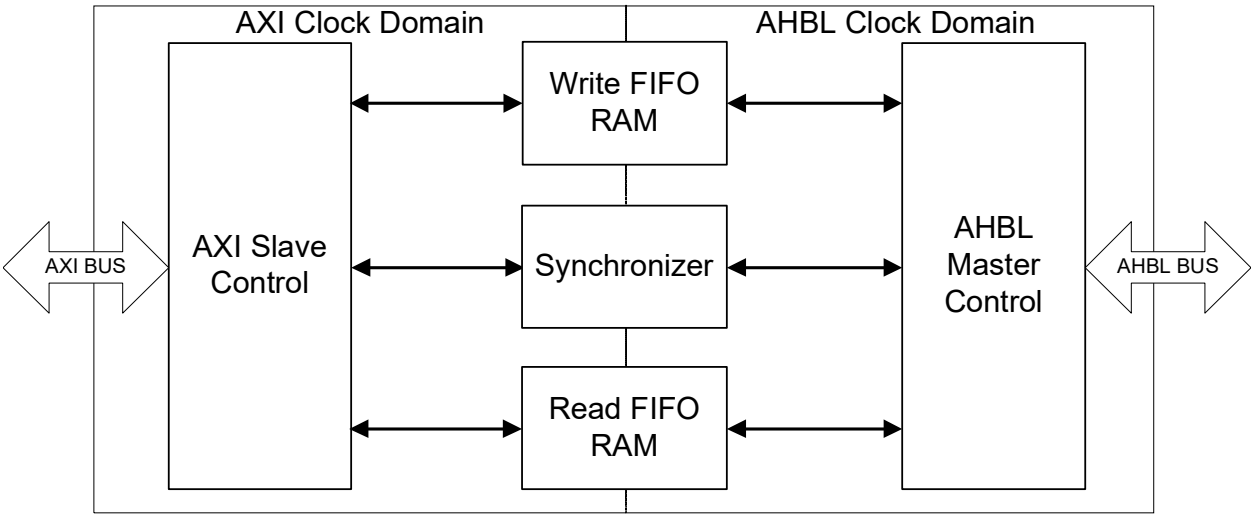
The ACLK and HCLK clocks are configurable to be synchronous or asynchronous via parameter/generic. The core implements the clock-domain-crossing (CDC) logic, where the AHB clock and AXI clock are asynchronous to each other.

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, page 6.

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, AXI_DWIDTH bits 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, AXI_DWIDTH bits 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

When AXI_DWIDTH is set to 64, the core supports 64-bit transfer size (ARSIZE and AWSIZE = 3’b011), 32-bit transfer size (ARSIZE and AWSIZE = 3’b010), 16-bit transfer size (ARSIZE and AWSIZE = 3'b001), and 8-bit transfer (ARSIZE and AWSIZE = 3'b000). A slave error response will be returned to the AXI master, if a transfer of size other than 64/32/16/8 bit is attempted.

When AXI_DWIDTH is set to 32, the core supports 32-bit transfer size (ARSIZE and AWSIZE = 3’b010), 16-bit transfer size (ARSIZE and AWSIZE = 3'b001) and 8-bit transfer (ARSIZE and AWSIZE = 3'b000). A slave error response will be returned to the AXI master, if a transfer of size other than 32/16/8 bit is attempted.

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 transfer size) and WSTRB = 8’h05 (for 32-bit transfer size).

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-beat, -8-beat, and 16-beat incrementing burst AHB transfers. If an unaligned AXI transaction is received, which is not aligned to the transfer size, the AHB Master Control block will perform a number of non-sequential transfers to move to a address aligned to the transfer size 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 the AXI transfer size boundary, 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, page 8 and Table 3, page 9 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 (AXI_DWIDTH = 64)**

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

Table 2 • **Fixed Address AXI Transaction to AHB Transfer Conversion (AXI_DWIDTH = 64) *(continued)***

3'b000 (16-bit aligned)	16-bit	8'h03	2	0	2	-
3'b110 (16-bit aligned)	16-bit	8'hC0	2	0	2	-
3'b001 (unaligned)	16-bit	8'h02	2	2	0	-
3'b111 (unaligned)	16-bit	8'h80	2	2	0	-
3'b000	8-bit	8'h01	2	2	-	-
3'b011	8-bit	8'h08	2	2	-	-
3'b110	8-bit	8'h40	2	2	-	-

Note: CoreAXItoAHBL does not permit the use of narrow transfers for fixed address AXI write transactions. For example, WSTRB = 8'h7F (for 64-bit transfers), WSTRB = 8'h07 (for 32-bit transfers) and WSTRB = 8'h01 (for 16-bit transfers).

Table 3 • **Fixed Address AXI Transaction to AHB Transfer Conversion (AXI_DWIDTH = 32)**

AWADDR/ARADDR[1:0]	AXI Transfer Size	WSTRB (in case of write transactions)	Burst Length (AxLEN + 1)	Resultant AHB Transfers		
				Non-sequential		
				8-bit	16-bit	32-bit
2'b00 (aligned)	32-bit	4'hF	2	0	0	2
2'b01 (unaligned)	32-bit	4'hE	2	2	2	0
2'b11 (unaligned)	32-bit	4'h8	2	2	0	0
2'b10 (unaligned)	32-bit	4'hC	2	0	2	0
2'b00 (aligned)	16-bit	4'h3	2	0	2	-
2'b10 (aligned)	16-bit	4'hC	2	0	2	-
2'b01 (unaligned)	16-bit	4'h2	2	2	0	-
2'b11 (unaligned)	16-bit	4'h8	2	2	0	-
2'b00	8-bit	4'h1	2	2	-	-
2'b11	8-bit	4'h8	2	2	-	-
2'b10	8-bit	4'h4	2	2	-	-
2'b01	8-bit	4'h2	2	2	-	-

Note: CoreAXItoAHBL does not permits the use of narrow transfers for fixed address AXI write transactions. For example, WSTRB = 4'h7 (for 32-bit transfers) and WSTRB = 4'h1 (for 16-bit transfers).

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 address aligned to the transfer size, before generating a combination of 4-beat, 8-beat, and 16-beat, 32/16/8-bit AHB transfers based on the AXI transfer size. 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 4, page 10, Table 5, page 12, Table 6, page 14 and Table 7, page 15 shows the resultant AHB transfers generated when the AXI master performs incrementing address AXI write transactions.

Table 4 • **Incrementing Address AXI Write Transaction to AHB Transfer Conversion (AXI_DWIDTH = 64)**

AWADDR [2:0] (Offset optional)	AXI Transfer Size	WSTRB First Beat	WSTRB 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				
					8-bit single	16-bit single	32-bit single	4-beat	8-beat	16-beat	4-beat	8-beat	16-beat	4-beat	8-beat	16-beat		
3'b000	64-bit	8'hFF (aligned)	8'hFF	16	0	0	0	-	-	-	-	-	-	0	0	2		
3'b000	64-bit	8'hFF (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'b000	64-bit	8'hFF (aligned)	8'h03 (narrow)	8	0	1	2	-	-	-	-	-	-	1	1	0		
3'b101	64-bit	8'hE0 (unaligned)	8'h07 (narrow)	8	2	2	0	--	-	-	-	-	-	1	1	0		
3'b010	64-bit	8'h1C (unaligned & narrow)	N/A	1	1	1	0	-	-	-	-	-	-	0	0	0		
3'b000	32-bit	8'h0F (aligned)	8'hF0	16	0	0	0	-	-	-	-	-	-	0	0	1		
3'b100	32-bit	8'hF0 (aligned)	8'h0F	8	0	0	0	-	-	-	-	-	-	0	1	0		
3'b110	32-bit	8'hC0 (unaligned)	8'h0F	8	0	1	3	-	-	-	-	-	-	1	0	0		
3'b000	32-bit	8'h0F (aligned)	8'h70 (narrow)	8	1	1	3	-	-	-	-	-	-	1	0	0		
3'b011	32-bit	8'h08 (unaligned)	8'h30 (narrow)	8	1	1	2	-	-	-	-	-	-	1	0	0		
3'b101	32-bit	8'h60 (unaligned & narrow)	N/A	1	2	0	0	-	-	-	-	-	-	0	0	0		
3'b000	16-bit	8'h03 (aligned)	8'hC0	16	0	0	-	-	-	-	0	0	1	-	-	-		

Table 4 • **Incrementing Address AXI Write Transaction to AHB Transfer Conversion (AXI_DWIDTH = 64)**

AWADDR [2:0] (Offset optional)	AXI Transfer Size	WSTRB First Beat	WSTRB Last Beat	Burst Length (AWLEN + 1)	Resultant AHB Transfers (Assuming transaction doesn't cross a 1 KB boundary and NO_BURST_TRANS = 0)											
3'b110	16-bit	8'hC0 (aligned)	8'h30	8	0	0	-	-	-	-	0	1	0	-	-	-
3'b101	16-bit	8'h20 (unaligned)	8'h0C	8	1	3	-	-	-	-	1	0	0	-	-	-
3'b100	16-bit	8'h30 (aligned)	8'h04 (narrow)	8	1	3	-	-	-	-	1	0	0	-	-	-
3'b111	16-bit	8'h80 (unaligned)	8'h10 (narrow)	8	2	2	-	-	-	-	1	0	0	-	-	-
3'b000	8-bit	8'h01	8'h80	16	0	-	-	0	0	1	-	-	-	-	-	-
3'b101	8-bit	8'h10	8'h04	8	0	-	-	0	1	0	-	-	-	-	-	-
3'b010	8-bit	8'h04	8'h80	6	2	-	-	1	0	0	-	-	-	-	-	-
3'b110	8-bit	8'h40	8'h01	3	3	-	-	0	0	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.

Note: For AWSIZE = 3'b001 (16-bit), CoreAXItoAHBL expects all byte lanes to contain valid data (WSTRB = 8'h03 or WSTRB = 8'h0C or WSTRB = 8'h30 or WSTRB = 8'hC0) for all transfers other than the first and last, in transactions greater than two transfers in length.

Table 5 • **Incrementing Address AXI Write Transaction to AHB Transfer Conversion (AXI_DWIDTH = 32)**

AWADDR [1:0] (Offset optional)	AXI Transfer Size	WSTRB First Beat	WSTRB 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				
					8-bit single	16-bit single	32-bit single	4-beat	8-beat	16-beat	4-beat	8-beat	16-beat	4-beat	8-beat	16-beat		
2'b00	32-bit	4'hF (aligned)	4'hF	16	0	0	0	-	-	-	-	-	-	0	0	1		
2'b00	32-bit	4'hF (aligned)	4'hF	8	0	0	0	-	-	-	-	-	-	0	1	0		
2'b11	32-bit	4'h8 (Unaligned)	4'hF	8	1	0	3	-	-	-	-	-	-	1	0	0		
2'b00	32-bit	4'hF (aligned)	4'h3 (narrow)	8	0	1	3	-	-	-	-	-	-	1	0	0		
2'b01	32-bit	4'hE (Unaligned)	4'h1 (narrow	8	2	1	2	-	-	-	-	-	-	1	0	0		
2'b10	32-bit	4'h4 (Unaligned and narrow)	N/A	1	1	0	0	-	-	-	-	-	-	0	0	0		
2'b00	16-bit	4'h3 (aligned)	4'hC	16	0	0	-	-	-	-	0	0	1	-	-	-		
2'b10	16-bit	4'hC (aligned)	4'h3	8	0	0	-	-	-	-	0	1	0	-	-	-		
2'b01	16-bit	4'h2 (Unaligned)	4'hC	8	1	3	-	-	-	-	1	0	0	-	-	-		
2'b10	16-bit	4'hC (aligned)	4'h1 (narrow)	8	1	3	-	-	-	-	1	0	0	-	-	-		
2'b11	16-bit	4'h8 (unaligned)	4'h1 (narrow)	8	2	2	-	-	-	-	1	0	0	-	-	-		
2'b00	8-bit	4'h1	4'h8	16	0	-	-	0	0	1	-	-	-	-	-	-		
2'b10	8-bit	4'h4	4'h4	13	1	-	-	1	1	0	-	-	-	-	-	-		
2'b11	8-bit	4'h8	4'h4	8	0	-	-	0	1	0	-	-	-	-	-	-		
2'b01	8-bit	4'h2	4'h4	2	2	-	-	0	0	0	-	-	-	-	-	-		

Note: For AWSIZE = 3'b010 (32-bit), CoreAXItoAHBL expects all byte lanes to contain valid data (WSTRB = 4'hF) for all transfers other than the first and last, in transactions greater than two transfers in length.

Note: For AWSIZE = 3'b001 (16-bit), CoreAXItoAHBL expects all byte lanes to contain valid data (WSTRB = 4'h3 or WSTRB = 4'hC) for all transfers other than the first and last, in transactions greater than two transfers in length.

Table 6 • **Incrementing Address AXI Read Transaction to AHB Transfer Conversion (AXI_DWIDTH = 64)**

ARADDR [2:0]	AXI Transfer Size	Burst Length (ARLEN + 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			
			8-bit single	16-bit single	32-bit single	4-beat	8-beat	16-beat	4-beat	8-beat	16-beat	4-beat	8-beat	16-beat
3'b000 (aligned)	64-bit	16	0	0	0	-	-	-	-	-	-	0	0	2
3'b000 (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'b110 (unaligned)	64-bit	8	0	1	2	-	-	-	-	-	-	1	1	0
3'b000 (aligned)	32-bit	16	0	0	0	-	-	-	-	-	-	0	0	1
3'b100 (aligned)	32-bit	8	0	0	0	-	-	-	-	-	-	0	1	0
3'b011 (unaligned)	32-bit	8	1	0	3	-	-	-	-	-	-	1	0	0
3'b110 (unaligned)	32-bit	8	0	1	3	-	-	-	-	-	-	1	0	0
3'b000 (aligned)	16-bit	16	0	0	-	-	-	-	0	0	1	-	-	-
3'b010 (aligned)	16-bit	8	0	0	-	-	-	-	0	1	0	-	-	-
3'b001 (unaligned)	16-bit	8	1	3	-	-	-	-	1	0	0	-	-	-
3'b111 (unaligned)	16-bit	8	1	3	-	-	-	-	1	0	0	-	-	-
3'b000	8-bit	16	0	-	-	0	0	1	-	-	-	-	-	-
3'b010	8-bit	14	2	-	-	1	1	0	-	-	-	-	-	-
3'b101	8-bit	9	1	-	-	0	1	0	-	-	-	-	-	-
3'b111	8-bit	2	2	-	-	0	0	0	-	-	-	-	-	-

Table 7 • **Incrementing Address AXI Read Transaction to AHB Transfer Conversion (AXI_DWIDTH = 32)**

ARADDR [1:0]	AXI Transfer Size	Burst Length (ARLEN + 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			
			8-bit single	16-bit single	32-bit single	4-beat	8-beat	16-beat	4-beat	8-beat	16-beat	4-beat	8-beat	16-beat
2'b00 (aligned)	32-bit	16	0	0	0	-	-	-	-	-	-	0	0	1
2'b00 (aligned)	32-bit	8	0	0	0	-	-	-	-	-	-	0	1	0
2'b11 (unaligned)	32-bit	8	1	0	3	-	-	-	-	-	-	1	0	0
2'b01 (unaligned)	32-bit	8	1	1	3	-	-	-	-	-	-	1	0	0
2'b00 (aligned)	16-bit	16	0	0	-	-	-	-	0	0	1	-	-	-
2'b10 (aligned)	16-bit	8	0	0	-	-	-	-	0	1	0	-	-	-
2'b01 (unaligned)	16-bit	8	1	3	-	-	-	-	1	0	0	-	-	-
2'b11 (unaligned)	16-bit	8	1	3	-	-	-	-	1	0	0	-	-	-
2'b00	8-bit	16	0	-	-	0	0	1	-	-	-	-	-	-
2'b01	8-bit	11	3	-	-	0	1	0	-	-	-	-	-	-
2'b10	8-bit	5	1	-	-	1	0	0	-	-	-	-	-	-
2'b11	8-bit	1	1	-	-	0	0	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.

When AXI_DWIDTH is set to 64:

- 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), WSTRB = 8’h70, 8’h08 (for 32-bit AXI transactions) and WSTRB = 8'h04, 8'h10 (for 16-bit AXI transfers).
- 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'hF8, 8’h80, for 32-bit AXI transactions WSTRB = 8’h0E, 8’hC0, 8’h08 and for 16-bit AXI transactions WSTRB = 8’h02, 8’h80, 8’h08, 8'h20), and narrow during the last data beat (for 64-bit AXI transactions WSTRB = 8’h7F, 8’h03, 8’h01, for 32-bit AXI transactions WSTRB = 8’h30, 8’h07 and for 16-bit AXI transactions WSTRB = 8’h01, 8’h40).
- 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 example, WSTRB = 8’hFE, 8’hF0, 8’hC0 (for 64-bit AXI transactions), WSTRB = 8’hE0, 8’h0C (for 32-bit AXI transactions), and WSTRB = 8'h02, 8'h80 (for 16-bit AXI transfers).
- For all other circumstances (other than first and last transfer of a beat) the core expects the AXI master to assert all 8 write strobes (WSTRB = 8'hFF) for 64-bit AXI transfers; 4 write strobes (WSTRB = 8'h0F or WSTRB = 8'hF0) for 32-bit AXI transfers; 2 write strobes (WSTRB = 8'h03 or WSTRB = 8'h0C or WSTRB = 8'h30 or WSTRB = 8'hC0) for 16-bit AXI transfers.

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).

When AXI_DWIDTH is set to 32:

- 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 = 4’h4, 4’h6 (for 32-bit AXI transactions) and WSTRB = 4'h4, 4'h1 (for 16-bit AXI transfers).
- For AXI write transactions that consist of multiple transfers, the core permits the transfer to be unaligned during the first data beat (example: for 32-bit AXI transactions WSTRB = 4’hE, 4’hC, 4’h8 and for 16-bit AXI transactions WSTRB = 4’h2, 4’h8), and narrow during the last data beat (example: for 32-bit AXI transactions WSTRB = 4’h3, 4’h7 and for 16-bit AXI transactions WSTRB = 4’h1, 4’h4).
- 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 example, WSTRB = 4’hE, 4’hC (for 32-bit AXI transactions) and WSTRB = 4'h2, 4'h8 (for 16-bit AXI transfers).
- For all other circumstances (other than first and last transfer of a beat) the core expects the AXI master to assert all 4 write strobes (WSTRB = 4'hF) for 32-bit AXI transfers; 2 write strobes (WSTRB = 4'h3 or WSTRB = 4'hC) for 16-bit AXI transfers.

Note: CoreAXItoAHBL does not support sparse assertion of the write strobes. Example WSTRB = 4’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-bit, 16-bit,and32-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:

- When AXI_DWIDTH parameter is configured as 64 and AXI master attempts a write or read transaction with a transfer size other than 64/32/16/8 bit (AWSIZE/ARSIZE = 3'b011, AWSIZE/ARSIZE = 3'b010, AWSIZE/ARSIZE = 3'b001, or AWSIZE/ARSIZE = 3'b000).
- When AXI_DWIDTH parameter is configured as 32 and AXI master attempts a write or read transaction with a transfer size other than 32/16/8 bit (AWSIZE/ARSIZE = 3'b010, AWSIZE/ARSIZE = 3'b001, or AWSIZE/ARSIZE = 3'b000).
- AXI master attempts a wrapping burst transaction without the wrapping burst logic instantiated (WRAP_SUPPORT = 0 && AWBURST/ARBURST = 2'b10)
- AXI master attempts a wrapping burst when the burst length is something other than 2, 4, 8, or 16 (AWLEN/ARLEN = 4'b0001, 4'b0011, 4'b0111, 4'b1111)
- 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 = 0x00000009, WSTRB = 0xFE; for 32-bit AXI transaction: AWADDR = 0x0000000A, WSTRB = 0x0C; for 16-bit AXI transactions: AWADDR = 0x00000003, WSTRB = 0x08.
- 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 = 0x00000000, WSTRB = 0xC0; for 32-bit AXI transactions: AWADDR = 0x00000000, WSTRB = 0x0E; for 16-bit AXI transactions: AWADDR = 0x00000000, WSTRB = 0x02.

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 8, page 18). 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 8 • **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.
AXI_DWIDTH	32 or 64	64	Sets the width of AXI read data, AXI write data and AXI write-strobe signals. The width of AXI write strobe signal is calculated using AXI_DWIDTH parameter. AXI WRITE STROBE WIDTH = AXI_DWIDTH/8
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.
AXI_SEL_MM_S	0 or 1	0	Selects between the AXI Mirror Master BIF or AXI Slave BIF for the AXI interface when core is used in the Libero Smart Design. 0: AXI Slave BIF 1: AXI Mirror Master BIF
EXPOSE_WID	0 or 1	0	This parameter is valid only when the parameter AXI_SEL_MM_S is 0 (AXI Slave BIF is selected). 0: WID signal is part of AXI BIF 1: WID signal is exposed out of the AXI BIF.

4.2 I/O Signals

Signal descriptions for CoreAXItoAHBL are defined in Table 9, page 19.

Table 9 • **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.
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

Table 9 • **CoreAXItoAHBL I/O Signals *(continued)***

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.
AWSIZE	3	In	Burst size – Indicates the size of each transfer in the transaction. Supported burst sizes: 3’b000: 8-bit (byte) transactions 3’b001: 16-bit (half-word) transactions 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

Table 9 • **CoreAXItoAHBL I/O Signals *(continued)***

AXI Write Data Channel			
WID	ID_WIDTH	In	Write Data ID tag – The Identification tag for the write data transaction. This signal is generated and is exposed outside the AXI Slave BIF when the parameters EXPOSE_WID = 1 and AXI_SEL_MM_S = 0. This signal is generated only for this configuration.
WID_BIF	ID_WIDTH	In	Write Data ID tag – The Identification tag for the write data transaction. This signal is generated as a part of the AXI Slave BIF when the parameters EXPOSE_WID = 0 and AXI_SEL_MM_S = 0. This signal is generated and as a part of the AXI Mirror Master BIF when AXI_SEL_MM_S = 1. This signal is generated only for these two configurations.
WDATA	AXI_DWIDTH	In	Write data bus is AXI_DWIDTH bits wide.
WSTRB	AXI_DWIDTH/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
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.

Table 9 • **CoreAXItoAHBL I/O Signals *(continued)***

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’b000: 8-bit (byte) transactions 3’b001: 16-bit (half-word) transactions 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 9 • **CoreAXItoAHBL I/O Signals *(continued)***

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	AXI_DWIDTH	Out	Read data – Read data bus is AXI_DWIDTH 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

No license is required to use this core.

5.1.1 RTL

Complete RTL source code is provided for the core and testbench.

5.2 SmartDesign

CoreAXItoAHBL is pre-installed in the Libero SmartDesign IP deployment design environment or downloaded from the online repository. Figure 3, page 24 shows an example instantiated.

For information on using SmartDesign to instantiate and generate cores, refer to Libero User Guide.

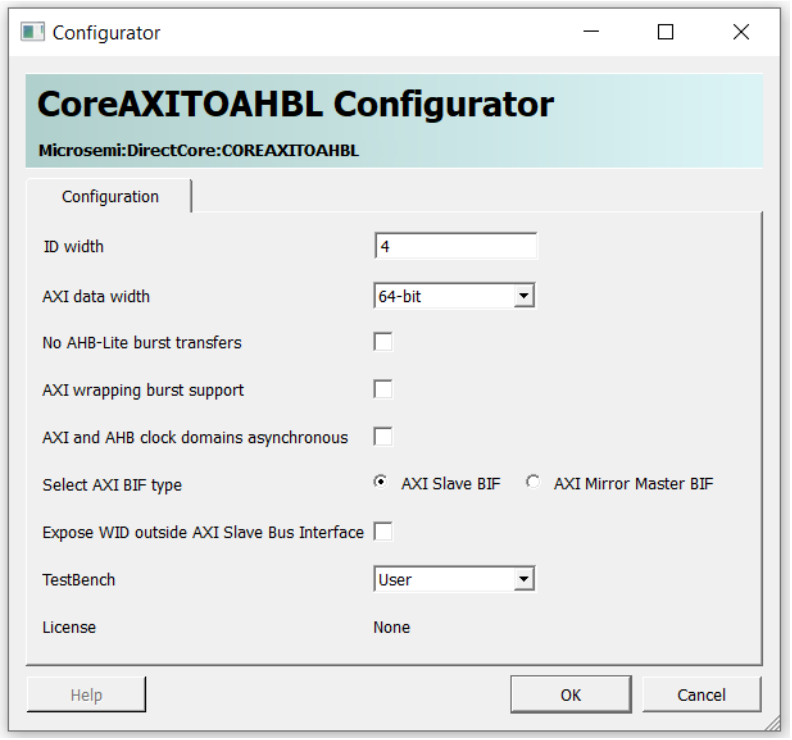
Note: CoreAXItoAHBL is compatible with Libero System-on-Chip (SoC) and Libero System-on-Chip (SoC) PolarFire. Unless specified otherwise, this document uses the name Libero to identify Libero SoC and Libero SoC PolarFire.

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 is as shown in Figure 4, page 25.

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 design hierarchy pane and click **Simulation** in the **Libero 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. This will invoke Synplify Pro and automatically runs the synthesis.

5.6 Place-and-Route in Libero

After design is synthesized, click **Place and Route** in the Libero Design Flow pane to run place and route on the CoreAXItoAHBL. No special place and route settings are required.

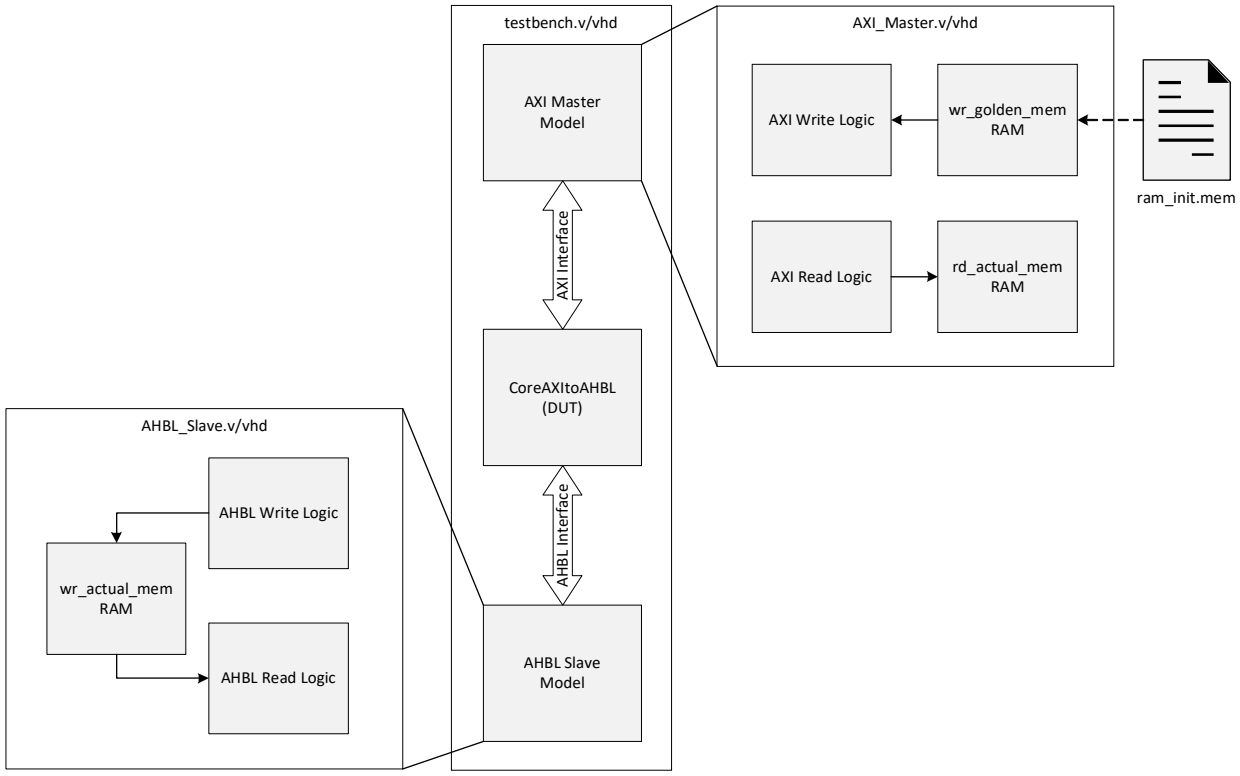
6 Testbench

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, page 26 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 through the RAM_ADDR_WIDTH parameter.