Connecting User Logic to AXI Interfaces of High-Performance Communication Blocks in the SmartFusion2 Devices - Libero SoC v11.7

Table of Contents

Purpose . . . 1
Introduction . . . 1
References . . . 2
Design Requirements . . . 2
AXI Protocol . . . 2
AXI - Write Transaction . . . 4
AXI - Read Transaction . . . 6
AXI Interface in the SmartFusion2 Device . . . 7
Implementing the AXI Master Interface on the User Logic . . . 10
Implementing the AXI Slave Interface on the User Logic . . . 13
Conclusion . . . 17
Appendix A: Adding BIF to AXI Master and Slave Blocks . . . 18
Appendix B: AXI Interface Signals . . . 22
Appendix C: Design Files . . . 24
List of Changes . . . 25

Purpose

This application note describes the user logic connection with advanced extensible interface (AXI) master and slave interfaces of high-performance communication blocks in the SmartFusion®2 system-on-chip (SoC) field programmable gate array (FPGA) device.

Introduction

This application note provides information on how to connect the user logic to AXI master and slave interfaces of high-performance communication blocks in the SmartFusion2 device.

The SmartFusion2 device integrates:

- Inherently reliable flash-based FPGA fabric
- 166 MHz ARM® Cortex®-M3 processor
- Advanced security processing accelerators
- DSP blocks
- Static random access memory (SRAM)
- Embedded non-volatile memory (eNVM)
- Industry-required high-performance communication blocks on a single-chip

© 2016 Microsemi Corporation

The high-performance communication blocks include high-speed serial interface (SERDESIF) block and high-speed double-data rate (DDR) blocks, microcontroller subsystem double-data rate (MDDR), and fabric double-data rate (FDDR). The FPGA fabric communicates with the SERDESIF, MDDR, and FDDR blocks through the AXI or AMBA® high-performance bus AHB interface. This application note describes how to create a simple AXI master and slave interfaces on user logic to communicate with the SERDESIF, MDDR, and FDDR blocks. Refer to the *AC333: Connecting User Logic to the SmartFusion Microcontroller Subsystem Application Note* for more information on creating a custom AHB interface on user logic.

References

The following documents are referred in this document. The references complement and help in understanding the relevant Microsemi SmartFusion2 SoC FPGA device flows and features that are demonstrated in this document.

- *UG0331: SmartFusion2 Microcontroller Subsystem User Guide*
- *UG0447: IGLOO2 and SmartFusion2 High Speed Serial Interfaces User Guide*
- *UG0446: SmartFusion2 and IGLOO2 FPGA High Speed DDR Interfaces User Guide*

Design Requirements

Table 1 lists the design requirements.

Table 1 • **Design Requirements**

Hardware Requirements	**Signal**
Desktop or Laptop	Any 64-bit Windows
Software Requirements	
Libero® System-on-Chip (SoC)	v11.7

AXI Protocol

This section provides a simple overview of AXI protocol. The AMBA AXI protocol is targeted at high-performance and high-frequency system designs, and it includes a number of features that make it suitable for a high-speed sub-micron interconnections.

The following are the AXI protocol features:

- Separate address
- Control and data phases
- Supports unaligned data transfers using byte strobes
- Burst-based transactions with only start address issued
- Separate read and write data channels
- Issues multiple outstanding addresses
- Out-of-order transaction completion
- Easy addition of register stages to provide timing closure

The AXI protocol is burst based. Every transaction has an address and control information on the address channel that describes the nature of the data to be transferred. The AXI protocol specifies the following five independent channels:

- Write address channel
- Write data channel
- Write response channel
- Read address channel
- Read data channel

Figure 1 shows how a write transaction uses the write address, write data, and write response channels.

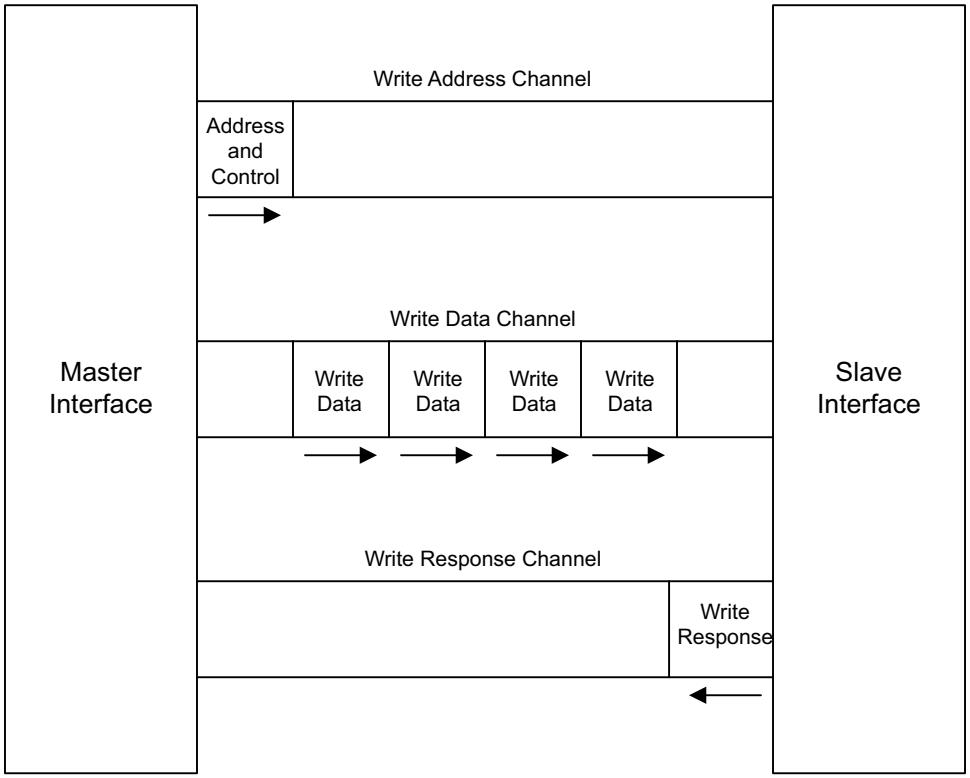

***Figure 1* • AXI Write Flow**

Figure 2 shows how a read transaction uses the read address and read data channels.

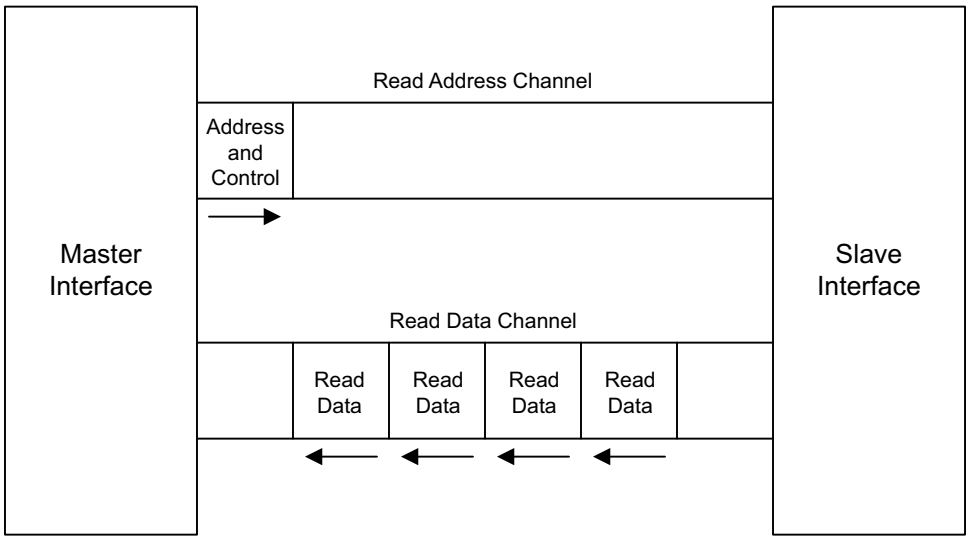

***Figure 2* • AXI Read Flow**

Each of the five independent channels consists a set of information signals and uses a two-way VALID and READY handshake mechanism. The source displays the VALID signal on the channel whenever the valid data or control information is available on the channel. The destination displays the READY signal to show when it can accept the data. Both the read data and write data channels display the LAST signal when it transfers the final data item. Refer to "Appendix B: AXI Interface Signals" on page 22 for AXI interface signals and their description.

AXI - Write Transaction

The AXI master sends the write address using the write address channel and then sends the write data using the write data channel. Finally, the slave sends the response using the write response channel. The following sub-sections describe the write transaction mechanism in the AXI protocol:

- Write Address Channel Handshake Mechanism
- Write Data Channel Handshake Mechanism
- Write Response Channel Handshake Mechanism

Write Address Channel Handshake Mechanism

The AXI master asserts the AWVALID signal (Time T1 on Figure 3 on page 5) only when it drives the valid address and control. The signals must remain asserted until the AXI slave accepts the address and control the information and asserts the associated AWREADY signal (Time T2 on Figure 3 on page 5).

Write Data Channel Handshake Mechanism

During a write transaction, the AXI master asserts the WVALID signal (Time T3 on Figure 3 on page 5) only when it drives the valid write data. The WVALID signal must remain asserted until the AXI slave accepts the write data and asserts the WREADY signal (Time T4 on Figure 3 on page 5).

Write Response Channel Handshake Mechanism

The AXI slave asserts the BVALID signal (Time T10 on Figure 3 on page 5) only when it drives the valid write response, BRESP. The BVALID signal must remain asserted until the master accepts the write response and asserts the BREADY signal.

Note: The master can assert the BREADY signal before the slave asserts the BVALID signal to complete the response transfer in one cycle, refer to Figure 3 on page 5.

Figure 3 shows the AXI write transaction timing diagram with burst length = 4.

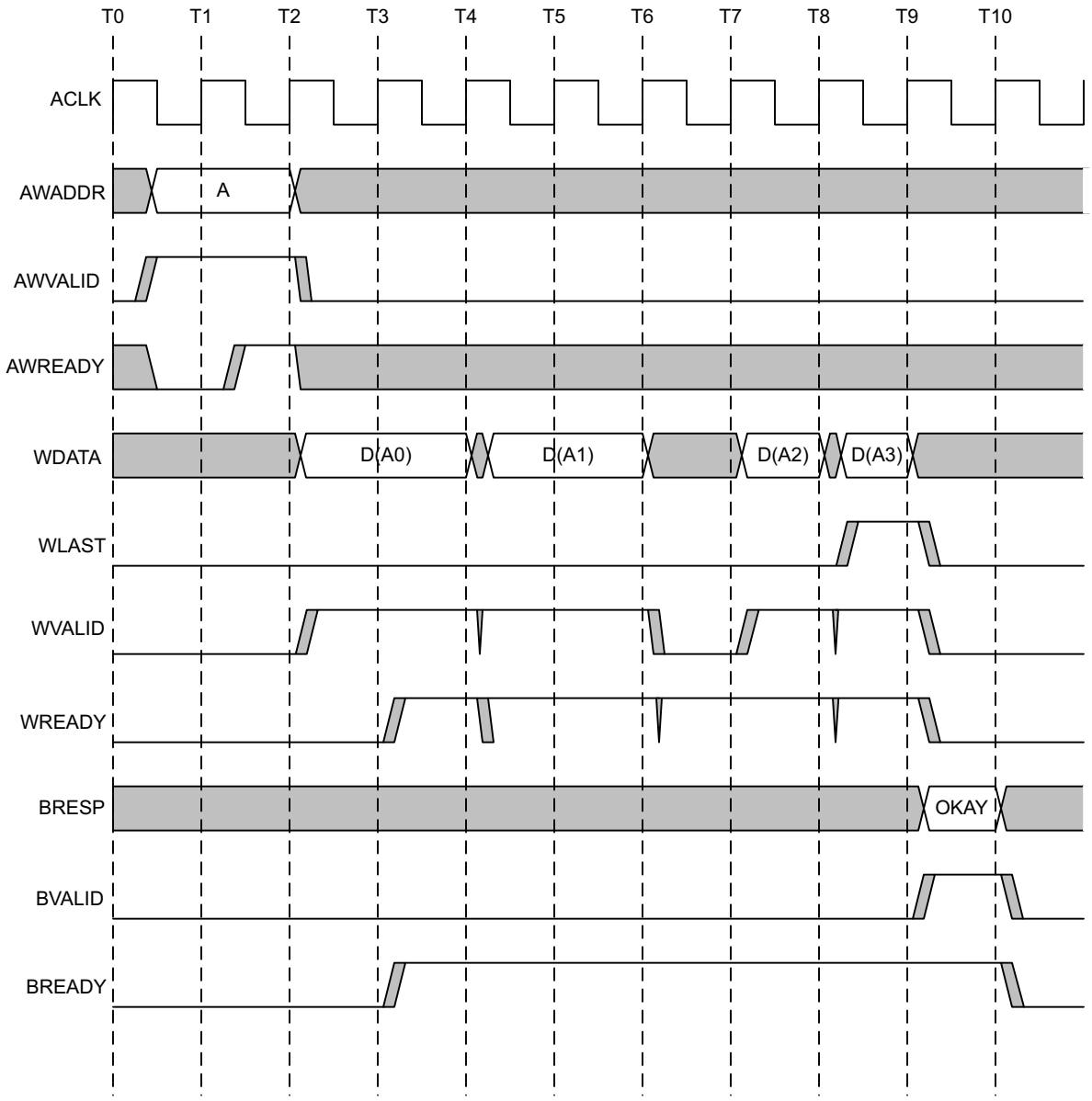

Figure 3 **• Write Transaction Timing Diagram with a Burst Length of 4**

The AXI protocol provides an ID field to enable a master to issue a number of separate transactions, each of which must be returned in order. A master can use the ARID or AWID signal of a transaction to provide the additional information on ordering requirements of the master. The slave transfers a BID to match the AWID and WID of the transaction to respond. If a master requires that all the transactions need to be completed in the same order that they are issued, then all of the transactions must have the same ID tag. In addition, the AXI protocol also provides burst type support, protection unit support, error support, and so on by using various AXI interface signals. Refer to "Appendix B: AXI Interface Signals" on page 22 for a description of AXI signals. The design examples provided in this application note provide the basic write transactions with the incremental burst transactions. To add the advanced features, you can modify the state machine to support the required features.

AXI - Read Transaction

The AXI master sends the read address using the read address channel, then the slave sends read data back using the read data channel. The following sections describe the read transaction mechanisms in the AXI protocol:

- Read Address Channel Handshake Mechanism
- Read Data Channel Handshake Mechanism

Read Address Channel Handshake Mechanism

The AXI master asserts the ARVALID (Time T1 on Figure 4) signal only when it drives the valid address and control information. It must remain asserted until:

- The AXI slave accepts the address and control information
- Asserts the associated the ARREADY (Time T2 on Figure 4) signal

Read Data Channel Handshake Mechanism

The AXI slave asserts the RVALID (Time T6 on Figure 4) signal with the appropriate ID tags only when it drives the valid read data. The RVALID signal must remain asserted until the AXI master accepts the data and asserts the RREADY signal.

Note: If the master is ready to accept data, it can assert RREADY before the slave asserts the RVALID signal, refer to Figure 4. Even if an AXI slave has only one source of read data, it must assert the RLAST signal only in response to a request for the data.

Figure 4 shows AXI read transaction timing diagram.

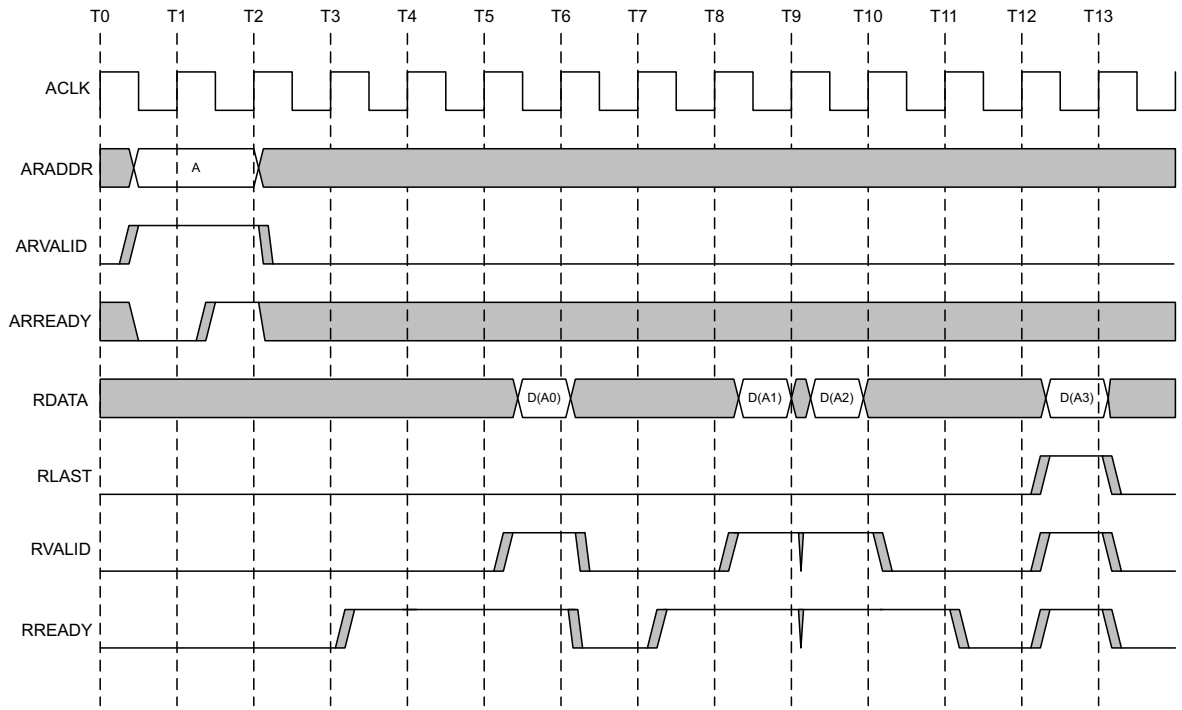

***Figure 4* • Read Transaction Timing Diagram with a Burst Length of 4**

The master uses an ID tag during the read operation using the ARID signal. The slave must send the data back with the same ID tags using the RID signal. Similarly, AWID and WID signals are used for the write transactions. Also, the read transfer provides burst type support, protection unit support, error support, similar to write transfer. The design examples provided in this application note support the basic read transactions with the incremental burst transaction.

AXI Interface in the SmartFusion2 Device

This section provides an overview of the AXI interface in the SmartFusion2 SERDESIF, MDDR, and FDDR blocks.

Figure 5 shows the system-level block diagram of the MDDR subsystem in the microcontroller subsystem (MSS). The DDR_FIC in the MDDR facilitates communication between the FPGA fabric masters and AXI transaction controller in the MDDR block. It can be configured to provide 64-bit AXI slave interface interfaces to the FPGA fabric. When configured in 64-bit AXI slave mode, the user logic requires to have an AXI master interface to initiate read or write transactions to MDDR. The AXI transaction controller receives read and write requests from the AXI masters in the Fabric and translates them into DDR controller commands.

Note: The DDR_FIC can also be configured to provide the AHB-Lite slave interface.

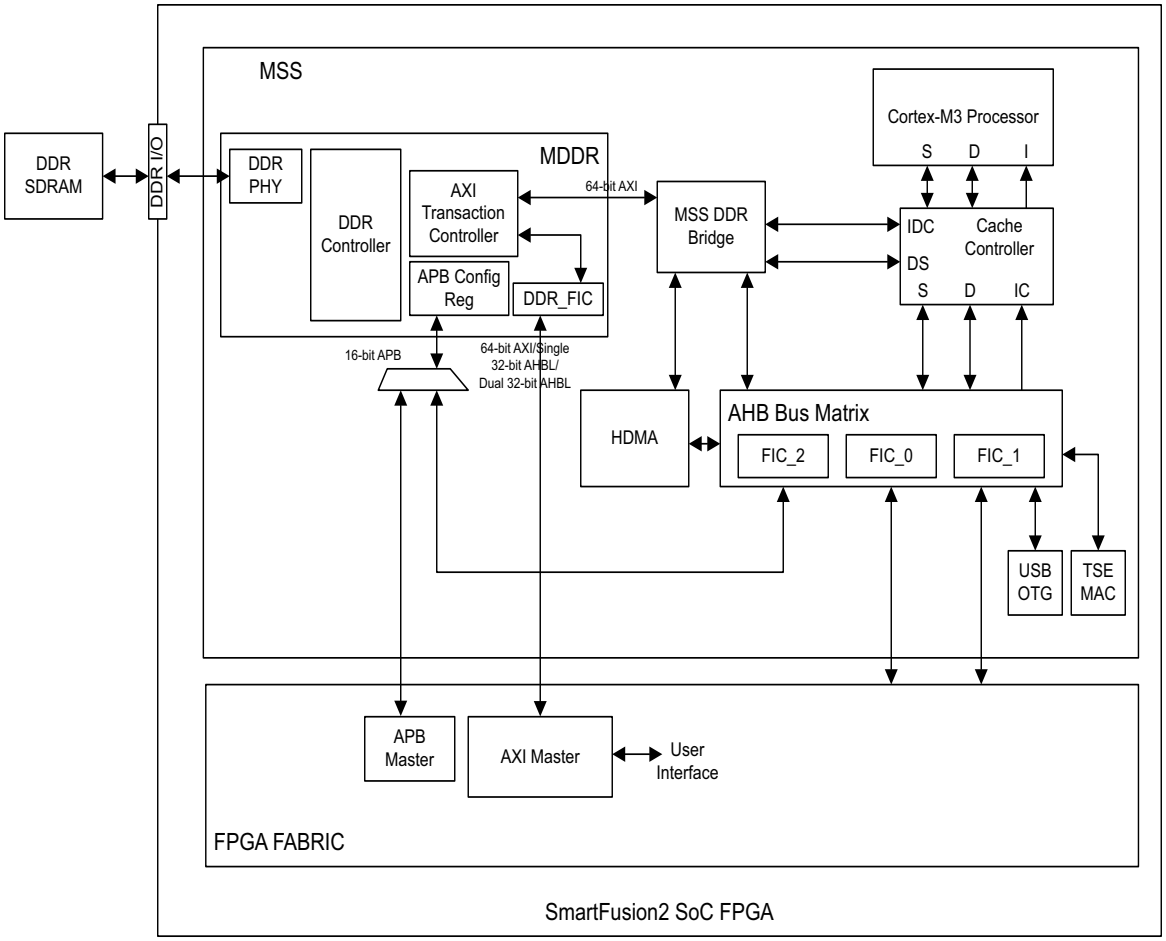

***Figure 5* • System-Level Block Diagram of the MDDR Subsystem**

Similarly, the FPGA fabric communicates with the FDDR subsystem through the AXI or AHB interfaces. The DDR_FIC in FDDR can be configured to allow the FDDR subsystem to accept the data transfer requests from an AXI or AHB master. To allow read or write transactions to the DDR memories from the FPGA fabric, you can configure the DDR_FIC in AXI slave mode and connect the user logic through an AXI master interface.

Figure 6 shows the system-level block diagram of the FDDR subsystem.

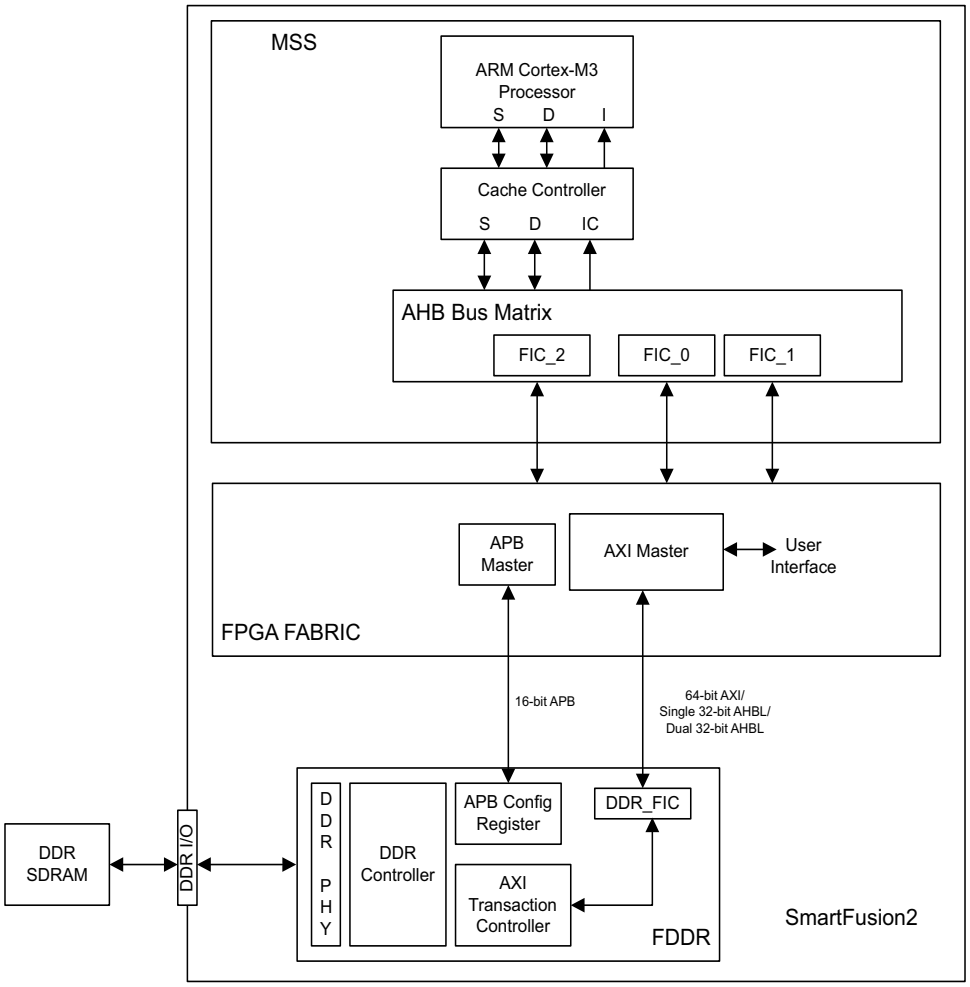

***Figure 6* • System-Level Block Diagram of the FDDR Subsystem**

The SERDESIF block interfaces with the FPGA fabric can be programmed to be either AXI master, AXI slave, AHB master, or AHB slave interface. Figure 7 shows the system-level block diagram for SERDESIF block interface, where SERDESIF application interface is configured with an AXI master and an AXI slave interface. The AXI master in FPGA fabric is used to initiate transfers to the PCIe link while the AXI slave in FPGA fabric is used to receive data from the PCIe link.

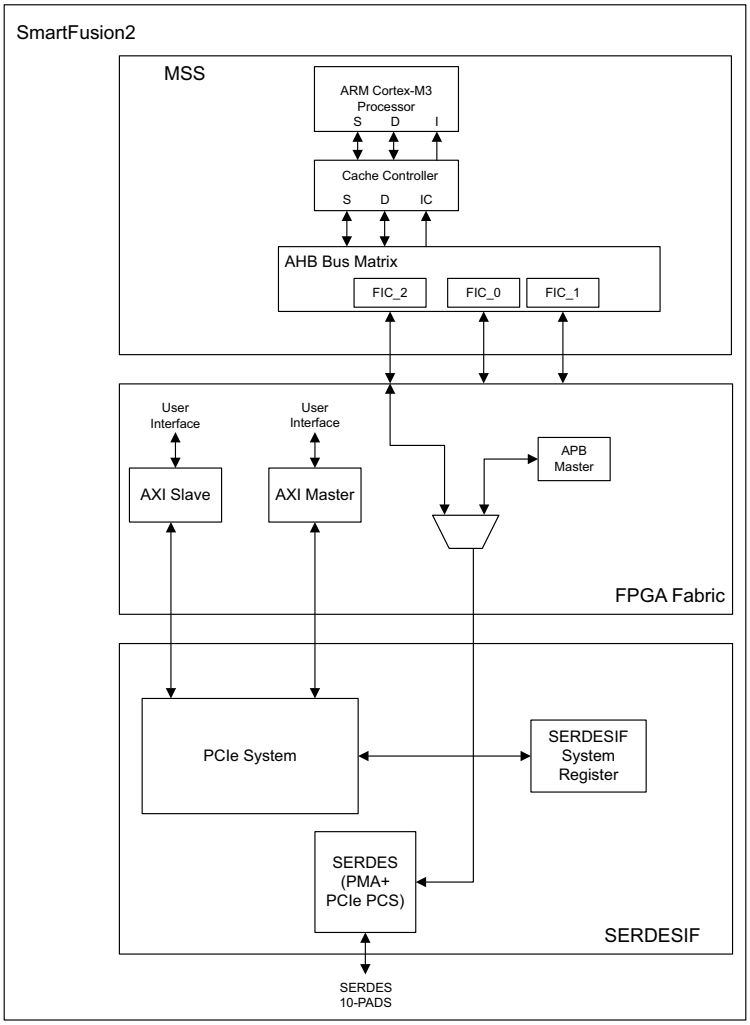

***Figure 7* • System-Level Block Diagram of the SERDESIF Block**

To summarize, the MDDR, FDDR, and SERDESIF blocks have an AXI interface and the user logic with the AXI interface must be included in the FPGA fabric to communicate with these blocks.

Implementing the AXI Master Interface on the User Logic

The following sub-sections provide detailed information on creating an AXI master interface:

- Designing AXI Master Interface Block
- Connecting AXI Master Interface Block to the High-Speed Serial Blocks
- Simulating AXI Master Interface Block

After creating the AXI master interface block, it can be connected to the MDDR, FDDR, or SERDESIF blocks and can initiate transactions through the AXI interface. The design example is available for download. Refer to "Appendix C: Design Files" on page 24 for the design file.

Note: The AXI master design example supports the basic AXI protocol. It does not support various advanced AXI protocol operations such as the unaligned addressing, out-of-order transaction completion, and low-power operation.

Designing AXI Master Interface Block

To design an AXI master interface, you can use a simple state machine.

To initiate a write transaction, the AXI master interface uses the user interface information and sends the AXI write address and AXI write control information on the write address channel. The master needs to keep the address and control on the bus until the slave accepts and asserts the AWREADY signal. The master then sends each item of write data from user interface, over the write data channel. The master must keep the write data on the bus until the slave accepts the write data and asserts the WREADY signal. During Burst mode, the next data should be on the bus only after the slave receives the previous data by asserting the WREADY signal. When the master sends the last data item, the WLAST signal goes HIGH. When the slave accepts all the data items, it drives a write response back to the master to indicate that the write transaction is complete. The master accepts the response and asserts the BREADY signal and also checks the response ID.

Figure 8 shows the AXI master write transaction state machine.

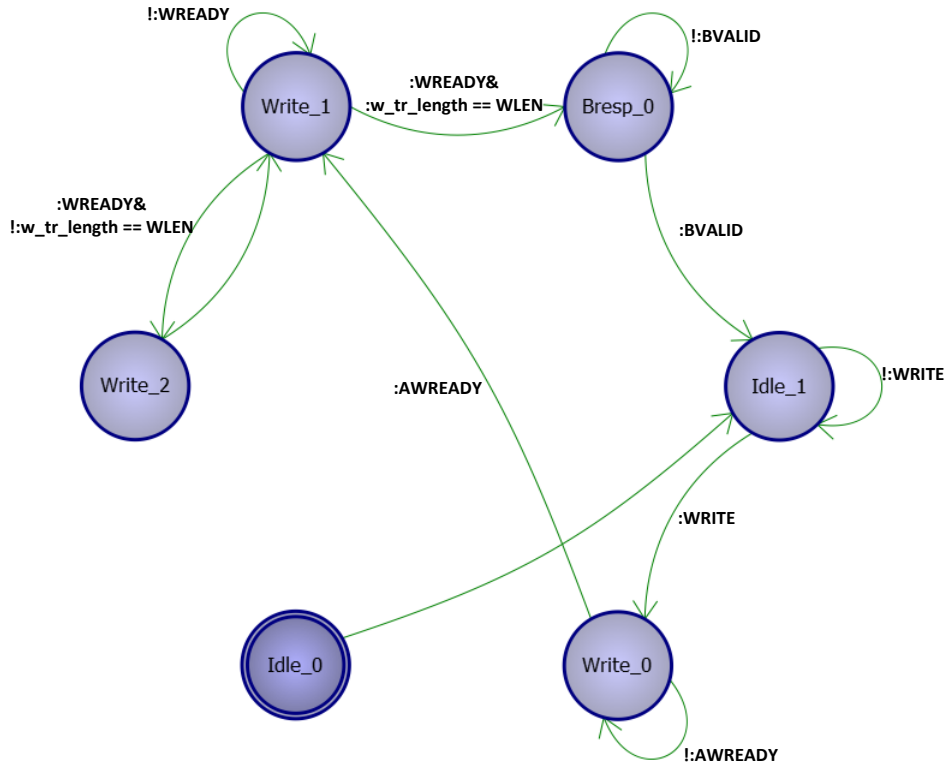

***Figure 8* • AXI Master Write Transaction State Machine**

To initiate a read transaction, the AXI master interface uses the user interface information and sends the AXI read address and AXI read control information waits for the slave to accept it. The master also drives a set of control signals that gives the length and type of the burst. The master keeps the address and control signals on the address bus, until the slave accepts and asserts the ARREADY signal. The data transfer occurs on the read data channel, the master asserts the RREADY signal to indicate that it can accept the read data. For the final data transfer of the burst, the slave asserts the RLAST signal to indicate the transfer of last data item and the read state machine moves to the Idle state.

Figure 9 shows the AXI master read transaction state machine.

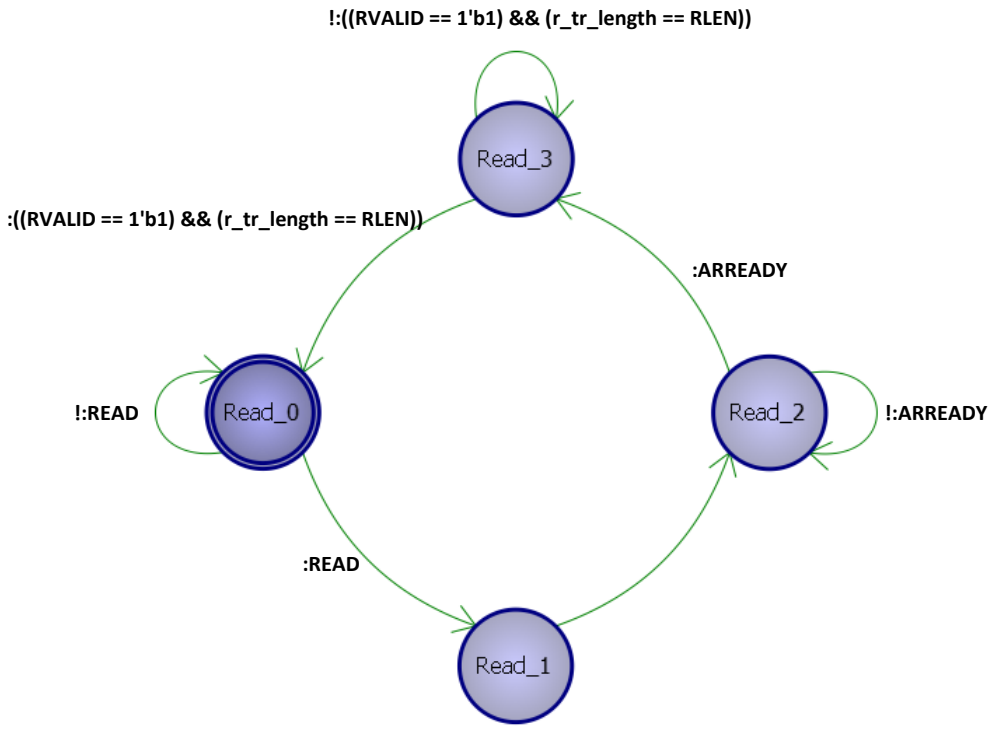

Figure 9 **• AXI Master Read Transaction State Machine**

Connecting AXI Master Interface Block to the High-Speed Serial Blocks

After compiling the RTL code, you can connect an AXI master bus interface (BIF) on the AXI master block to easily connect the AXI slave interface of MDDR, FDDR, or SERDESIF block. To connect, add the AXI master BIF in the RTL code using **Create Core** feature in Libero SoC. Refer to "Appendix A: Adding BIF to AXI Master and Slave Blocks" on page 18 for more Information.

Figure 10 shows the design example showing an AXI master connected to the FDDR block.

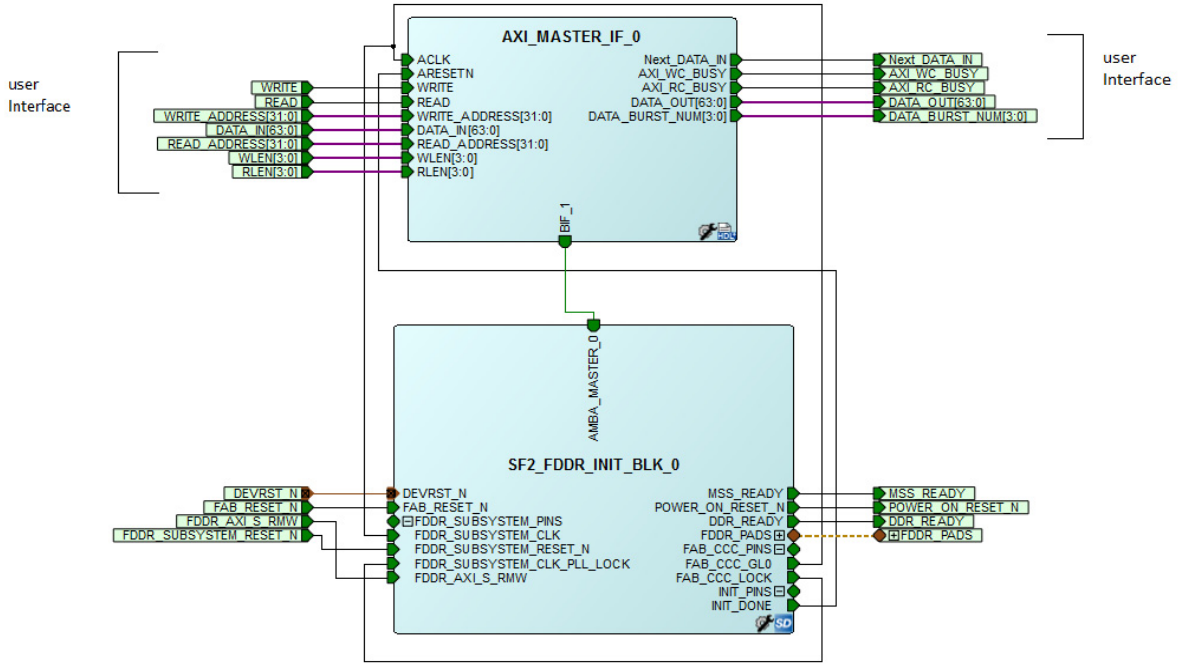

***Figure 10* • Design Example Showing the AXI Master Interface Connected to the FDDR Block**

Simulating AXI Master Interface Block

The design example includes a testbench that you can use to initiate write and read transactions on the AXI master block. The testbench uses the user interface on AXI master block to initiate write and read transactions to the FDDR. You need to open the project in the Libero SoC software, run the simulation, and view the AXI data transfer. The design example includes a `wave.do` that adds the AXI signal to the waveform.

Figure 11 shows the simulation waveforms, where the testbench initiates four writes and four reads to and from FDDR using various burst length.

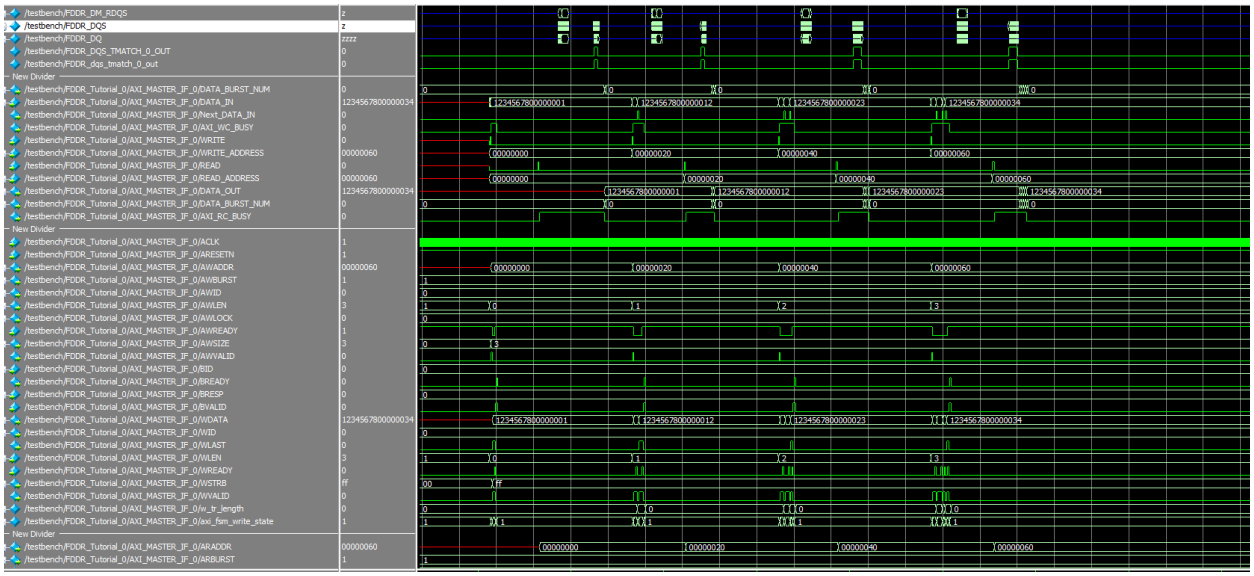

Figure 11 **• Simulation Waveform for FDDR Design with User AXI Master**

Implementing the AXI Slave Interface on the User Logic

The following sections provide detailed information on creating an AXI slave interface:

- Designing AXI Slave Interface Block
- Connecting the AXI Slave Interface Block to the SmartFusion2 SERDESIF Block
- Simulating AXI Slave Interface Block

The AXI slave supports the basic AXI protocol, it does not support various advanced AXI protocol operations including the unaligned addressing, out-of-order transaction completion, or low-power operation. The slave responds to the AXI Interface signals only when the supported modes are used. After finishing the AXI slave block, you can connect it to the SERDESIF block and receive a data transfer from the PCIe link. The design example is available to download. Refer to "Appendix C: Design Files" on page 24 for download information.

Designing AXI Slave Interface Block

This section describes the design example for creating a custom AXI slave on a memory block. On your logic, you can follow the description below and create your own AXI slave block using the state machine described below. The state machine generates the required write and read signals for the memory. The same technique can be used to create an AXI slave on your custom logic block.

During a write transfer, the user logic (AXI slave block) waits for AWVALID signal from the master and accepts the address and control information by asserting the associated AWREADY signal. The slave waits for the WVALID signal when the master drives valid write data. The slave acknowledges receipt of the write data by asserting the WREADY signal. The slave receives data until the WLAST signal is asserted by the master. The slave indicates receipt of the data by asserting the BVALID signal with a valid write response including write response ID. The BVALID signal must remain asserted until the master accepts the write response and asserts the BREADY signal.

Figure 12 shows the write transaction state diagrams for the AXI slave block.

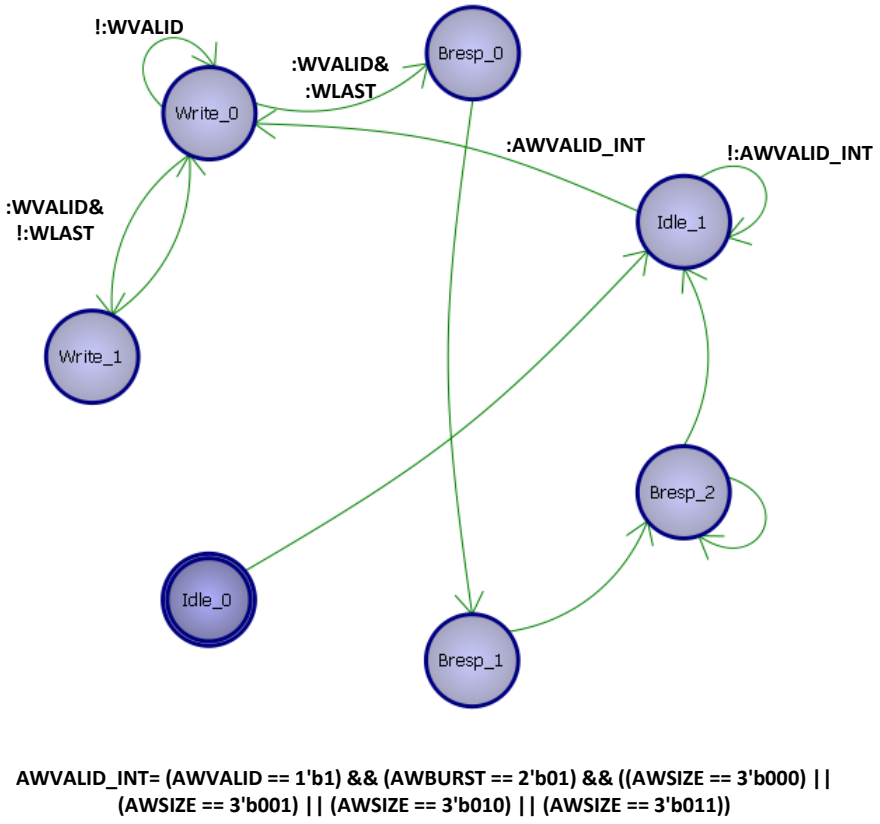

***Figure 12* • AXI Slave Write Transaction State Machine**

During a read transfer, the user logic waits for the ARVALID signal from the master and accepts the address and control information and asserts the associated ARREADY signal. The slave reads (or drives) the read data from the user interface and indicates a valid read data by driving the RVALID signal High. The slave waits for the RREADY signal before driving new data. The slave sends the RLAST signal when it drives the last data.

Figure 13 shows the read state diagram for the AXI slave block.

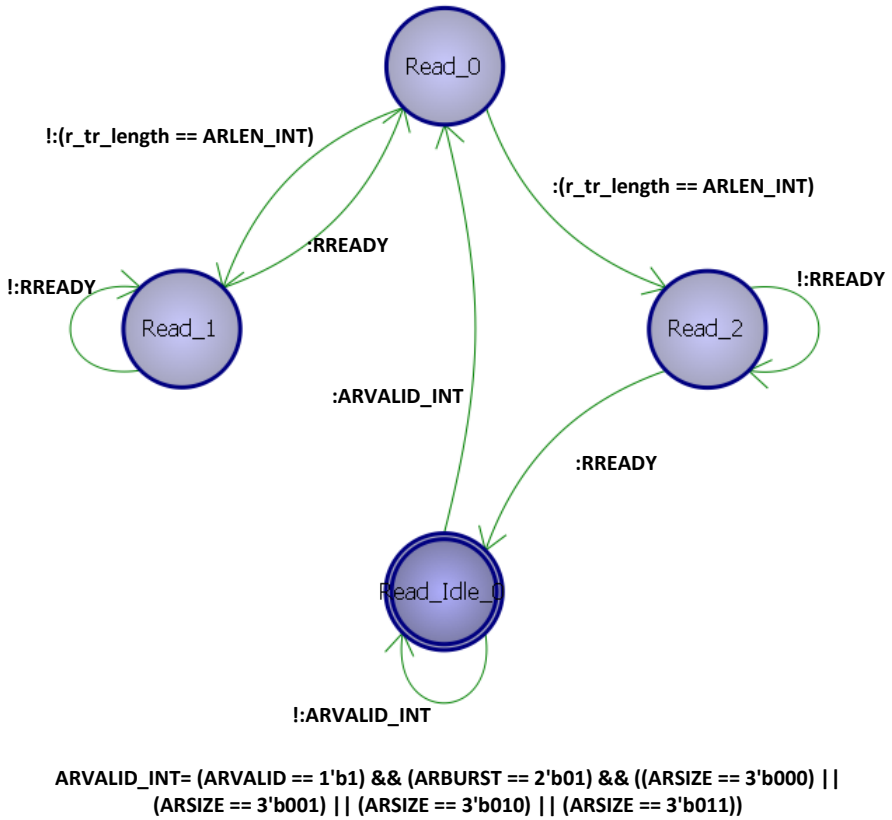

***Figure 13* • AXI Slave Read Transaction State Machine**

Connecting the AXI Slave Interface Block to the SmartFusion2 SERDESIF Block

After implementing the RTL code, connect the AXI slave interface to the AXI master interface on the SERDESIF block. To connect, add the AXI slave BIF in the RTL code using **Create Core** feature in the Libero software. Refer to "Appendix A: Adding BIF to AXI Master and Slave Blocks" on page 18 for more details.

Figure 14 shows the block diagram of the design example. The AXI slave block interfaces with the SERDESIF block AXI master interface.

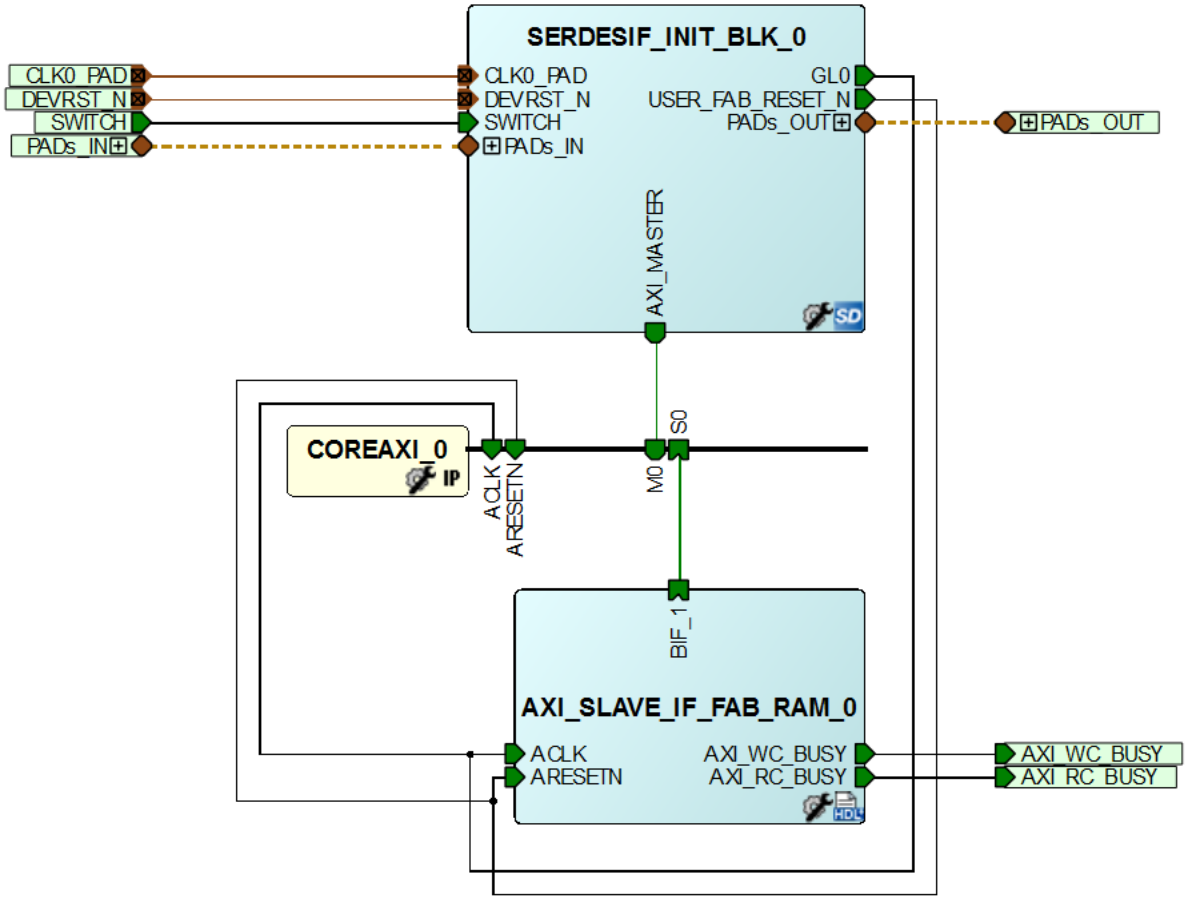

***Figure 14* • Design Example Showing AXI Slave Interface Connected to the SERDESIF Block**

Simulating AXI Slave Interface Block

The design example includes all the files required to run the simulation. The BFM simulation is used to initiate write and read transactions on the AXI bus from the SERDESIF block and test the AXI slave block. The `SERDESIF_1_user.bfm` file (included in the project) has the BFM command for writing and reading from the SERDESIF AXI master interface. Refer to the *SERDESIF Block Simulation User Guide* for more information on using the SERDESIF BFM flow. Open the project in the Libero software, run the simulation, and view the AXI data transfer.

The design example includes a `wave.do` that adds the AXI signals to the waveform. Figure 15 shows the simulation waveforms.

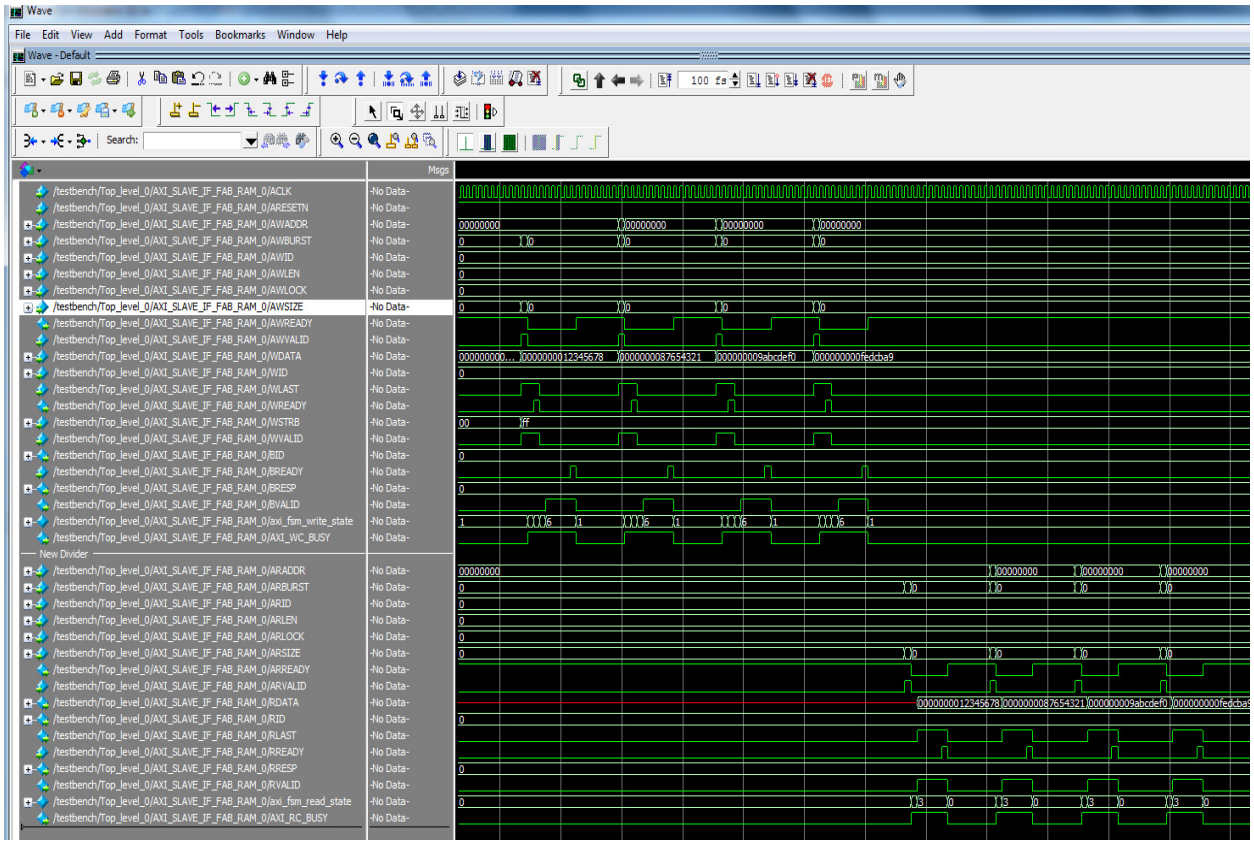

Figure 15 **• PCIe BFM Simulation Using Fabric AXI Slave**

Conclusion

This application note provides information on how to connect the user logic to AXI master and slave interfaces of high-performance communication blocks in the SmartFusion2 SoC FPGA device.

Appendix A: Adding BIF to AXI Master and Slave Blocks

To connect the AXI master or slave BIF to the AXI master block or AXI slave block:

1. Add or create the RTL source code for the custom AXI master block in the Libero software.
2. To add a bus definition to the user logic, right-click **AXI_MASTER_IF (AXI_MASTER_IF.v)** in the **Design Hierarchy** tab and select **Create Core from HDL**, as shown in Figure 16.

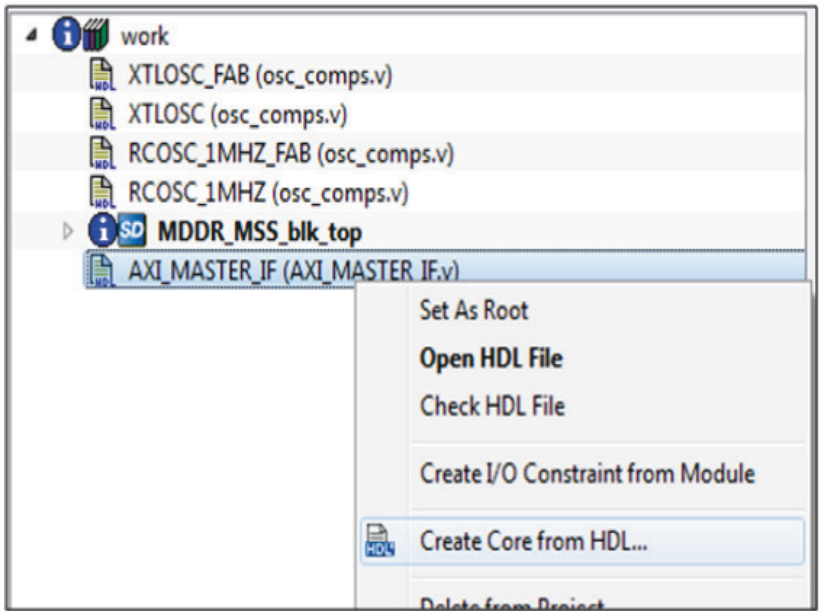

***Figure 16* • Adding BIF using Create Core from HDL**

3. Figure 17 shows the message as **Signals and Parameters were automatically extracted from HDL 'D:\....' and core 'AXI_MASTER_IF' was created**. **Would you like to remove any of the parameters, or add bus interfaces to your core?** displayed after creating the AXI_MASTER_IF and asks for confirmation. Click **Yes**

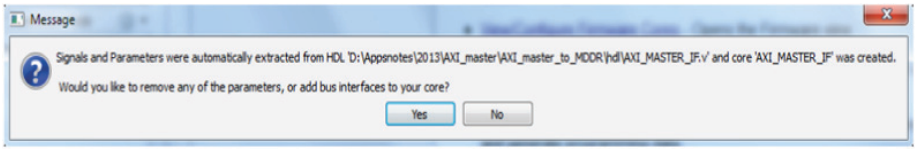

***Figure 17* • AXI_MASTER_IF Confirmation Message**

4. Click **Add Bus Interface** in the **Edit Core Definition** dialog box, as shown in Figure 18. The **Edit Core Definition** dialog shows the **Extracted Ports** and **Extracted Parameters** in AX_MASTER_IF block.

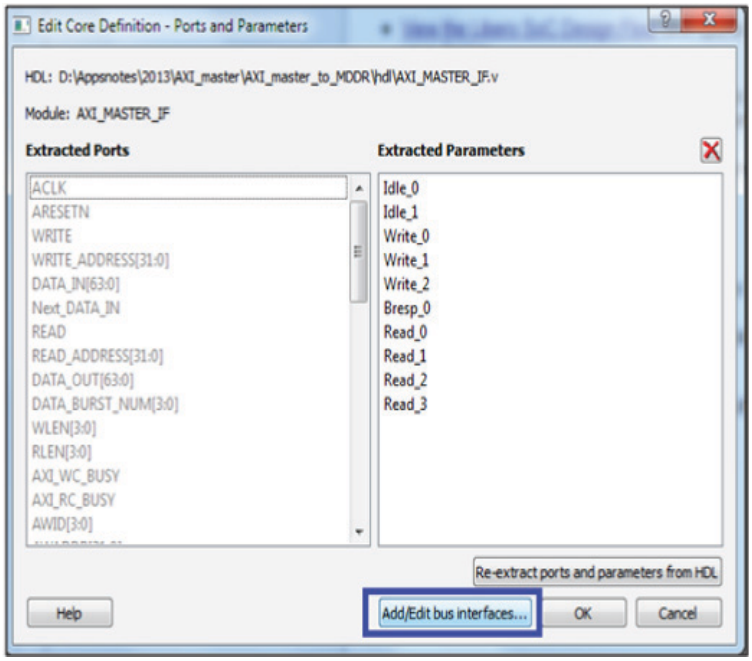

Figure 18 **• Edit Core Definition - Ports and Parameters Dialog Box**

5. Click **Add Bus Interface** from **Edit core Definition - Bus Interfaces** dialog box, as shown in Figure 19.

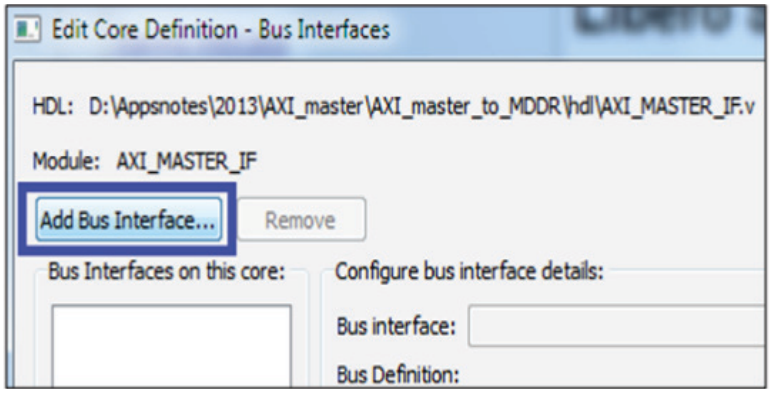

Figure 19 **• Edit Core Definition - Bus Interfaces Dialog Box**

6. Click a **AXI master** as bus definition from the **Select Bus Definition** dialog box and click **OK**, as shown in Figure 20.

Note: Select **AXI slave** as bus definition to add AXI slave block.

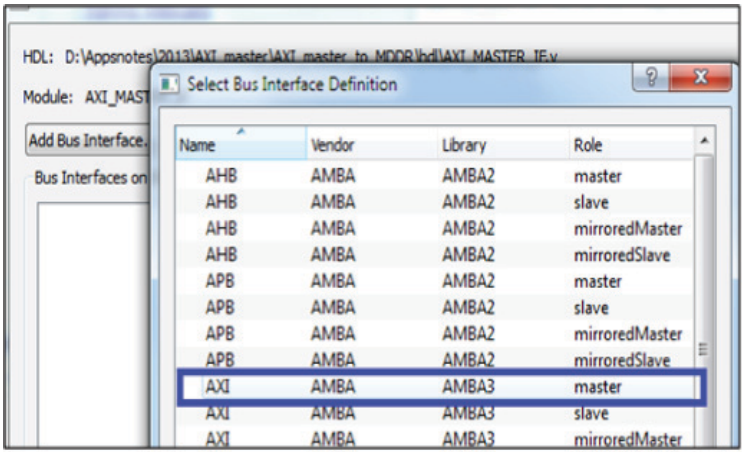

***Figure 20* • Selecting AXI Master as Bus Definition from the Select Bus Interface Definition Dialog Box**

7. Click **Map by Name** to map the signals automatically and the configurator maps similar signal names between the bus definition and pin names on the instance. Map other signals manually and click **OK**.

Note: Microsemi recommends to use the exact AXI signals in the RTL code when you use the Map by Name feature. Else manually map the AXI signal between BIF and the signal name in the RTL code.

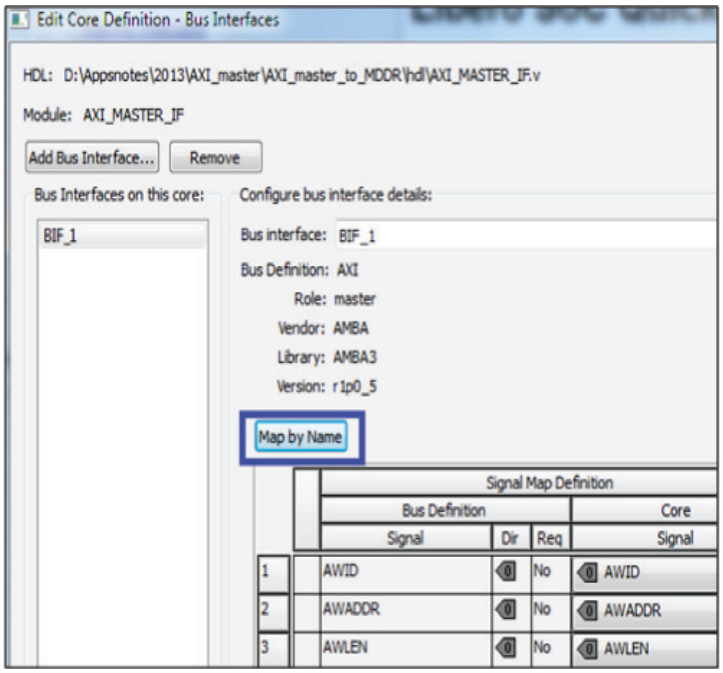

***Figure 21* • Edit Core Definition - Bus Interfaces Dialog**

8. Click **OK** to add the bus interface.

Figure 22 lists the AXI master block after adding the BIF.

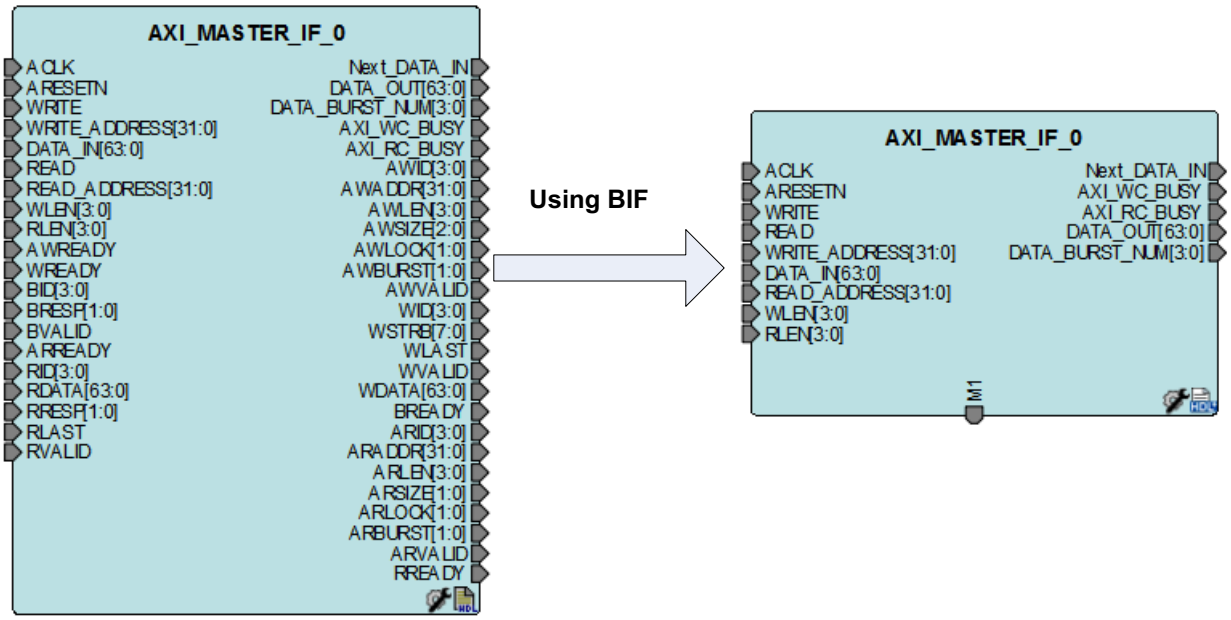

***Figure 22* • AXI Master Block after Adding the BIF**

Use the same procedure to add the AXI slave BIF in to the AXI slave interface block.

Appendix B: AXI Interface Signals

Table 2 lists the write address channel signals.

Table 2 • **Write Address Channel Signals**

Signal	Source	Description
AWID[3:0]	Master	Write address ID. This signal is an identification tag for the write address group of signals.
AWADDR[31:0]	Master	Write address. The write address bus gives the address of the first transfer in a write burst transaction. The associated control signals are used to determine the addresses of the remaining transfers in the burst.
AWLEN[3:0]	Master	Burst length. Indicates the exact number of transfers in a burst. This information determines the number of data transfers that are associated with the address.
AWSIZE[2:0]	Master	Burst size. Indicates the size of each transfer in the burst. Byte lane strobes indicate the byte lanes that are need to be updated.
AWBURST[1:0]	Master	Burst type. Burst type, coupled with the size information, details how the address for each transfer within the burst is calculated.
AWLOCK[1:0]	Master	Lock type. Provides additional information on the atomic characteristics of the transfer.
AWCACHE[3:0]	Master	Cache type. Indicates the bufferable, cacheable, write-through, write-back, and allocate attributes of the transaction.
AWPROT[2:0]	Master	Protection type. Indicates the normal, privileged, or secured protection level of the transaction and whether the transaction is a data access or an instruction access.
AWVALID	Master	Write address valid. Indicates that valid write address and control information are available. 1: Address and control information is available 0: Address and control information is not available The address and control information remain stable until the address acknowledges that the signal, AWREADY, is High.
AWREADY	Slave	Write address ready. Indicates that the slave is ready to accept an address and associated control signals. 1: Slave is ready 0: Slave is not ready

Table 3 list the write data channel signals.

Table 3 • **Write Data Channel Signals**

Signal	Source	Description
WID[3:0]	Master	Write ID tag. This signal is an identification tag of the write data transfer. The WID value must match the AWID value of the write transaction.
WDATA[31:0]	Master	Write data. The write data bus can be 8, 16, 32, 64, 128, 256, 512, or 1024 bits wide.
WSTRB[3:0]	Master	Write strobes. Indicates the byte lanes that are need to be updated in memory. There is one write strobe for each eight bits of the write data bus. Therefore, WSTRB[n] corresponds to WDATA[(8 × n) + 7:(8 × n)].
WLAST	Master	Write last. This signal indicates the last transfer in a write burst.

Table 3 • **Write Data Channel Signals (continued)**

Signal	Source	Description
WVALID	Master	Write valid. Indicates that valid write data and strobes are available. 1: Write data and strobes are available 0: Write data and strobes are not available
WREADY	Slave	Write ready. Indicates that the slave can accept the write data. 1: Slave is ready 0: Slave is not ready

Table 4 lists the write response channel signals.

Table 4 • **Write Response Channel Signals**

Signal	Source	Description
BID[3:0]	Slave	Response ID. Identification tag of the write response. The BID value must match the AWID value of the write transaction to which the slave responds.
BRESP[1:0]	Slave	Write response. Indicates the status of the write transaction. Responses are OKAY, EXOKAY, SLVERR, and DECERR allowed.
BVALID	Slave	Write response valid. Indicates that a valid write response is available. 1: Write response is available 0: Write response is not available
BREADY	Master	Response ready. Indicates that the master can accept the response. 1: Master is ready 0: Master is not ready

Table 5 lists the read address channel signals.

Table 5 • **Read Address Channel Signals**

Signal	Source	Description
ARID[3:0]	Master	Read address ID. This signal is an identification tag for the read address group of signals.
ARADDR[31:0]	Master	Read address. Indicates the initial address of a read burst transaction. Only the start address of the burst is provided and the control signals that are issued alongside the address detail how the address is calculated for the remaining transfers in the burst.
ARLEN[3:0]	Master	Burst length. Indicates the exact number of transfers in a burst. It determines the number of data transfers associated with the address.
ARSIZE[2:0]	Master	Burst size. Indicates the size of each transfer in the burst.
ARBURST[1:0]	Master	Burst type. Burst type, coupled with the size information, details how the address for each transfer within the burst is calculated.
ARLOCK[1:0]	Master	Lock type. Provides additional information on the atomic characteristics of the transfer.
ARCACHE[3:0]	Master	Cache type. Provides additional information on the cacheable characteristics of the transfer.
ARPROT[2:0]	Master	Protection type. Provides protection unit information for the transaction.

Table 5 • **Read Address Channel Signals (continued)**

Signal	Source	Description
ARVALID	Master	Read address valid. Indicates, when HIGH, that the read address and control information is valid and remain stable until the address acknowledges that the signal, ARREADY, is High. 1: Address and control information is valid 0: Address and control information is not valid
ARREADY	Slave	Read address ready. This signal indicates that the slave is ready to accept an address and associated control signals: 1: Slave is ready 0: Slave is not ready

Table 6 lists the read data channel signals.

Table 6 • **Read Data Channel Signals**

Signal	Source	Description
RID[3:0]	Slave	Read ID tag. This signal is an identification tag of the read data group of signals. The RID value is generated by the slave and must match the ARID value of the read transaction to which it is responding.
RDATA[31:0]	Slave	Read data. The read data bus can be 8, 16, 32, 64, 128, 256, 512, or 1024 bits wide.
RRESP[1:0]	Slave	Read response. Indicates the status of the read transfer. The allowable responses are OKAY, EXOKAY, SLVERR, and DECERR.
RLAST	Slave	Read last. Indicates the last transfer in a read burst.
RVALID	Slave	Read valid. Indicates that the required read data is available and the read transfer is completed: 1: Read data is available 0: Read data is not available
RREADY	Master	Read ready. Indicates that the master can accept the read data and response information: 1: Master is ready 0: Master is not ready

Appendix C: Design Files

Design files for SmartFusion2 are download from the Microsemi website: *http://soc.microsemi.com/download/rsc/?f=m2s_ac409_liberov11p7_df*. The design file consists of a Libero Verilog project. Refer to the `Readme.txt` file included in the design file for the directory structure and description.

List of Changes

The following table shows important changes made in this document for each revision.

Revision*	Changes	Page
Revision 5 (March 2016)	Updated the document for Libero v11.7 software release (SAR 76920).	NA
Revision 4 (September 2015)	Updated the document for Libero v11.6 software release (SAR 71480).	NA
Revision 3 (August 2014)	Updated the design files link under "Appendix C: Design Files" section.	24
Revision 2 (July 2014)	Updated the document for Libero v11.4 software release (SAR 58817).	NA
Revision 1 (November 2013)	Initial release.	NA

*Note: *The revision number is located in the part number after the hyphen. The part number is displayed at the bottom of the last page of the document. The digits following the slash indicate the month and year of publication.*

Microsemi Corporation (Nasdaq: MSCC) offers a comprehensive portfolio of semiconductor and system solutions for communications, defense & security, aerospace 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**.

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

© 2016 Microsemi Corporation. All rights reserved. Microsemi and the Microsemi logo are trademarks of Microsemi Corporation. All other trademarks and service marks are the property of their respective owners.

Microsemi makes no warranty, representation, or guarantee regarding the information contained herein or the suitability of its products and services for any particular purpose, nor does Microsemi assume any liability whatsoever arising out of the application or use of any product or circuit. The products sold hereunder and any other products sold by Microsemi have been subject to limited testing and should not be used in conjunction with mission-critical equipment or applications. Any performance specifications are believed to be reliable but are not verified, and Buyer must conduct and complete all performance and other testing of the products, alone and together with, or installed in, any end-products. Buyer shall not rely on any data and performance specifications or parameters provided by Microsemi. It is the Buyer's responsibility to independently determine suitability of any products and to test and verify the same. The information provided by Microsemi hereunder is provided "as is, where is" and with all faults, and the entire risk associated with such information is entirely with the Buyer. Microsemi does not grant, explicitly or implicitly, to any party any patent rights, licenses, or any other IP rights, whether with regard to such information itself or anything described by such information. Information provided in this document is proprietary to Microsemi, and Microsemi reserves the right to make any changes to the information in this document or to any products and services at any time without notice.

51900277-5/03.16