UG0644

User Guide

DDR AXI Arbiter

February 2018

Contents

1 Revision History 1
1.1 Revision 5.0 1
1.2 Revision 4.0 1
1.3 Revision 3.0 1
1.4 Revision 2.0 1
1.5 Revision 1.0 1
2 Introduction 2
3 Hardware Implementation 3
3.1 Design Description 3
3.2 Inputs and Outputs 5
3.3 Configuration Parameters 13
3.4 Timing Diagrams 14
3.5 Testbench 16
3.5.1 Simulating MSS SmartDesign 25
3.5.2 Simulating Testbench 30
3.6 Resource Utilization 31

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

In revision 5.0 of this document, the Resource Utilization section and the Resource Utilization Report were updated. For more information, see Resource Utilization (see page 31).

1.2 Revision 4.0

The following is a summary of the changes in revision 4.0 of this document.

- Added testbench configuration parameters in the table. For more information, see Configuration Parameters (see page 16)..
- Added information to simulate core using testbench. For more information, see Testbench (see page 16).
- Updated the Resource Utilization for DDR AXI Arbiter values in the table. For more information, see Resource Utilization (see page 31).

1.3 Revision 3.0

The following is a summary of the changes in revision 3.0 of this document.

- Added 8-bit information for write channel 1 and 2. For more information, see Design Description (see page 3).
- Updated Testbench section. For more information, see Testbench (see page 16).

1.4 Revision 2.0

In revision 2.0 of this document, the figures and tables in the were updated in the Testbench section. For more information, see Testbench (see page 16).

1.5 Revision 1.0

Revision 1.0 was the first publication of this document

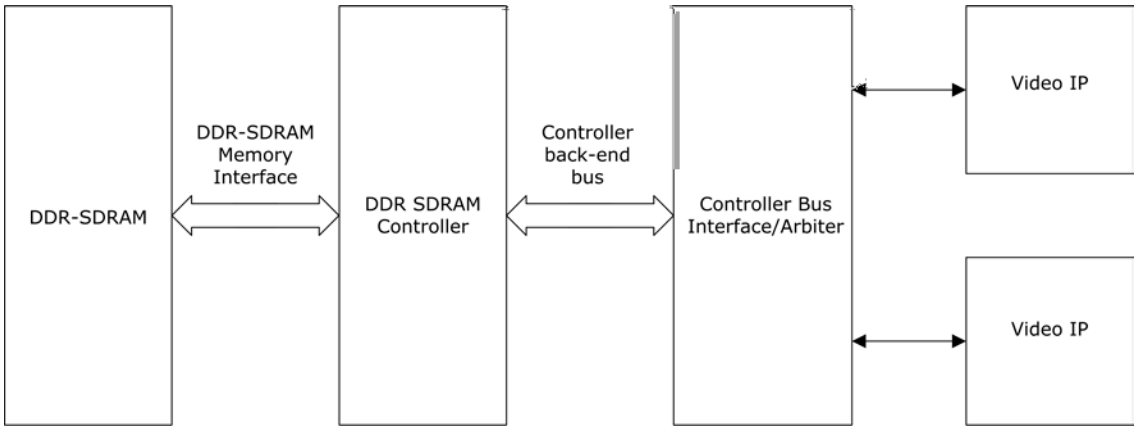
2 Introduction

Memories are an integral part of any typical video and graphics applications. They are used for buffering video pixel data. One common buffering example is display frame buffers in which the complete video pixel data for a frame is buffered in the memory.

Dual data rate (DDR)-synchronous DRAM (SDRAM) is one of the commonly used memories in video applications for buffering. SDRAM is used because of its speed which is required for fast processing in video systems.

The following figure shows an example of a system-level diagram of DDR-SDRAM memory interfacing with video application.

Figure 1 • DDR-SDRAM Memory Interfacing



In Microsemi SmartFusion®2 System-on-Chip (SoC), there are two on-chip DDR controllers with 64-bit advanced extensible interface (AXI) and 32-bit advanced high-performance bus (AHB) slave interfaces towards the field programmable gate array (FPGA) fabric. An AXI or AHB master interface is required to read and write the DDR-SDRAM memory interfaced to the on-chip DDR controllers.

3 Hardware Implementation

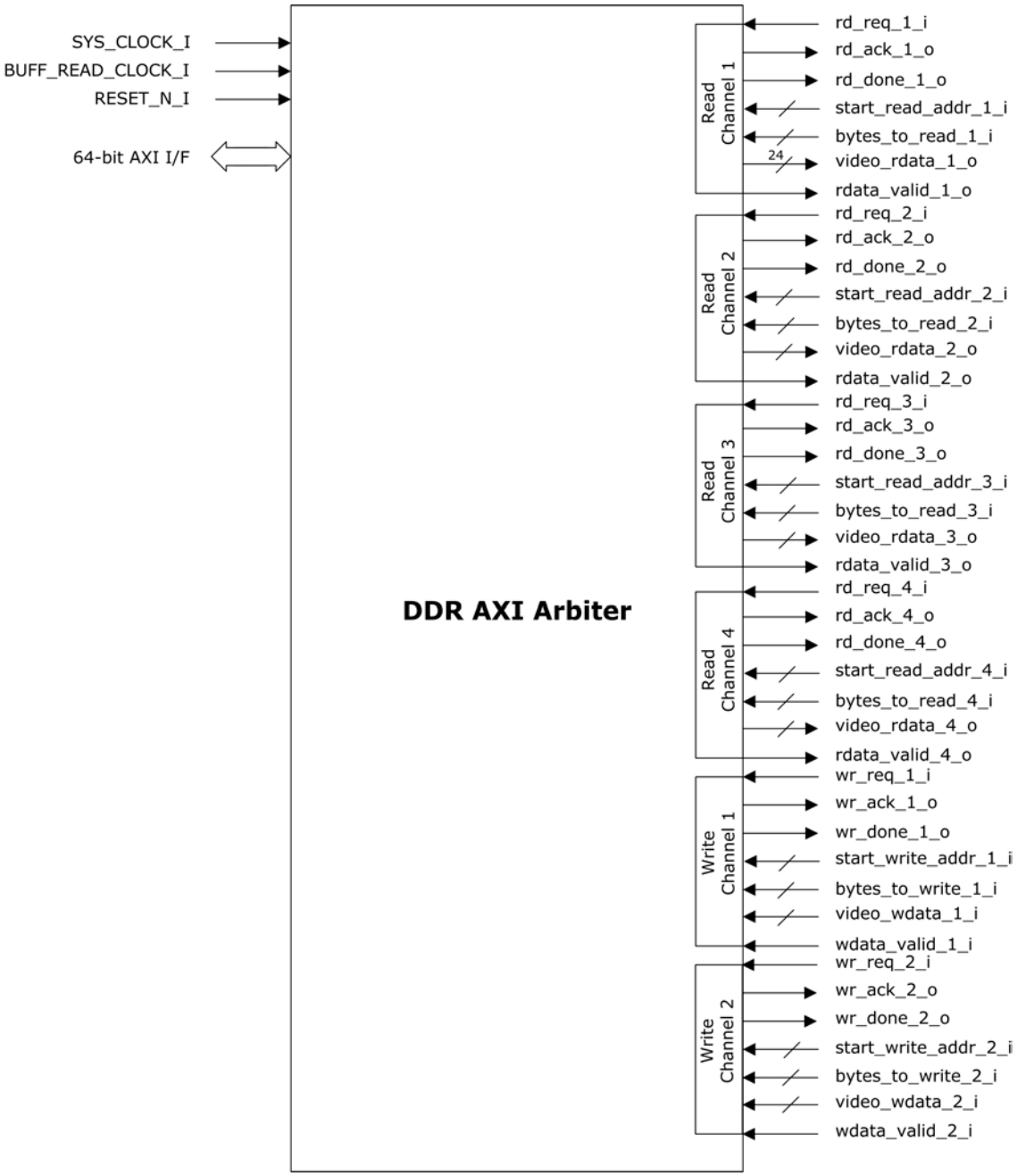
3.1 Design Description

The DDR AXI Arbiter provides a 64-bit AXI master interface to the DDR-SDRAM on-chip controllers of SmartFusion2 devices. The DDR AXI Arbiter has four read channels and two write channels towards the user logic. The block arbitrates between the four read channels to provide access to the AXI read channel in a round-robin manner. As long as the read channel 1 master's read request is high, the AXI read channel is allocated to it. Read channel 1 has fixed output data width of 24-bit. Read channels 2, 3, and 4 can be configured as 8-bit, 24-bit, or 32-bit data output width. This is selected by global configuration parameter.

The block also arbitrates between the two write channels to provide access to the AXI write channel in a round-robin manner. Both the write channels have equal priority. Write channel 1 and 2 can be configured as 8-bit, 24-bit, or 32-bit input data width.

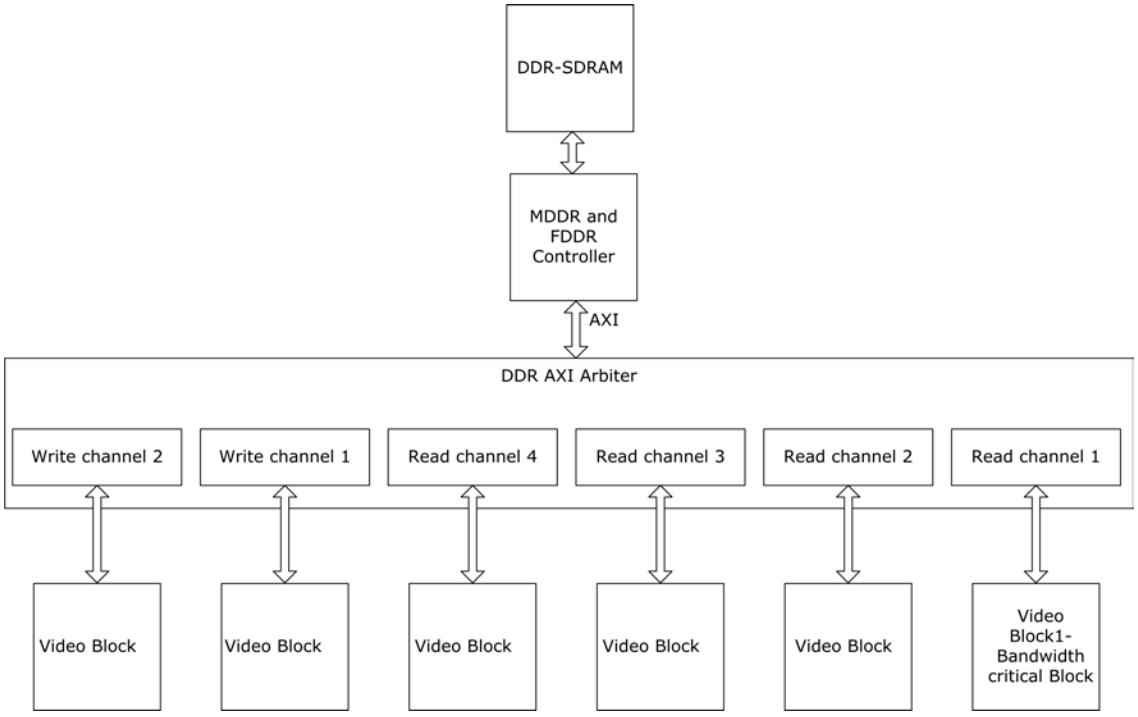
The following figure shows the top-level pin-out diagram of the DDR AXI Arbiter.

Figure 2 • Top-Level Block Diagram of DDR AXI Arbiter Block



The following figure shows the top-level block diagram of a system with DDR AXI Arbiter block ported into the SmartFusion2 device.

Figure 3 • System-Level Block Diagram of DDR AXI Arbiter on the SmartFusion2 Device



3.2 Inputs and Outputs

The following table lists the input and output ports of the DDR AXI Arbiter.

Table 1 • Input and Output Ports of the DDR AXI Arbiter

Signal Name	Direction	Width	Description
RESET_N_I	Input		Active low asynchronous reset signal to design
SYS_CLOCK_I	Input		System clock
BUFF_READ_CLOCK_I	Input		Write channel's internal buffer read clock, must be double the SYS_CLOCK_I frequency
rd_req_1_i	Input		Read request from Master 1
rd_ack_o	Output		Arbiter acknowledgment to read request from Master 1
rd_done_1_o	Output		Read completion to Master 1
start_read_addr_1_i	Input	[(g_AXI_AWIDTH-1):0]	DDR address from where read has to be started for read channel 1
bytes_to_read_1_i	Input	[(g_RD_CHANNEL1_AXI_BUFF_ AWIDTH + 3) - 1 : 0]	Bytes to be read out from read channel 1
video_rdata_1_o	Output	[(g_RD_CHANNEL1_VIDEO_DATA_WIDTH-1):0]	Video data output from read channel 1

Signal Name	Direction	Width	Description
rdata_valid_1_o	Output		Read data valid from read channel 1
rd_req_2_i	Input		Read request from Master 2
rd_ack_2_o	Output		Arbiter acknowledgment to read request from Master 2
rd_done_2_o	Output		Read completion to Master 2
start_read_addr_2_i	Input	[(g_AXI_AWIDTH-1):0]	DDR address from where read has to be started for read channel 2
bytes_to_read_2_i	Input	[(g_RD_CHANNEL2_AXI_BUFF_AWIDTH + 3) - 1 : 0]	Bytes to be read out from read channel 2
video_rdata_2_o	Output	[(g_RD_CHANNEL2_VIDEO_DATA_WIDTH-1):0]	Video data output from read channel 2
rdata_valid_2_o	Output		Read data valid from read channel 2
rd_req_3_i	Input		Read request from Master 3
rd_ack_3_o	Output		Arbiter acknowledgment to read request from Master 3
rd_done_3_o	Output		Read completion to Master 3
start_read_addr_3_i	Input	[(g_AXI_AWIDTH-1):0]	DDR address from where read has to be started for read channel 3
bytes_to_read_3_i	Input	[(g_RD_CHANNEL3_AXI_BUFF_AWIDTH + 3) - 1 : 0]	Bytes to be read out from read channel 3
video_rdata_3_o	Output	[(g_RD_CHANNEL3_VIDEO_DATA_WIDTH-1):0]	Video data output from read channel 3
rdata_valid_3_o	Output		Read data valid from read channel 3
rd_req_4_i	Input		Read request from Master 4
rd_ack_4_o	Output		Arbiter acknowledgment to read request from Master 4
rd_done_4_o	Output		Read completion to Master 4
start_read_addr_4_i	Input	[(g_AXI_AWIDTH-1):0]	DDR address from where read has to be started for read channel 4
bytes_to_read_4_i	Input	[(g_RD_CHANNEL4_AXI_BUFF_AWIDTH + 3) - 1 : 0]	Bytes to be read out from read channel 4
video_rdata_4_o	Output	[(g_RD_CHANNEL4_VIDEO_DATA_WIDTH-1):0]	Video data output from read channel 4
rdata_valid_4_o	Output		Read data valid from read channel 4
wr_req_1_i	Input		Write request from Master 1
wr_ack_1_o	Output		Arbiter acknowledgment to write request from Master 1
wr_done_1_o	Output		Write completion to Master 1
start_write_addr_1_i	Input	[(g_AXI_AWIDTH-1):0]	DDR address to which write has to happen from write channel 1
bytes_to_write_1_i	Input	[(g_WR_CHANNEL1_AXI_BUFF_AWIDTH + 3) - 1 : 0]	Bytes to be written from write channel 1
video_wdata_1_i	Input	[(g_WR_CHANNEL1_VIDEO_DATA_WIDTH-1):0]	Video data Input to write channel 1
wdata_valid_1_i	Input		Write data valid to write channel 1
wr_req_2_i	Input		Write request from Master 1

Signal Name	Direction	Width	Description
wr_ack_2_o	Output		Arbiter acknowledgment to write request from Master 2
wr_done_2_o	Output		Write completion to Master 2
start_write_addr_2_i	Input	[(g_AXI_AWIDTH-1):0]	DDR address to which write has to happen from write channel 2
bytes_to_write_2_i	Input	[(g_WR_CHANNEL2_AXI_BUFF_AWIDTH + 3) - 1 : 0]	Bytes to be written from write channel 2
video_wdata_2_i	Input	[(g_WR_CHANNEL2_VIDEO_DATA_WIDTH-1):0]	Video data Input to write channel 2
wdata_valid_2_i	Input		Write data valid to write channel 2
AXI I/F signals			
Read Address Channel			
m_arid_o	Output	[3:0]	Read address ID. Identification tag for the read address group of signals.
m_araddr_o	Output	[(g_AXI_AWIDTH-1):0]	Read address. Provides the initial address of a read burst transaction. Only the start address of the burst is provided.
m_arlen_o	Output	[3:0]	Burst length. Provides the exact number of transfers in a burst. This information determines the number of data transfers associated with the address
m_arsize_o	Output	[2:0]	Burst size. Size of each transfer in the burst
m_arburst_o	Output	[1:0]	Burst type. Coupled with the size information, details how the address for each transfer within the burst is calculated. Fixed to 2'b01 à Incremental address burst
m_arlock_o	Output	[1:0]	Lock type. Provides additional information about the atomic characteristics of the transfer. Fixed to 2'b00 à Normal Access
m_arcache_o	Output	[3:0]	Cache type. Provides additional information about the cacheable characteristics of the transfer. Fixed to 4'b0000 à Non-cacheable and non-bufferable
m_arprot_o	Output	[2:0]	Protection type. Provides protection unit information for the transaction. Fixed to 3'b000 à Normal, secure data access

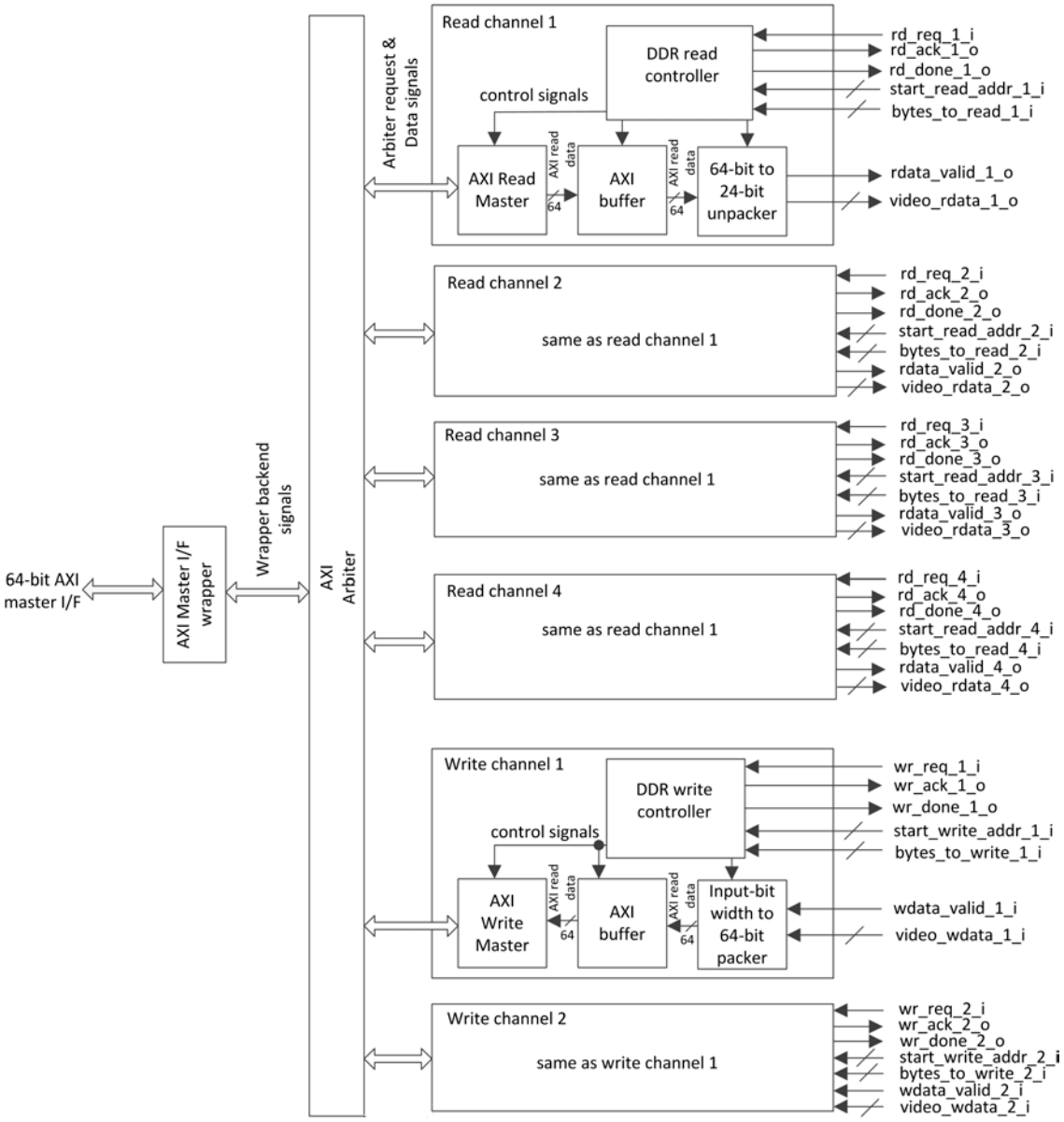
Signal Name	Direction	Width	Description
m_arvalid_o	Output		Read address valid. When HIGH, the read address and control information is valid and remain high until the address acknowledge signal, **m_arready**, is high. '1' = Address and control information valid '0' = Address and control information not valid.
m_arready_i	Input		Read address ready. The slave is ready to accept an address and associated control signals: 1 = slave ready 0 = slave not ready.
Read Data Channel			
m_rid_i	Input	[3:0]	Read ID tag. ID tag of the read data group of signals. The **m_rid** value is generated by the Slave and must match the **m_arid** value of the read transaction to which it is responding.
m_rdata_i	Input	[(g_AXI_DWIDTH-1):0]	Read data.
m_rresp_i	Input	[1:0]	Read response. The status of the read transfer. Allowable responses are OKAY, EXOKAY, SLVERR, and DECERR.
m_rlast_i	Input		Read last. Last transfer in a read burst.
m_rvalid_i	Input		Read valid. Required read data is available and the read transfer can complete: 1 = read data available 0 = read data not available.
m_rready_o	Output		Read ready. Master can accept the read data and response information: 1= master ready 0 = master not ready.
Write Address Channel			
m_awid_o	Output	[3:0]	Write address ID. Identification tag for the write address group of signals.
m_awaddr_o	Output	[(g_AXI_AWIDTH-1):0]	Write address. Provides 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.

Signal Name	Direction	Width	Description
m_awlen_o	Output	[3:0]	Burst length. Provides the exact number of transfers in a burst. This information determines the number of data transfers associated with the address.
m_awsize_o	Output	[2:0]	Burst size. Size of each transfer in the burst. Byte lane strobes indicate exactly which byte lanes to update. Fixed to 3'b011 à 8 bytes per data transfer or 64-bit transfer
m_awburst_o	Output	[1:0]	Burst type. Coupled with the size information, details how the address for each transfer within the burst is calculated. Fixed to 2'b01 à Incremental address burst
m_awlock_o	Output	[1:0]	Lock type. Provides additional information about the atomic characteristics of the transfer. Fixed to 2'b00 à Normal Access
m_awcache_o	Output	[3:0]	Cache type. Indicates the bufferable, cacheable, write-through, write-back, and allocate attributes of the transaction. Fixed to 4'b0000 à Non-cacheable and non-bufferable
m_awprot_o	Output	[2:0]	Protection type. Indicates the normal, privileged, or secure protection level of the transaction and whether the transaction is a data access or an instruction access. Fixed to 3'b000 à Normal, secure data access
m_awvalid_o	Output		Write address valid. Indicates that valid write address and control information are available: 1 = address and control information available 0 = address and control information not available. The address and control information remain stable until the address acknowledge signal, **m_awready**, goes HIGH.

Signal Name	Direction	Width	Description
m_awready_i	Input		Write address ready. Indicates that the slave is ready to accept an address and associated control signals: 1 = slave ready 0 = slave not ready.
Write Data Channel			
m_wid_o	Output	[3:0]	Write ID tag. ID tag of the write data transfer. The **m_wid** value must match the **m_awid** value of the write transaction.
m_wdata_o	Output	[(g_AXI_DWIDTH-1):0]AXI_DWDITH parameter	Write data
m_wstrb_o	Output	[7:0]	Write strobes. This signal indicates which byte lanes to update in memory. There is one write strobe for each eight bits of the write data bus
m_wlast_o	Output		Write last. Last transfer in a write burst.
m_wvalid_o	Output		Write valid. Valid write data and strobes are available: 1 = write data and strobes available 0 = write data and strobes not available.
m_wready_i	Input		Write ready. Slave can accept the write data: 1 = slave ready 0 = slave not ready.
Write Response Channel Signals			
m_bid_i	Input	[3:0]	Response ID. The identification tag of the write response. The **m_bid** value must match the **m_awid** value of the write transaction to which the slave is responding.
m_bresp_i	Input	[1:0]	Write response. Status of the write transaction. The allowable responses are OKAY, EXOKAY, SLVERR, and DECERR.
m_bvalid_i	Input		Write response valid. Valid write response is available: 1 = write response available 0 = write response not available.
m_bready_o	Output		Response ready. Master can accept the response information. 1 = master ready 0 = master not ready.

The following figure shows the internal block diagram of the DDR AXI arbiter.

Figure 4 • Internal Block Diagram of the DDR AXI Arbiter



Each read channel gets triggered when it gets a high input signal on the read_req_(x)_i input. Then it samples the starting AXI address and the bytes to read inputs which are input from the external master. The channel acknowledges the external master by toggling read_ack_(x)_o. The channel processes the inputs and generates the required AXI transactions to read the data from DDR-SDRAM. The data read out in 64-bit AXI format is stored into internal buffer. After the required data is read out and stored into the internal buffer, the un-packer module is enabled. The un-packer module unpacks each 64-bit word into the output data bit length required for that particular channel for example if the channel is configured as 32-bit output data width, each 64-bit word is sent out as two 32-bit output data words. For channel 1 which is a 24-bit channel, the un-packer unpacks each 64-bit word into 24-bit output data. As 64 is not a multiple of 24, the un-packer for read channel 1 combines a group of three 64-bit words to generate eight 24-bit data words. This puts a constraint on read channel 1 that the data bytes requested by the external master should be divisible by 8. Read channels 2, 3, and 4 can be configured as 8-bit, 24-bit, and 32-bit data width, which is determined by g_RD_CHANNEL(X) _VIDEO_DATA_WIDTH global configuration parameter. If they are configured as 24-bit, the above mentioned constraint will be applicable to each of them also. But if they are configured as 8-bit or 32-bit, there is no such constraint as 64 is multiple of 32 and 8. In these cases, each 64-bit word is unpacked into either two 32-bit data words or eight 8-bit data words.

Read Channel 1 unpacks 64-bit data words read out of DDR-SDRAM to 24-bit output data words in batches of 48 64-bit words, that is whenever 48 64-bit words are available in the internal buffer of read channel 1, the un-packer starts unpacking them to give 24-bit output data. If the requested data bytes to read are less than 48 64-bit words, the un-packer is only enabled after the complete data is read out of the DDR-SDRAM. In remaining three read channels, the un-packer starts sending out read data only after the complete requested number of bytes is read out from the DDR-SDRAM.

When a read channel configured for 24-bit output width, the starting read address must be aligned to 24-bytes boundary. This is required to satisfy the constraint that the un-packer unpacks a group of three 64-bit words to produce eight 24-bit output words.

All read channels generate the read done output to the external master after the requested bytes are sent to the external master.

In case of write channels, the external master has to input the required data to the particular channel. The write channel takes the input data and packs them into 64-bit words and stores them in the internal storage. After the required data is stored, the external master has to provide the write request along with the starting address and bytes to write. On sampling these inputs, the write channel acknowledges the external master. After this, the channel generates the AXI write transactions to write the stored data into DDR-SDRAM. All write channels generate the write done output to the external master once the requested bytes are written into DDR-SDRAM. After a write request is given to any write channel, new data must not be written into the write channel, until the current transaction completion is indicated by assertion of wr_done_(x)_o

Write channels 1 and 2 can be configured as 8-bit, 24-bit, and 32-bit data width, which is determined by g_WR_CHANNEL(X)_VIDEO_DATA_WIDTH global configuration parameter. If they are configured as 24-bit, then the bytes to be written must be multiple of eight as the internal packer packs eight 24-bit data words to generate three 64-bit data words. But if they are configured as 8-bit or 32-bit, there is no such constraint.

For a 32-bit channel, minimum two 32-bit words have to be read. For a 8-bit channel, minimum 8-bit words need to be read, because there is no padding provided by the arbiter module. In all the read and write channels, the depth of the internal buffers is multiple of the display horizontal width. The internal buffer depth is calculated as follows:

g_RD_CHANNEL(X)_HORIZONTAL_RESOLUTION* g_RD_CHANNEL(X)_VIDEO_DATA_WIDTH * g_RD_CHANNEL(X)_BUFFER_LINE_STORAGE) / g_AXI_DWIDTH

Where, X = Channel number

The internal buffer width is determined by AXI data bus width that is, configuration parameter g_AXI_DWIDTH.

The AXI read and write transactions are performed according to the ARM AMBA AXI specifications. The transaction size for each data transfer is fixed to 64-bit. The block generates AXI transactions of fixed burst length of16 beats. The block also checks whether any single burst crosses the AXI address boundary of 4 KByte. If a single burst crosses the 4 KByte boundary, the burst is split into 2 burst at the 4 KByte boundary.

3.3 Configuration Parameters

The following table lists the configuration parameters used in the hardware implementation of the DDR AXI Arbiter. These are generic parameters and can be varied based on the application requirements.

Table 2 • Configuration Parameters

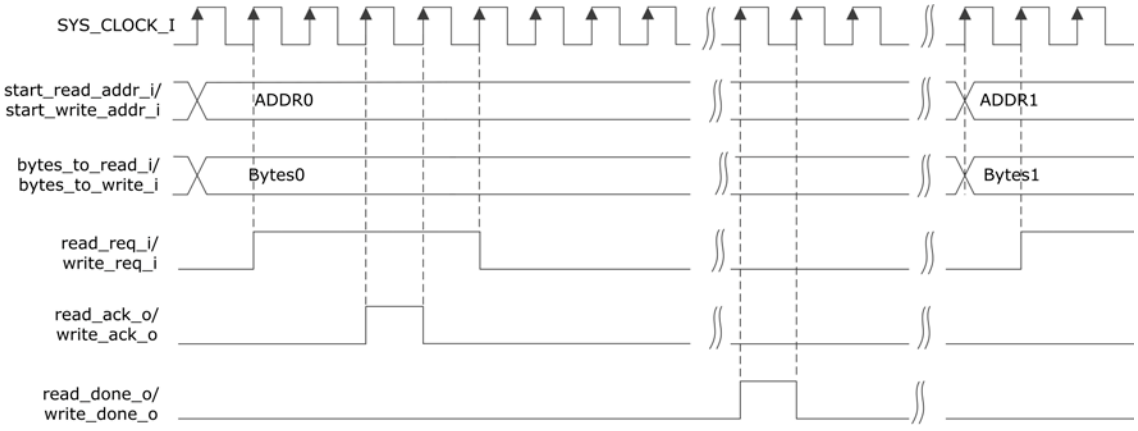
Name	Description
g_AXI_AWIDTH	AXI address bus width
g_AXI_DWIDTH	AXI data bus width
g_RD_CHANNEL1_AXI_BUFF_AWIDTH	Address bus width for the read Channel 1 internal buffer, which stores the AXI read data.
g_RD_CHANNEL2_AXI_BUFF_AWIDTH	Address bus width for the read Channel 2 internal buffer, which stores the AXI read data.
g_RD_CHANNEL3_AXI_BUFF_AWIDTH	Address bus width for the read Channel 3 internal buffer, which stores the AXI read data.
g_RD_CHANNEL4_AXI_BUFF_AWIDTH	Address bus width for the read Channel 4 internal buffer, which stores the AXI read data.
g_WR_CHANNEL1_AXI_BUFF_AWIDTH	Address bus width for the write Channel 1 internal buffer, which stores the AXI write data.
g_WR_CHANNEL2_AXI_BUFF_AWIDTH	Address bus width for the write Channel 2 internal buffer, which stores the AXI write data.
g_RD_CHANNEL1_HORIZONTAL_RESOLUTION	Video display horizontal resolution for read Channel 1
g_RD_CHANNEL2_HORIZONTAL_RESOLUTION	Video display horizontal resolution for read Channel 2
g_RD_CHANNEL3_HORIZONTAL_RESOLUTION	Video display horizontal resolution for read Channel 3
g_RD_CHANNEL4_HORIZONTAL_RESOLUTION	Video display horizontal resolution for read Channel 4
g_WR_CHANNEL1_HORIZONTAL_RESOLUTION	Video display horizontal resolution for write Channel 1
g_WR_CHANNEL2_HORIZONTAL_RESOLUTION	Video display horizontal resolution for write Channel 2
g_RD_CHANNEL1_VIDEO_DATA_WIDTH	Read Channel 1 video output bit width
g_RD_CHANNEL2_VIDEO_DATA_WIDTH	Read Channel 2 video output bit width
g_RD_CHANNEL3_VIDEO_DATA_WIDTH	Read Channel 3 video output bit width
g_RD_CHANNEL4_VIDEO_DATA_WIDTH	Read Channel 4 video output bit width
g_WR_CHANNEL1_VIDEO_DATA_WIDTH	Write Channel 1 video Input bit width.
g_WR_CHANNEL2_VIDEO_DATA_WIDTH	Write Channel 2 video Input bit width.
g_RD_CHANNEL1_BUFFER_LINE_STORAGE	Depth of the internal buffer for read Channel 1 in terms of number of display horizontal lines. The depth of the buffer is g_RD_CHANNEL1_HORIZONTAL_RESOLUTION * g_RD_CHANNEL1_VIDEO_DATA_WIDTH * g_RD_CHANNEL1_BUFFER_LINE_STORAGE) / g_AXI_DWIDTH

Name	Description
g_RD_CHANNEL2_BUFFER_LINE_STORAGE	Depth of the internal buffer for read Channel 2 in terms of number of display horizontal lines. The depth of the buffer is g_RD_CHANNEL2_HORIZONTAL_RESOLUTION * g_RD_CHANNEL2_VIDEO_DATA_WIDTH * g_RD_CHANNEL2_BUFFER_LINE_STORAGE) / g_AXI_DWIDTH
g_RD_CHANNEL3_BUFFER_LINE_STORAGE	Depth of the internal buffer for read Channel 3 in terms of number of display horizontal lines. The depth of the buffer is g_RD_CHANNEL3_HORIZONTAL_RESOLUTION * g_RD_CHANNEL3_VIDEO_DATA_WIDTH * g_RD_CHANNEL3_BUFFER_LINE_STORAGE) / g_AXI_DWIDTH
g_RD_CHANNEL4_BUFFER_LINE_STORAGE	Depth of the internal buffer for read Channel 4 in terms of number of display horizontal lines. The depth of the buffer is g_RD_CHANNEL4_HORIZONTAL_RESOLUTION * g_RD_CHANNEL4_VIDEO_DATA_WIDTH * g_RD_CHANNEL4_BUFFER_LINE_STORAGE) / g_AXI_DWIDTH
g_WR_CHANNEL1_BUFFER_LINE_STORAGE	Depth of the internal buffer for write Channel 1 in terms of number of display horizontal lines. The depth of the buffer is g_WR_CHANNEL1_HORIZONTAL_RESOLUTION * g_WR_CHANNEL1_VIDEO_DATA_WIDTH * g_WR_CHANNEL1_BUFFER_LINE_STORAGE) / g_AXI_DWIDTH
g_WR_CHANNEL2_BUFFER_LINE_STORAGE	Depth of the internal buffer for write Channel 2 in terms of number of display horizontal lines. The depth of the buffer is g_WR_CHANNEL2_HORIZONTAL_RESOLUTION * g_WR_CHANNEL2_VIDEO_DATA_WIDTH * g_WR_CHANNEL2_BUFFER_LINE_STORAGE) / g_AXI_DWIDTH

3.4 Timing Diagrams

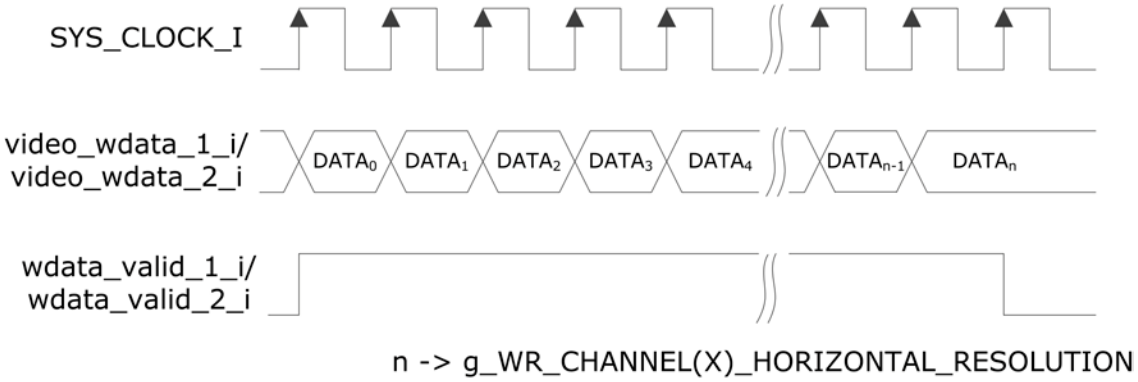
The following figure shows the connection of the read and write request inputs, starting memory address, bytes to read or write inputs from external master, read or write acknowledgment, and read or write completion outputs given by arbiter.

Figure 5 • Timing Diagram for Signals Used in Writing/Reading through AXI Interface



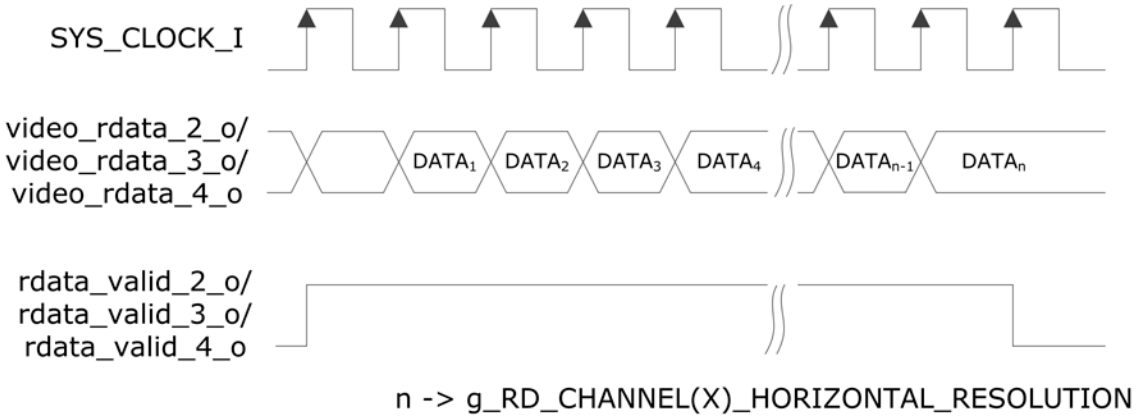
The following figure shows the connection between the write data input from the external master along with the data input valid for both write channels.

Figure 6 • Timing Diagram for Writing into Internal Storage



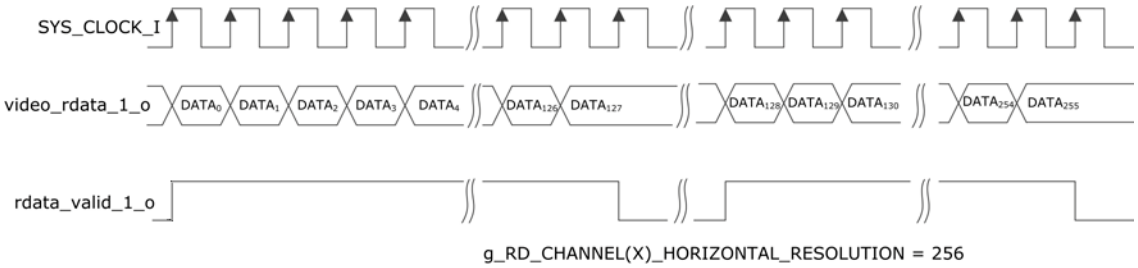
The following figure shows the connection between the read data output towards the external master along with the data output valid for all read channels 2, 3, and 4.

Figure 7 • Timing Diagram for Data Received through DDR AXI Arbiter for Read Channels 2, 3, and 4



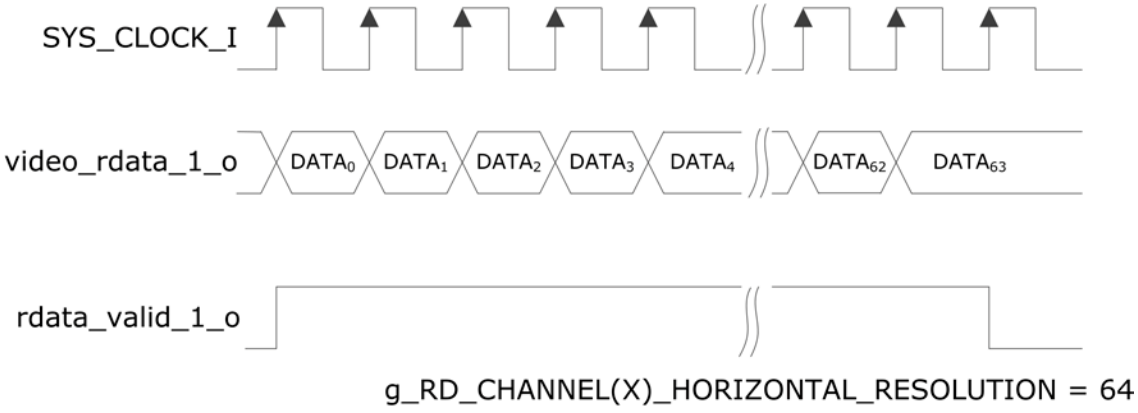
The following figure shows the connection between the read data output for the read Channel 1 when g_RD_CHANNEL 1_HORIZONTAL_RESOLUTION is greater than 128 (in this case = 256).

Figure 8 • Timing Diagram for Data Received through DDR AXI Arbiter Read Channel 1 (greater than 128 bytes)



The following figure shows the connection between the read data output for the read Channel 1 when g_RD_CHANNEL 1_HORIZONTAL_RESOLUTION is less than or equal to 128 (in this case = 64).

Figure 9 • Timing Diagram for Data Received through DDR AXI Arbiter Read Channel 1 (less than or equal to 128 bytes)



3.5 Testbench

A testbench is provided to check the functionality of the DDR Arbiter core. The following table lists the parameters that can be configured according to the application.

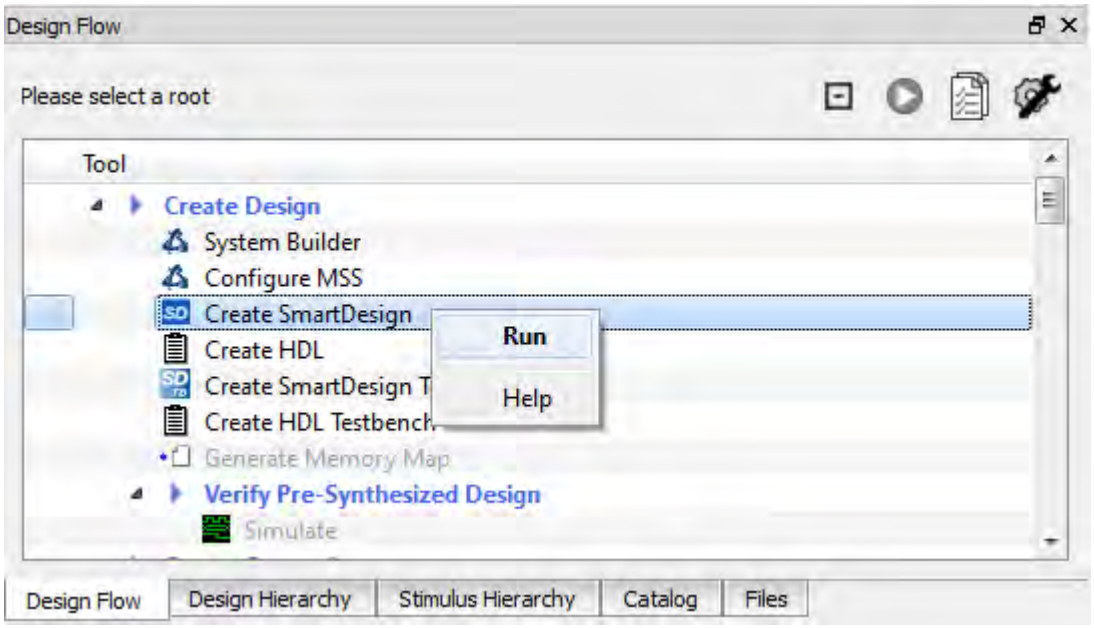
Table 3 • Testbench Configuration Parameters

Name	Description
IMAGE_1_FILE_NAME	Input file name for image to be written by write channel 1
IMAGE_2_FILE_NAME	Input file name for image to be written by write channel 2
g_DATA_WIDTH	Video data width of the read or write channel
WIDTH	Horizontal resolution of the image to be written and read by the write and read channels
HEIGHT	Vertical resolution of the image to be written and read by the write and read channels

The following steps describe how testbench is used to simulate the core through Libero SoC.

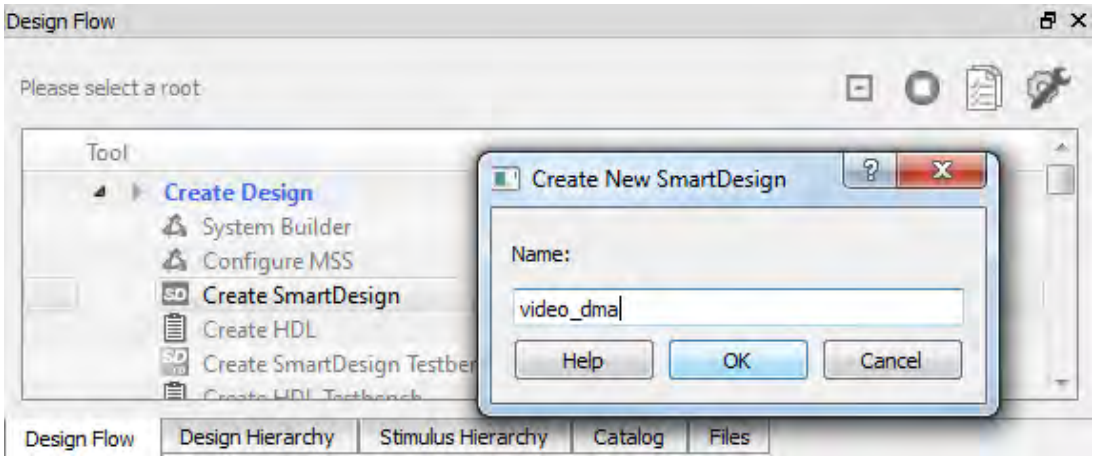
1. In the **Design Flow** window, right-click **Create SmartDesign** and click Run to create a SmartDesign.

Figure 10 • Create SmartDesign



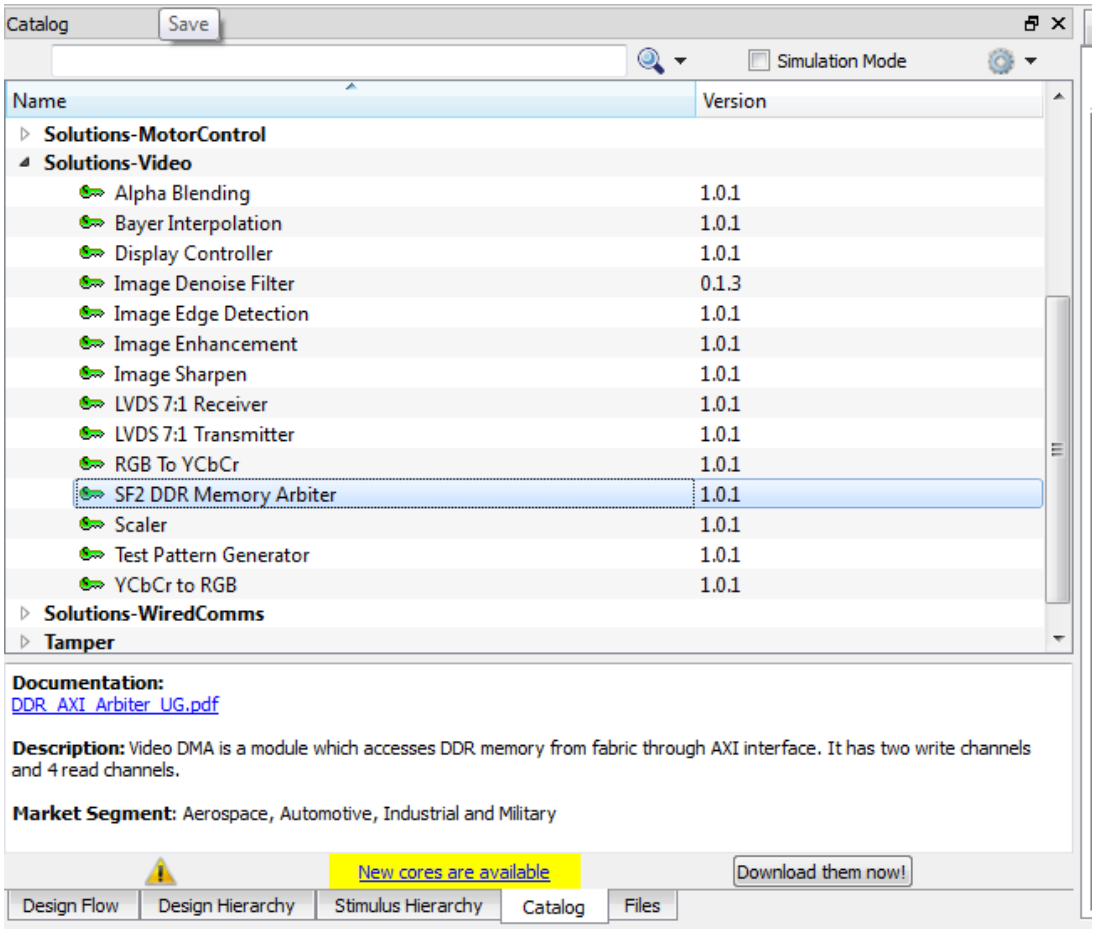
2. Enter the name of the new design as video_dma in the **Create New SmartDesign** dialog box and click **OK**. A SmartDesign is created, and a canvas is displayed on right of the **Design Flow** pane.

Figure 11 • Naming SmartDesign



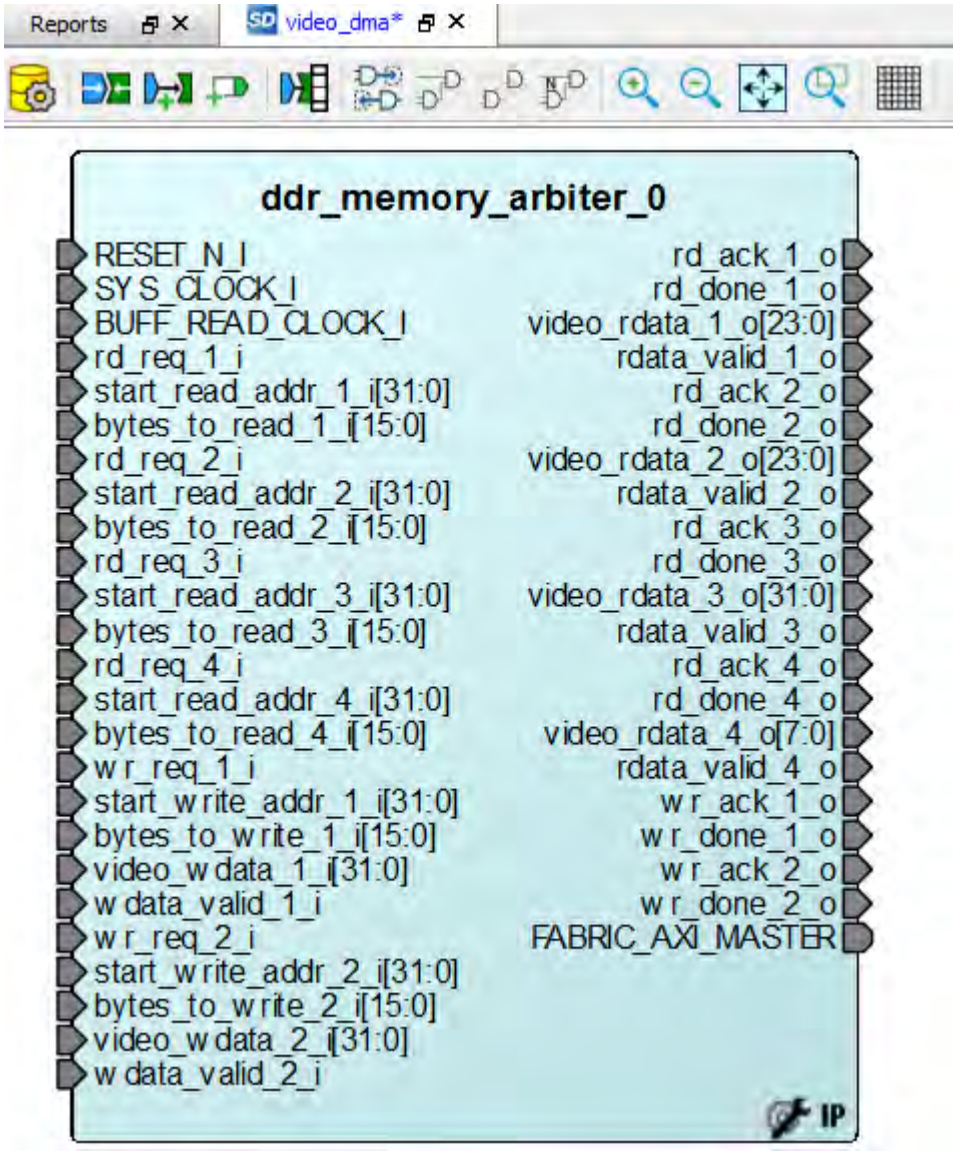
3. In the **Catalog** window, expand **Solutions-Video** and drag-and-drop **SF2 DDR Memory Arbiter** in the SmartDesign canvas.

Figure 12 • DDR Memory Arbiter in Libero SoC Catalog



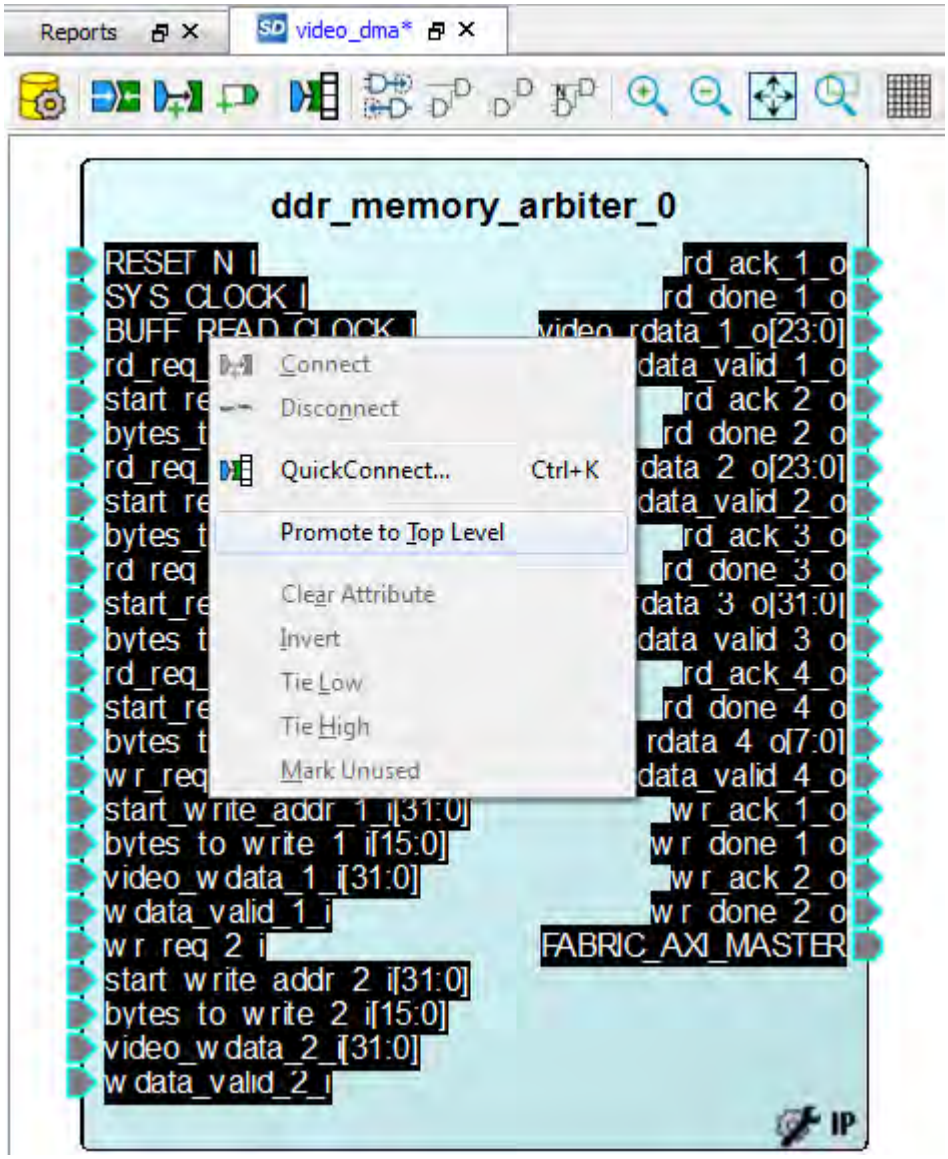
The **DDR Memory Arbiter Core** is displayed, as shown in the following figure. Double-click the core to configure the arbiter if required.

Figure 13 • DDR Memory Arbiter Core in SmartDesign Canvas



4. Select all the ports of the core and right-click and then click **Promote to Top Level**, as shown in the following figure.

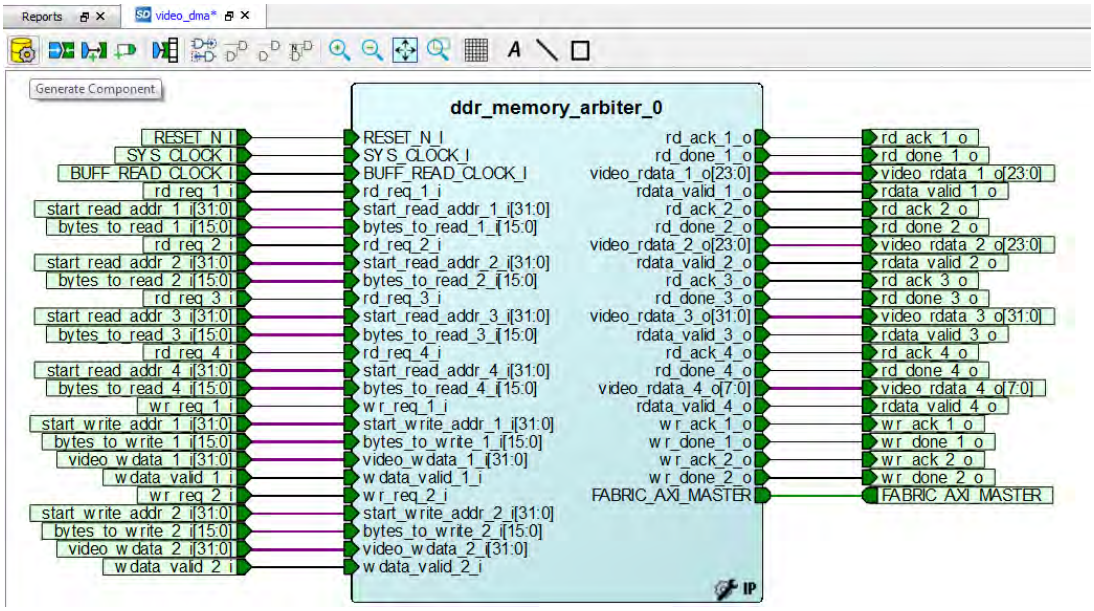
Figure 14 • Promote to Top Level Option



> ⚠ Ensure to promote all ports to top level before clicking the generate component icon in the toolbar.

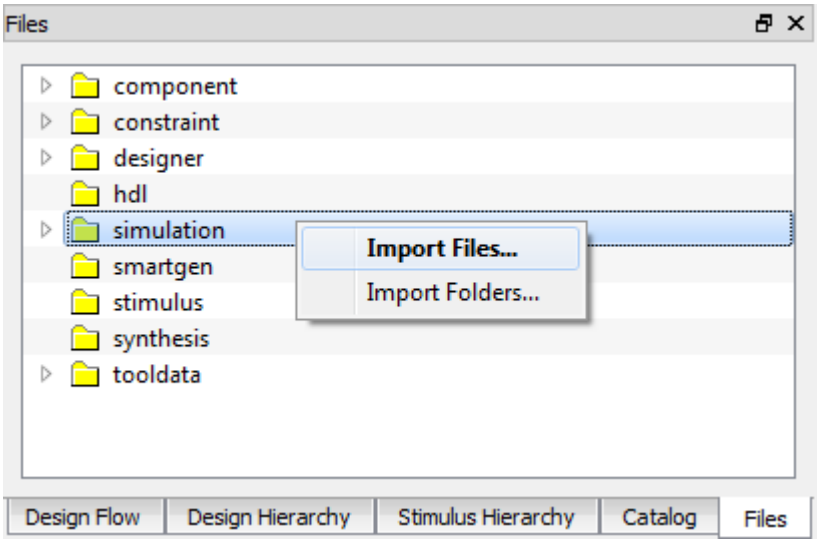
5. Click the **Generate Component** icon in the SmartDesign toolbar, as shown in the following figure. The SmartDesign component is generated.

Figure 15 • Generate Component



6. Navigate to **View** > **Windows** > **Files**. The **Files** dialog box is displayed.
7. Right-click the **simulation** folder and click **Import Files**, as shown in the following figure.

Figure 16 • Import File

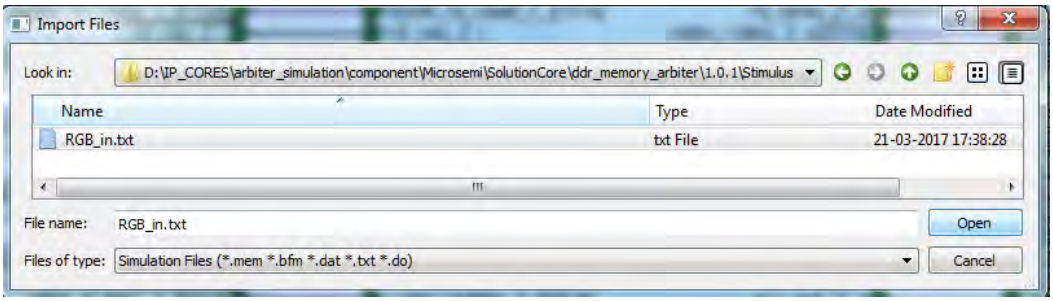


8. To Import the image stimulus file, navigate and import one of the following files and click **Open**.
   a. A sample RGB_in.txt file is provided with the testbench at the following path:

   ```
   ..\Project_name\component\Microsemi\SolutionCore\ ddr_memory_arbiter \2.0.0\Stimulu
   ```

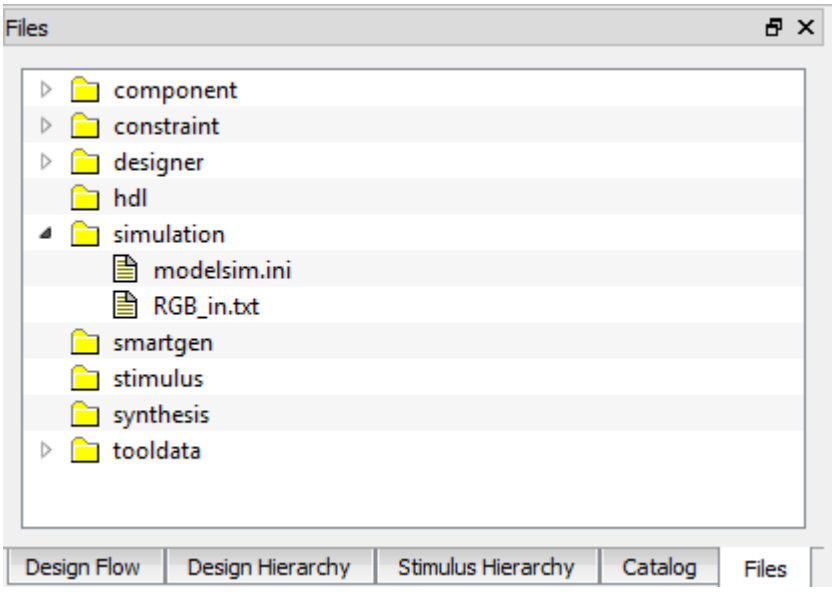
   To import the sample test bench input image, browse to the sample testbench input image file, and click Open, as shown in the following figure.

Figure 17 • Input Image File Selection



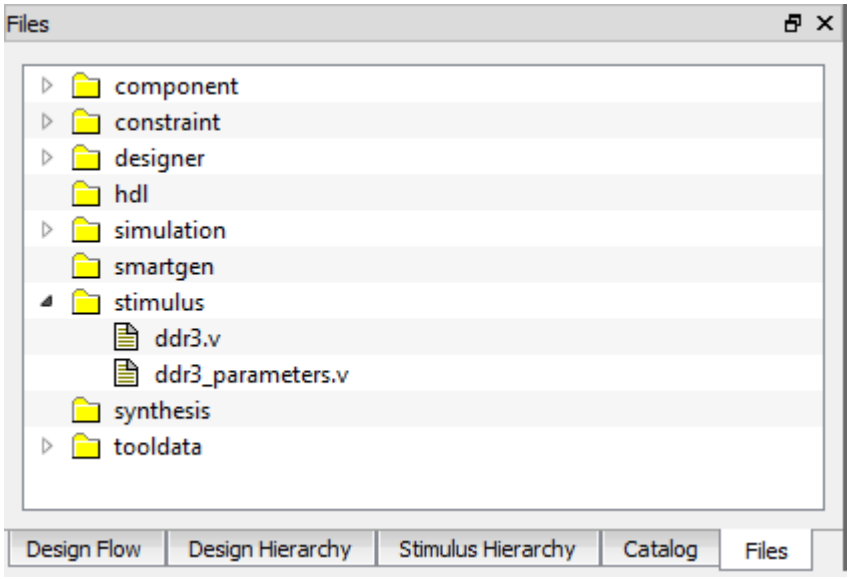
   b. To import a different image, browse to the folder containing the desired image file, and click **Open**. The imported image stimulus file is listed under simulation directory, as shown in the following figure.

Figure 18 • Input Image File in Simulation Directory



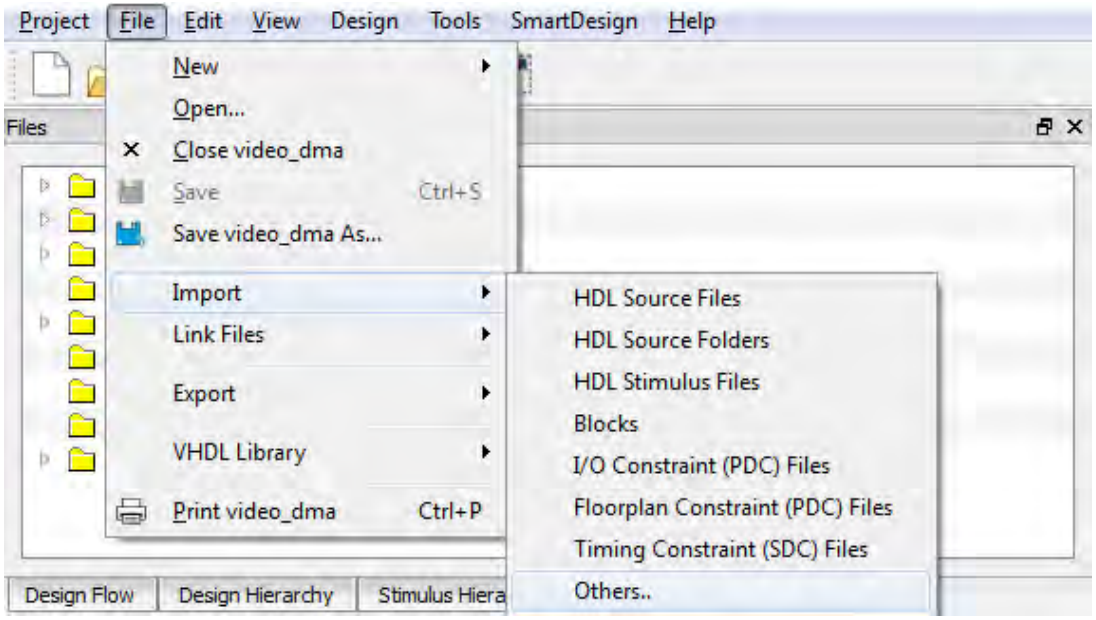
9. Import the ddr BFM files. Two files which are equivalent of DDR BFM – `ddr3.v` and `ddr3_parameters.v` are provided with the testbench at the following path: `..\Project_name\component\Microsemi\SolutionCore\ddr_memory_arbiter \2.0.0\Stimulus`.
   Right-click the **stimulus** folder and select **Import Files** option, and then select the aforementioned BFM files. The imported DDR BFM files are listed under stimulus, as shown in the following figure.

Figure 19 • Imported File



10. Navigate to **File** > **Import** > **Others**. The Import Files dialog box is displayed.

Figure 20 • Import Testbench File



11. `Import the testbench and MSS component files (top_tb.cxf, mss_top_sb_MSS.cxf, mss_top.`

    `..\Project_name\component\Microsemi\SolutionCore\ddr_memory_arbiter \2.0.0\Stimulus`

Figure 21 • Import Testbench and MSS Component Files

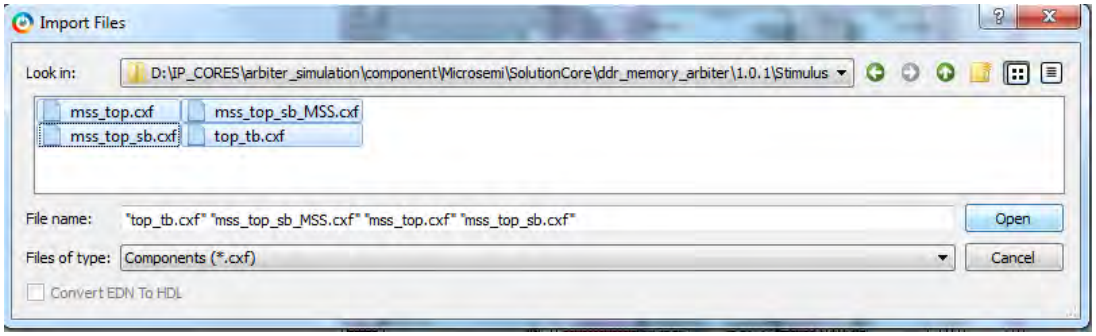
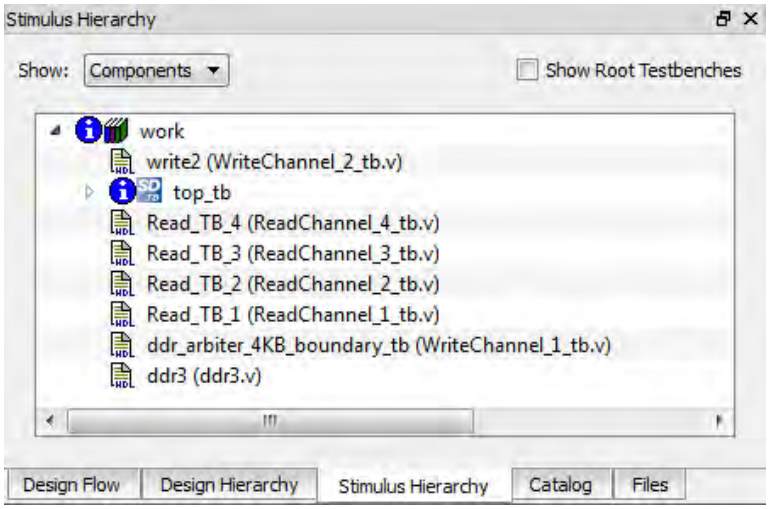


Figure 22 • top_tb Created

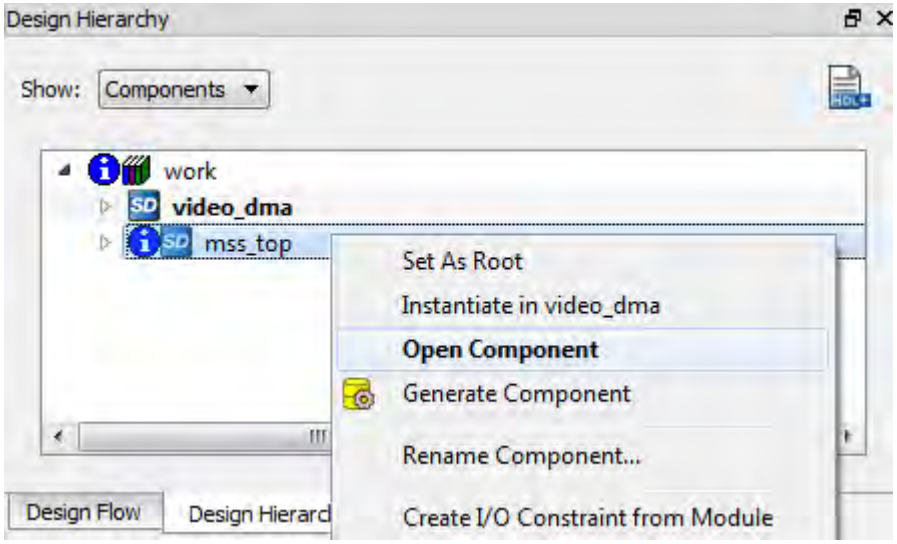


3.5.1 Simulating MSS SmartDesign

The following instructions describe how to simulate MSS SmartDesign:

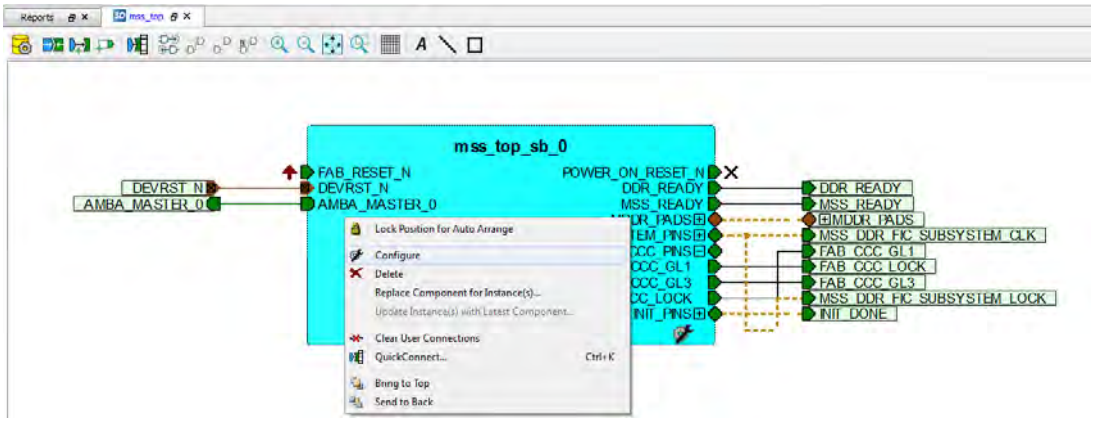
1. Click the **Design Hierarchy** tab and select Component from the show drop-down list. The imported MSS SmartDesign is displayed.
2. Right-click **mss_top** under **Work** and click **Open Component**, as shown in the following figure. The **mss_top_sb_0** component is displayed.

Figure 23 • Open Component



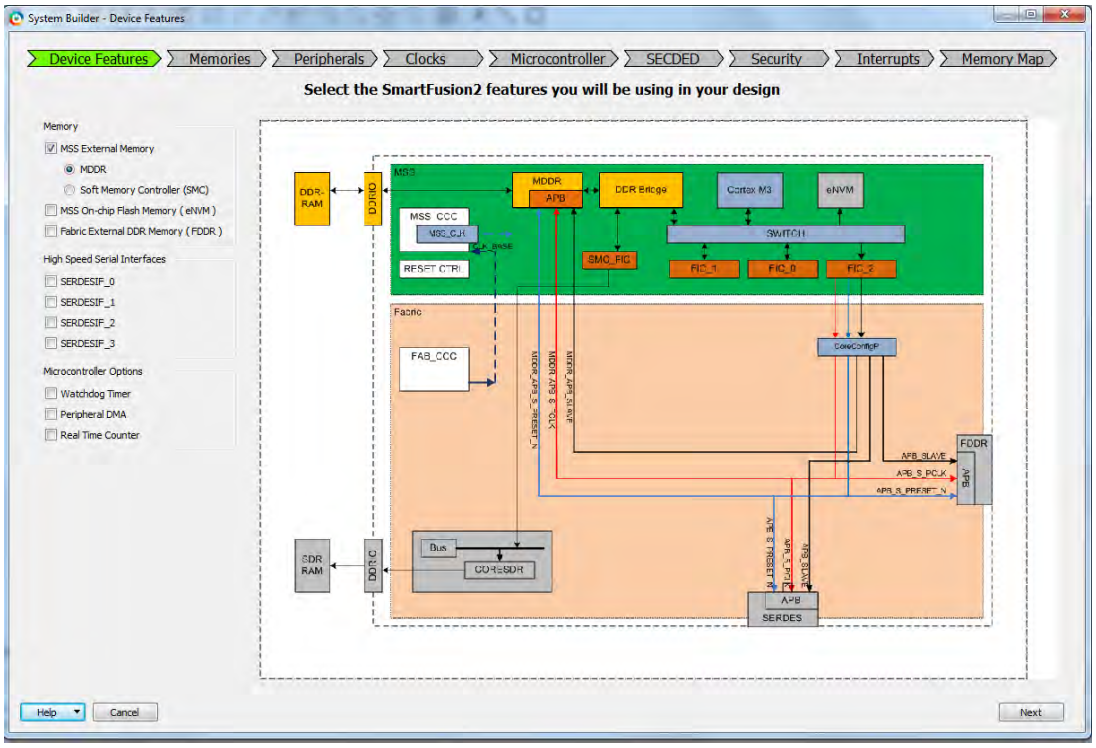
3. Right-click the mss_top_sb_0 component and click Configure, as shown in the following figure.

Figure 24 • Configure Component



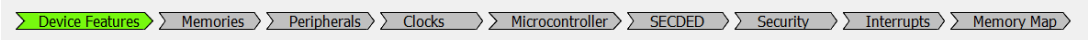
The **MSS Configuration** window is displayed, as shown in the following figure.

Figure 25 • MSS Configuration Window



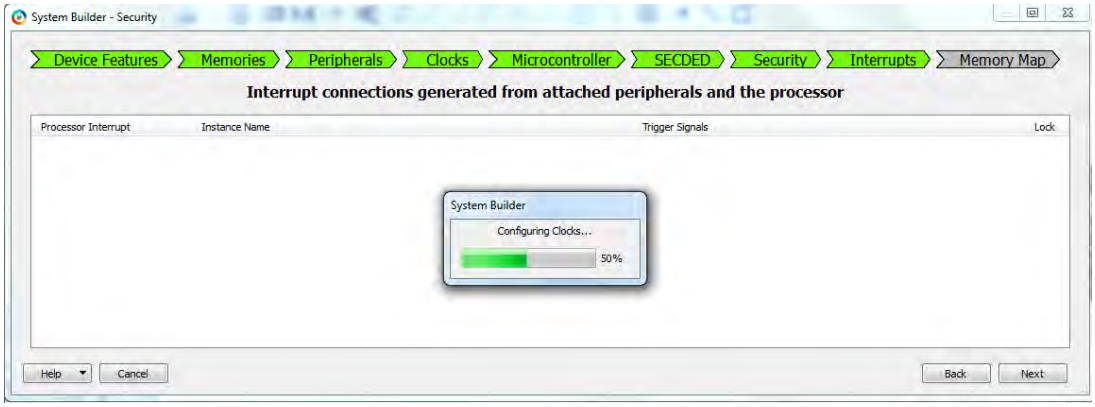
4. Click **Next** through all the configuration tabs, as shown in the following image.

Figure 26 • Configuration Tabs



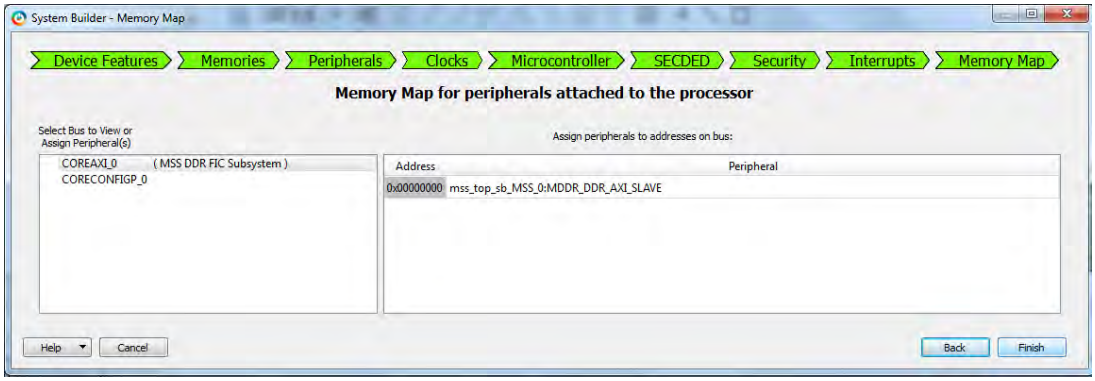
The MSS is configured after Interrupts tab is configured. The following figure shows the progression of **MSS Configuration**.

Figure 27 • MSS Configuration Window After Configuration



5. Click Next after the configuration is complete. The Memory Map window is displayed, as shown in the following figure.

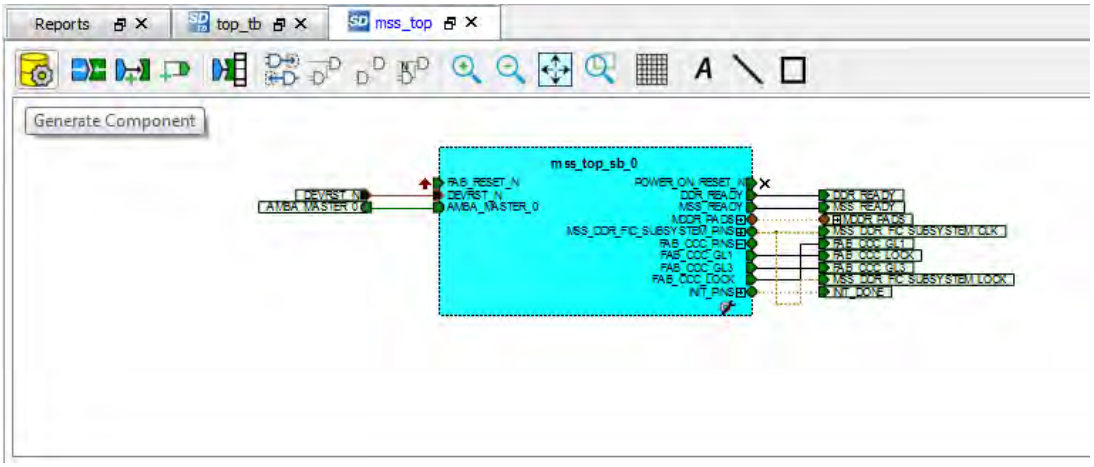
Figure 28 • Memory Map



6. Click **Finish**.

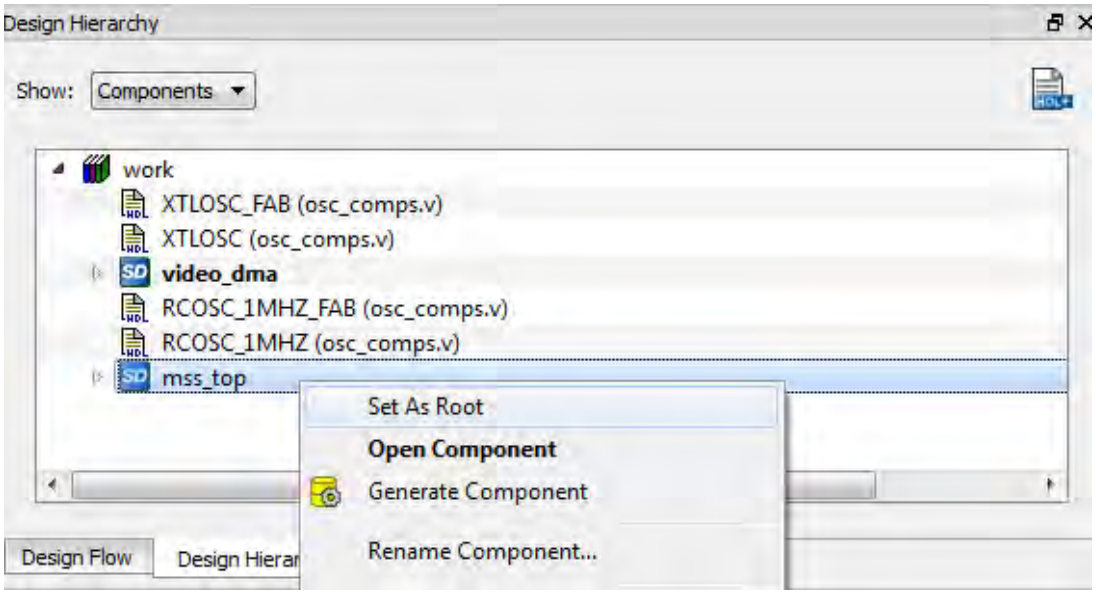
7. Click **Generate Component** from the SmartDesign toolbar to generate the MSS, as shown in the following figure.

Figure 29 • Generate Component



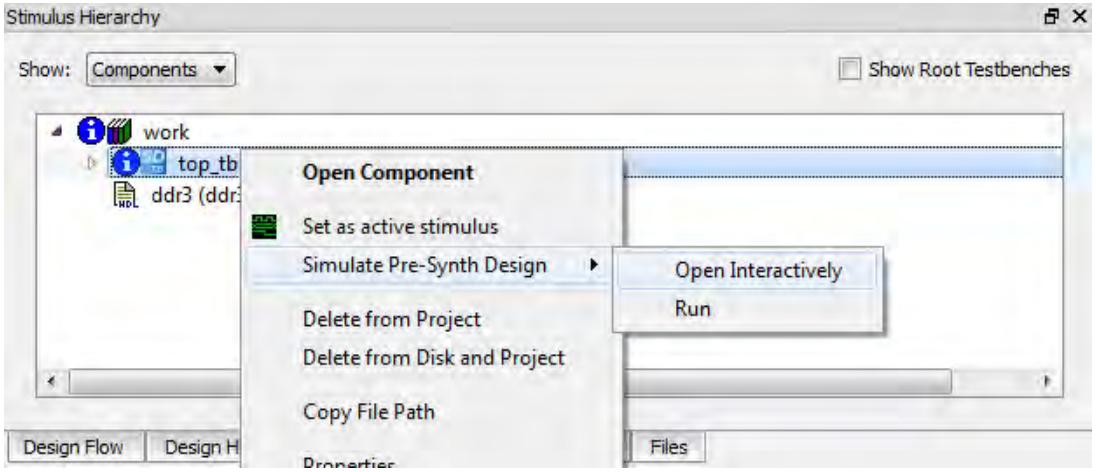
8. In the **Design Hierarchy** window, right-click **mss_top** under **Work** and click **Set As Root**, as shown in the following figure.

Figure 30 • Set MSS as Root



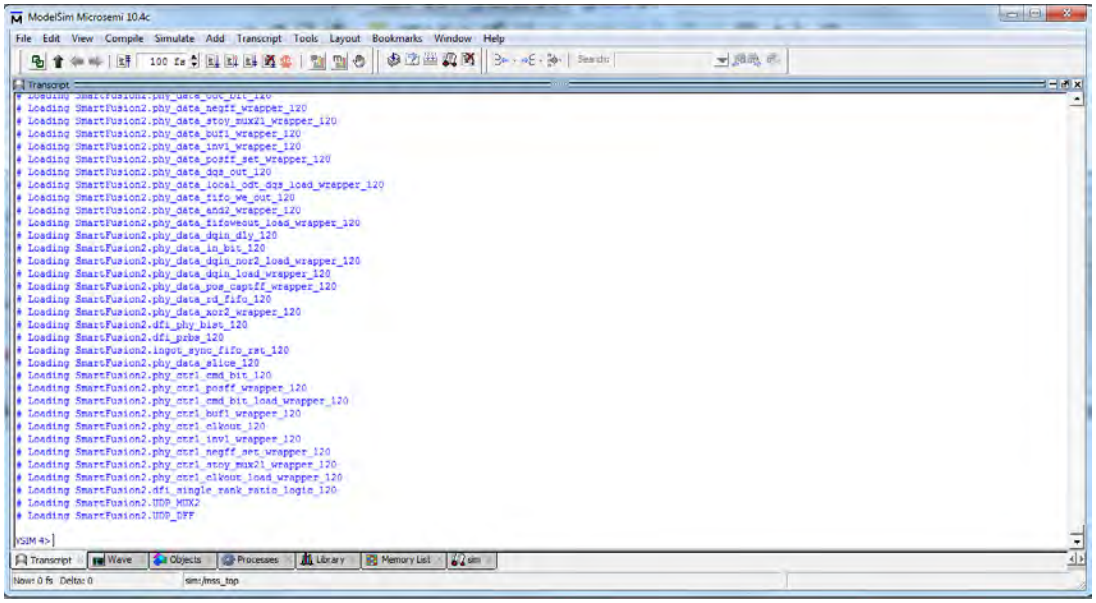
9. In the **Design Flow** window, expand **Verify Pre-synthesized Design** under **Create Design**, right-click **Simulate** and click **Open Interactively**. It simulates the MSS.

Figure 31 • Simulate the Pre-synthesized Design



10. Click **No** if an alert message is displayed to associate Testbench stimulus with MSS.
11. Close the **Modelsim** window after the simulation is complete.

Figure 32 • Simulation Window

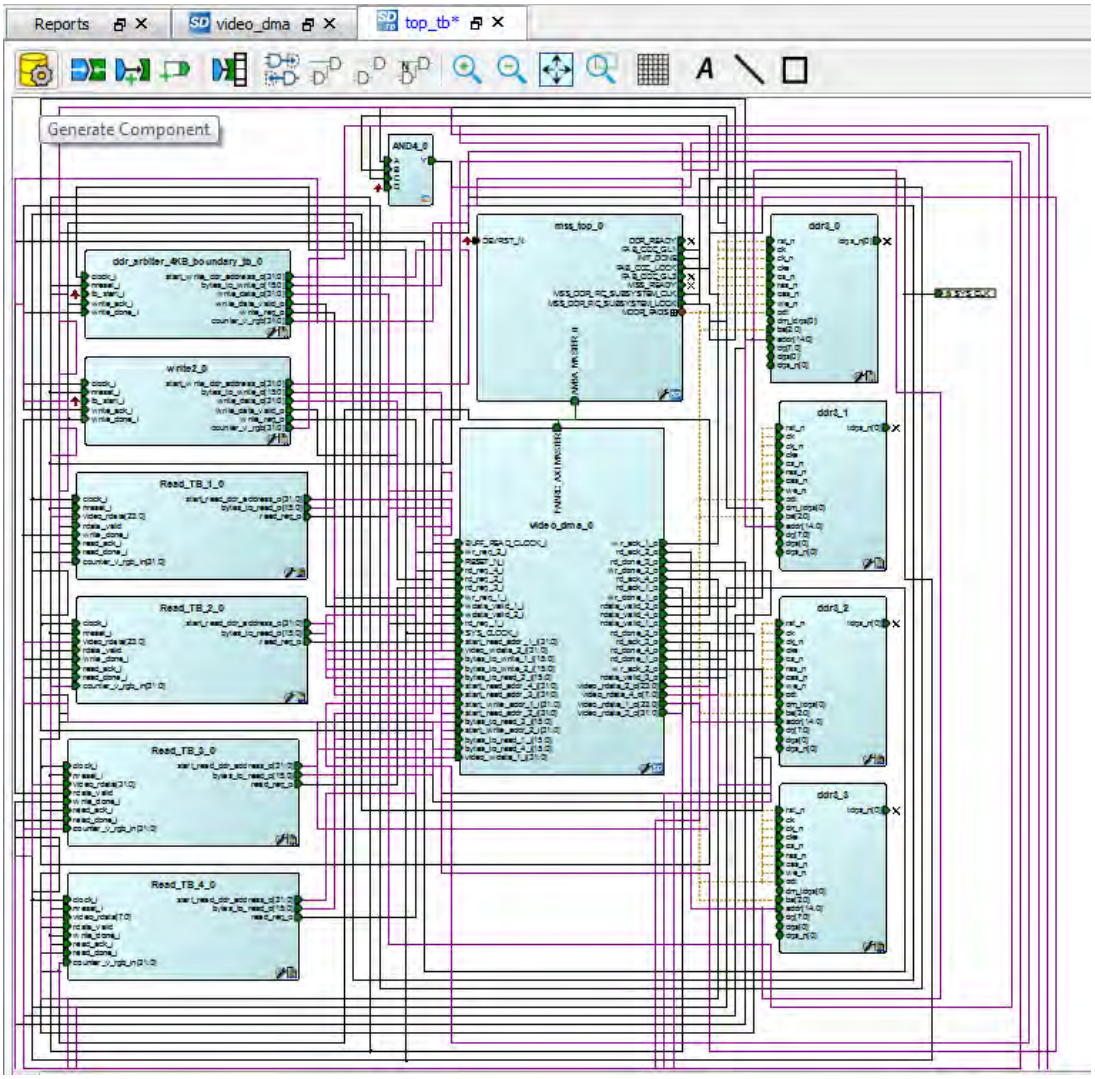


3.5.2 Simulating Testbench

The following instructions describe how to simulate testbench:

1. Select the **top_tb SmartDesign Testbench** and click **Generate Component** from the SmartDesign toolbar to generate the testbench, as shown in the following figure.

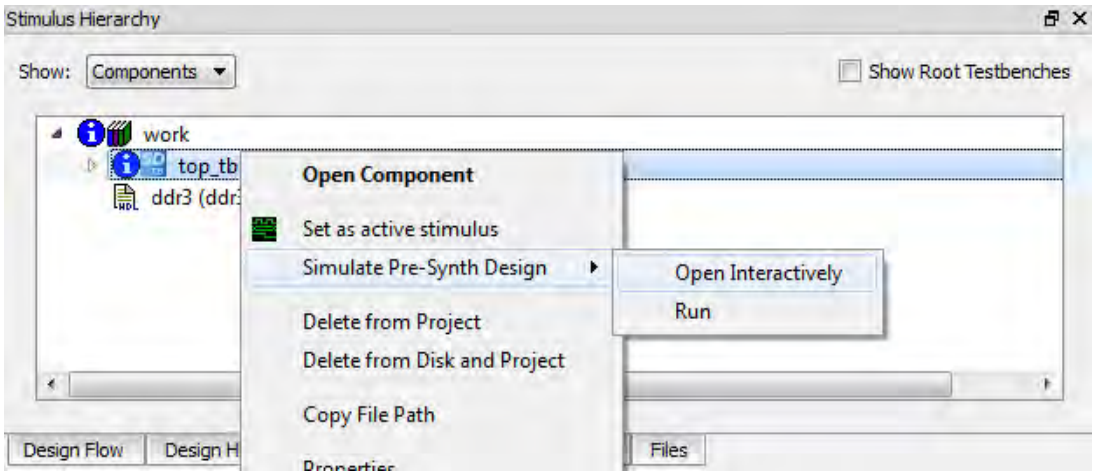
Figure 33 • Generating a Component



2. In the **Stimulus Hierarchy** window, right-click **top_tb** (`top_tb.v`) testbench file and click **Set as active stimulus**. The stimulus is activated for the top_tb testbench file.

3. In the **Stimulus Hierarchy** window, right-click **top_tb** (`top_tb.v`) testbench file and click **Open Interactively** from **Simulate Pre-Synth Design**. This simulates the core for one frame.

Figure 34 • Simulating Pre-Synthesis Design



4. If the simulation is interrupted because of the runtime limit in the DO file, use the run -all command to complete the simulation. After the simulation is completed, navigate to **View** > **Files** > **simulation** to view the test bench output image file in the simulation folder.

The output of the simulation – the text equivalent of one frame of the image, is stored in the `Read_out_rd_ch(x).txt` text file depending on the read channel used. This can be converted into an image and compared with the original image.

3.6 Resource Utilization

The DDR Arbiter block is implemented on an M2S150T SmartFusion®2 System-on-Chip (SoC) FPGA in the FC1152 package) and PolarFire FPGA (MPF300TS_ES - 1FCG1152E package).

Table 4 • Resource Utilization for DDR AXI Arbiter

Resource	Usage
DFFs	2992
4-input LUTs	4493
MACC	0
RAM1Kx18	20 (For: g_RD_CHANNEL(X)_HORIZONTAL_RESOLUTION = 1280 g_RD_CHANNEL(X)_BUFFER_LINE_STORAGE = 1 g_WR_CHANNEL(X)_BUFFER_LINE_STORAGE = 1 g_AXI_DWIDTH = 64 g_RD_CHANNEL(X)_VIDEO_DATA_WIDTH = 24 g_WR_CHANNEL(X)_VIDEO_DATA_WIDTH = 32)
RAM64x18	0

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

50200644