AXI BFM Cores v5.0

LogiCORE IP Product Guide

Vivado Design Suite

PG129 November 18, 2015

XILINX
ALL PROGRAMMABLE™

Table of Contents

Appendix A: Verification, Compliance, and Interoperability

Appendix B: Migrating and Upgrading

Appendix C: Debugging

Appendix D: Additional Resources and Legal Notices

Introduction

The Xilinx® LogiCORE™ IP AXI Bus Functional Model (BFM) cores, developed for Xilinx by Cadence® Design Systems, support the simulation of customer-designed AXI-based IP. AXI BFM cores support all versions of AXI (AXI3, AXI4, AXI4-Lite, and AXI4-Stream). The BFMs are encrypted Verilog modules. BFM operation is controlled by using a sequence of Verilog tasks contained in a Verilog-syntax text file.

Features

- Supports all protocol data widths and address widths, transfer types and responses
- Transaction-level protocol checking (burst type, length, size, lock type, cache type)
- Behavioral Verilog Syntax
- Verilog Task-based API

LogiCORE IP Facts Table	
Core Specifics	
Supported Device Family(1)	UltraScale+™ Families, UltraScale™ Architecture, Zynq®-7000 All Programmable SoC, 7 Series
Supported User Interfaces	AXI4, AXI4-Lite, AXI4-Stream, AXI3
Resources	N/A
Provided with Core	
Design Files	N/A
Example Design	Verilog
Test Bench	Verilog
Constraints File	N/A
Simulation Model	Encrypted Verilog
Supported S/W Driver	N/A
Tested Design Flows(2)(3)	
Design Entry	Vivado® Design Suite Vivado
Simulation	For supported simulators, see the Xilinx Design Tools: Release Notes Guide.
Synthesis	N/A
Support	
Provided by Xilinx at the Xilinx Support web page	

Notes:

1. For a complete list of supported derivative devices, see the Vivado IP catalog.
2. Windows XP 64-bit is not supported.
3. For the supported versions of the tools, see the Xilinx Design Tools: Release Notes Guide.
4. This IP does not deliver BFM for Zynq PS. It only delivers the BFM cores for AXI3, AXI4, AXI4-Lite, and AXI4-Stream interfaces.

Overview

Core Architecture

The general AXI BFM core architecture is shown in Figure 1-1.

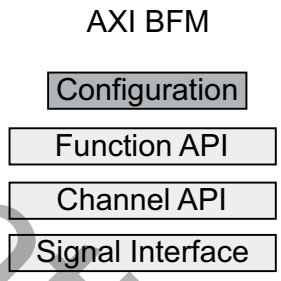

Figure 1-1: **Core Architecture**

All of the AXI BFM cores consist of three main layers:

- Signal interface
- Channel API
- Function API

The signal interface includes the typical Verilog input/output ports and associated signals. The channel API is a set of defined Verilog tasks (see Test Writing API) that operate at the basic transaction level inherent in the AXI protocol, including:

- Read Address Channel
- Write Address Channel
- Read Data Channel
- Write Data Channel
- Write Response Channel

This split enables the tasks associated with each channel to be executed concurrently or sequentially. This allows the test writer to control and implement out-of-order transfers, interleaved data transfers, and other features.

The next level up in the API hierarchy is the function level API (see Test Writing API). This level has complete transaction level control. For example, a complete AXI read burst process is encapsulated in a single Verilog task.

An important component of the AXI BFM core architecture is the configuration mechanism. This is implemented using Verilog parameters and/or BFM internal variables and is used to set the address bus width, data bus width, and other parameters. The reason Verilog parameters are used instead of defines is so that each BFM can be configured separately within a single test bench.

For example, it is reasonable to have an AXI master that has a different data bus width than one of the slaves it is attached to (in this case the interconnect needs to handle this). BFM internal variables are used for configuration variables that maybe changed during simulation. For a complete list of configuration options, see Configuration Options.

The intended use of the AXI BFM cores are shown in Figure 1-2.

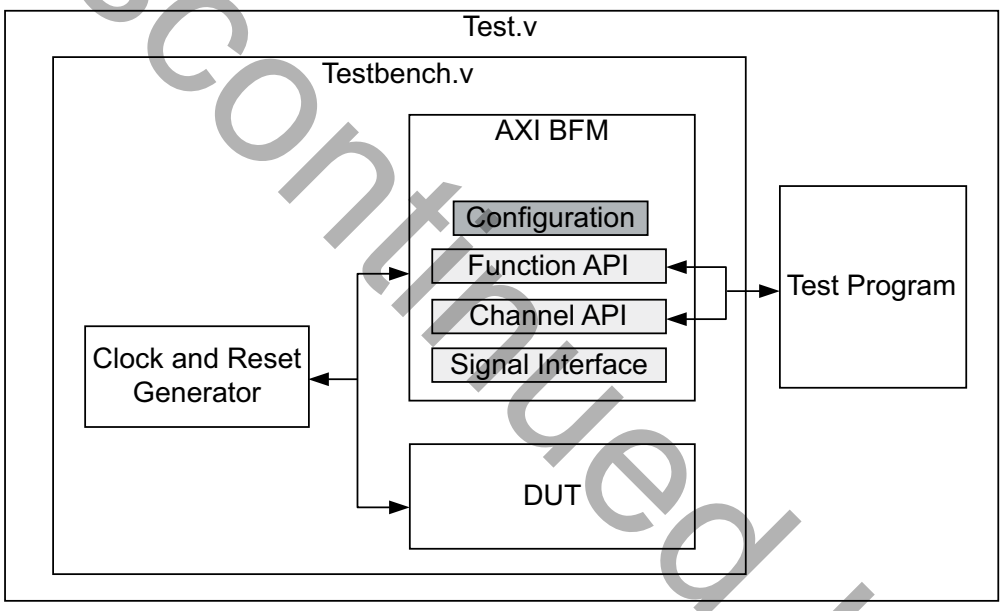

Figure 1-2: **AXI BFM Cores Use Case**

Figure 1-2 shows a single AXI BFM core. However, the test bench can contain multiple instances of AXI BFM core. The DUT and the AXI BFM core are instantiated in a test bench that also contains a clock and reset generator. Then, the test writer instantiates the test bench into the test module and creates a test program using the BFM API layers. The test program would call API tasks either sequentially or concurrently using fork and join. See Chapter 5, Example Design and Chapter 6, Test Bench for practical examples of test programs and test bench setups.

Configuration Options

In most cases, the configuration options are passed to the AXI BFM cores through Verilog parameters. AXI BFM core internal variables are used for options that can be dynamically controlled by the test writer because Verilog parameters do not support run time modifications.

To change the AXI BFM core internal variables during simulation, the correct BFM API task should be called. For example, to change the CHANNEL_LEVEL_INFO from 0 to 1, the `set_channel_level_info(1)` task call should be made. For more information on the API for changing internal variables, see Test Writing API.

Applications

The purpose of the AXI BFM cores are to verify connectivity and basic functionality of AXI masters and AXI slaves. A basic level of protocol checking is included with the AXI BFM cores. For comprehensive protocol checking, the Cadence AXI UVC [Ref 1] should be deployed.

The following aspects of the AXI3 and AXI4 protocol are checked by the AXI BFM cores:

- Reset conditions are checked:
  - Reset values of signals
  - Synchronous release of reset
- Inputs into the test writing API are checked to ensure they are valid to prevent protocol violations.
- Signal inputs into master and slave BFM, respectively, are checked to ensure they are valid to prevent protocol violations.
- Address ranges are checked in the Slave BFM.

This section describes the checkers that are implemented as Verilog tasks.

BFM Limitations

The purpose of this IP is to enable Xilinx customers to verify that their designs are able to communicate with Xilinx IP using AXI3 or AXI4 protocol. The complete verification of such interfaces with regards to protocol compliance is not within the scope of the AXI BFM cores; for compliance testing and complete verification of AXI interfaces then the Cadence AXI UVC should be deployed.

The BFM cores are implemented in behavioral Verilog-2001 and as such are limited to the constructs available for this version of Verilog. For that reason, some of the checking might seem limited compared with other VIP offerings that can leverage from assertion languages like SVA or PSL. Furthermore, there are no constructs inside Verilog-2001 to prevent or handle test bench race conditions. This means that the test writer must ensure that they are not causing race conditions by calling the Function Level API tasks within concurrent blocks (for example, inside a fork... join block). It is possible to use the concurrent blocks to create certain scenarios as illustrated in the example tests provided with this VIP.

The most common protocol violation caused by such race conditions is violation of the AXI3 write ordering rules: the first write transfer of each burst MUST be in the same order as the address phase transfers that is. the WID of the first transfer in each burst must be in the same order as the associated AWIDs. This is an incredibly difficult check to write in Verilog-2001 but a limited debug check is available to help detect and debug such a condition (see the function "`set_write_id_order_check_feature_value`" in Section 5 for more details on how to enable this).

Another limitation for BFM is that AXI Master BFM does not allow the same ID for outstanding transactions. Each outstanding transaction must be given an unique ID.

The BFM cores are encrypted using the Verilog P1735 IEEE standard. End users should note that while there are no import/export restrictions on this verification IP there maybe be a need for the correct simulator license feature for 256-bit AES decryption.

Licensing and Ordering Information

This Xilinx® LogiCORE™ IP module is provided under the terms of the Xilinx Core License Agreement. The module is shipped as part of the Vivado® Design Suite. For full access to all core functionalities in simulation, you must purchase a license for the core. Contact your local Xilinx sales representative for information on pricing and availability.

For more information, visit the AXI Bus Functional Model web page.

Information about other Xilinx LogiCORE IP modules is available at the Xilinx Intellectual Property page. For information on pricing and availability of other Xilinx LogiCORE IP modules and tools, contact your local Xilinx sales representative.

IMPORTANT: *When simulating AXI BFM cores, the license is checked out and held until the simulation is completed.*

Product Specification

Standards

The AXI BFM cores are AXI4, AXI4-Lite, AXI4-Stream, and AXI3 compliant.

Designing with the Core

This chapter includes guidelines and additional information to facilitate designing with the core.

AXI BFM Cores Design Parameters

Note: Run Time Parameters can also be changed during simulation from test bench using respective APIs.

AXI3 BFM

AXI3 Master BFM

Table 3-1 contains a list of parameters and configuration variables supported by the AXI3 Master BFM.

Table 3-1: **AXI3 Master BFM Parameters**

BFM Parameters	Description
Static Parameters	
NAME	String name for the master BFM. This is used in the messages coming from the BFM. The default for the master BFM is "MASTER_0."
DATA_BUS_WIDTH	Read and write data buses can be 32, 64, 128, 256, 512, or 1,024 bits wide. Default is 32.
ADDRESS_BUS_WIDTH	This parameter can take the values from 12 to 64. Default is 32.
ID_BUS_WIDTH	Default is 4.
MAX_OUTSTANDING_TRANSACTIONS	This defines the maximum number of outstanding transactions. Any attempt to generate more traffic while this limit has been reached is handled by stalling until at least one of the outstanding transactions has finished. Default is 8.

Table 3-1: **AXI3 Master BFM Parameters *(Cont'd)***

BFM Parameters	Description
EXCLUSIVE_ACCESS_SUPPORTED	This parameter informs the master that exclusive access is supported by the slave. A value of 1 means it is supported so the response check expects an EXOKAY, or else give a warning, in response to an exclusive access. A value of 0 means the slave does not support this so a response of OKAY is expected in response to an exclusive access. Default is 1.
Run Time Parameters	
WRITE_BURST_DATA_TRANSFER_GAP	The configuration variable can be set dynamically during the run of a test. It controls the gap between the write data transfers that comprise a write data burst. This value is an integer number and is measured in clock cycles. Default is 0. ***Note:*** If this is set to a value greater than zero *and* concurrent write bursts are called. Then write data interleaving occurs. The depth of this data interleaving depends on the number of parallel writes being performed. Care must be taken to ensure that write order protocol is not violated by the test writer.
RESPONSE_TIMEOUT	This value, measured in clock cycles, is the value used to determine if a task that is waiting for a response should timeout. Default is 500 clock cycles. A value of zero means that the timeout feature is disabled.
DISABLE_RESET_VALUE_CHECKS	This configuration value is used to enable/disable the checks for the reset values of input signals to the BFM. For example, the slave BFM checks at reset time if the signals from the master are at the expected reset values.
STOP_ON_ERROR	This configuration variable is used to enable/disable the stopping of the simulation on an error condition. The default (1) means stop on error. ***Note:*** This is not used for timeout errors; such errors always stop simulation.
CHANNEL_LEVEL_INFO	This configuration variable controls the printing of channel level information messages. When set to 1 info messages are printed, when set to zero no channel level information is printed. Default (0) means channel level info messages are disabled.
FUNCTION_LEVEL_INFO	This configuration variable controls the printing of function level information messages. When set to 1 info messages are printed, when set to zero no function level information is printed. Default (1) means function level info messages are enabled.
CLEAR_SIGNALS_AFTER_HANDSHAKE	This configuration value is used to enable/disable the setting of BFM output signals to reset values between transfers. Default is 0.
WRITE_ID_ORDER_CHECK_FEATURE	This configuration value can be used to disable the write ID ordering checks which might be required for error testing.
ERROR_ON_SLVERR	This configuration value is used to enable/disable errors on SLVERR responses to reads or writes. Default is 0, which means these are reported as warnings instead of errors.

Table 3-1: **AXI3 Master BFM Parameters *(Cont'd)***

BFM Parameters	Description
ERROR_ON_DECERR	This configuration value is used to enable/disable errors on SLVERR responses to reads or writes. Default is 0, which means these are reported as warnings instead of errors.
INPUT_SIGNAL_DELAY	This is used to move the BFM input signals off the simulation clock edge if needed. The default is 0.
TASK_RESET_HANDLING	0 = ignore reset and continue to process task (default) 1 = stall task execution until out of reset and print info message 2 = issue an error and stop (depending on STOP_ON_ERROR value) 3 = issue a warning and continue

AXI3 Slave BFM

Table 3-2 contains a list of parameters and configuration variables supported by the AXI3 Slave BFM:

Table 3-2: **AXI3 Slave BFM Parameters**

BFM Parameters	Description
Static Parameters	
NAME	String name for the slave BFM. This is used in the messages coming from the BFM. The default for the slave BFM is "SLAVE_0."
DATA_BUS_WIDTH	Read and write data buses can be 32, 64, 128, 256, 512, or 1,024 bits wide. Default is 32.
ADDRESS_BUS_WIDTH	Address parameter takes the values from 12 to 64. Default is 32.
ID_BUS_WIDTH	Slaves can have different ID bus widths compared to the master. The default is 4.
SLAVE_ADDRESS	This is the start address of the slave memory range.
SLAVE_MEM_SIZE	This is the size of the memory that the slave models. Starting from address = SLAVE_ADDRESS. This is measured in bytes therefore a value of 4,096 = 4 KB. The default value is 4 bytes, meaning, one 32-bit entry.
MAX_OUTSTANDING_TRANSACTIONS	This defines the maximum number of outstanding transactions. Any attempt to generate more traffic while this limit has been reached is handled by stalling until at least one of the outstanding transactions has finished. Default is 8.

Table 3-2: **AXI3 Slave BFM Parameters *(Cont'd)***

BFM Parameters	Description
MEMORY_MODEL_MODE	The parameter puts the slave BFM into a simple memory model mode. This means that the slave BFM automatically responds to all transfers and does not require any of the API functions to be called by the test. The memory mode is very simple and only supports aligned and normal INCR transfers. Narrow transfers are not supported, and WRAP and FIXED bursts are also not supported. The size and address range of the memory are controlled by the parameters SLAVE_ADDRESS and SLAVE_MEM_SIZE. The value 1 enables this memory model mode. A value of 0 disables it. Default is 0. The slave channel level API and function level API should not be used while this mode is active.
EXCLUSIVE_ACCESS_SUPPORTED	This parameter informs the slave that exclusive access is supported. A value of 1 means it is supported so the automatic generated response is an EXOKAY to exclusive accesses. A value of 0 means the slave does not support this so a response of OKAY is automatically generated in response to exclusive accesses. Default is 1.
	Run Time Parameters
READ_BURST_DATA_TRANSFER_GAP	The configuration variable controls the gap between the read data transfers that comprise a read data burst. This value is an integer number and is measured in clock cycles. Default is 0. ***Note:*** If this is set to a value greater than zero and concurrent read bursts are called, read data interleaving occurs. The depth of this data interleaving depends on the number of parallel writes being performed.
WRITE_RESPONSE_GAP	This configuration variable controls the gap, measured in clock cycles, between the reception of the last write transfer and the write response. Default is 0. ***Note:*** Care must be taken to ensure that write order protocol is not violated by the test writer.
READ_RESPONSE_GAP	This configuration variable controls the gap, measured in clock cycles, between the reception of the read address transfer and the start of the first read data transfer. Default is 0.
RESPONSE_TIMEOUT	This configuration variable, measured in clock cycles, is the value used to determine if a task that is waiting for a response should timeout. Default = 500 clock cycles. A value of zero means that the timeout feature is disabled. The value of this variable cannot be set when memory_model_mode is enabled.
DISABLE_RESET_VALUE_CHECKS	This configuration value is used to enable/disable the checks for the reset values of input signals to the BFM. For example, the slave BFM checks at reset time if the signals from the master are at the expected reset values.
WRITE_ID_ORDER_CHECK_FEATURE	This configuration value can be used to disable the write ID ordering checks which might be required for error testing.

Table 3-2: **AXI3 Slave BFM Parameters *(Cont'd)***

BFM Parameters	Description
STOP_ON_ERROR	This configuration variable is used to enable/disable the stopping of the simulation on an error condition. The default value of one stops the simulation on an error. ***Note:*** This is not used for timeout errors; such errors always stop simulation.
CHANNEL_LEVEL_INFO	This configuration variable controls the printing of channel level information messages. When set to 1 info messages are printed; when set to zero no channel level information is printed. The default (0) disables the channel level info messages.
FUNCTION_LEVEL_INFO	This configuration variable controls the printing of function level information messages. When set to 1 info messages are printed; when set to zero no function level information is printed. The default (1) enables the function level info messages.
CLEAR_SIGNALS_AFTER_HANDSHAKE	This configuration value is used to enable/disable the setting of BFM output signals to reset values between transfers. Default is 0.
INPUT_SIGNAL_DELAY	This is used to move the BFM input signals off the simulation clock edge if needed. The default is 0.
TASK_RESET_HANDLING	0 = ignore reset and continue to process task (default) 1 = stall task execution until out of reset and print info message 2 = issue an error and stop (depending on STOP_ON_ERROR value) 3 = issue a warning and continue

AXI4 BFM

AXI4 Master BFM

Table 3-3 contains a list of parameters and configuration variables supported by the AXI4 Master BFM.

Table 3-3: **AXI4 Master BFM Parameters**

BFM Parameters	Description
Static Parameters	
NAME	String name for the master BFM. This is used in the messages coming from the BFM. The default for the master BFM is "MASTER_0."
DATA_BUS_WIDTH	Read and write data buses can be 32, 64, 128, 256, 512, or 1,024 bits wide. Default is 32.
ADDRESS_BUS_WIDTH	Address width can vary from 12 to 64. Default is 32.
ID_BUS_WIDTH	Default is 4.
AWUSER_BUS_WIDTH	Default is 1.
ARUSER_BUS_WIDTH	Default is 1.
RUSER_BUS_WIDTH	Default is 1.

Table 3-3: **AXI4 Master BFM Parameters *(Cont'd)***

BFM Parameters	Description
WUSER_BUS_WIDTH	Default is 1.
BUSER_BUS_WIDTH	Default is 1.
MAX_OUTSTANDING_TRANSACTIONS	This defines the maximum number of outstanding transactions. Any attempt to generate more traffic while this limit has been reached is handled by stalling until at least one of the outstanding transactions has finished. Default is 8.
EXCLUSIVE_ACCESS_SUPPORTED	This parameter informs the master that exclusive access is supported by the slave. A value of 1 means it is supported so the response check expects an EXOKAY, or else give a warning, in response to an exclusive access. A value of 0 means the slave does not support this so a response of OKAY is expected in response to an exclusive access. Default is 1.
Run Time Parameters	
WRITE_BURST_DATA_TRANSFER_GAP	It controls the gap between the write data transfers that comprise a write data burst. This value is an integer number and is measured in clock cycles. Default is 0. ***Note:*** If this is set to a value greater than zero and concurrent read bursts are called, then the BFM attempts to perform read data interleaving.
WRITE_BURST_ADDRESS_DATA_PHASE_GAP	It controls the gap between the write address phase and the write data burst inside the WRITE_BURST task. This value is an integer number and is measured in clock cycles. Default is 0.
WRITE_BURST_DATA_ADDRESS_PHASE_GAP	It controls the gap between the write data burst and the write address phase inside the WRITE_BURST_CONCURRENT. This enables you to start the address phase at anytime during the data burst. This value is an integer number and is measured in clock cycles. Default is 0.
RESPONSE_TIMEOUT	This value, measured in clock cycles, is the value used to determine if a task that is waiting for a response should timeout. Default is 500 clock cycles. A value of zero means that the timeout feature is disabled.
DISABLE_RESET_VALUE_CHECKS	This configuration value is used to enable/disable the checks for the reset values of input signals to the BFM. For example, the slave BFM checks at reset time if the signals from the master are at the expected reset values.
STOP_ON_ERROR	This configuration variable is used to enable/disable the stopping of the simulation on an error condition. The default value of one stops the simulation on an error. ***Note:*** This is not used for timeout errors; such errors always stop simulation.
CHANNEL_LEVEL_INFO	This configuration variable controls the printing of channel level information messages. When set to 1 info messages are printed, when set to zero no channel level information is printed. The default (0) disables the channel level info messages.

Table 3-3: **AXI4 Master BFM Parameters *(Cont'd)***

BFM Parameters	Description
FUNCTION_LEVEL_INFO	This configuration variable controls the printing of function level information messages. When set to 1 info messages are printed, when set to zero no function level information is printed. The default (1) enables the function level info messages.
CLEAR_SIGNALS_AFTER_HANDSHAKE	This configuration value is used to enable/disable the setting of BFM output signals to reset values between transfers. Default is 0.
ERROR_ON_SLVERR	This configuration value is used to enable/disable errors on SLVERR responses to reads or writes. Default is 0, which means these are reported as warnings instead of errors.
ERROR_ON_DECERR	This configuration value is used to enable/disable errors on SLVERR responses to reads or writes. Default is 0, which means these are reported as warnings instead of errors.
INPUT_SIGNAL_DELAY	This is used to move the BFM input signals off the simulation clock edge if needed. The default is 0.
TASK_RESET_HANDLING	0 = ignore reset and continue to process task (default) 1 = stall task execution until out of reset and print info message 2 = issue an error and stop (depending on STOP_ON_ERROR value) 3 = issue a warning and continue

AXI4 Slave BFM

Table 3-4 contains a list of parameters and configuration variables supported by the AXI4 Slave BFM.

Table 3-4: **AXI4 Slave BFM Parameters**

BFM Parameters	Description
Static Parameters	
NAME	String name for the slave BFM. This is used in the messages coming from the BFM. The default for the slave BFM is "SLAVE_0."
DATA_BUS_WIDTH	Read and write data buses can be 32, 64, 128, 256, 512, or 1,024 bits wide. Default is 32.
ADDRESS_BUS_WIDTH	Address width can vary from 12 to 64. Default is 32.
ID_BUS_WIDTH	Slaves can have different ID bus widths compared to the master. Default is 4.
AWUSER_BUS_WIDTH	Default is 1.
ARUSER_BUS_WIDTH	Default is 1.
RUSER_BUS_WIDTH	Default is 1.
WUSER_BUS_WIDTH	Default is 1.

Table 3-4: **AXI4 Slave BFM Parameters *(Cont'd)***

BFM Parameters	Description
BUSER_BUS_WIDTH	Default is 1.
SLAVE_ADDRESS	This is the start address of the slave memory range.
SLAVE_MEM_SIZE	This is the size of the memory that the slave models. Starting from address = SLAVE_ADDRESS. This is measured in bytes therefore a value of 4,096 = 4 KB. The default value is 4 bytes (one 32-bit entry).
MAX_OUTSTANDING_TRANSACTIONS	This defines the maximum number of outstanding transactions. Any attempt to generate more traffic while this limit has been reached is handled by stalling until at least one of the outstanding transactions has finished. Default is 8.
MEMORY_MODEL_MODE	The parameter puts the slave BFM into a simple memory model mode. This means that the slave BFM automatically responds to all transfers and does not require any of the API functions to be called by the test. The memory mode is very simple and only supports, aligned and normal INCR transfers. Narrow transfers are not supported, and WRAP and FIXED bursts are also not supported. The size and address range of the memory are controlled by the parameters SLAVE_ADDRESS and SLAVE_MEM_SIZE. The value 1 enables this memory model mode. A value of 0 disables it. Default is 0. The slave channel level API and function level API should not be used while this mode is active.
Run Time Parameters	
EXCLUSIVE_ACCESS_SUPPORTED	This parameter informs the slave that exclusive access is supported. A value of 1 means it is supported so the automatic generated response is an EXOKAY to exclusive accesses. A value of 0 means the slave does not support this so a response of OKAY is automatically generated in response to exclusive accesses. Default is 1.
READ_BURST_DATA_TRANSFER_GAP	The configuration variable controls the gap between the read data transfers that comprise a read data burst. This value is an integer number and is measured in clock cycles. Default is 0. ***Note:*** If this is set to a value greater than zero *and* concurrent read bursts are called, then AXI4 protocol is violated as the BFM attempts to perform data interleaving.
WRITE_RESPONSE_GAP	This configuration variable controls the gap, measured in clock cycles, between the reception of the last write transfer and the write response. Default is 0.
READ_RESPONSE_GAP	This configuration variable controls the gap, measured in clock cycles, between the reception of the read address transfer and the start of the first read data transfer. Default is 0.

Table 3-4: **AXI4 Slave BFM Parameters *(Cont'd)***

BFM Parameters	Description
RESPONSE_TIMEOUT	This configuration variable, measured in clock cycles, is the value used to determine if a task that is waiting for a response should timeout. Default = 500 clock cycles. A value of zero means that the timeout feature is disabled. The value of this variable cannot be set when memory_model_mode is enabled.
DISABLE_RESET_VALUE_CHECKS	This configuration value is used to enable/disable the checks for the reset values of input signals to the BFM. For example, the slave BFM checks at reset time if the signals from the master are at the expected reset values.
STOP_ON_ERROR	This configuration variable is used to enable/disable the stopping of the simulation on an error condition. The default value of 1 stops the simulation on an error. ***Note:*** This is not used for timeout errors; such errors always stop simulation.
CHANNEL_LEVEL_INFO	This configuration variable controls the printing of channel level information messages. When set to 1 info messages are printed; when set to zero no channel level information is printed. The default (0) disables the channel level info messages.
FUNCTION_LEVEL_INFO	This configuration variable controls the printing of function level information messages. When set to 1 info messages are printed; when set to zero no function level information is printed. The default (1) enables the function level info messages.
CLEAR_SIGNALS_AFTER_HANDSHAKE	This configuration value is used to enable/disable the setting of BFM output signals to reset values between transfers. Default is 0.
INPUT_SIGNAL_DELAY	This is used to move the BFM input signals off the simulation clock edge if needed. The default is 0.
TASK_RESET_HANDLING	0 = ignore reset and continue to process task (default) 1 = stall task execution until out of reset and print info message 2 = issue an error and stop (depending on STOP_ON_ERROR value) 3 = issue a warning and continue

AXI4-Lite Master BFM

Table 3-5 contains a list of parameters and configuration variables supported by the AXI4-Lite Master BFM.

Table 3-5: **AXI4-Lite Master BFM Parameters**

BFM Parameters	Description
Static Parameters	
NAME	String name for the master BFM. This is used in the messages coming from the BFM. The default for the master BFM is "MASTER_0."
DATA_BUS_WIDTH	Read and write data buses can 32 or 64 bits wide only. Default is 32.
ADDRESS_BUS_WIDTH	Address width can vary from 1 to 64. Default is 32.

Table 3-5: **AXI4-Lite Master BFM Parameters *(Cont'd)***

BFM Parameters	Description
MAX_OUTSTANDING_TRANSACTIONS	This defines the maximum number of outstanding transactions. Any attempt to generate more traffic while this limit has been reached is handled by stalling until at least one of the outstanding transactions has finished. Default is 8.
Run Time Parameters	
RESPONSE_TIMEOUT	This value, measured in clock cycles, is the value used to determine if a task that is waiting for a response should timeout. Default is 500 clock cycles. A value of zero means that the timeout feature is disabled.
DISABLE_RESET_VALUE_CHECKS	This configuration value is used to enable/disable the checks for the reset values of input signals to the BFM. For example, the slave BFM checks at reset time if the signals from the master are at the expected reset values.
STOP_ON_ERROR	This configuration variable is used to enable/disable the stopping of the simulation on an error condition. The default value of one stops the simulation on an error. ***Note:*** This is not used for timeout errors; such errors always stop simulation.
CHANNEL_LEVEL_INFO	This configuration variable controls the printing of channel level information messages. When set to 1 info messages are printed, when set to zero no channel level information is printed. The default (0) disables the channel level info messages.
FUNCTION_LEVEL_INFO	This configuration variable controls the printing of function level information messages. When set to 1 info messages are printed, when set to zero no function level information is printed. The default (1) enables the function level info messages.
CLEAR_SIGNALS_AFTER_HANDSHAKE	This configuration value is used to enable/disable the setting of BFM output signals to reset values between transfers. Default is 0.
ERROR_ON_SLVERR	This configuration value is used to enable/disable errors on SLVERR responses to reads or writes. Default is 0, which means these are reported as warnings instead of errors.
ERROR_ON_DECERR	This configuration value is used to enable/disable errors on SLVERR responses to reads or writes. Default is 0, which means these are reported as warnings instead of errors.
INPUT_SIGNAL_DELAY	This is used to move the BFM input signals off the simulation clock edge if needed. The default is 0.
TASK_RESET_HANDLING	0 = ignore reset and continue to process task (default) 1 = stall task execution until out of reset and print info message 2 = issue an error and stop (depending on STOP_ON_ERROR value) 3 = issue a warning and continue

AXI4-Lite Slave BFM

Table 3-6 contains a list of parameters and configuration variables supported by the AXI4-Lite Slave BFM.

Table 3-6: **AXI4-Lite Slave BFM Parameters**

BFM Parameters	Description
Static Parameters	
NAME	String name for the slave BFM. This is used in the messages coming from the BFM. The default for the slave BFM is "SLAVE_0."
DATA_BUS_WIDTH	Read and write data buses can be 32 or 64 bits wide only. Default is 32.
ADDRESS_BUS_WIDTH	Address width can vary from 1 to 64. Default is 32.
SLAVE_ADDRESS	This is the start address of the slave memory range.
SLAVE_MEM_SIZE	This is the size of the memory that the slave models. Starting from address = SLAVE_ADDRESS. This is measured in bytes therefore a value of 4,096 = 4 KB. The default value is 4 bytes, that is, one 32-bit entry.
MAX_OUTSTANDING_TRANSACTIONS	This defines the maximum number of outstanding transactions. Any attempt to generate more traffic while this limit has been reached is handled by stalling until at least one of the outstanding transactions has finished. Default is 8.
MEMORY_MODEL_MODE	The parameter puts the slave BFM into a simple memory model mode. This means that the slave BFM automatically responds to all transfers and does not require any of the API functions to be called by the test. The memory mode is very simple and only supports, aligned and normal INCR transfers. Narrow transfers are not supported, and WRAP and FIXED bursts are also not supported. The size and address range of the memory are controlled by the parameters SLAVE_ADDRESS and SLAVE_MEM_SIZE. The value 1 enables this memory model mode. A value of 0 disables it. Default is 0. The slave channel level API and function level API should not be used while this mode is active.
Run Time Parameters	
WRITE_RESPONSE_GAP	This configuration variable controls the gap, measured in clock cycles, between the reception of the last write transfer and the write response. Default is 0. The value of this variable cannot be set when memory_model_mode is enabled.
READ_RESPONSE_GAP	This configuration variable controls the gap, measured in clock cycles, between the reception of the read address transfer and the start of the first read data transfer. Default is 0.

Table 3-6: **AXI4-Lite Slave BFM Parameters *(Cont'd)***

BFM Parameters	Description
RESPONSE_TIMEOUT	This configuration variable, measured in clock cycles, is the value used to determine if a task that is waiting for a response should timeout. Default = 500 clock cycles. A value of zero means that the timeout feature is disabled.
DISABLE_RESET_VALUE_CHECKS	This configuration value is used to enable/disable the checks for the reset values of input signals to the BFM. For example, the slave BFM checks at reset time if the signals from the master are at the expected reset values.
STOP_ON_ERROR	This configuration variable is used to enable/disable the stopping of the simulation on an error condition. The default value of one stops the simulation on an error. ***Note:*** This is not used for timeout errors; such errors always stop simulation.
CHANNEL_LEVEL_INFO	This configuration variable controls the printing of channel level information messages. When set to 1 info messages are printed, when set to zero no channel level information is printed. The default (0) disables the channel level info messages.
FUNCTION_LEVEL_INFO	This configuration variable controls the printing of function level information messages. When set to 1 info messages are printed, when set to zero no function level information is printed. The default (1) enables the function level info messages.
CLEAR_SIGNALS_AFTER_HANDSHAKE	This configuration value is used to enable/disable the setting of BFM output signals to reset values between transfers. Default is 0.
INPUT_SIGNAL_DELAY	This is used to move the BFM input signals off the simulation clock edge if needed. The default is 0.
TASK_RESET_HANDLING	0 = ignore reset and continue to process task (default) 1 = stall task execution until out of reset and print info message 2 = issue an error and stop (depending on STOP_ON_ERROR value) 3 = issue a warning and continue

AXI4-Stream Master BFM

Table 3-7 contains a list of parameters and configuration variables supported by the AXI4-Stream Master BFM.

Table 3-7: **AXI4-Stream BFM Parameters**

BFM Parameters	Description
Static Parameters	
NAME	String name for the master BFM. This is used in the messages coming from the BFM. The default for the master BFM is "MASTER_0."
DATA_BUS_WIDTH	Read and write data buses can be 8 to 1,024, in multiples of 8 bits wide. Default is 32.
ID_BUS_WIDTH	Default is 8.

Table 3-7: **AXI4-Stream BFM Parameters *(Cont'd)***

BFM Parameters	Description
DEST_BUS_WIDTH	Default is 4.
USER_BUS_WIDTH	Default is 8.
MAX_PACKET_SIZE	This parameter is an integer value that controls the maximum size of a packet. It is used to size the packet data vector. The value must be specified as an integer multiple of the DATA_BUS_WIDTH. For example, if DATA_BUS_WIDTH = 32 bits and MAX_PACKET_SIZE = 2, then the maximum packet size is 64 bits. The default value is 10.
MAX_OUTSTANDING_TRANSACTIONS	This defines the maximum number of outstanding transactions. Any attempt to generate more traffic while this limit has been reached is handled by stalling until at least one of the outstanding transactions has finished. Default is 8.
STROBE_NOT_USED	Enables and disables the strobe signal check. 0 = Strobe signals used 1 = Strobe signals not used The default is 0. A value of 1 disables the associated checks.
KEEP_NOT_USED	Enables and disables the keep signal checks. 0 = Keep signals used 1 = Keep signals not used The default is 0. Changing the value to 1 disables the associated checks.
Run Time Parameters	
PACKET_TRANSFER_GAP	The configuration variable controls the gap between the transfers in a packet. This value is an integer number and is measured in clock cycles. The default is 0. ***Note:*** If this is set to a value greater than zero and concurrent SEND_PACKET tasks are called, then the BFM attempts to perform write data interleaving.
RESPONSE_TIMEOUT	This value, measured in clock cycles, is the value used to determine if a task that is waiting for a response should timeout. Default is 500 clock cycles. A value of zero means that the timeout feature is disabled.
STOP_ON_ERROR	This configuration variable is used to enable/disable the stopping of the simulation on an error condition. The default value of 1 stops the simulation on an error. ***Note:*** This is not used for timeout errors; such errors always stop simulation.
CHANNEL_LEVEL_INFO	This configuration variable controls the printing of channel level information messages. When set to 1, info messages are printed, when set to zero no channel level information is printed. The default (1) enables channel level info messages.
CLEAR_SIGNALS_AFTER_HANDSHAKE	This configuration value is used to enable/disable the setting of BFM output signals to reset values between transfers. Default is 0.

Table 3-7: **AXI4-Stream BFM Parameters *(Cont'd)***

BFM Parameters	Description
INPUT_SIGNAL_DELAY	This is used to move the BFM input signals off the simulation clock edge if needed. The default is 0.
TASK_RESET_HANDLING	0 = ignore reset and continue to process task (default) 1 = stall task execution until out of reset and print info message 2 = issue an error and stop (depending on STOP_ON_ERROR value) 3 = issue a warning and continue

AXI4-Stream Slave BFM

Table 3-8 contains a list of parameters and configuration variables supported by the AXI4-Stream Slave BFM.

Table 3-8: **AXI4-Stream Slave BFM Parameters**

BFM Parameters	Description
Static Parameters	
NAME	String name for the slave BFM. This is used in the messages coming from the BFM. The default for the slave BFM is "SLAVE_0."
DATA_BUS_WIDTH	Read and write data buses can be 8 to 1,024, in multiples of 8 bits wide. Default is 32.
ID_BUS_WIDTH	Default is 8.
DEST_BUS_WIDTH	Default is 4.
USER_BUS_WIDTH	Default is 8.
MAX_PACKET_SIZE	This parameter is an integer value that controls the maximum size of a packet. It is used to size the packet data vector. The value must be specified as an integer multiple of the DATA_BUS_WIDTH. For example, if DATA_BUS_WIDTH = 32 bits and MAX_PACKET_SIZE = 2, then the maximum packet size is 64 bits. The default value is 10.
MAX_OUTSTANDING_TRANSACTIONS	This defines the maximum number of outstanding transactions. Any attempt to generate more traffic while this limit has been reached is handled by stalling until at least one of the outstanding transactions has finished. Default is 8.
STROBE_NOT_USED	Enables and disables the strobe signal check. 0 = Strobe signals used 1 = Strobe signals not used The default is 0. A value of 1 only disables the associated checks.
KEEP_NOT_USED	Enables and disables the keep signal checks. 0 = Keep signals used 1 = Keep signals not used The default is 0. Changing the value to 1 only disables the associated checks.

Table 3-8: **AXI4-Stream Slave BFM Parameters *(Cont'd)***

BFM Parameters	Description
Run Time Parameters	
RESPONSE_TIMEOUT	This configuration variable, measured in clock cycles, is the value used to determine if a task that is waiting for a response should timeout. Default = 500 clock cycles. A value of zero means that the timeout feature is disabled.
DISABLE_RESET_VALUE_CHECKS	This configuration value is used to enable/disable the checks for the reset values of input signals to the BFM. For example, the slave BFM checks at reset time if the signals from the master are at the expected reset values.
STOP_ON_ERROR	This configuration variable is used to enable/disable the stopping of the simulation on an error condition. The default value of 1 stops the simulation on an error. ***Note:*** This is not used for timeout errors; such errors always stop simulation.
CHANNEL_LEVEL_INFO	This configuration variable controls the printing of channel level information messages. When set to 1, info messages are printed, when set to zero no channel level information is printed. The default (1) enables the channel level info messages.
INPUT_SIGNAL_DELAY	This is used to move the BFM input signals off the simulation clock edge if needed. The default is 0.
TASK_RESET_HANDLING	0 = ignore reset and continue to process task (default) 1 = stall task execution until out of reset and print info message 2 = issue an error and stop (depending on STOP_ON_ERROR value) 3 = issue a warning and continue

Test Writing API

The test writing API is layered to allow you to implement more complex protocol features. This approach enables very complex test cases to be written. For a complete overview of the general AXI BFM core architecture, see Chapter 1, Overview.

For all functions in the API, the input and output values used for burst length and burst size are encoded as specified in the *AMBA® AXI Specifications* [Ref 7]. For example, LEN = 0 as an input means a burst of length 1.

Tasks and functions common to all BFM are described in Table 3-9.

Argument Data Types to APIs

Input arguments for AXI Master Function/Channel APIs and AXI Slave Function APIs use integer data type for AXI protocol transaction arguments (for example, BURST_TYPE, PROT TYPE, etc.) or integer variables (only if it is a not vector input like WUSER). If a vector is used for an input argument, a reg data type is required.

AXI Slave Channel API inputs typically use inputs that are AXI Protocol transaction arguments which use the data type reg signals (for example, AXI protocol BURST_TYPE, PROT TYPE, etc.) that are passed between APIs.

Output arguments from APIs use the reg data type. For further clarification, see Chapter 6, Test Bench delivered with the core.

Utility API Tasks/Functions

Table 3-9: **Utility API Tasks/Functions**

API Task Name and Description	Inputs	Outputs
report_status This function can be called at the end of a test to report the final status of the associated BFM.	dummy_bit: The value of this input can be 1 or 0 and does not matter. It is only required because a Verilog function needs at least 1 input.	report_status: This is an integer number which is calculated as: report_status = error_count + warning_count + pending_transactions_count
report_config This task prints out the current configuration as set by the configuration parameters and variables. This task can be called at any time.	None	None
set_channel_level_info This function sets the CHANNEL_LEVEL_INFO internal control variable to the specified input value.	LEVEL: A bit input for the info level.	None
set_function_level_info This function sets the FUNCTION_LEVEL_INFO internal control variable to the specified input value.	LEVEL: A bit input for the info level.	None
set_response_timeout This task sets the RESPONSE_TIMEOUT internal control variable to the specified input value.	Number of clock cycles for timeout. A value of zero means that the timeout feature is disabled	None
set_stop_on_error This function sets the STOP_ON_ERROR internal control variable to the specified input value.	LEVEL: A bit input for the info level.	None
set_read_burst_data_transfer_gap This function sets the SLAVE READ_BURST_DATA_TRANSFER_GAP internal control variable to the specified input value.	TIMEOUT: An integer value measured in clock cycles.	None
set_write_response_gap This function sets the SLAVE WRITE_RESPONSE_GAP internal control variable to the specified input value.	TIMEOUT: An integer value measured in clock cycles.	None

Table 3-9: **Utility API Tasks/Functions *(Cont'd)***

API Task Name and Description	Inputs	Outputs
set_read_response_gap This function sets the SLAVE READ_RESPONSE_GAP internal control variable to the specified input value.	TIMEOUT: An integer value measured in clock cycles.	None
set_write_burst_data_transfer_gap This function sets the MASTER WRITE_BURST_DATA_TRANSFER_GAP internal control variable to the specified input value.	TIMEOUT: An integer value measured in clock cycles.	None
set_wrtie_burst_address_data_phase_gap This function sets the AXI4 FULL MASTER WRITE_BURST_ADDRESS_DATA_PHASE_GAP internal control variable to the specified input value.	GAP_LENGTH: An integer value measured in clock cycles.	None
set_write_burst_data_address_phase_gap This function sets the AXI4 FULL MASTER WRITE_BURST_DATA_ADDRESS_PHASE_GAP internal control variable to the specified input value.	GAP_LENGTH: An integer value measured in clock cycles.	None
set_packet_transfer_gap This function sets the AXI4 Streaming MASTER PACKET_TRANSFER_GAP internal control variable to the specified input value.	GAP_LENGTH: An integer value measured in clock cycles.	None
set_task_call_and_reset_handling This task sets the TASK_RESET_HANDLING internal variable to the specified input value: 0x0 = Ignore reset and continue to process task (default) 0x1 = Stall task execution until out of reset and print info message 0x2 = Issue an error and stop (depending on STOP_ON_ERROR value) 0x3 = Issue a warning and continue	task_reset_handling: An integer value used to define BFM behavior during reset when a channel level API task is called.	None
remove_pending_transaction This task is only required if the test writer is using the channel level API task RECEIVE_READ_DATA instead of RECEIVE_READ_BURST. The RECEIVE_READ_DATA does not decrease the pending transaction counter so this task must be called manually after the full read data transfer is complete.	None	None

Table 3-9: **Utility API Tasks/Functions *(Cont'd)***

API Task Name and Description	Inputs	Outputs
set_input_signal_delay This task sets the internal variable INPUT_SIGNAL_DELAY to the specified input value. This is used to move the BFM input signals off the simulation clock edge if needed. The default value is zero. If used, it must be applied to each BFM separately.	INPUT_DELAY: An integer value used for the #INPUT_SIGNAL_DELAY on BFM input signals.	None
set_write_id_order_check_feature_value This task sets the WRITE_ID_ORDER_CHECK_FEATURE _CHECKS internal variable to the specified input value: 0 = disabled 1 = enabled These checks are for the AXI 3 write ID ordering rules and are mainly to help detect and debug any test issues. For example, using fork...join to call any of the write_burst master API tasks can cause race conditions. Such conditions get handled differently from simulator to simulator as the Verilog event queue is implemented differently by each vendor. For that reason these checks are not a full solution but a guide and debug tool only.	value: A simple bit value to enable/ disable reset value checks.	None
set_disable_reset_value_checks This task sets the DISABLE_RESET_VALUE_CHECKS internal variable to the specified input value: 0 = enabled 1 = disabled These checks are for the reset values of input signals to the BFM. For example, the slave BFM checks at reset if the signals from the master are at the expected reset values.	disable_value: A simple bit value to enable/ disable reset value checks.	None
set_clear_signals_after_handshake This task sets the CLEAR_SIGNALS_AFTER_HANDSHAKE internal variable to the specified input value: 0 = disabled 1 = enabled When disabled the last driven value is left on the output BFM signal until a new value is transferred.	A simple bit value to enable/ disable driving signals to reset values between transfers.	None

Table 3-9: **Utility API Tasks/Functions *(Cont'd)***

API Task Name and Description	Inputs	Outputs
set_error_on_slverr This task sets the ERROR_ON_SLVERR internal variable to the specified input value: 0 = warning reported on slverr 1 = error reported on slverr	A simple bit value to enable/ disable errors on slverr responses.	None
set_error_on_decerr This task sets the ERROR_ON_DECERR internal ariable to the specified input value: 0 = warning reported on decerr 1 = error reported on decerr	A simple bit value to enable/ disable errors on decerr responses.	None

API Instantiation Example

Table 3-9 lists out different APIs supported by `cdn_axi_bfm` IP. These APIs are called from the test bench to change the values of internal variables associated with them during simulation run time. The syntax and an example to demonstrate how to use these APIs in the test bench are given here:

Syntax to use an API:

```
<Instance name>.<api_name>(Input value to the API);
```

Example to use an API:

```
tb.master_0.cdn_axi4_master_bfm_inst.set_write_burst_data_transfer_gap(0);
```

AXI3 Master BFM Test Writing API

The channel level API for the AXI3 Master BFM is detailed in Table 3-10.

Table 3-10: **Channel Level API for AXI3 Master BFM**

API Task Name and Description	Inputs	Outputs
SEND_WRITE_ADDRESS Creates a write address channel transaction. This task returns after the write address has been acknowledged by the slave.	ID: Write Address ID tag ADDR: Write Address LEN: Burst Length SIZE: Burst Size BURST: Burst Type LOCK: Lock Type CACHE: Cache Type PROT: Protection Type	None
SEND_WRITE_DATA Creates a single write data channel transaction. The ID tag should be the same as the write address ID tag it is associated with. The data should be the same size as the width of the data bus. This task returns after is has been acknowledged by the slave. The data input is used as raw bus data, that is, no realignment for narrow or unaligned data. Should be called multiple times for a burst with correct control of the LAST flag.	ID: Write ID tag STOBE: Strobe signals DATA: Data for transfer LAST: Last transfer flag	None
SEND_READ_ADDRESS Creates a read address channel transaction. This task returns after the read address has been acknowledged by the slave.	ID: Read Address ID tag ADDR: Read Address LEN: Burst Length SIZE: Burst Size BURST: Burst Type LOCK: Lock Type CACHE: Cache Type PROT: Protection Type	None
RECEIVE_READ_DATA This task drives the RREADY signal and monitors the read data bus for read transfers coming from the slave that have the specified ID tag. It then returns the data associated with the transaction and the status of the last flag. The data output here is raw bus data, that is, no realignment for narrow or unaligned data. This would need to be called multiple times for a burst > 1. Also, you must call the “remove_pending_transaction” task when all data is received to ensure that the pending transaction counter is decremented. This is done automatically by the RECEIVE_READ_BURST and RECEIVE_WRITE_RESPONSE channel level API tasks.	ID: Read ID tag	DATA: Data transferred by the slave RESPONSE: The slave read response from the following: [OKAY, EXOKAY, SLVERR, DECERR] LAST: Last transfer flag
RECEIVE_WRITE_RESPONSE This task drives the BREADY signal and monitors the write response bus for write responses coming from the slave that have the specified ID tag. It then returns the response associated with the transaction.	ID: Write ID tag	RESPONSE: The slave write response from the following: [OKAY, EXOKAY, SLVERR, DECERR]

Table 3-10: **Channel Level API for AXI3 Master BFM *(Cont'd)***

API Task Name and Description	Inputs	Outputs
RECEIVE_READ_BURST This task receives a read channel burst based on the ID input. The RECEIVE_READ_DATA from the channel level API is used. This task returns when the read transaction is complete. The data returned by the task is the valid only data, that is, re-aligned data. This task also checks each response and issues a warning if it is not as expected.	ID: Read ID tag ADDR: Read Address LEN: Burst Length SIZE: Burst Size BURST: Burst Type LOCK: Lock Type	DATA: Valid Data transferred by the slave RESPONSE: This is a vector that is created by concatenating all slave read responses together
SEND_WRITE_BURST This task does a write burst on the write data lines. It does not execute the write address transfer. This task uses the SEND_WRITE_DATA task from the channel level API. This task returns when the complete write burst is complete. This task automatically supports the generation of narrow transfers and unaligned transfers; that is, this task aligns the input data with the burst so data padding is not required.	ID: Write ID tag ADDR: Write Address LEN: Burst Length SIZE: Burst Size BURST: Burst Type DATA: Data to send DATASIZE: The size in bytes of the valid data contained in the input data vector	None

The function level API for the AXI3 Master BFM is detailed in Table 3-11.

Table 3-11: **Function Level API for AXI3 Master BFM**

API Task Name and Description	Inputs	Outputs
READ_BURST This task does a full read process. It is composed of the tasks SEND_READ_ADDRESS and RECEIVE_READ_BURST from the channel level API. This task returns when the read transaction is complete.	ID: Read ID tag ADDR: Read Address LEN: Burst Length SIZE: Burst Size BURST: Burst Type LOCK: Lock Type CACHE: Cache Type PROT: Protection Type	DATA: Valid data transferred by the slave RESPONSE: This is a vector that is created by concatenating all slave read responses together
WRITE_BURST This task does a full write process. It is composed of the tasks SEND_WRITE_ADDRESS, SEND_WRITE_BURST and RECEIVE_WRITE_RESPONSE from the channel level API. This task returns when the complete write transaction is complete. This task automatically supports the generation of narrow transfers and unaligned transfers. This API should be used in scenario wherein before the slave asserts AWREADY and/or WREADY the slave can wait for AWVALID.	ID: Write ID tag ADDR: Write Address LEN: Burst Length SIZE: Burst Size BURST: Burst Type LOCK: Lock Type CACHE: Cache Type PROT: Protection Type DATA: Data to send DATASIZE: The size in bytes of the valid data contained in the input data vector	RESPONSE: The slave write response from the following: [OKAY, EXOKAY, SLVERR, DECERR]

Table 3-11: **Function Level API for AXI3 Master BFM *(Cont'd)***

API Task Name and Description	Inputs	Outputs
WRITE_BURST_CONCURRENT This task does the same function as the WRITE_BURST task; however, it performs the write address and write data phases concurrently.	ID: Write ID tag ADDR: Write Address LEN: Burst Length SIZE: Burst Size BURST: Burst Type LOCK: Lock Type CACHE: Cache Type PROT: Protection Type DATA: Data to send DATASIZE: The size in bytes of the valid data contained in the input data vector	RESPONSE: The slave write response from the following: [OKAY, EXOKAY, SLVERR, DECERR]
WRITE_BURST_DATA_FIRST This task does the same function as the WRITE_BURST task; however, it sends the write data burst before sending the associated write address transfer on the write address channel. This is used in scenario wherein before the slave asserts AWREADY and/or WREADY the slave can wait for WVALID.	ID: Write ID tag ADDR: Write Address LEN: Burst Length SIZE: Burst Size BURST: Burst Type LOCK: Lock Type CACHE: Cache Type PROT: Protection Type DATA: Data to send DATASIZE: The size in bytes of the valid data contained in the input data vector	RESPONSE: The slave write response from the following: [OKAY, EXOKAY, SLVERR, DECERR]

AXI3 Slave BFM Test Writing API

The channel level API for the AXI3 Slave BFM is detailed in Table 3-12.

Table 3-12: **Channel Level API for AXI3 Slave BFM**

API Task Name and Description	Inputs	Outputs
SEND_WRITE_RESPONSE Creates a write response channel transaction. The ID tag must match the associated write transaction. This task returns after it has been acknowledged by the master.	ID: Write ID tag RESPONSE: The chosen write response from the following [OKAY, EXOKAY, SLVERR, DECERR]	None
SEND_READ_DATA Creates a read channel transaction. The ID tag must match the associated read transaction. This task returns after it has been acknowledged by the master. ***Note:*** This would need to be called multiple times for a burst > 1.	ID: Read ID tag DATA: Data to send to the master RESPONSE: The read response to send to the master from the following: [OKAY, EXOKAY, SLVERR, DECERR] LAST: Last transfer flag	None
RECEIVE_WRITE_ADDRESS This task drives the AWREADY signal and monitors the write address bus for write address transfers coming from the master that have the specified ID tag (unless the IDValid bit = 0). It then returns the data associated with the write address transaction. If the IDValid bit is 0 then the input ID tag is not used and the next available write address transfer is sampled. This task uses the SLAVE_ADDRESS and SLAVE_MEM_SIZE parameters to determine if the address is valid.	ID: Write Address ID tag IDValid: Bit to indicate if the ID input parameter is to be used. When set to 1 the ID is valid and used, when set to 0 it is ignored.	ADDR: Write Address LEN: Burst Length SIZE: Burst Size BURST: Burst Type LOCK: Lock Type CACHE: Cache Type PROT: Protection Type IDTAG: Sampled ID tag
RECEIVE_READ_ADDRESS This task drives the ARREADY signal and monitors the read address bus for read address transfers coming from the master that have the specified ID tag (unless the IDValid bit = 0). It then returns the data associated with the read address transaction. If the IDValid bit is 0 then the input ID tag is not used and the next available read address transfer is sampled. This task uses the SLAVE_ADDRESS and SLAVE_MEM_SIZE parameters to determine if the address is valid.	ID: Write Address ID tag IDValid: Bit to indicate if the ID input parameter is to be used. When set to 1 the ID is valid and used, when set to 0 it is ignored.	ADDR: Write Address LEN: Burst Length SIZE: Burst Size BURST: Burst Type LOCK: Lock Type CACHE: Cache Type PROT: Protection Type IDTAG: Sampled ID tag

Table 3-12: **Channel Level API for AXI3 Slave BFM *(Cont'd)***

API Task Name and Description	Inputs	Outputs
RECEIVE_WRITE_DATA This task drives the WREADY signal and monitors the write data bus for write transfers coming from the master that have the specified ID tag (unless the IDValid bit = 0). It then returns the data associated with the transaction and the status of the last flag. ***Note:*** This would need to be called multiple times for a burst > 1. If the IDValid bit is 0 then the input ID tag is not used and the next available write data transfer is sampled.	ID: Write ID tag IDValid: Bit to indicate if the ID input parameter is to be used. When set to 1 the ID is valid and used, when set to 0 it is ignored.	DATA: Data transferred from the master STRB: Strobe signals used to validate the data LAST: Last transfer flag IDTAG: Sampled ID tag
RECEIVE_WRITE_BURST This task receives and processes a write burst on the write data channel with the specified ID (unless the IDValid bit =0). It does not wait for the write address transfer to be received. This task uses the RECEIVE_WRITE_DATA task from the channel level API. If the IDValid bit is 0 then the input ID tag is not used and the next available write burst is sampled. This task returns when the complete write burst is complete. This task automatically supports narrow transfers and unaligned transfers; that is, this task aligns the output data with the burst so the final output data should only contain valid data (up to the size of the burst data, shown by the output datasize).	ID: Write ID tag IDValid: Bit to indicate if the ID input parameter is to be used. When set to 1 the ID is valid and used, when set to 0 it is ignored. ADDR: Write Address LEN: Burst Length SIZE: Burst Size BURST: Burst Type	DATA: Data received from the write burst DATASIZE: The size in bytes of the valid data contained in the output data vector IDTAG: Sampled ID tag
RECEIVE_WRITE_BURST_NO_CHECKS This task receives and processes a write burst on the write data channel blindly, that is, with no checking of length, size or anything else. This task uses the RECEIVE_WRITE_DATA task from the channel level API. This task returns when the complete write burst is complete. This task automatically supports narrow transfers and unaligned transfers; that is, this task aligns the output data with the burst so the final output data should only contain valid data (up to the size of the burst data, shown by the output datasize).	ID: Write ID tag	DATA: Data received from the write burst DATASIZE: The size in bytes of the valid data contained in the output data vector

Table 3-12: **Channel Level API for AXI3 Slave BFM *(Cont'd)***

API Task Name and Description	Inputs	Outputs
SEND_READ_BURST This task does a read burst on the read data lines. It does not wait for the read address transfer to be received. This task uses the SEND_READ_DATA task from the channel level API. This task returns when the complete read burst is complete. This task automatically supports the generation of narrow transfers and unaligned transfers; that is, this task aligns the input data with the burst so data padding is not required.	ID: Read ID tag ADDR: Read Address LEN: Burst Length SIZE: Burst Size BURST: Burst Type LOCK: Lock Type DATA: Data to be sent over the burst	None
SEND_READ_BURST_RESP_CTRL This task is the same as SEND_READ_BURST except that the response sent to the master can be specified.	ID: Read ID tag ADDR: Read Address LEN: Burst Length SIZE: Burst Size BURST: Burst Type DATA: Data to be sent over the burst RESPONSE: This is a vector that should contain all of the desired responses for each read data transfer	None

The function level API for the AXI3 Slave BFM is detailed in Table 3-13.

Table 3-13: **Function Level API for AXI3 Slave BFM**

API Task Name and Description	Inputs	Outputs
READ_BURST_RESPOND Creates a semi-automatic response to a read request from the master. It checks if the ID tag for the read request is as expected and then provides a read response using the data provided. It is composed of the tasks RECEIVE_READ_ADDRESS and SEND_READ_BURST from the channel level API. This task returns when the complete write transaction is complete. This task automatically supports the generation of narrow transfers and unaligned transfers; that is, this task aligns the input data with the burst so data padding is not required.	ID: Read ID tag DATA: Data to send in response to the master read	None
WRITE_BURST_RESPOND This is a semi-automatic task which waits for a write burst with the specified ID tag and responds appropriately. The data received in the write burst is delivered as an output data vector. This task is composed of the tasks RECEIVE_WRITE_ADDRESS, RECEIVE_WRITE_BURST and SEND_WRITE_RESPONSE from the channel level API. This task returns when the complete write transaction is complete. This task automatically supports the generation of narrow transfers and unaligned transfers; that is, this task aligns the input data with the burst so data padding is not required.	ID: Write ID tag	DATA: Data received by slave DATASIZE: The size in bytes of the valid data contained in the output data vector
WRITE_BURST_RESPOND_DATA_FIRST This is a semi-automatic task which waits for a write burst with the specified ID tag and responds appropriately. It expects the write data to start arriving before the write address phase. It returns the data received in the write as a data vector. It is composed of the tasks RECEIVE_WRITE_BURST_NO_CHECKS, RECEIVE_WRITE_ADDRESS and SEND_WRITE_RESPONSE from the channel level API. This task returns when the complete write transaction is complete.	ID: Write ID tag	DATA: Data received by slave DATASIZE: The size in bytes of the valid data contained in the output data vector

Table 3-13: **Function Level API for AXI3 Slave BFM *(Cont'd)***

API Task Name and Description	Inputs	Outputs
READ_BURST_RESP_CTRL This task is the same as READ_BURST_RESPONSE except that the responses sent to the master can be specified.	ID: Read ID tag DATA: Data to send in response to the master read. RESPONSE: This is a vector that should contain all of the desired responses for each read data transfer.	None
WRITE_BURST_RESP_CTRL This task is the same as WRITE_BURST_RESPONSE except that the response sent to the master can be specified.	ID: Write ID tag RESPONSE: The chosen write response from the following [OKAY, EXOKAY, SLVERR, DECERR]	DATA: Data received by slave DATASIZE: The size in bytes of the valid data contained in the output data vector

AXI4 Master BFM Test Writing API

The channel level API for the AXI4 Master BFM is detailed in Table 3-14.

Table 3-14: **Channel Level API for AXI4 Master BFM**

API Task Name	Inputs	Outputs
SEND_WRITE_ADDRESS Creates a write address channel transaction. This task returns after the write address has been acknowledged by the slave.	ID: Write Address ID tag ADDR: Write Address LEN: Burst Length SIZE: Burst Size BURST: Burst Type LOCK: Lock Type CACHE: Cache Type PROT: Protection Type REGION: Region Identifier QOS: Quality of Service Signals AWUSER: Address Write User Defined Signals	None
SEND_WRITE_DATA Creates a single write data channel transaction. The data should be the same size as the width of the data bus. This task returns after is has been acknowledged by the slave. The data input is used as raw bus data; that is, no realignment for narrow or unaligned data. ***Note:*** Should be called multiple times for a burst with correct control of the LAST flag	STOBE: Strobe signals DATA: Data for transfer LAST: Last transfer flag WUSER: Write User Defined Signals	None

Table 3-14: **Channel Level API for AXI4 Master BFM *(Cont'd)***

API Task Name	Inputs	Outputs
SEND_READ_ADDRESS Creates a read address channel transaction. This task returns after the read address has been acknowledged by the slave.	ID: Read Address ID tag ADDR: Read Address LEN: Burst Length SIZE: Burst Size BURST: Burst Type LOCK: Lock Type CACHE: Cache Type PROT: Protection Type REGION: Region Identifier QOS: Quality of Service Signals ARUSER: Address Read User Defined Signals	None
RECEIVE_READ_DATA This task drives the RREADY signal and monitors the read data bus for read transfers coming from the slave that have the specified ID tag. It then returns the data associated with the transaction and the status of the last flag. The data output here is raw bus data; that is, no realignment for narrow or unaligned data. ***Note:*** This would need to be called multiple times for a burst > 1. Also, you must call the "remove_pending_transaction" task when all data is received to ensure that the pending transaction counter is decremented. This is done automatically by the RECEIVE_READ_BURST and RECEIVE_WRITE_RESPONSE channel level API tasks.	ID: Read ID tag	DATA: Data transferred by the slave RESPONSE: The slave read response from the following: [OKAY, EXOKAY, SLVERR, DECERR] LAST: Last transfer flag RUSER: Read User Defined Signals
RECEIVE_WRITE_RESPONSE This task drives the BREADY signal and monitors the write response bus for write responses coming from the slave that have the specified ID tag. It then returns the response associated with the transaction.	ID: Write ID tag	RESPONSE: The slave write response from the following: [OKAY, EXOKAY, SLVERR, DECERR] BUSER: Write Response User Defined Signals

Table 3-14: **Channel Level API for AXI4 Master BFM *(Cont'd)***

API Task Name	Inputs	Outputs
RECEIVE_READ_BURST This task receives a read channel burst based on the ID input. The RECEIVE_READ_DATA from the channel level API is used. This task returns when the read transaction is complete. The data returned by the task is the valid only data, that is, re-aligned data. This task also checks each response and issues a warning if it is not as expected.	ID: Read ID tag ADDR: Read Address LEN: Burst Length SIZE: Burst Size BURST: Burst Type LOCK: Lock Type	DATA: Valid Data transferred by the slave RESPONSE: This is a vector that is created by concatenating all slave read responses together RUSER: This is a vector that is created by concatenating all slave read user signal data together
SEND_WRITE_BURST This task does a write burst on the write data lines. It does not execute the write address transfer. This task uses the SEND_WRITE_DATA task from the channel level API. This task returns when the complete write burst is complete. This task automatically supports the generation of narrow transfers and unaligned transfers; that is, this task aligns the input data with the burst so data padding is not required.	ADDR: Write Address LEN: Burst Length SIZE: Burst Size BURST: Burst Type DATA: Data to send DATASIZE: The size in bytes of the valid data contained in the input data vector WUSER: This is a vector that is created by concatenating all write transfer user signal data together	None

The function level API for the AXI4 Master BFM is detailed in Table 3-15.

Table 3-15: **Function Level API for AXI4 Master BFM**

API Task Name and Description	Inputs	Outputs
READ_BURST This task does a full read process. It is composed of the tasks SEND_READ_ADDRESS and RECEIVE_READ_BURST from the channel level API. This task returns when the read transaction is complete.	ID: Read ID tag ADDR: Read Address LEN: Burst Length SIZE: Burst Size BURST: Burst Type LOCK: Lock Type CACHE: Cache Type PROT: Protection Type REGION: Region Identifier QOS: Quality of Service Signals ARUSER: Address Read User Defined Signals	DATA: Valid data transferred by the slave RESPONSE: This is a vector that is created by concatenating all slave read responses together RUSER: This is a vector that is created by concatenating all slave read user signal data together

Table 3-15: **Function Level API for AXI4 Master BFM *(Cont'd)***

API Task Name and Description	Inputs	Outputs
WRITE_BURST This task does a full write process. It is composed of the tasks SEND_WRITE_ADDRESS, SEND_WRITE_BURST and RECEIVE_WRITE_RESPONSE from the channel level API. This task returns when the complete write transaction is complete. This task automatically supports the generation of narrow transfers and unaligned transfers. This is used in scenario wherein before the slave asserts AWREADY and/or WREADY the slave can wait for AWVALID.	ID: Write ID tag ADDR: Write Address LEN: Burst Length SIZE: Burst Size BURST: Burst Type LOCK: Lock Type CACHE: Cache Type PROT: Protection Type DATA: Data to send DATASIZE: The size in bytes of the valid data contained in the input data vector REGION: Region Identifier QOS: Quality of Service Signals AWUSER: Address Write User Defined Signals WUSER: This is a vector that is created by concatenating all write transfer user signal data together	RESPONSE: The slave write response from the following: [OKAY, EXOKAY, SLVERR, DECERR] BUSER: Write Response Channel User Defined Signals
WRITE_BURST_CONCURRENT This task does the same function as the WRITE_BURST task; however, it performs the write address and write data phases concurrently.	ID: Write ID tag ADDR: Write Address LEN: Burst Length SIZE: Burst Size BURST: Burst Type LOCK: Lock Type CACHE: Cache Type PROT: Protection Type DATA: Data to send DATASIZE: The size in bytes of the valid data contained in the input data vector REGION: Region Identifier QOS: Quality of Service Signals AWUSER: Address Write User Defined Signals WUSER: This is a vector that is created by concatenating all write transfer user signal data together	RESPONSE: The slave write response from the following: [OKAY, EXOKAY, SLVERR, DECERR] BUSER: Write Response Channel User Defined Signals

AXI4 Slave BFM Test Writing API

The channel level API for the AXI4 Slave BFM is detailed in Table 3-16.

Table 3-16: **Channel Level API for AXI4 Slave BFM**

API Task Name and Description	Inputs	Outputs
SEND_WRITE_RESPONSE Creates a write response channel transaction. The ID tag must match the associated write transaction. This task returns after it has been acknowledged by the master.	ID: Write ID tag RESPONSE: The chosen write response from the following [OKAY, EXOKAY, SLVERR, DECERR] BUSER: Write Response User Defined Signals	None
SEND_READ_DATA Creates a read channel transaction. The ID tag must match the associated read transaction. This task returns after it has been acknowledged by the master. ***Note:*** This would need to be called multiple times for a burst > 1.	ID: Read ID tag DATA: Data to send to the master RESPONSE: The read response to send to the master from the following: [OKAY, EXOKAY, SLVERR, DECERR] LAST: Last transfer flag RUSER: Read User Defined Signals	None
RECEIVE_WRITE_ADDRESS This task drives the AWREADY signal and monitors the write address bus for write address transfers coming from the master that have the specified ID tag (unless the IDValid bit = 0). It then returns the data associated with the write address transaction. If the IDValid bit is 0 then the input ID tag is not used and the next available write address transfer is sampled. This task uses the SLAVE_ADDRESS and SLAVE_MEM_SIZE parameters to determine if the address is valid.	ID: Write Address ID tag IDValid: Bit to indicate if the ID input parameter is to be used. When set to 1 the ID is valid and used, when set to 0 it is ignored.	ADDR: Write Address LEN: Burst Length SIZE: Burst Size BURST: Burst Type LOCK: Lock Type CACHE: Cache Type PROT: Protection Type REGION: Region Identifier QOS: Quality of Service Signals AWUSER: Address Write User Defined Signals IDTAG: Sampled ID tag

Table 3-16: **Channel Level API for AXI4 Slave BFM *(Cont'd)***

API Task Name and Description	Inputs	Outputs
RECEIVE_READ_ADDRESS This task drives the ARREADY signal and monitors the read address bus for read address transfers coming from the master that have the specified ID tag (unless the IDValid bit = 0). It then returns the data associated with the read address transaction. If the IDValid bit is 0 then the input ID tag is not used and the next available read address transfer is sampled. This task uses the SLAVE_ADDRESS and SLAVE_MEM_SIZE parameters to determine if the address is valid.	ID: Read Address ID tag IDValid: Bit to indicate if the ID input parameter is to be used. When set to 1 the ID is valid and used, when set to 0 it is ignored.	ADDR: Read Address LEN: Burst Length SIZE: Burst Size BURST: Burst Type LOCK: Lock Type CACHE: Cache Type PROT: Protection Type REGION: Region Identifier QOS: Quality of Service Signals ARUSER: Address Read User Defined Signals IDTAG: Sampled ID tag
RECEIVE_WRITE_DATA This task drives the WREADY signal and monitors the write data bus for write transfers coming from the master. It then returns the data associated with the transaction and the status of the last flag. ***Note:*** This would need to be called multiple times for a burst > 1.	None	DATA: Data transferred from the master STRB: Strobe signals used to validate the data LAST: Last transfer flag WUSER: Write User Defined Signals
RECEIVE_WRITE_BURST This task receives and processes a write burst on the write data channel. It does not wait for the write address transfer to be received. This task uses the RECEIVE_WRITE_DATA task from the channel level API. This task returns when the complete write burst is complete. This task automatically supports narrow transfers and unaligned transfers; that is, this task aligns the output data with the burst so the final output data should only contain valid data (up to the size of the burst data).	ADDR: Write Address LEN: Burst Length SIZE: Burst Size BURST: Burst Type	DATA: Data received from the write burst DATASIZE: The size in bytes of the valid data contained in the output data vector WUSER: This is a vector that is created by concatenating all master write user signal data together

Table 3-16: **Channel Level API for AXI4 Slave BFM *(Cont'd)***

API Task Name and Description	Inputs	Outputs
SEND_READ_BURST This task does a read burst on the read data lines. It does not wait for the read address transfer to be received. This task uses the SEND_READ_DATA task from the channel level API. This task returns when the complete read burst is complete. This task automatically supports the generation of narrow transfers and unaligned transfers; that is, this task aligns the input data with the burst so data padding is not required.	ID: Read ID tag ADDR: Read Address LEN: Burst Length SIZE: Burst Size BURST: Burst Type LOCK: Lock Type DATA: Data to be sent over the burst RUSER: This is a vector that is created by concatenating all required slave read user signal data together	None
SEND_READ_BURST_RESP_CTRL This task does a read burst on the read data lines. It does not wait for the read address transfer to be received. This task uses the SEND_READ_DATA task from the channel level API. This task returns when the complete read burst is complete. This task automatically supports the generation of narrow transfers and unaligned transfers; that is, this task aligns the input data with the burst so data padding is not required.	ID: Read ID tag ADDR: Read Address LEN: Burst Length SIZE: Burst Size BURST: Burst Type DATA: Data to be sent over the burst RESPONSE: This is a vector that should contain all of the desired responses for each read data transfer RUSER: This is a vector that is created by concatenating all required slave read user signal data together	None

The function level API for the AXI4 Slave BFM is detailed in Table 3-17.

Table 3-17: **Function Level API for AXI4 Slave BFM**

API Task Name and Description	Inputs	Outputs
READ_BURST_RESPOND Creates a semi-automatic response to a read request from the master. It checks if the ID tag for the read request is as expected and then provides a read response using the data provided. It is composed of the tasks RECEIVE_READ_ADDRESS and SEND_READ_BURST from the channel level API. This task returns when the complete write transaction is complete. This task automatically supports the generation of narrow transfers and unaligned transfers; that is, this task aligns the input data with the burst so data padding is not required.	ID: Read ID tag DATA: Data to send in response to the master read RUSER: This is a vector that is created by concatenating all required read user signal data together	None
WRITE_BURST_RESPOND This is a semi-automatic task which waits for a write burst with the specified ID tag and responds appropriately. The data received in the write burst is delivered as an output data vector. This task is composed of the tasks RECEIVE_WRITE_ADDRESS, RECEIVE_WRITE_BURST and SEND_WRITE_RESPONSE from the channel level API. This task returns when the complete write transaction is complete. This task automatically supports the generation of narrow transfers and unaligned transfers; that is, this task aligns the input data with the burst so data padding is not required.	ID: Write ID tag BUSER: Write Response Channel User Defined Signals	DATA: Data received by slave DATASIZE: The size in bytes of the valid data contained in the output data vector WUSER: This is a vector that is created by concatenating all master write transfer user signal data together

Table 3-17: **Function Level API for AXI4 Slave BFM *(Cont'd)***

API Task Name and Description	Inputs	Outputs
READ_BURST_RESP_CTRL Creates a semi-automatic response to a read request from the master. It checks if the ID tag for the read request is as expected and then provides a read response using the data and response vector provided. It is composed of the tasks RECEIVE_READ_ADDRESS and SEND_READ_BURST_RESP_CTRL from the channel level API. This task returns when the complete write transaction is complete. This task automatically supports the generation of narrow transfers and unaligned transfers; that is, this task aligns the input data with the burst so data padding is not required.	ID: Read ID tag DATA: Data to send in response to the master read RESPONSE: This is a vector that should contain all of the desired responses for each read data transfer RUSER: This is a vector that is created by concatenating all required read user signal data together	None
WRITE_BURST_RESP_CTRL This is a semi-automatic task which waits for a write burst with the specified ID tag and responds appropriately using the specified response. The data received in the write burst is delivered as an output data vector. This task is composed of the tasks RECEIVE_WRITE_ADDRESS, RECEIVE_WRITE_BURST and SEND_WRITE_RESPONSE from the channel level API. This task returns when the complete write transaction is complete. This task automatically supports the generation of narrow transfers and unaligned transfers; that is, this task aligns the input data with the burst so data padding is not required.	ID: Write ID tag RESPONSE: The chosen write response from the following [OKAY, EXOKAY, SLVERR, DECERR] BUSER: Write Response Channel User Defined Signals	DATA: Data received by slave DATASIZE: The size in bytes of the valid data contained in the output data vector WUSER: This is a vector that is created by concatenating all master write transfer user signal data together

AXI4-Lite Master BFM Test Writing API

The channel level API for the AXI4-Lite Master BFM is detailed in Table 3-18.

Table 3-18: **Channel Level API for AXI4-Lite Master BFM**

API Task Name and Description	Inputs	Outputs
SEND_WRITE_ADDRESS Creates a write address channel transaction. This task returns after the write address has been acknowledged by the slave.	ADDR: Write Address PROT: Protection Type	None
SEND_WRITE_DATA Creates a single write data channel transaction. The data should be the same size as the width of the data bus. This task returns after is has been acknowledged by the slave.	STOBE: Strobe signals DATA: Data for transfer	None
SEND_READ_ADDRESS Creates a read address channel transaction. This task returns after the read address has been acknowledged by the slave.	ADDR: Read Address PROT: Protection Type	None
RECEIVE_READ_DATA This task drives the RREADY signal and monitors the read data bus for read transfers coming from the slave. It returns the data associated with the transaction and the response from the slave.	None	DATA: Data transferred by the slave RESPONSE: The slave read response from the following: [OKAY, SLVERR, DECERR]
RECEIVE_WRITE_RESPONSE This task drives the BREADY signal and monitors the write response bus for write responses coming from the slave. It returns the response associated with the transaction.	None	RESPONSE: The slave write response from the following: [OKAY, SLVERR, DECERR]

The function level API for the AXI4-Lite Master BFM is detailed in Table 3-19.

Table 3-19: **Function Level API for AXI4-Lite Master BFM**

API Task Name and Description	Inputs	Outputs
READ_BURST This task does a full read process. It is composed of the tasks SEND_READ_ADDRESS and RECEIVE_READ_DATA from the channel level API. This task returns when the read transaction is complete.	ADDR: Read Address PROT: Protection Type	DATA: Valid data transferred by the slave RESPONSE: The slave write response from the following: [OKAY, SLVERR, DECERR]
WRITE_BURST This task does a full write process. It is composed of the tasks SEND_WRITE_ADDRESS, SEND_WRITE_DATA and RECEIVE_WRITE_RESPONSE from the channel level API. This task returns when the complete write transaction is complete. This is used in scenarios wherein before the slave asserts AWREADY and/or WREADY the slave can wait for AWVALID.	ADDR: Write Address PROT: Protection Type DATA: Data to send DATASIZE: The size in bytes of the valid data contained in the input data vector	RESPONSE: The slave write response from the following: [OKAY, SLVERR, DECERR]
WRITE_BURST_CONCURRENT This task does the same function as the WRITE_BURST task; however, it performs the write address and data phases concurrently.	ADDR: Write Address PROT: Protection Type DATA: Data to send DATASIZE: The size in bytes of the valid data contained in the input data vector	RESPONSE: The slave write response from the following: [OKAY, SLVERR, DECERR]
WRITE_BURST_DATA_FIRST This task does the same function as the WRITE_BURST task; however, it sends the write data burst before sending the associated write address transfer on the write address channel. This is used in scenarios wherein before the slave asserts AWREADY and/or WREADY the slave can wait for WVALID.	ADDR: Write Address PROT: Protection Type DATA: Data to send DATASIZE: The size in bytes of the valid data contained in the input data vector	RESPONSE: The slave write response from the following: [OKAY, SLVERR, DECERR]

AXI4-Lite Slave BFM Test Writing API

The channel level API for the AXI4-Lite Slave BFM is detailed in Table 3-20.

Table 3-20: **Channel Level API for AXI4-Lite Slave BFM**

API Task Name and Description	Inputs	Outputs
SEND_WRITE_RESPONSE Creates a write response channel transaction. This task returns after it has been acknowledged by the master.	RESPONSE: The chosen write response from the following [OKAY, SLVERR, DECERR]	None
SEND_READ_DATA Creates a read channel transaction. This task returns after it has been acknowledged by the master.	DATA: Data to send to the master RESPONSE: The read response to send to the master from the following: [OKAY, SLVERR, DECERR]	None
RECEIVE_WRITE_ADDRESS This task drives the AWREADY signal and monitors the write address bus for write address transfers coming from the master. It returns the data associated with the write address transaction. This task uses the SLAVE_ADDRESS and SLAVE_MEM_SIZE parameters to determine if the address is valid.	ADDR: Write Address ADDRValid: Bit to indicate if the address input parameter is to be used. When set to 1 the ADDR is valid and used, when set to 0 it is ignored.	PROT: Protection Type SADDR: Sampled Write Address
RECEIVE_READ_ADDRESS This task drives the ARREADY signal and monitors the read address bus for read address transfers coming from the master. It returns the data associated with the read address transaction. This task uses the SLAVE_ADDRESS and SLAVE_MEM_SIZE parameters to determine if the address is valid.	ADDR: Read Address ADDRValid: Bit to indicate if the address input parameter is to be used. When set to 1 the ADDR is valid and used, when set to 0 it is ignored.	PROT: Protection Type SADDR: Sampled Read Address
RECEIVE_WRITE_DATA This task drives the WREADY signal and monitors the write data bus for write transfers coming from the master. It returns the data associated with the transaction.	None	DATA: Data transferred from the master STRB: Strobe signals used to validate the data

The function level API for the AXI4-Lite Slave BFM is detailed in Table 3-21.

Table 3-21: **Function Level API for AXI4-Lite Slave BFM**

API Task Name and Description	Inputs	Outputs
READ_BURST_RESPOND Creates a semi-automatic response to a read request from the master. It is composed of the tasks RECEIVE_READ_ADDRESS and SEND_READ_DATA from the channel level API. This task returns when the complete write transaction is complete. If ADDRVALID = 0 the input ADDR is ignored and the first read request is used and responded to. If the ADDRVALID = 1 then the ADDR input is used and the DATA input is used to respond to the read burst with the specified address.	ADDR: Read Address ADDRValid: Bit to indicate if the address input parameter is to be used. When set to 1 the ADDR is valid and used, when set to 0 it is ignored. DATA: Data to send in response to the master read	None
WRITE_BURST_RESPOND This is a semi-automatic task which waits for a write burst from the master and responds appropriately. The data received in the write burst is delivered as an output data vector. This task is composed of the tasks RECEIVE_WRITE_ADDRESS, RECEIVE_WRITE_DATA and SEND_WRITE_RESPONSE from the channel level API. This task returns when the complete write transaction is complete. If ADDRVALID = 0 the input ADDR is ignored and the first write request is used for the DATA output. If the ADDRVALID = 1 then the ADDR input is used and the DATA associated with that transfer is output using the DATA output.	ADDR: Write Address ADDRValid: Bit to indicate if the address input parameter is to be used. When set to 1 the ADDR is valid and used, when set to 0 it is ignored.	DATA: Data received by slave DATASIZE: The size in bytes of the valid data contained in the output data vector

Table 3-21: **Function Level API for AXI4-Lite Slave BFM *(Cont'd)***

API Task Name and Description	Inputs	Outputs
READ_BURST_RESP_CTRL This task is the same as READ_BURST_RESPOND except that the response sent to the master can be specified.	ADDR: Read Address ADDRValid: Bit to indicate if the address input parameter is to be used. When set to 1 the ADDR is valid and used, when set to 0 it is ignored. DATA: Data to send in response to the master read RESPONSE: The chosen write response from the following [OKAY, SLVERR, DECERR]	None
WRITE_BURST_RESP_CTRL This task is the same as WRITE_BURST_RESPOND except that the response sent to the master can be specified.	ADDR: Write Address ADDRValid: Bit to indicate if the address input parameter is to be used. When set to 1 the ADDR is valid and used, when set to 0 it is ignored. RESPONSE: The chosen write response from the following [OKAY, SLVERR, DECERR]	DATA: Data received by slave DATASIZE: The size in bytes of the valid data contained in the output data vector

AXI4-Stream Master BFM Test Writing API

The channel level API for the AXI4-Stream Master BFM is detailed in Table 3-22.

Table 3-22: **Channel Level API for AXI4-Stream Master BFM**

API Task Name and Description	Inputs	Outputs
SEND_TRANSFER Creates a single AXI4-Stream transfer.	ID: Transfer ID Tag DEST: Transfer Destination DATA: Transfer Data STRB: Transfer Strobe Signals KEEP: Transfer Keep Signals LAST: Transfer Last Signal USER: Transfer User Signals	None
SEND_PACKET This task sends a complete packet over the streaming interface. It uses the SEND_TRANSFER task from the channel level API.	ID: Transfer ID Tag DEST: Transfer Destination DATA: Vector of Transfer data to send DATASIZE: The size in bytes of the valid data contained in the input data vector (This must be aligned to the multiples of the data bus width) USER: This is a vector that is created by concatenating all transfer user signal data together	None

AXI4-Stream Slave BFM Test Writing API

The channel level API for the AXI4-Stream Slave BFM is detailed in Table 3-23.

Table 3-23: **Channel Level API for AXI4-Stream Slave BFM**

API Task Name and Description	Inputs	Outputs
RECEIVE_TRANSFER Receives a single AXI4-Stream transfer.	ID: Transfer ID Tag IDValid: Bit to indicate if the ID input parameter is to be used. When set to 1 the ID is valid and used, when set to 0 it is ignored DEST: Transfer Destination DESTValid: Bit to indicate if the DEST input parameter is to be used	ID: Transfer ID Tag DEST: Transfer Destination DATA: Transfer Data STRB: Transfer Strobe Signals KEEP: Transfer Keep Signals LAST: Transfer Last Signal USER: Transfer User Signals
RECEIVE_PACKET This task receives and processes a packet from the transfer channel. It returns when the complete packet has been sampled. This task uses the RECEIVE_TRANSFER task from the channel level API. If the IDValid or DESTValid bits are 0, the input ID tag and the DEST values are not used. In this case, the next values from the first valid transfer are sampled and used for the full packet irrespective of the ID tag or DEST input values.	ID: Packet ID Tag IDValid: Bit to indicate if the ID input parameter is to be used. When set to 1, the ID is valid and used; when set to 0, it is ignored DEST: Packet Destination DESTValid: Bit to indicate if the DEST input parameter is to be used	PID: Packet ID Tag PDEST: Packet Destination DATA: Packet data vector DATASIZE: The size in bytes of the valid data contained in the output packet data vector USER: This is a vector that is created by concatenating all master user signal data together

Protocol Description

Fore more information on AXI specification, see the ARM® *AMBA AXI4-Stream Protocol Specification* [Ref 2].

Design Flow Steps

This chapter describes customizing and generating the core, constraining the core, and the simulation, synthesis and implementation steps that are specific to this IP core. More detailed information about the standard Vivado® design flows and the Vivado IP integrator can be found in the following Vivado Design Suite user guides:

- *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994) [Ref 5]
- *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 3]
- *Vivado Design Suite User Guide: Getting Started* (UG910) [Ref 4]
- *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 6]

Customizing and Generating the Core

This section includes information about using Xilinx® tools to customize and generate the core in the Vivado Design Suite.The AXI BFM cores can be found in the following place in the Vivado IP catalog:

If you are customizing and generating the core in the IP integrator, see the *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994) [Ref 5] for detailed information. IP integrator might auto-compute certain configuration values when validating or generating the design, as noted in Using AXI BFM Cores in Vivado IP Integrator. To check whether the values change, see the description of the parameter in this chapter. To view the parameter value, run the `validate_bd_design` command in the Tcl Console.

```
Embedded_Processing\Debug & Verification\Verification
```

You can customize the IP for use in your design by specifying values for the various parameters associated with the IP core using the following steps:

1. Select the IP from the IP catalog.
2. Double-click the selected IP or select the Customize IP command from the toolbar or right-click menu.

For details, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 3] and the *Vivado Design Suite User Guide: Getting Started* (UG910) [Ref 4].

Note: Figures in this chapter are illustrations of the Vivado IDE. This layout might vary from the current version.

Figure 4-1 and Figure 4-2 show the AXI BFM cores Customize IP dialog box with information about customizing ports.

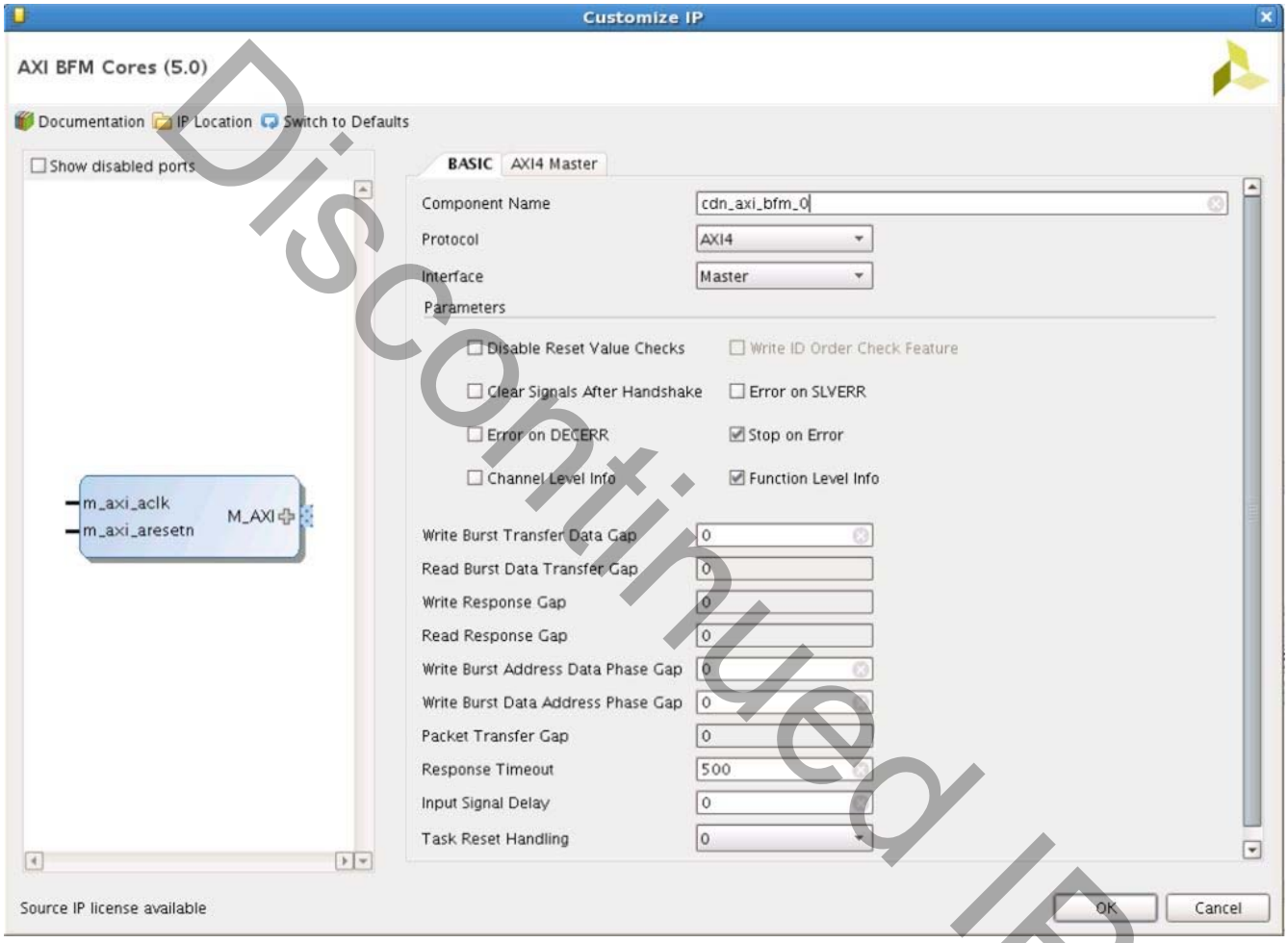

Figure 4-1: **Vivado Customize IP Dialog Box – Basic Tab**

Basic

Note: For the run time parameter descriptions, see Table 4-1.

- **Component Name** – The base name of the output files generated for the core. Names must begin with a letter and can be composed of any of the following characters: a to z, 0 to 9, and "_".
- **Protocol** – Choose the specific AXI specification.
- **Select the Master or Slave Mode** – Select the Master or Slave mode.

Note: Based on the selection of Protocol and Mode, the next tab is updated accordingly. This guide only shows the AXI4 Master tab.

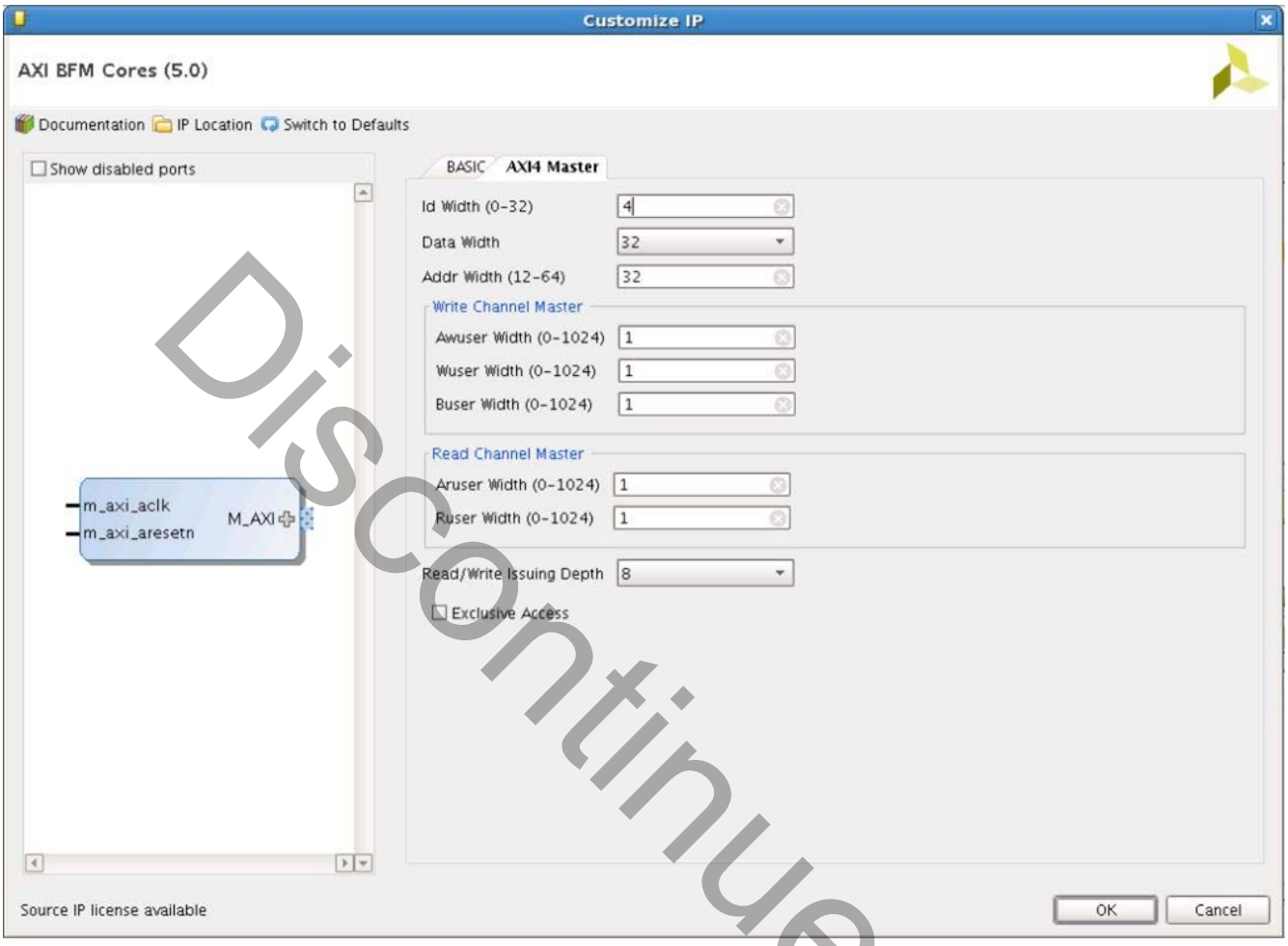

Figure 4-2: **Vivado Customize IP Dialog Box – AXI4 Master Tab**

AXI4 Master

- **ID Width** – ID Width default is 4.
- **Data Width** – Read and write data buses can be 8, 16, 32, 64, 128, 256, 512, or 1,024 bits wide. Default is 32.
- **Addr Width** – Address width can be configured between 12 to 64 bits. Default is 32.
- **Read/Write Issuing Depth** – Default is 8.
- **Exclusive Access** – This informs the master that exclusive access is supported by the slave. A value of 1 means it is supported so the response check expects an EXOKAY, or else give a warning, in response to an exclusive access. A value of 0 means the slave does not support this so a response of OKAY is expected in response to an exclusive access. Default is 1.

Write Channel Master

- **Awuser Width** – Range of 1 to 1,024 with default set to 1.
- **Wuser Width** – Range of 1 to 1,024 with default set to 1.
- **Buser Width** – Range of 1 to 1,024 with default set to 1.

Read Channel Master

- **Aruser Width** – Range of 1 to 1,024 with default set to 1.
- **Ruser Width** – Range of 1 to 1,024 with default set to 1.

BFM Instantiation Names

When the IP is configured and generated, the top-level wrapper is named with `<component_name>.v`.

The BFM instantiation names are the following:

- AXI4
  - **Master** – `cdn_axi4_master_bfm_inst`
  - **Slave** – `cdn_axi4_slave_bfm_inst`
- AXI3
  - **Master** – `cdn_axi3_master_bfm_inst`
  - **Slave** – `cdn_axi3_slave_bfm_inst`
- AXI4-Lite
  - **Master** – `cdn_axi4_lite_master_bfm`
  - **Slave** – `cdn_axi4_lite_slave_bfm`
- AXI4-Stream
  - **Master** – `cdn_axi4_streaming_master_bfm`
  - **Slave** – `cdn_axi4_streaming_slave_bfm`

Using AXI BFM Cores in Vivado IP Integrator

When the IP is used in IP integrator, certain parameters are auto set based on the connections.

Figure 4-3 shows the AXI4 Slave Mode.

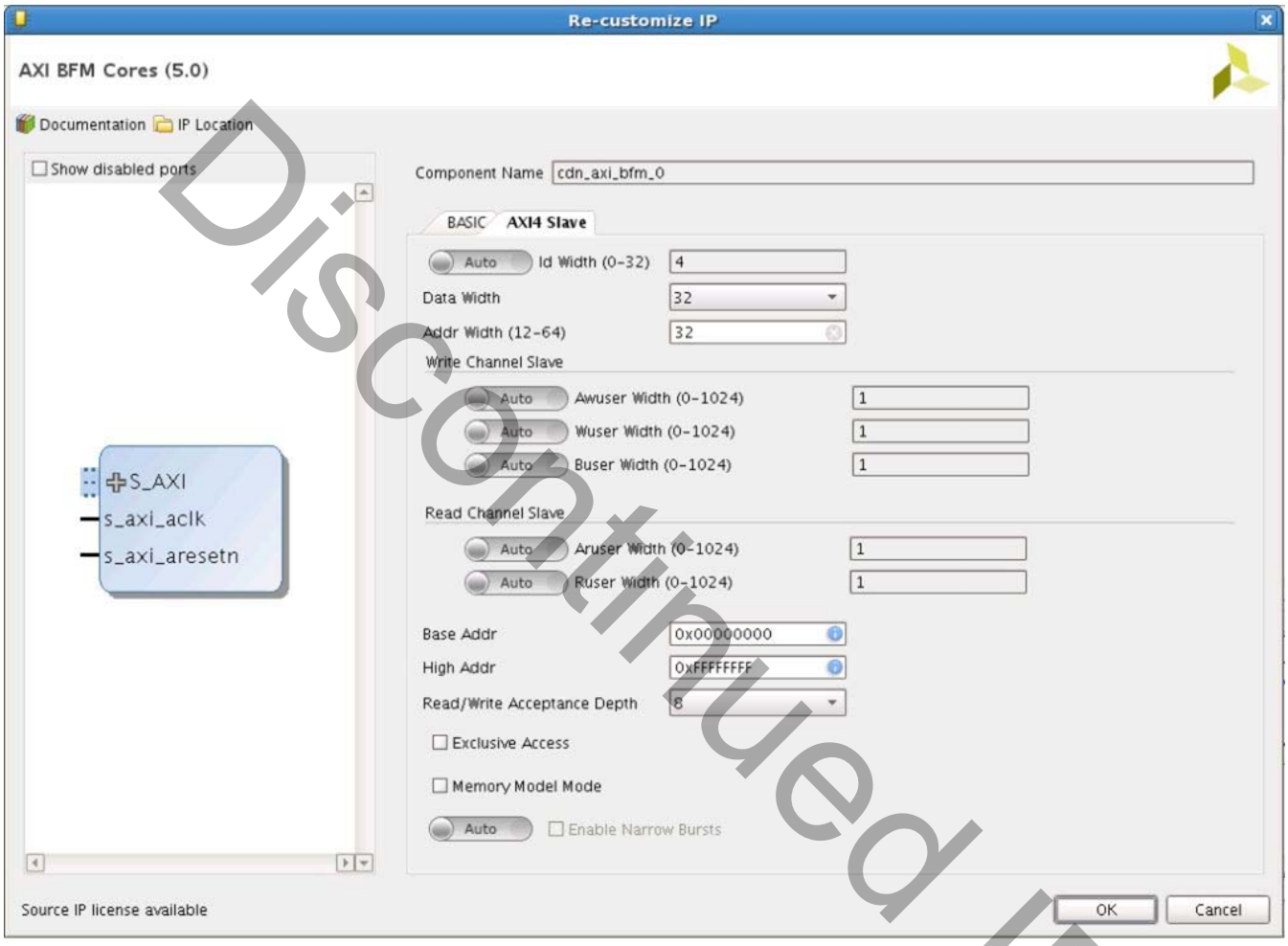

Figure 4-3: **AXI4 Slave Mode**

In this case, all of the ID width parameters (ID Width, Awuser Width, Wuser Width, Buser Width, Aruser Width, Ruser Width) are auto set based on the AXI4 interface.

IMPORTANT: *By default these parameters are automatically updated. You can override the propagated value by changing the switch to* ***Manual****.*

Figure 4-4 shows the AXI3 Slave Mode.

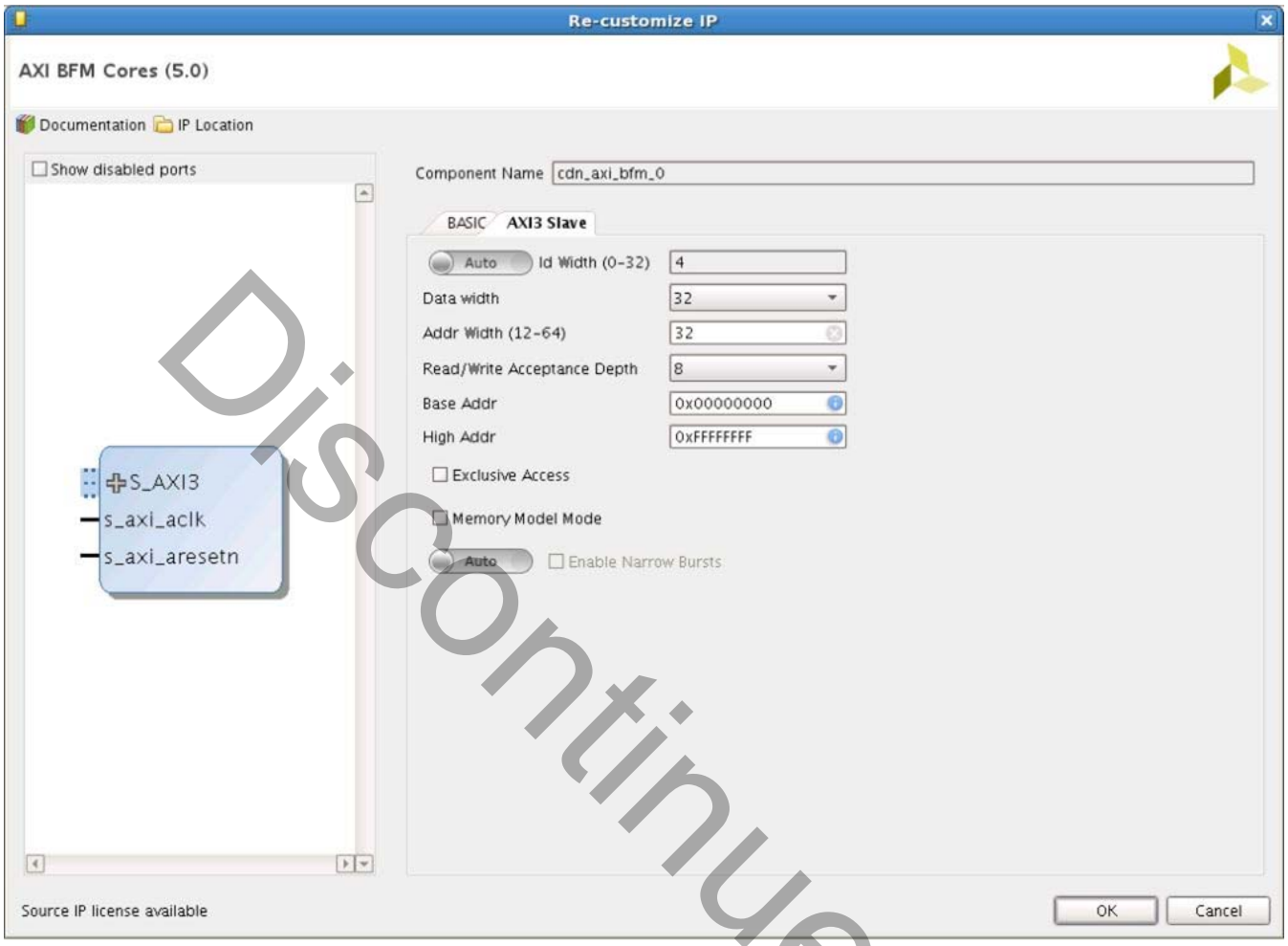

Figure 4-4: **AXI3 Slave Mode**

In this case, the ID Width parameter is auto set based on the AXI3 interface.

IMPORTANT: *By default these parameters are automatically updated in IP integrator. You can override the propagated value by changing the switch to* ***Manual****.*

Figure 4-5 shows the AXI-Stream Slave Mode.

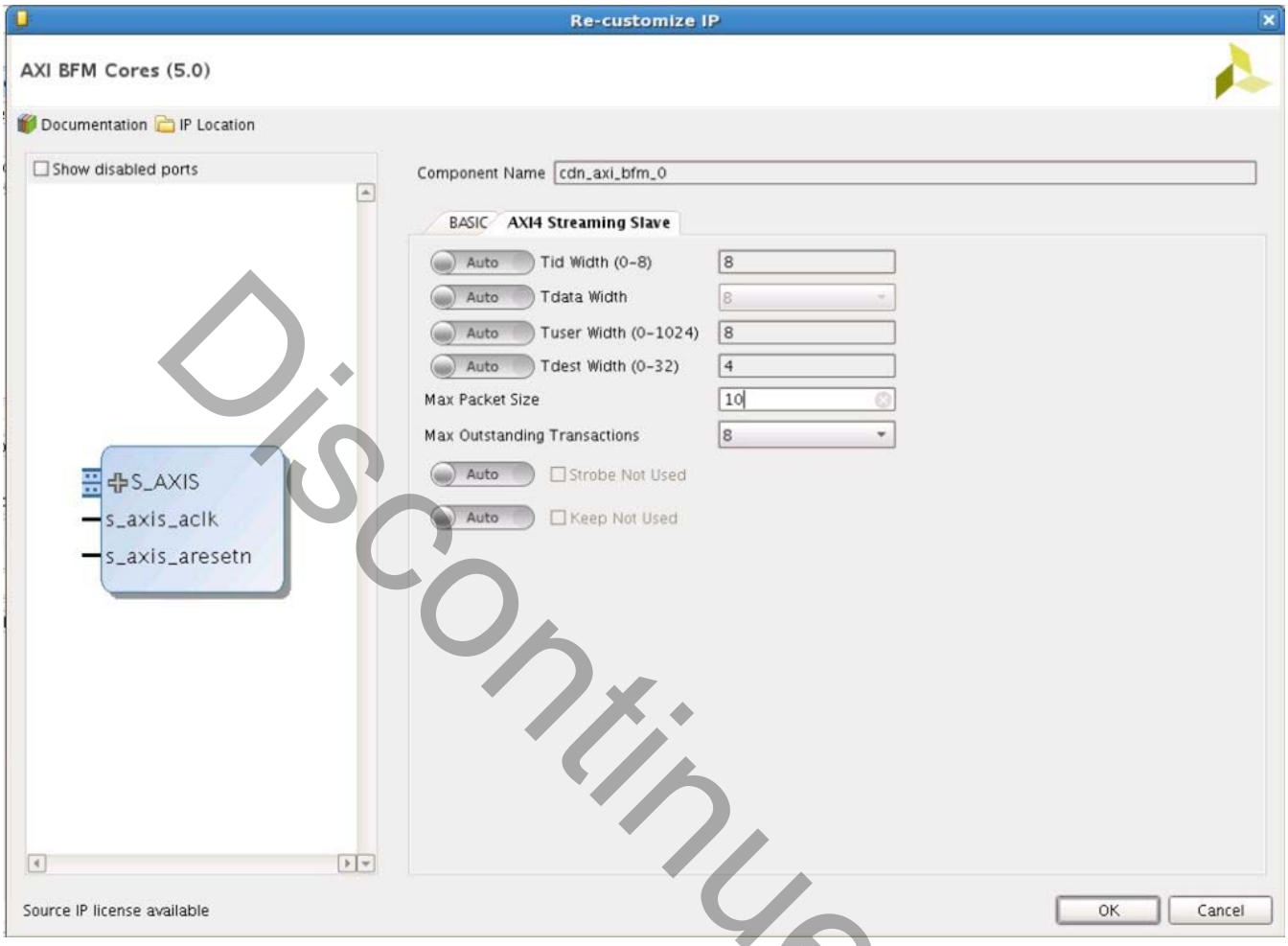

Figure 4-5: **AXI-Stream Slave Mode**

In this case, all of the parameters related to AXI-Stream slave interface are auto set based on the interface connection.

IMPORTANT: *By default these parameters are automatically updated in IP integrator. You can override the propagated value by changing the switch to* ***Manual****.*

AXI BFM cores can be used in IP integrator design to drive any of the supported interfaces. You can instantiate this IP in IP integrator just like any other IP. For more information on IP integrator, see *Vivado Design User Guide: Designing IP Subsystems Using IP Integrator* (UG994) [Ref 5].

IMPORTANT: *While using the AXI BFM cores in IP integrator, it is important to find the hierarchical path of BFM instance in the IP integrator generated wrapper so that it can be called/driven from a Verilog test bench.*

The following example explains how to determine the hierarchical path of any BFM instance in an IP integrator design.

Figure 4-6 shows a simple IP integrator design that has AXI4-Lite Master BFM on one side of AXI Interconnect and two peripherals on the other side of the AXI Interconnect.

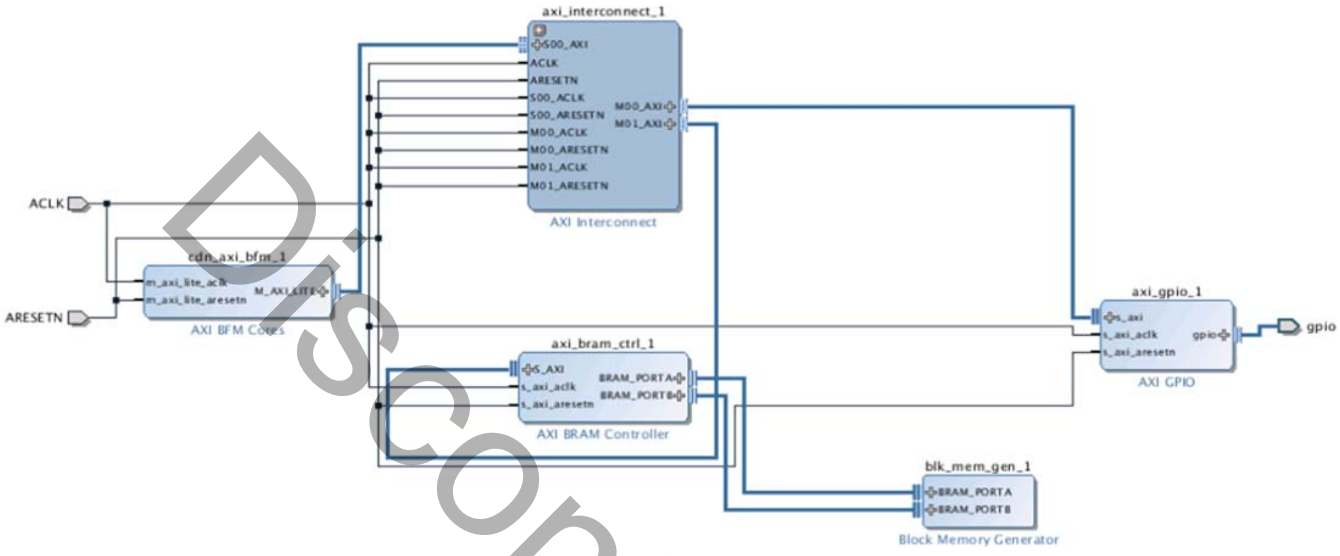

Figure 4-6: **Example IP Integrator Design**

Follow these guidelines to find the BFM hierarchical path:

1. Right-click **bd design** and select **Generate Output Products**.

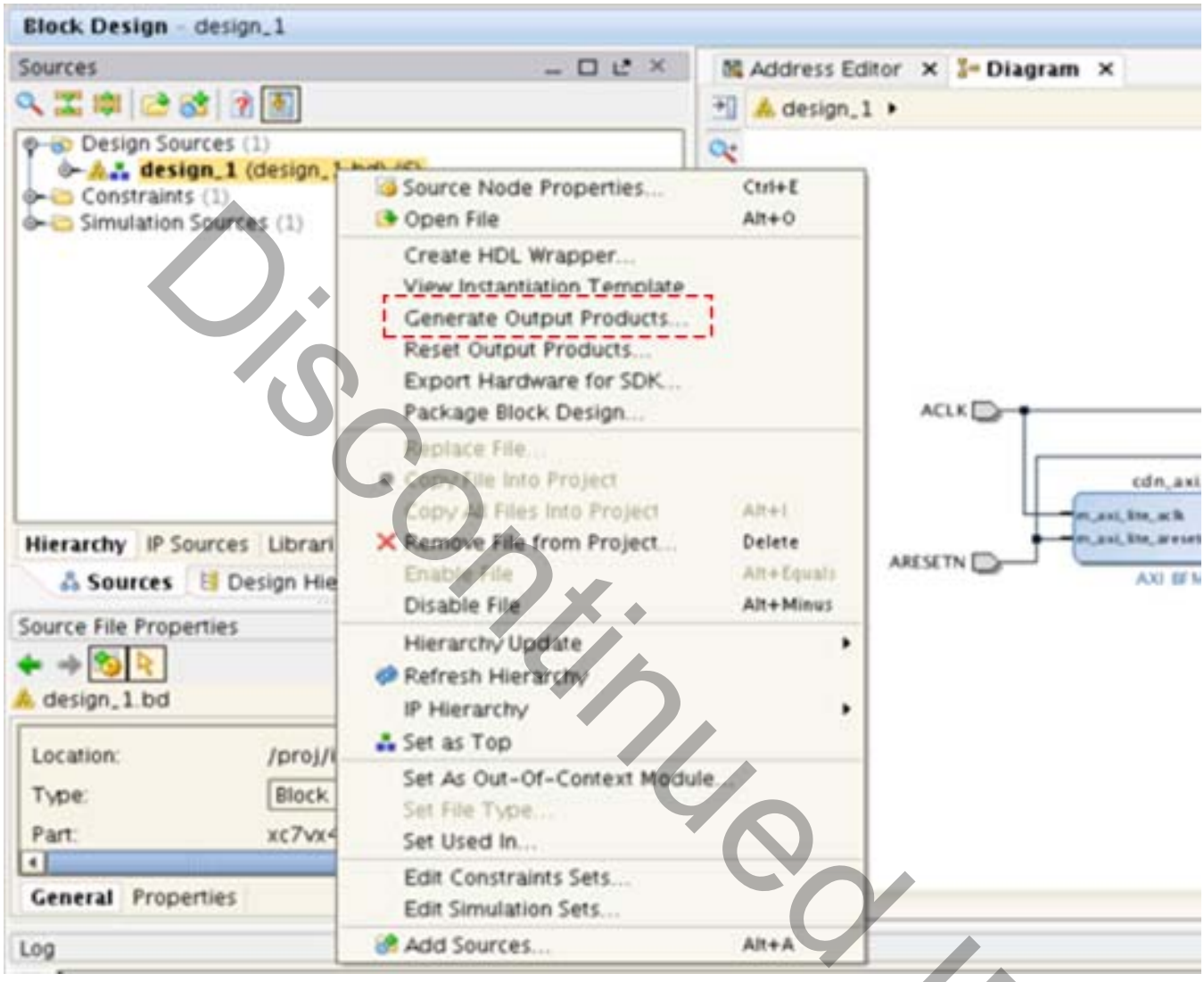

Figure 4-7: **Generate Output Products**

2. After the Output Products have been delivered, right-click **bd design** again and select **Create HDL Wrapper**.

   Note: AXI BFM cores support only Verilog language.

3. Figure 4-8 shows the complete hierarchy of the instances after the wrapper has been generated.

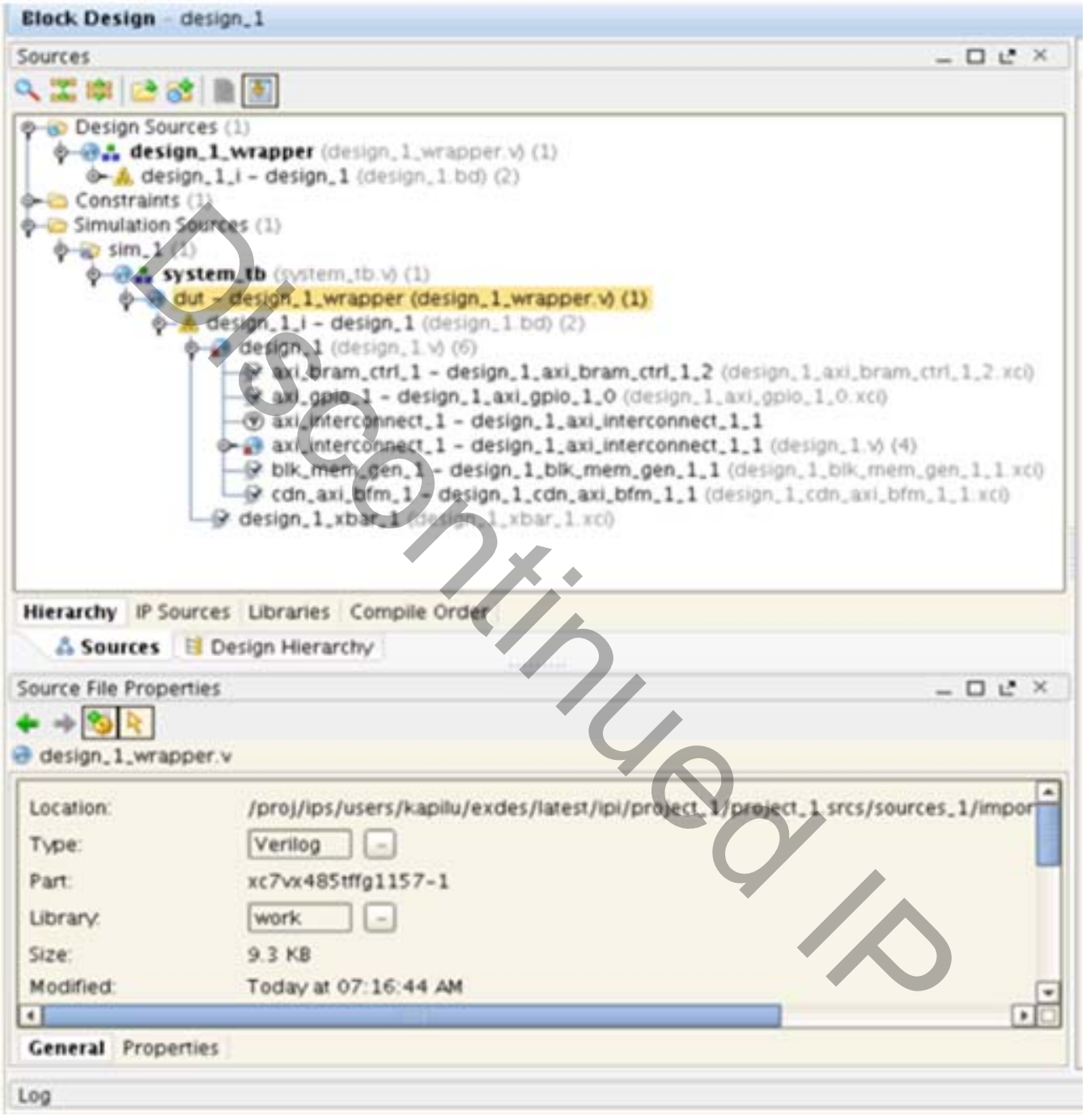

Figure 4-8: **Complete Design Hierarchy**

4. The generated wrapper should be used as DUT module in the test bench.

5. Figure 4-9 and Figure 4-10 show how to identify the BFM instance in the hierarchy. After the hierarchy is identified it can used in the Verilog test bench to drive the BFM APIs.

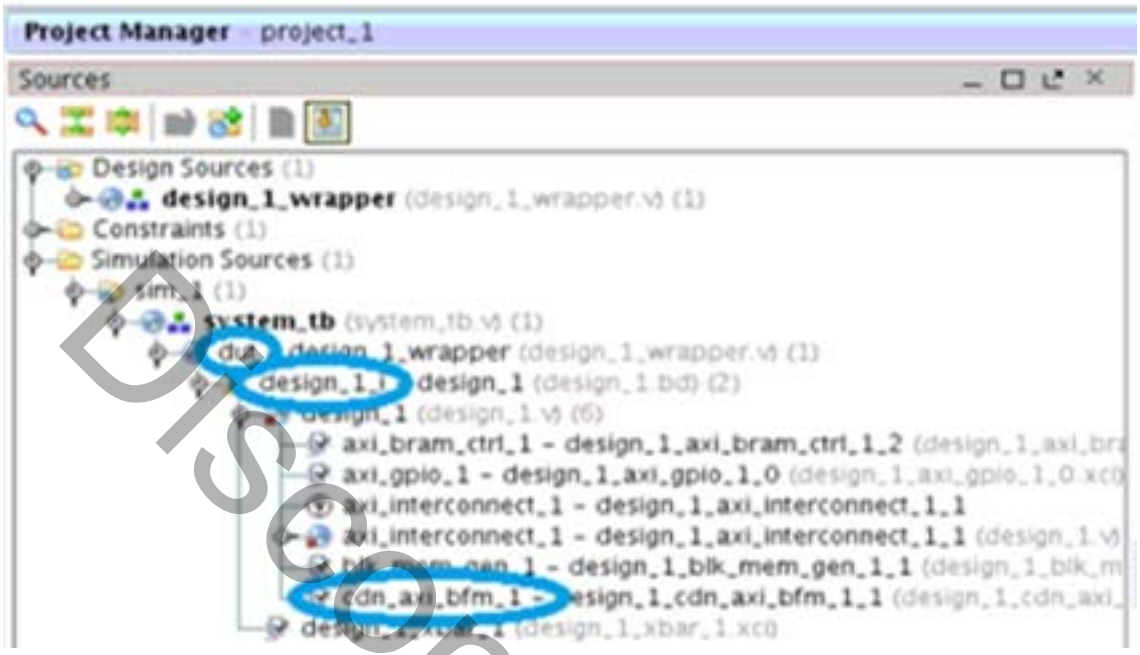

Figure 4-9: **BFM Instance in Design Hierarchy – Source Window**

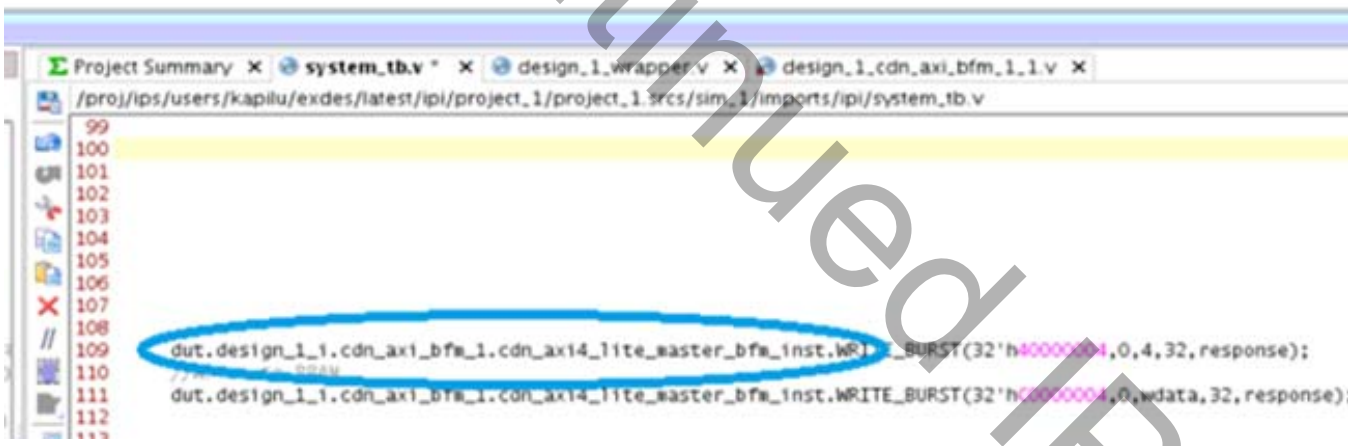

Figure 4-10: **BFM Instance in Design Hierarchy – system_tb.v Window**

User Parameters

Table 4-1 shows the relationship between the GUI fields in the Vivado IDE and the User Parameters (which can be viewed in the Tcl Console).

Table 4-1: **GUI Parameter to User Parameter Relationship**

GUI Parameter/Value[1]	User Parameter/Value[1]	Default Value[1]
Component_Name	C_XIP_MODE	cdn_axi_bfm_v5_0
Protocol	C_PROTOCOL_SELECTION	1
Interface	C_MODE_SELECT	0
Write Burst Transfer Data Gap	C_WRITE_BURST_TRANSFER_DATA_GAP	0
Response Timeout	C_RESPONSE_TIMEOUT	500
Disable Reset Value Checks	C_DISABLE_RESET_VALUE_CHECKS	0
Write ID Order Check Feature	C_WRITE_ID_ORDER_CHECK_FEATURE	0
Clear Signals After Handshake	C_CLEAR_SIGNALS_AFTER_HANDSHAKE	0
Error on SLVERR	C_ERROR_ON_SLVERR	0
Error on DECERR	C_ERROR_ON_DECERR	0
Stop on Error	C_STOP_ON_ERROR	1
Channel Level Info	C_CHANNEL_LEVEL_INFO	0
Function Level Info	C_FUNCTION_LEVEL_INFO	1
Read Burst Data Transfer Gap	C_READ_BURST_DATA_TRANSFER_GAP	0
Write Response Gap	C_WRITE_RESPONSE_GAP	0
Read Response Gap	C_READ_RESPONSE_GAP	0
Write Burst Address Data Phase Gap	C_WRITE_BURST_ADDRESS_DATA_PHASE_GAP	0
Write Burst Data Address Phase Gap	C_WRITE_BURST_DATA_ADDRESS_PHASE_GAP	0
Packet Transfer Gap	C_PACKET_TRANSFER_GAP	0
Input Signal Delay	C_INPUT_SIGNAL_DELAY	0
Task Reset Handling	C_TASK_RESET_HANDLING	0
Id Width (0–32)	C_M_AXI4_ID_WIDTH	4
Data Width	C_M_AXI4_DATA_WIDTH	32
Addr Width (12–64)	C_M_AXI4_ADDR_WIDTH	32
Awuser Width (0–1,024)	C_M_AXI4_AWUSER_WIDTH	1
Wuser Width (0–1,024)	C_M_AXI4_WUSER_WIDTH	1
Buser Width (0–1,024)	C_M_AXI4_BUSER_WIDTH	1
Aruser Width (0–1,024)	C_M_AXI4_ARUSER_WIDTH	1
Ruser Width (0–1,024)	C_M_AXI4_RUSER_WIDTH	1
Read/Write Issuing Depth	C_INTERCONNECT_M_AXI4_READ_ISSUING	8
Exclusive Access	C_M_AXI4_EXCLUSIVE_ACCESS_SUPPORTED	0

1. Parameter values are listed in the table where the GUI parameter value differs from the user parameter value. Such values are shown in this table as indented below the associated parameter.

Output Generation

For more information, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 3].

Constraining the Core

IMPORTANT: *This section is not applicable to this IP core.*

This section contains information about constraining the core in the Vivado Design Suite.

Required Constraints

This section is not applicable for this IP core.

Device, Package, and Speed Grade Selections

This section is not applicable for this IP core.

Clock Frequencies

This section is not applicable for this IP core.

Clock Management

This section is not applicable for this IP core.

Clock Placement

This section is not applicable for this IP core.

Banking

This section is not applicable for this IP core.

Transceiver Placement

This section is not applicable for this IP core.

I/O Standard and Placement

This section is not applicable for this IP core.

Simulation

This section contains information about simulating IP in the Vivado Design Suite. For comprehensive information about Vivado simulation components, as well as information about using supported third-party tools, see the *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 6].

The IP and its example design can be simulated directly from Vivado by clicking the **Run Simulation** button.

This version does not deliver any scripts.

Synthesis and Implementation

IMPORTANT: *This section is not applicable to this IP core.*

This section contains information about synthesis and implementation in the Vivado Design Suite. For details about synthesis and implementation, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 3].

Example Design

This chapter contains information about the example design provided in the Vivado® Design Suite.

IMPORTANT: *The example design of this IP is a generic one. It is not customized to the IP configuration. The intent of this example design is to demonstrate the usage of various APIs.*

Overview

This section describes the example tests used to demonstrate the abilities of each AXI BFM core pair. Example tests are delivered in Verilog. These example designs are available in the AXI_BFM installation area in the Tcl Console folder in encrypted format. When the core example design is open, the example files are delivered in standard path (`example.srcs/sim_1/imports/simulation`). Each AXI master is connected to a single AXI slave, and then direct tests are used to transfer data from the master to the slave and from the slave to the master.

RECOMMENDED: *The AXI BFM cores are not fully autonomous. For example, the AXI Master BFM is only a user-driven verification component that enables you to generate valid AXI protocol scenarios. Furthermore, if tests are written using the channel level API it is possible that the AXI protocol can be accidentally violated. For this reason, Xilinx® recommends using the function level API for each BFM.*

The *ARM® AMBA® AXI Protocol Specification*, Section 3.3, Dependencies between Channel Handshake Signals [Ref 7], states that:

- Slave can wait for AWVALID or WVALID, or both, before asserting AWREADY
- Slave can wait for AWVALID or WVALID, or both, before asserting WREADY

This implies that the slave does not need to support all three possible scenarios. However, if the AXI Master BFM operates in such a way that is not supported by the slave, then the simulation stalls. Each scenario is handled by the function level API:

Scenario 1

Before the slave asserts AWREADY and/or WREADY, the slave can wait for AWVALID. This is modeled using the function level API, WRITE_BURST.

Scenario 2

Before the slave asserts AWREADY and/or WREADY, the slave can wait for WVALID. This is modeled using the function level API, WRITE_BURST_DATA_FIRST.

Scenario 3

Before the slave asserts AWREADY and/or WREADY, the slave can wait for both AWVALID and WVALID. This is modeled using the function level API, WRITE_BURST_CONCURRENT.

Using AXI BFM Cores for Standalone RTL Design

The AXI BFM cores can be used to verify connectivity and basic functionality of AXI masters and AXI slaves with the custom RTL design flow. The AXI BFM provides example test benches and tests that demonstrate the abilities of AXI3, AXI4, AXI4-Lite, and AXI4-Stream Master/Slave BFM pair. These examples can be used as a starting point to create tests for custom RTL design with AXI3, AXI4, AXI4-Lite, and AXI4-Stream interface.

Test Bench

This chapter contains information about the test bench provided in the Vivado® Design Suite.

AXI3 BFM Example Test Bench and Test

The Verilog example test bench and example test case for the AXI3 BFM is shown in Figure 6-1.

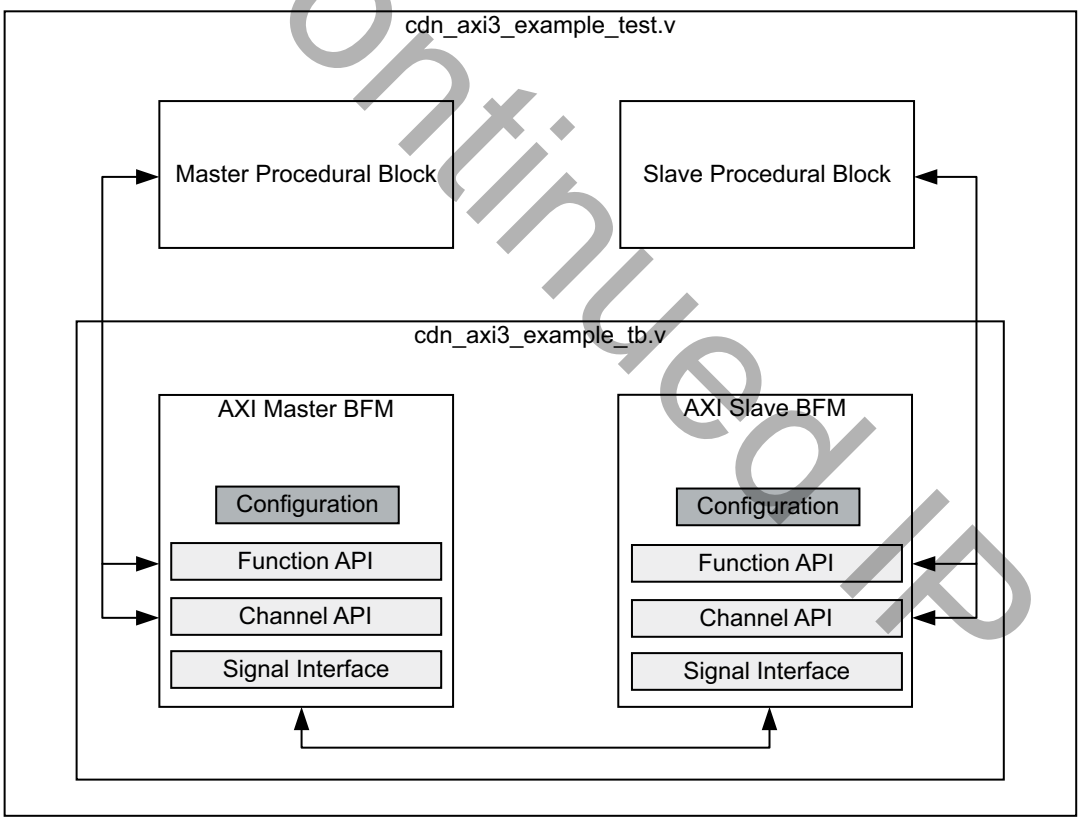

Figure 6-1: **Verilog Example Test Bench and Test Case Structure**

The example test bench has the master and slave BFM connected directly to each other. This gives visibility into both sides of the code (master code and slave code) required to hit the scenarios detailed in the example test.

cdn_axi3_example_test.v

The example test (`simulation/cdn_axi3_example_test.v`) contains the master and slave test code to simulate the following scenarios:

- Simple sequential write and read burst transfers example.
- Looped sequential write and read transfers example.
- Parallel write and read burst transfers example.
- Narrow write and read transfers example.
- Unaligned write and read transfers example.
- Narrow and unaligned write and read transfers example.
- Out of order write and read burst example.
- Write Bursts performed in two different ways; Data before address and data with address concurrently.
- Write data interleaving example.
- Read data interleaving example.
- Outstanding transactions example.
- Slave read and write bursts error response example.
- Write and read bursts with different length gaps between data transfers example.
- Write and Read bursts with different length gaps between channel transfers example.
- Write burst that is longer than the data it is sending example.

AXI4 BFM Example Test Bench and Test

The AXI4 Verilog example test bench structure is identical to the one used for AXI3 shown in Figure 6-1. The following section provides details about the example test available.

cdn_axi4_example_test.v

The example test (`simulation/cdn_axi4_example_test.v`) contains the master and slave test code to simulate the following scenarios:

- Simple sequential write and read burst transfers example.
- Looped sequential write and read transfers example.
- Parallel write and read burst transfers example.
- Narrow write and read transfers example.

- Unaligned write and read transfers example.
- Narrow and unaligned write and read transfers example.
- Write Bursts performed with address and data channel transfers concurrently.
- Outstanding transactions example.
- Slave read and write bursts error response example.
- Write and read bursts with different length gaps between data transfers example.
- Write and Read bursts with different length gaps between channel transfers example.
- Write burst that is longer than the data it is sending example.
- Read data interleaving example.

AXI4-Lite BFM Example Test Bench and Test

The AXI4-Lite Verilog example test bench structure is identical to the one used for AXI3 shown in Figure 6-1. The following section provides details about the example test available.

cdn_axi4_lite_example_test.v

The example test (`simulation/cdn_axi4_lite_example_test.v`) contains the master and slave test code to simulate the following scenarios:

- Simple sequential write and read burst transfers example.
- Looped sequential write and read transfers example.
- Parallel write and read burst transfers example.
- Write Bursts performed in two different ways; Data before address and data with address concurrently.
- Outstanding transactions example.
- Slave read and write bursts error response example.
- Write and Read bursts with different length gaps between channel transfers example.
- Unaligned write and read transfers example.
- Write burst that has valid data size less than the data bus width.

AXI4-Stream BFM Example Test Bench and Test

The AXI4-Stream Verilog example test bench structure is identical to the one used for AXI3 shown in Figure 6-1. The following section provides details about the example test available.

cdn_axi4_streaming_example_test.v

The example test (`simulation/cdn_axi4_streaming_example_test.v`) contains the master and slave test code to simulate the following scenarios:

- Simple master to slave transfer example.
- Looped master to slave transfers example.
- Simple master to slave packet example.
- Looped master to slave packet example.
- Ragged (less data at the end of the packet than can be supported) master to slave packet example.
- Packet data interleaving example.

Useful Coding Guidelines and Examples

Loop Construct Simple Example

While coding directed tests, "for loops" are typically employed frequently to efficiently generate large volumes of stimulus for both the master and/or slave BFM. For example:

```
for (m=0;m<2;m =m+1) begin  // Burst Type variable
 for (k=0;k<3;k=k+1)  begin  // Burst Size variable
  $display("-------------------------------------------------");
  $display("EXAMPLE TEST LOCKED and NORMAL ");
  $display("-------------------------------------------------");

  for (i=0; i<16;i=i+1) begin // Burst Length variable
   tb.master_0.WRITE_BURST(mtestID+i,  // Master ID
                           mtestAddr,  // Master Address
                           i,          // Master Burst Length
                           k,          // Master Burst Size
                           m,     // Master Access Type FIXED, INCR
                           `LOCKED_TYPE_FIXED, // Use define
                           4'b0000,     // Buffer/Cachable Hardcoded
                           3'b000,      // Protection Type Hardcoded
                       test_data[i],// Write Data from array
                           response,    // response from slave
    end
  end
end
```

This "for loop" cycles through the following stimulus:

- Access Type (m) – FIXED, INCR
- Burst Size (k) – 1_BYTE, 2_BYTES, 4_BYTES
- Burst Length (i) – 1 to 16

Nested for loops can be used to generate numerous combinations of traffic types, but care must be taken to not violate protocol. AXI BFM cores check the input parameters of the API calls, but this does not prevent higher level protocol being violated.

Loop Construct Complex Example

In some cases, a nested for loop can lead to invalid stimulus if not used correctly. A good example of this is WRAP bursts. The AXI Specification requires that WRAP bursts must be 2, 4, 8, or 16 transfers in length. For this type of burst, the nested for loop from the Loop Construct Simple Example cannot be used because the nested for loop cycles through burst lengths of 1 to 16. For exhaustive WRAP tests, another for loop declaration is widely used to drive legal stimulus:

```
for (i=2; i <= 16; i=i*2) begin
```

Thus giving a burst length of 2, 4, 8, and 16 transfers.

DUT Modeling Using the AXI BFM Cores – Memory Model Example

In most cases, the behavior of a master or slave is more complicated than simple transfer generation. For this reason, AXI BFM API enables you to model higher level DUT functionality. A simple example is a slave memory model. Such a memory model is available as a configuration option in most of the AXI slave BFM. This example shows the code used for the AXI3 Slave BFM memory model mode, starting with the write datapath.

```
//------------------------------------------------------------------// Write Path
//------------------------------------------------------------------
always @(posedge ACLK) begin : WRITE_PATH
  //----------------------------------------------------------------
  //- Local Variables
  //----------------------------------------------------------------
  reg [ID_BUS_WIDTH-1:0] id;
  reg [ADDRESS_BUS_WIDTH-1:0] address;
  reg [`LENGTH_BUS_WIDTH-1:0] length;
  reg [`SIZE_BUS_WIDTH-1:0] size;
  reg [`BURST_BUS_WIDTH-1:0] burst_type;
  reg [`LOCK_BUS_WIDTH-1:0] lock_type;
  reg [`CACHE_BUS_WIDTH-1:0] cache_type;
  reg [`PROT_BUS_WIDTH-1:0] protection_type;
  reg [ID_BUS_WIDTH-1:0] idtag;
  reg [(DATA_BUS_WIDTH*(`MAX_BURST_LENGTH+1))-1:0] data;
  reg [ADDRESS_BUS_WIDTH-1:0] internal_address;
  reg [`RESP_BUS_WIDTH-1:0] response;
  integer i;
  integer datasize;
  //----------------------------------------------------------------
  // Implementation Code
  //----------------------------------------------------------------
  if (MEMORY_MODEL_MODE == 1) begin
    // Receive the next available write address
    RECEIVE_WRITE_ADDRESS(id,`IDVALID_FALSE,address,length,size,
    burst_type,lock_type,cache_type,protection_type,idtag);
    // Get the data to send to the memory.
    RECEIVE_WRITE_BURST(idtag,`IDVALID_TRUE,address,length,size,
    burst_type,data,datasize,idtag);
    // Put the data into the memory array
    internal_address = address - SLAVE_ADDRESS;
    for (i=0; i < datasize; i=i+1) begin
      memory_array[internal_address+i] = data[i*8 +: 8];
    end
    // End the complete write burst/transfer with a write response
    // Work out which response type to send based on the lock type.
    response = calculate_response(lock_type);
    repeat(WRITE_RESPONSE_GAP) @(posedge ACLK);
    SEND_WRITE_RESPONSE(idtag,response);
  end
end
```

As shown in the code, it is possible to create the write datapath for a simple memory model using three of the tasks from the slave channel level API. This is achieved in the following four steps:

1. Wait for any write address request on the write address bus. This is done by calling RECEIVE_WRITE_ADDRESS with IDVALID_FALSE. This ensures that the first detected and valid write address handshake is recorded and the details passed back. This task is blocking; so the WRITE_PATH process does not proceed until it has found a write address channel transfer.
2. Get the write data burst that corresponds to the write address request in the previous step. This is done by calling RECEIVE_WRITE_BURST with the ID tag output from the RECEIVE_WRITE_ADDRESS call and with IDVALID_TRUE. This ensures that the entire write data burst that has the specified id tag is captured before execution returns to the WRITE_PATH process.
3. Take the data from the write data burst and put it into a memory array. In this case, the memory array is an array of bytes.
4. Complete the AXI3 protocol is to send a response. The internal function calculate_reponse is used to work out if the transfer was exclusive or not and to deliver an EXOKAY or OK response (more code could be added here to support DECERR or SLVERR response types). When the response has been calculated, the WRITE_PATH process waits for the defined internal control variable WRITE_RESPONSE_GAP in clock cycles before sending the response back to the slave with the same ID tag as the write data transfer.

The following code illustrates the steps required to make the read datapath for a simple slave memory model:

```
//------------------------------------------------------------
// Read Path
//------------------------------------------------------------always @(posedge
ACLK) begin : READ_PATH
  //------------------------------------------------------------
  // Local Variables
  //------------------------------------------------------------
  reg [ID_BUS_WIDTH-1:0] id;
  reg [ADDRESS_BUS_WIDTH-1:0] address;
  reg [`LENGTH_BUS_WIDTH-1:0] length;
  reg [`SIZE_BUS_WIDTH-1:0] size;
  reg [`BURST_BUS_WIDTH-1:0] burst_type;
  reg [`LOCK_BUS_WIDTH-1:0] lock_type;
  reg [`CACHE_BUS_WIDTH-1:0] cache_type;
  reg [`PROT_BUS_WIDTH-1:0] protection_type;
  reg [ID_BUS_WIDTH-1:0] idtag;
  reg [(DATA_BUS_WIDTH*(`MAX_BURST_LENGTH+1))-1:0] data;
  reg [ADDRESS_BUS_WIDTH-1:0] internal_address;
  integer i;
  integer number_of_valid_bytes;
  //------------------------------------------------------------
  // Implementation Code
  //------------------------------------------------------------
```

```
  if (MEMORY_MODEL_MODE == 1) begin
  // Receive a read address transfer
  RECEIVE_READ_ADDRESS(id,`IDVALID_FALSE,address,length,size,
  burst_type,lock_type,cache_type,protection_type,idtag);
  // Get the data to send from the memory.
  internal_address = address - SLAVE_ADDRESS;
  data = 0;
  number_of_valid_bytes =
(decode_burst_length(length)*transfer_size_in_bytes(size))-(address %
(DATA_BUS_WIDTH/8));

  for (i=0; i < number_of_valid_bytes; i=i+1) begin
    data[i*8 +: 8] = memory_array[internal_address+i];
  end
  // Send the read data
  repeat(READ_RESPONSE_GAP) @(posedge ACLK);
  SEND_READ_BURST(idtag,address,length,size,burst_type,
  lock_type,data);
  end
end
```

As shown in the code, it is possible to create the read datapath for a simple memory model using two of the tasks from the slave channel level API. This is achieved in the following two steps:

1. Wait for any read address request on the read address bus. This is done by calling RECEIVE_READ_ADDRESS with IDVALID_FALSE. This ensures that the first detected and valid read address handshake is recorded and the details are passed back. This task is blocking; so the READ_PATH process does not proceed until it has found a read address channel transfer.
2. Take the requested data from the memory array and send it in a read burst. This is done by extracting the data byte by byte into a data vector which is used as an input into the SEND_READ_BURST task. Before sending the read data burst, the READ_PATH process waits for the clock cycles determined in the internal control variable READ_RESPONSE_GAP.

Appendix A

Verification, Compliance, and Interoperability

AXI BFM cores are compliant to AXI3, AXI4, AXI4-Lite, and AXI4-Stream protocols.

Migrating and Upgrading

This appendix contains information about migrating a design from ISE® to the Vivado® Design Suite, and for upgrading to a more recent version of the IP core. For customers upgrading in the Vivado Design Suite, important details (where applicable) about any port changes and other impact to user logic are included.

Migrating to the Vivado Design Suite

For information on migrating to the Vivado Design Suite, see *ISE to Vivado Design Suite Migration Guide* (UG911) [Ref 8].

Upgrading in the Vivado Design Suite

This section provides information about any changes to the user logic or port designations that take place when you upgrade to a more current version of this IP core in the Vivado Design Suite.

There is no special instructions for migration except that all of the wrappers are unified into the AXI BFM cores.

Debugging

This appendix includes details about resources available on the Xilinx® Support website and debugging tools.

Finding Help on Xilinx.com

To help in the design and debug process when using the AXI BFM, the Xilinx Support web page contains key resources such as product documentation, release notes, answer records, information about known issues, and links for obtaining further product support.

Documentation

This product guide is the main document associated with the AXI BFM. This guide, along with documentation related to all products that aid in the design process, can be found on the Xilinx Support web page or by using the Xilinx Documentation Navigator.

Download the Xilinx Documentation Navigator from the Downloads page. For more information about this tool and the features available, open the online help after installation.

Answer Records

Answer Records include information about commonly encountered problems, helpful information on how to resolve these problems, and any known issues with a Xilinx product. Answer Records are created and maintained daily ensuring that users have access to the most accurate information available.

Answer Records for this core can be located by using the Search Support box on the main Xilinx support web page. To maximize your search results, use proper keywords such as:

- Product name
- Tool message(s)
- Summary of the issue encountered

A filter search is available after results are returned to further target the results.

Master Answer Record for the AXI BFM

AR: 54678

Technical Support

Xilinx provides technical support in the Xilinx Support web page for this LogiCORE™ IP product when used as described in the product documentation. Xilinx cannot guarantee timing, functionality, or support if you do any of the following:

- Implement the solution in devices that are not defined in the documentation.
- Customize the solution beyond that allowed in the product documentation.
- Change any section of the design labeled DO NOT MODIFY.

To contact Xilinx Technical Support, navigate to the Xilinx Support web page.

Interface Debug

AXI4-Lite Interfaces

Read from a register that does not have all 0s as a default to verify that the interface is functional. If the interface is unresponsive, ensure that the following conditions are met:

- The `s_axi_aclk and aclk` inputs are connected and toggling.
- The interface is not being held in reset, and `s_axi_areset` is an active-Low reset.
- The interface is enabled, and `s_axi_aclken` is active-High (if used).
- The main core clocks are toggling and that the enables are also asserted.
- If the simulation has been run, verify in simulation and/or Vivado lab tools capture that the waveform is correct for accessing the AXI4-Lite interface.

Additional Resources and Legal Notices

Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see Xilinx Support.

References

These documents provide supplemental material useful with this product guide:

1. *Cadence AXI UVC User Guide* (VIPP 9.2/VIPP 10.2 releases)
2. *ARM® AMBA® AXI4-Stream Protocol v1.0 Specification* (ARM IHI 0051A)
3. *Vivado® Design Suite User Guide: Designing with IP* (UG896)
4. *Vivado Design Suite User Guide: Getting Started* (UG910)
5. *Vivado Design Suite User Guide: Designing IP Subsystems Using IP Integrator* (UG994)
6. *Vivado Design Suite User Guide: Logic Simulation* (UG900)
7. *ARM AMBA AXI Protocol v2.0 Specification* (ARM IHI 0022C)
8. *ISE® to ISE to Vivado Design Suite Migration Guide* (UG911)
9. *LogiCORE™ IP AXI Interconnect Product Guide* (PG059)

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
11/18/2015	5.0	• Added support for UltraScale+ families. • Updated description in Example Design Overview section.
10/01/2014	5.0	• Document updates only for revision change. • Added BFM Limitations in Overview chapter. • Updated tables in AXI BFM Cores Design Parameters section. • Added WRITE_ID_ORDER_CHECK_FEATURE in Table 3-1: AXI3 Master BFM Parameters and Table 3-2: AXI3 Slave BFM Parameters. • Added DISABLE_RESET_VALUE_CHECKS in Table 3-1: AXI3 Master BFM Parameters to Table 3-6: AXI4-Lite Slave BFM Parameters and Table 3-8: AXI4-Stream BFM Parameters. • Deleted ERROR rows in Table 3-7: AXI4-Stream BFM Parameters. • Added set_response_timeout, set_input_signal_delay, set_disable_reset_value_checks, and set_write_id_order_check_feature_value in Table 3-9: Utility API Tasks/ Functions. • Added Argument Data Types to APIs and API Instantiation Example sections in Test Writing API. • Updated descriptions in Table 3-10: Channel Level API for AXI3 Master BFM. • Updated descriptions in Table 3-12: Channel Level API for AXI3 Slave BFM. • Updated descriptions in Table 3-14: Channel Level API for AXI4 Master BFM. • Updated descriptions in Table 3-16: Channel Level API for AXI4 Slave BFM. • Updated descriptions in Table 3-18: Channel Level API for AXI4-Lite Master BFM. • Updated descriptions in Table 3-20: Channel Level API for AXI4-Lite Slave BFM. • Updated descriptions in Table 3-22: Channel Level API for AXI4-Stream Master BFM. • Updated descriptions in Table 3-23: Channel Level API for AXI4-Stream Slave BFM. • Added Design Flow Steps chapter. • Updated GUIs in Customizing and Generating the Core section. • Added BFM Instantiations Names section in Design Flow chapter. • Added User Parameter section in Design Flow chapter. • Added Important note in the Example Design chapter.
12/18/2013	5.0	Added UltraScale support.

Date	Version	Revision
10/02/2013	5.0	• Updated Figs. 4-1 to 4-2. • Added Using AXI BFM Cores in Vivado IP Integrator section. • Added Output Generation in Generating the Core chapter. • Added Simulation, Synthesis, and Test Bench chapters. • Updated Migrating Appendix. • Updated Debug Appendix.
06/19/2013	4.1	• Revision number advanced to 4.1 to align with core version number. • Updated DATA_BUS_WIDTH and added CLEAR_SIGNALS_AFTER_HANDSHAKE, ERROR_ON_SLVERR, and ERROR_ON_DECERR in Table 3-1 AXI3 Master BFM Parameters. • Updated DATA_BUS_WIDTH and added CLEAR_SIGNALS_AFTER_HANDSHAKE in Table 3-2 AXI3 Slave BFM Parameters. • Updated DATA_BUS_WIDTH and added CLEAR_SIGNALS_AFTER_HANDSHAKE, ERROR_ON_SLVERR, and ERROR_ON_DECERR in Table 3-3 AXI4 Master BFM Parameters. • Updated DATA_BUS_WIDTH and added CLEAR_SIGNALS_AFTER_HANDSHAKE in Table 3-4 AXI4 Slave BFM Parameters. • Added CLEAR_SIGNALS_AFTER_HANDSHAKE, ERROR_ON_SLVERR, and ERROR_ON_DECERR in Table 3-5 AXI4-Lite Master BFM Parameters. • Added CLEAR_SIGNALS_AFTER_HANDSHAKE in Table 3-6 AXI4-Lite Slave BFM Parameters. • Updated DATA_BUS_WIDTH and added CLEAR_SIGNALS_AFTER_HANDSHAKE, ERROR_ON_SLVERR, and ERROR_ON_DECERR in Table 3-7 AXI4-Stream BFM Parameters. • Updated DATA_BUS_WIDTH in Table 3-8 AXI4-Stream Slave BFM Parameters. • Added set_clear_signals_after_handshake, set_error_on_slverr, and set_error_on_decerr in Table 3-9 Utility API Tasks/Functions. • Added Inputs description in Table 3-23 Channel Level API for AXI4-Stream Slave BFM. • Updated Figs. 4-1 to 4-2.
03/20/2013	1.0	Initial Xilinx release of the product guide and replaces DS824.

Please Read: Important Legal Notices