

LogiCORE IP AXI Bus Functional Models v4.0

Product Guide for Vivado Design Suite

PG129 March 20, 2013

Table of Contents

IP Facts

Chapter 1: Overview

Configuration Options	6
Applications	7
Licensing and Ordering Information	8

Chapter 2: Product Specification

Standards	9
-----------------	---

Chapter 3: Designing with the Core

AXI BFM Design Parameters	10
Test Writing API	23
Protocol Description	49

Chapter 4: Customizing and Generating the Core

Vivado Integrated Design Environment (IDE)	50
--	----

Chapter 5: Detailed Example Design

Example Design	54
Using AXI BFM for Standalone RTL Design	55
Demonstration Test Bench	56
Simulation	62

Appendix A: Verification, Compliance, and Interoperability

Appendix B: Migrating

Appendix C: Debugging

Finding Help on Xilinx.com	65
Interface Debug	67

Appendix D: Additional Resources

Xilinx Resources	68
References	68
Revision History	69
Notice of Disclaimer.....	69

Introduction

The Xilinx LogiCORE™ IP AXI Bus Functional Models (BFMs), developed for Xilinx by Cadence® Design Systems, support the simulation of customer-designed AXI-based IP. AXI BFM support all versions of AXI (AXI3, AXI4, AXI4-Lite, and AXI4-Stream). The BFM 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⁾	Zynq™-7000, Virtex®-7, Kintex™-7, Artix™-7
Supported User Interfaces	AXI4, AXI4-Lite, AXI4-Stream, AXI3
Resources	N/A
Provided with Core	
Design Files	RTL
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
Simulation	Mentor Graphics Questa® SIM Vivado Simulator
Synthesis	N/A
Support	
Provided by Xilinx @ www.xilinx.com/support	

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 for AXI3, AXI4, AXI4-Lite, and AXI4-Stream interfaces.

Overview

The general AXI Bus Functional Model (BFM) architecture is shown in [Figure 1-1](#).

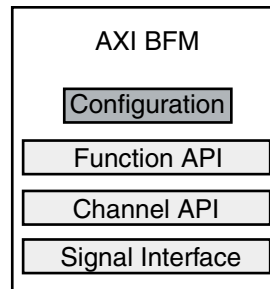


Figure 1-1: AXI BFM Architecture

All of the AXI BFM 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 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 is shown in [Figure 1-2](#).

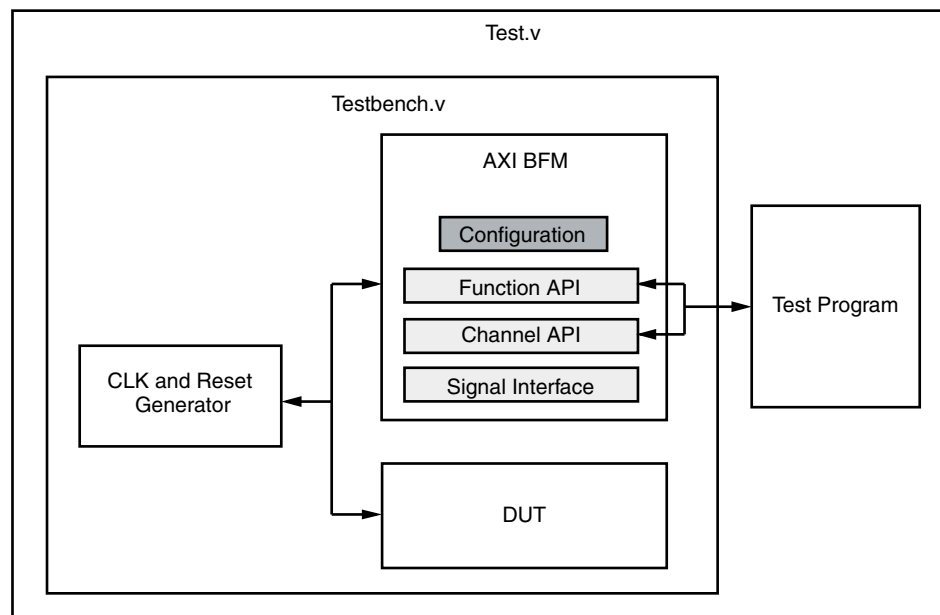


Figure 1-2: AXI BFM Use Case

[Figure 1-2](#) shows a single AXI BFM. However, the test bench can contain multiple instances of AXI BFM. The DUT and the AXI BFM 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 [Example Design](#) for practical examples of test programs and test bench setups.

Configuration Options

In most cases, the configuration options are passed to the BFM through Verilog parameters. BFM 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 BFM 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 is to verify connectivity and basic functionality of AXI masters and AXI slaves. A basic level of protocol checking is included with the AXI BFM. For comprehensive protocol checking, the Cadence AXI UVC [\[Ref 7\]](#) should be deployed.

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

- 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 Specific Checkers

[Table 1-1](#) details the Verilog checking tasks added to each BFM for a specific check. These checkers are only required for the BFM that they are located in; so, they are not included in a common file.

Table 1-1: BFM Specific Checker Tasks

Checker Task Name	Inputs	Checker Locations	Description
check_address_range	ADDRESS BURST_TYPE LENGTH	SLAVE BFM	Checks to see if address is valid with respect to the SLAVE configuration, the burst_type and length.
check_strobe	STROBE TRANSFER_NUMBER ADDRESS LENGTH SIZE BURST_TYPE	SLAVE BFM	Checks to see if the input strobe is correct. This check handles normal, narrow and unaligned transfers.

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](#).

Product Specification

Standards

The AXI BFM is 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 Design Parameters

AXI3 BFM



TIP: File name is the same as the module name as specified by you.

The AXI3 BFM modules and files are named as follows:

- MASTER BFM
 - Module Name: `cdn_axi_bfm_v4_0_0`
 - File Name: `cdn_axi3_master_bfm.v`
- SLAVE BFM
 - Module Name: `cdn_axi_bfm_v4_0_0`
 - File Name: `cdn_axi3_slave_bfm.v`

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
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, or 256 bits wide. Default is 32.
ADDRESS_BUS_WIDTH	Default is 32.
ID_BUS_WIDTH	Default is 4.

Table 3-1: AXI3 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.
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.
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 <i>and</i> 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.
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 (1) means stop on error. This configuration variable can be changed during simulation for error testing. 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.

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
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, or 256 bits wide. Default is 32.
ADDRESS_BUS_WIDTH	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.
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.
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. This configuration variable can be changed during simulation.

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

BFM Parameters	Description
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: This configuration variable can be changed during simulation.
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. Note: This configuration variable can be changed during simulation.
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. This configuration variable can be changed during simulation.
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. This configuration variable can be changed during simulation for error testing. 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.

AXI4 BFM



TIP: File name is the same as the module name as specified by you.

The AXI4 BFM modules and files are named as follows:

- Full Master BFM
 - Module Name: `cdn_axi_bfm_v4_0_0`
 - File Name: `cdn_axi4_master_bfm.v`
- Full Slave BFM

- Module Name: `cdn_axi_bfm_v4_0_0`
- File Name: `cdn_axi4_slave_bfm.v`
- Lite Master BFM
 - Module Name: `cdn_axi_bfm_v4_0_0`
 - File Name: `cdn_axi4_lite_master_bfm.v`
- Lite Slave BFM
 - Module Name: `cdn_axi_bfm_v4_0_0`
 - File Name: `cdn_axi4_lite_slave_bfm.v`
- Streaming Master BFM
 - Module Name: `cdn_axi_bfm_v4_0_0`
 - File Name: `cdn_axi4_streaming_master_bfm.v`
- Streaming Slave BFM
 - Module Name: `cdn_axi_bfm_v4_0_0`
 - File Name: `cdn_axi4_streaming_slave_bfm.v`

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
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, or 256 bits wide. Default is 32.
ADDRESS_BUS_WIDTH	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.
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.

Table 3-3: AXI4 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.
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 read bursts are called, then the BFM attempts to perform read data interleaving.
WRITE_BURST_ADDRESS_DATA_PHASE_GAP	This configuration variable can be set dynamically during the run of a test. 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	This configuration variable can be set dynamically during the run of a test. 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.
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. This configuration variable can be changed during simulation for error testing. 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.

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
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, or 256 bits wide. Default is 32.
ADDRESS_BUS_WIDTH	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.
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.
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.

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

BFM Parameters	Description
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 <i>and</i> 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. This configuration variable can be changed during simulation.
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. This configuration variable can be changed during simulation.
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. This configuration variable can be changed during simulation.
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. This configuration variable can be changed during simulation for error testing. 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.

AXI4-Lite Master BFM

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

Table 3-5: AXI4-Lite Master BFM Parameters

BFM Parameters	Description
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	Default is 32.
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.
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 one stops the simulation on an error. This configuration variable can be changed during simulation for error testing. 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.

AXI4-Lite Slave BFM

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

Table 3-6: AXI4-Lite Slave BFM Parameters

BFM Parameters	Description
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	Default is 32.
SLAVE_ADDRESS	This is the start address of the slave memory range.

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

BFM Parameters	Description
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.
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. This configuration variable can be changed during simulation.
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. This configuration variable can be changed during simulation.
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. This configuration variable can be changed during simulation.
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. This configuration variable can be changed during simulation for error testing. Note: This is not used for timeout errors; such errors always stop simulation.

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

BFM Parameters	Description
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.

AXI4-Stream Master BFM

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

Table 3-7: AXI4-Stream BFM Parameters

BFM Parameters	Description
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 256, 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 signals. <ul style="list-style-type: none"> • 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 keeping unused signals. <ul style="list-style-type: none"> • 0 = Keep signals used • 1 = Keep signals not used The default is 0. Changing the value to 1 disables the associated checks.

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

BFM Parameters	Description
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. This configuration variable can be changed during simulation for error testing. 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.

AXI4-Stream Slave BFM

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

Table 3-8: AXI4-Stream Slave BFM Parameters

BFM Parameters	Description
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 256, 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.

Table 3-8: AXI4-Stream Slave 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.
STROBE_NOT_USED	Enables and disables the strobe signals. <ul style="list-style-type: none"> • 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 keeping unused signals. <ul style="list-style-type: none"> • 0 = Keep signals used • 1 = Keep signals not used The default is 0. Changing the value to 1 only disables the associated checks.
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. This configuration variable can be changed during simulation.
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. This configuration variable can be changed during simulation for error testing. 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.

Test Writing API

The test writing API starts simple and is layered 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 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 6]*. 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](#).

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: $\text{report_status} = \text{error_count} + \text{warning_count} + \text{pending_transactions_count}$
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_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_bfm_clk_delay This task sets the internal variable BFM_CLK_DELAY to the specified input value. This is used to move the BFM internal clock off the simulation clock edge if needed. The default value is zero. If used it must be applied to each BFM separately.	CLK_DELAY: An integer value used for the #BFM_CLK_DELAY on BFM internal clocking.	None

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

API Task Name and Description	Inputs	Outputs
<p>set_task_call_and_reset_handling</p> <p>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</p>	<p>task_reset_handling: An integer value used to define BFM behavior during reset when a channel level API task is called.</p>	None
<p>remove_pending_transaction</p> <p>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 decrement the pending transaction counter so this task must be called manually after the full read data transfer is complete.</p>	None	None

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
<p>SEND_WRITE_ADDRESS</p> <p>Creates a write address channel transaction. This task returns after the write address has been acknowledged by the slave. This task emits a "write_address_transfer_complete" event upon completion.</p>	<p>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</p>	None
<p>SEND_WRITE_DATA</p> <p>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. This task emits a "write_data_transfer_complete" event upon completion.</p> <p>Note: Should be called multiple times for a burst with correct control of the LAST flag</p>	<p>ID: Write ID tag STOBE: Strobe signals DATA: Data for transfer LAST: Last transfer flag</p>	None

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

API Task Name and Description	Inputs	Outputs
<p>SEND_READ_ADDRESS</p> <p>Creates a read address channel transaction. This task returns after the read address has been acknowledged by the slave.</p> <p>This task emits a "read_address_transfer_complete" event upon completion.</p>	<p>ID: Read Address ID tag</p> <p>ADDR: Read Address</p> <p>LEN: Burst Length</p> <p>SIZE: Burst Size</p> <p>BURST: Burst Type</p> <p>LOCK: Lock Type</p> <p>CACHE: Cache Type</p> <p>PROT: Protection Type</p>	<p>None</p>
<p>RECEIVE_READ_DATA</p> <p>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.</p> <p>This task emits a "read_data_transfer_complete" event upon completion.</p> <p>Note: This would need to be called multiple times for a burst > 1.</p> <p>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.</p>	<p>ID: Read ID tag</p>	<p>DATA: Data transferred by the slave</p> <p>RESPONSE: The slave read response from the following: [OKAY, EXOKAY, SLVERR, DECERR]</p> <p>LAST: Last transfer flag</p>
<p>RECEIVE_WRITE_RESPONSE</p> <p>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.</p> <p>This task emits a "write_response_transfer_complete" event upon completion.</p>	<p>ID: Write ID tag</p>	<p>RESPONSE: The slave write response from the following: [OKAY, EXOKAY, SLVERR, DECERR]</p>

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

API Task Name and Description	Inputs	Outputs
<p>RECEIVE_READ_BURST</p> <p>This task receives a read channel burst based on the ID input. The RECEIVE_READ_DATA from the channel level API is used.</p> <p>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.</p> <p>This task emits a "read_data_burst_complete" event upon completion.</p>	<p>ID: Read ID tag</p> <p>ADDR: Read Address</p> <p>LEN: Burst Length</p> <p>SIZE: Burst Size</p> <p>BURST: Burst Type</p> <p>LOCK: Lock Type</p>	<p>DATA: Valid Data transferred by the slave</p> <p>RESPONSE: This is a vector that is created by concatenating all slave read responses together</p>
<p>SEND_WRITE_BURST</p> <p>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.</p> <p>This task returns when the complete write burst is complete.</p> <p>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.</p> <p>This task emits a "write_data_burst_complete" event upon completion.</p>	<p>ID: Write ID tag</p> <p>ADDR: Write Address</p> <p>LEN: Burst Length</p> <p>SIZE: Burst Size</p> <p>BURST: Burst Type</p> <p>DATA: Data to send</p> <p>DATASIZE: The size in bytes of the valid data contained in the input data vector</p>	<p>None</p>

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.	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_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.	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
<p>SEND_WRITE_RESPONSE</p> <p>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.</p> <p>This task emits a "write_response_transfer_complete" event upon completion.</p>	<p>ID: Write ID tag</p> <p>RESPONSE: The chosen write response from the following [OKAY, EXOKAY, SLVERR, DECERR]</p>	None
<p>SEND_READ_DATA</p> <p>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.</p> <p>This task emits a "read_data_transfer_complete" event upon completion.</p> <p>Note: This would need to be called multiple times for a burst > 1.</p>	<p>ID: Read ID tag</p> <p>DATA: Data to send to the master</p> <p>RESPONSE: The read response to send to the master from the following: [OKAY, EXOKAY, SLVERR, DECERR]</p> <p>LAST: Last transfer flag</p>	None
<p>RECEIVE_WRITE_ADDRESS</p> <p>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.</p> <p>If the IDValid bit is 0 then the input ID tag is not used and the next available write address transfer is sampled.</p> <p>This task uses the SLAVE_ADDRESS and SLAVE_MEM_SIZE parameters to determine if the address is valid.</p> <p>This task emits a "write_address_transfer_complete" event upon completion.</p>	<p>ID: Write Address ID tag</p> <p>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.</p>	<p>ADDR: Write Address</p> <p>LEN: Burst Length</p> <p>SIZE: Burst Size</p> <p>BURST: Burst Type</p> <p>LOCK: Lock Type</p> <p>CACHE: Cache Type</p> <p>PROT: Protection Type</p> <p>IDTAG: Sampled ID tag</p>
<p>RECEIVE_READ_ADDRESS</p> <p>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.</p> <p>If the IDValid bit is 0 then the input ID tag is not used and the next available read address transfer is sampled.</p> <p>This task uses the SLAVE_ADDRESS and SLAVE_MEM_SIZE parameters to determine if the address is valid.</p> <p>This task emits a "read_address_transfer_complete" event upon completion.</p>	<p>ID: Write Address ID tag</p> <p>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.</p>	<p>ADDR: Write Address</p> <p>LEN: Burst Length</p> <p>SIZE: Burst Size</p> <p>BURST: Burst Type</p> <p>LOCK: Lock Type</p> <p>CACHE: Cache Type</p> <p>PROT: Protection Type</p> <p>IDTAG: Sampled ID tag</p>

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

API Task Name and Description	Inputs	Outputs
<p>RECEIVE_WRITE_DATA</p> <p>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.</p> <p>Note: This would need to be called multiple times for a burst > 1.</p> <p>If the IDValid bit is 0 then the input ID tag is not used and the next available write data transfer is sampled. This task emits a "write_data_transfer_complete" event upon completion.</p>	<p>ID: Write ID tag</p> <p>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.</p>	<p>DATA: Data transferred from the master</p> <p>STRB: Strobe signals used to validate the data</p> <p>LAST: Last transfer flag</p> <p>IDTAG: Sampled ID tag</p>
<p>RECEIVE_WRITE_BURST</p> <p>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.</p> <p>If the IDValid bit is 0 then the input ID tag is not used and the next available write burst is sampled.</p> <p>This task returns when the complete write burst is complete.</p> <p>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).</p> <p>This task emits a "write_data_burst_complete" event upon completion.</p>	<p>ID: Write ID tag</p> <p>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.</p> <p>ADDR: Write Address</p> <p>LEN: Burst Length</p> <p>SIZE: Burst Size</p> <p>BURST: Burst Type</p>	<p>DATA: Data received from the write burst</p> <p>DATASIZE: The size in bytes of the valid data contained in the output data vector</p> <p>IDTAG: Sampled ID tag</p>
<p>RECEIVE_WRITE_BURST_NO_CHECKS</p> <p>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.</p> <p>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).</p>	<p>ID: Write ID tag</p>	<p>DATA: Data received from the write burst</p> <p>DATASIZE: The size in bytes of the valid data contained in the output data vector</p>

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

API Task Name and Description	Inputs	Outputs
<p>SEND_READ_BURST</p> <p>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.</p> <p>This task returns when the complete read burst is complete.</p> <p>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.</p> <p>This task emits a "read_data_burst_complete" event upon completion.</p>	<p>ID: Read ID tag</p> <p>ADDR: Read Address</p> <p>LEN: Burst Length</p> <p>SIZE: Burst Size</p> <p>BURST: Burst Type</p> <p>LOCK: Lock Type</p> <p>DATA: Data to be sent over the burst</p>	<p>None</p>
<p>SEND_READ_BURST_RESP_CTRL</p> <p>This task is the same as SEND_READ_BURST except that the response sent to the master can be specified.</p>	<p>ID: Read ID tag</p> <p>ADDR: Read Address</p> <p>LEN: Burst Length</p> <p>SIZE: Burst Size</p> <p>BURST: Burst Type</p> <p>DATA: Data to be sent over the burst</p> <p>RESPONSE: This is a vector that should contain all of the desired responses for each read data transfer</p>	<p>None</p>

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
<p>READ_BURST_RESPOND</p> <p>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.</p> <p>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.</p>	<p>ID: Read ID tag</p> <p>DATA: Data to send in response to the master read</p>	<p>None</p>
<p>WRITE_BURST_RESPOND</p> <p>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.</p> <p>This task is composed of the tasks RECEIVE_WRITE_ADDRESS, RECEIVE_WRITE_BURST and SEND_WRITE_RESPONSE from the channel level API.</p> <p>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.</p>	<p>ID: Write ID tag</p>	<p>DATA: Data received by slave</p> <p>DATASIZE: The size in bytes of the valid data contained in the output data vector</p>
<p>WRITE_BURST_RESPOND_DATA_FIRST</p> <p>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.</p>	<p>ID: Write ID tag</p>	<p>DATA: Data received by slave</p> <p>DATASIZE: The size in bytes of the valid data contained in the output data vector</p>

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

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

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
<p>SEND_WRITE_ADDRESS</p> <p>Creates a write address channel transaction. This task returns after the write address has been acknowledged by the slave.</p> <p>This task emits a "write_address_transfer_complete" event upon completion.</p>	<p>ID: Write Address ID tag</p> <p>ADDR: Write Address</p> <p>LEN: Burst Length</p> <p>SIZE: Burst Size</p> <p>BURST: Burst Type</p> <p>LOCK: Lock Type</p> <p>CACHE: Cache Type</p> <p>PROT: Protection Type</p> <p>REGION: Region Identifier</p> <p>QOS: Quality of Service Signals</p> <p>AWUSER: Address Write User Defined Signals</p>	None
<p>SEND_WRITE_DATA</p> <p>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.</p> <p>This task emits a "write_data_transfer_complete" event upon completion.</p> <p>Note: Should be called multiple times for a burst with correct control of the LAST flag</p>	<p>STOBE: Strobe signals</p> <p>DATA: Data for transfer</p> <p>LAST: Last transfer flag</p> <p>WUSER: Write User Defined Signals</p>	None

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

API Task Name	Inputs	Outputs
<p>SEND_READ_ADDRESS</p> <p>Creates a read address channel transaction. This task returns after the read address has been acknowledged by the slave.</p> <p>This task emits a "read_address_transfer_complete" event upon completion.</p>	<p>ID: Read Address ID tag</p> <p>ADDR: Read Address</p> <p>LEN: Burst Length</p> <p>SIZE: Burst Size</p> <p>BURST: Burst Type</p> <p>LOCK: Lock Type</p> <p>CACHE: Cache Type</p> <p>PROT: Protection Type</p> <p>REGION: Region Identifier</p> <p>QOS: Quality of Service Signals</p> <p>ARUSER: Address Read User Defined Signals</p>	<p>None</p>
<p>RECEIVE_READ_DATA</p> <p>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.</p> <p>This task emits a "read_data_transfer_complete" event upon completion.</p> <p>Note: This would need to be called multiple times for a burst > 1.</p> <p>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.</p>	<p>ID: Read ID tag</p>	<p>DATA: Data transferred by the slave</p> <p>RESPONSE: The slave read response from the following: [OKAY, EXOKAY, SLVERR, DECERR]</p> <p>LAST: Last transfer flag</p> <p>RUSER: Read User Defined Signals</p>
<p>RECEIVE_WRITE_RESPONSE</p> <p>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.</p> <p>This task emits a "write_response_transfer_complete" event upon completion.</p>	<p>ID: Write ID tag</p>	<p>RESPONSE: The slave write response from the following: [OKAY, EXOKAY, SLVERR, DECERR]</p> <p>BUSER: Write Response User Defined Signals</p>

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

API Task Name	Inputs	Outputs
<p>RECEIVE_READ_BURST</p> <p>This task receives a read channel burst based on the ID input. The RECEIVE_READ_DATA from the channel level API is used.</p> <p>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.</p> <p>This task emits a "read_data_burst_complete" event upon completion.</p>	<p>ID: Read ID tag</p> <p>ADDR: Read Address</p> <p>LEN: Burst Length</p> <p>SIZE: Burst Size</p> <p>BURST: Burst Type</p> <p>LOCK: Lock Type</p>	<p>DATA: Valid Data transferred by the slave</p> <p>RESPONSE: This is a vector that is created by concatenating all slave read responses together</p> <p>RUSER: This is a vector that is created by concatenating all slave read user signal data together</p>
<p>SEND_WRITE_BURST</p> <p>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.</p> <p>This task returns when the complete write burst is complete.</p> <p>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.</p> <p>This task emits a "write_data_burst_complete" event upon completion.</p>	<p>ADDR: Write Address</p> <p>LEN: Burst Length</p> <p>SIZE: Burst Size</p> <p>BURST: Burst Type</p> <p>DATA: Data to send</p> <p>DATASIZE: The size in bytes of the valid data contained in the input data vector</p> <p>WUSER: This is a vector that is created by concatenating all write transfer user signal data together</p>	<p>None</p>

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
<p>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.</p>	<p>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</p>	<p>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</p>

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

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

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
<p>SEND_WRITE_RESPONSE</p> <p>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.</p> <p>This task emits a "write_response_transfer_complete" event upon completion.</p>	<p>ID: Write ID tag</p> <p>RESPONSE: The chosen write response from the following [OKAY, EXOKAY, SLVERR, DECERR]</p> <p>BUSER: Write Response User Defined Signals</p>	None
<p>SEND_READ_DATA</p> <p>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.</p> <p>This task emits a "read_data_transfer_complete" event upon completion.</p> <p>Note: This would need to be called multiple times for a burst > 1.</p>	<p>ID: Read ID tag</p> <p>DATA: Data to send to the master</p> <p>RESPONSE: The read response to send to the master from the following: [OKAY, EXOKAY, SLVERR, DECERR]</p> <p>LAST: Last transfer flag</p> <p>RUSER: Read User Defined Signals</p>	None
<p>RECEIVE_WRITE_ADDRESS</p> <p>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.</p> <p>This task uses the SLAVE_ADDRESS and SLAVE_MEM_SIZE parameters to determine if the address is valid.</p> <p>This task emits a "write_address_transfer_complete" event upon completion.</p>	<p>ID: Write Address ID tag</p> <p>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.</p>	<p>ADDR: Write Address</p> <p>LEN: Burst Length</p> <p>SIZE: Burst Size</p> <p>BURST: Burst Type</p> <p>LOCK: Lock Type</p> <p>CACHE: Cache Type</p> <p>PROT: Protection Type</p> <p>REGION: Region Identifier</p> <p>QOS: Quality of Service Signals</p> <p>AWUSER: Address Write User Defined Signals</p> <p>IDTAG: Sampled ID tag</p>

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

API Task Name and Description	Inputs	Outputs
<p>RECEIVE_READ_ADDRESS</p> <p>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.</p> <p>This task uses the SLAVE_ADDRESS and SLAVE_MEM_SIZE parameters to determine if the address is valid.</p> <p>This task emits a "read_address_transfer_complete" event upon completion.</p>	<p>ID: Read Address ID tag</p> <p>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.</p>	<p>ADDR: Read Address</p> <p>LEN: Burst Length</p> <p>SIZE: Burst Size</p> <p>BURST: Burst Type</p> <p>LOCK: Lock Type</p> <p>CACHE: Cache Type</p> <p>PROT: Protection Type</p> <p>REGION: Region Identifier</p> <p>QOS: Quality of Service Signals</p> <p>ARUSER: Address Read User Defined Signals</p> <p>IDTAG: Sampled ID tag</p>
<p>RECEIVE_WRITE_DATA</p> <p>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.</p> <p>Note: This would need to be called multiple times for a burst > 1.</p> <p>This task emits a "write_data_transfer_complete" event upon completion.</p>	<p>None</p>	<p>DATA: Data transferred from the master</p> <p>STRB: Strobe signals used to validate the data</p> <p>LAST: Last transfer flag</p> <p>WUSER: Write User Defined Signals</p>
<p>RECEIVE_WRITE_BURST</p> <p>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.</p> <p>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).</p> <p>This task emits a "write_data_burst_complete" event upon completion.</p>	<p>ADDR: Write Address</p> <p>LEN: Burst Length</p> <p>SIZE: Burst Size</p> <p>BURST: Burst Type</p>	<p>DATA: Data received from the write burst</p> <p>DATASIZE: The size in bytes of the valid data contained in the output data vector</p> <p>WUSER: This is a vector that is created by concatenating all master write user signal data together</p>

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

API Task Name and Description	Inputs	Outputs
<p>SEND_READ_BURST</p> <p>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.</p> <p>This task returns when the complete read burst is complete.</p> <p>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.</p> <p>This task emits a "read_data_burst_complete" event upon completion.</p>	<p>ID: Read ID tag</p> <p>ADDR: Read Address</p> <p>LEN: Burst Length</p> <p>SIZE: Burst Size</p> <p>BURST: Burst Type</p> <p>LOCK: Lock Type</p> <p>DATA: Data to be sent over the burst</p> <p>RUSER: This is a vector that is created by concatenating all required slave read user signal data together</p>	<p>None</p>
<p>SEND_READ_BURST_RESP_CTRL</p> <p>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.</p> <p>This task returns when the complete read burst is complete.</p> <p>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.</p> <p>This task emits a "read_data_burst_complete" event upon completion.</p>	<p>ID: Read ID tag</p> <p>ADDR: Read Address</p> <p>LEN: Burst Length</p> <p>SIZE: Burst Size</p> <p>BURST: Burst Type</p> <p>DATA: Data to be sent over the burst</p> <p>RESPONSE: This is a vector that should contain all of the desired responses for each read data transfer</p> <p>RUSER: This is a vector that is created by concatenating all required slave read user signal data together</p>	<p>None</p>

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
<p>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.</p> <p>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.</p>	<p>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</p>	<p>None</p>
<p>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.</p> <p>This task is composed of the tasks RECEIVE_WRITE_ADDRESS, RECEIVE_WRITE_BURST and SEND_WRITE_RESPONSE from the channel level API.</p> <p>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.</p>	<p>ID: Write ID tag BUSER: Write Response Channel User Defined Signals</p>	<p>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</p>

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

API Task Name and Description	Inputs	Outputs
<p>READ_BURST_RESP_CTRL</p> <p>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 <code>RECEIVE_READ_ADDRESS</code> and <code>SEND_READ_BURST_RESP_CTRL</code> from the channel level API. This task returns when the complete write transaction is complete.</p> <p>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.</p>	<p>ID: Read ID tag</p> <p>DATA: Data to send in response to the master read</p> <p>RESPONSE: This is a vector that should contain all of the desired responses for each read data transfer</p> <p>RUSER: This is a vector that is created by concatenating all required read user signal data together</p>	<p>None</p>
<p>WRITE_BURST_RESP_CTRL</p> <p>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.</p> <p>This task is composed of the tasks <code>RECEIVE_WRITE_ADDRESS</code>, <code>RECEIVE_WRITE_BURST</code> and <code>SEND_WRITE_RESPONSE</code> from the channel level API.</p> <p>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.</p>	<p>ID: Write ID tag</p> <p>RESPONSE: The chosen write response from the following [OKAY, EXOKAY, SLVERR, DECERR]</p> <p>BUSER: Write Response Channel User Defined Signals</p>	<p>DATA: Data received by slave</p> <p>DATASIZE: The size in bytes of the valid data contained in the output data vector</p> <p>WUSER: This is a vector that is created by concatenating all master write transfer user signal data together</p>

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. This task emits a "write_address_transfer_complete" event upon completion.	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. This task emits a "write_data_transfer_complete" event upon completion.	STOB: 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. This task emits a "read_address_transfer_complete" event upon completion.	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. This task emits a "read_data_transfer_complete" event upon completion.	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. This task emits a "write_response_transfer_complete" event upon completion.	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
<p>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.</p>	<p>ADDR: Read Address PROT: Protection Type</p>	<p>DATA: Valid data transferred by the slave RESPONSE: The slave write response from the following: [OKAY, SLVERR, DECERR]</p>
<p>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.</p>	<p>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</p>	<p>RESPONSE: The slave write response from the following: [OKAY, SLVERR, DECERR]</p>
<p>WRITE_BURST_CONCURRENT This task does the same function as the WRITE_BURST task; however, it performs the write address and data phases concurrently.</p>	<p>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</p>	<p>RESPONSE: The slave write response from the following: [OKAY, SLVERR, DECERR]</p>
<p>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.</p>	<p>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</p>	<p>RESPONSE: The slave write response from the following: [OKAY, SLVERR, DECERR]</p>

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
<p>SEND_WRITE_RESPONSE Creates a write response channel transaction. This task returns after it has been acknowledged by the master. This task emits a "write_response_transfer_complete" event upon completion.</p>	<p>RESPONSE: The chosen write response from the following [OKAY, SLVERR, DECERR]</p>	None
<p>SEND_READ_DATA Creates a read channel transaction. This task returns after it has been acknowledged by the master. This task emits a "read_data_transfer_complete" event upon completion.</p>	<p>DATA: Data to send to the master RESPONSE: The read response to send to the master from the following: [OKAY, SLVERR, DECERR]</p>	None
<p>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. This task emits a "write_address_transfer_complete" event upon completion.</p>	<p>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.</p>	<p>PROT: Protection Type SADDR: Sampled Write Address</p>

Table 3-20: Channel Level API for AXI4-Lite Slave BFM (Cont'd)

API Task Name and Description	Inputs	Outputs
<p>RECEIVE_READ_ADDRESS</p> <p>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.</p> <p>This task uses the SLAVE_ADDRESS and SLAVE_MEM_SIZE parameters to determine if the address is valid.</p> <p>This task emits a "read_address_transfer_complete" event upon completion.</p>	<p>ADDR: Read Address</p> <p>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.</p>	<p>PROT: Protection Type</p> <p>SADDR: Sampled Read Address</p>
<p>RECEIVE_WRITE_DATA</p> <p>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.</p> <p>This task emits a "write_data_transfer_complete" event upon completion.</p>	<p>None</p>	<p>DATA: Data transferred from the master</p> <p>STRB: Strobe signals used to validate the data</p>

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
<p>READ_BURST_RESPOND</p> <p>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.</p> <p>If ADDRVALID = 0 the input ADDR is ignored and the first read request is used and responded to.</p> <p>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.</p>	<p>ADDR: Read Address</p> <p>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.</p> <p>DATA: Data to send in response to the master read</p>	<p>None</p>
<p>WRITE_BURST_RESPOND</p> <p>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.</p> <p>This task returns when the complete write transaction is complete.</p> <p>If ADDRVALID = 0 the input ADDR is ignored and the first write request is used for the DATA output.</p> <p>If the ADDRVALID = 1 then the ADDR input is used and the DATA associated with that transfer is output using the DATA output.</p>	<p>ADDR: Write Address</p> <p>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.</p>	<p>DATA: Data received by slave</p> <p>DATASIZE: The size in bytes of the valid data contained in the output data vector</p>

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

API Task Name and Description	Inputs	Outputs
<p>READ_BURST_RESP_CTRL</p> <p>This task is the same as READ_BURST_RESPOND except that the response sent to the master can be specified.</p>	<p>ADDR: Read Address</p> <p>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.</p> <p>DATA: Data to send in response to the master read</p> <p>RESPONSE: The chosen write response from the following [OKAY, SLVERR, DECERR]</p>	None
<p>WRITE_BURST_RESP_CTRL</p> <p>This task is the same as WRITE_BURST_RESPOND except that the response sent to the master can be specified.</p>	<p>ADDR: Write Address</p> <p>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.</p> <p>RESPONSE: The chosen write response from the following [OKAY, SLVERR, DECERR]</p>	<p>DATA: Data received by slave</p> <p>DATASIZE: The size in bytes of the valid data contained in the output data vector</p>

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
<p>SEND_TRANSFER</p> <p>Creates a single AXI4-Stream transfer. This task emits a "transfer_complete" event upon completion.</p>	<p>ID: Transfer ID Tag</p> <p>DEST: Transfer Destination</p> <p>DATA: Transfer Data</p> <p>STRB: Transfer Strobe Signals</p> <p>KEEP: Transfer Keep Signals</p> <p>LAST: Transfer Last Signal</p> <p>USER: Transfer User Signals</p>	None
<p>SEND_PACKET</p> <p>This task sends a complete packet over the streaming interface. It uses the SEND_TRANSFER task from the channel level API.</p> <p>This task returns when the whole packet has been sent, and emits a "packet_complete" event upon completion.</p>	<p>ID: Transfer ID Tag</p> <p>DEST: Transfer Destination</p> <p>DATA: Vector of Transfer data to send</p> <p>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)</p> <p>USER: This is a vector that is created by concatenating all transfer user signal data together</p>	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. This task emits a "transfer_complete" event upon completion.	None	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, and emits a "packet_complete" event upon completion. 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

For more information on AXI specification, see the ARM[®] AMBA AXI4-Stream Protocol Specification [\[Ref 5\]](#).

Customizing and Generating the Core

This chapter includes information about using Xilinx tools to customize and generate the core in the Vivado™ Design Suite environment.

Vivado Integrated Design Environment (IDE)

The AXI BFM can be found in `/AXI Infrastructure` or `/Debug & Verification` in the Vivado IP Catalog.

To access the core name, perform the following:

1. Open a project by selecting **File** then **Open Project** or create a new project by selecting **File** then **New Project** in Vivado.
2. Open the IP catalog and navigate to any of the taxonomies.
3. Double-click **AXI Bus Functional Model** to bring up the AXI BFM Customize IP dialog box.

[Figure 4-1](#) and [Figure 4-2](#) show the AXI BFM Customize IP dialog box with information about customizing ports.

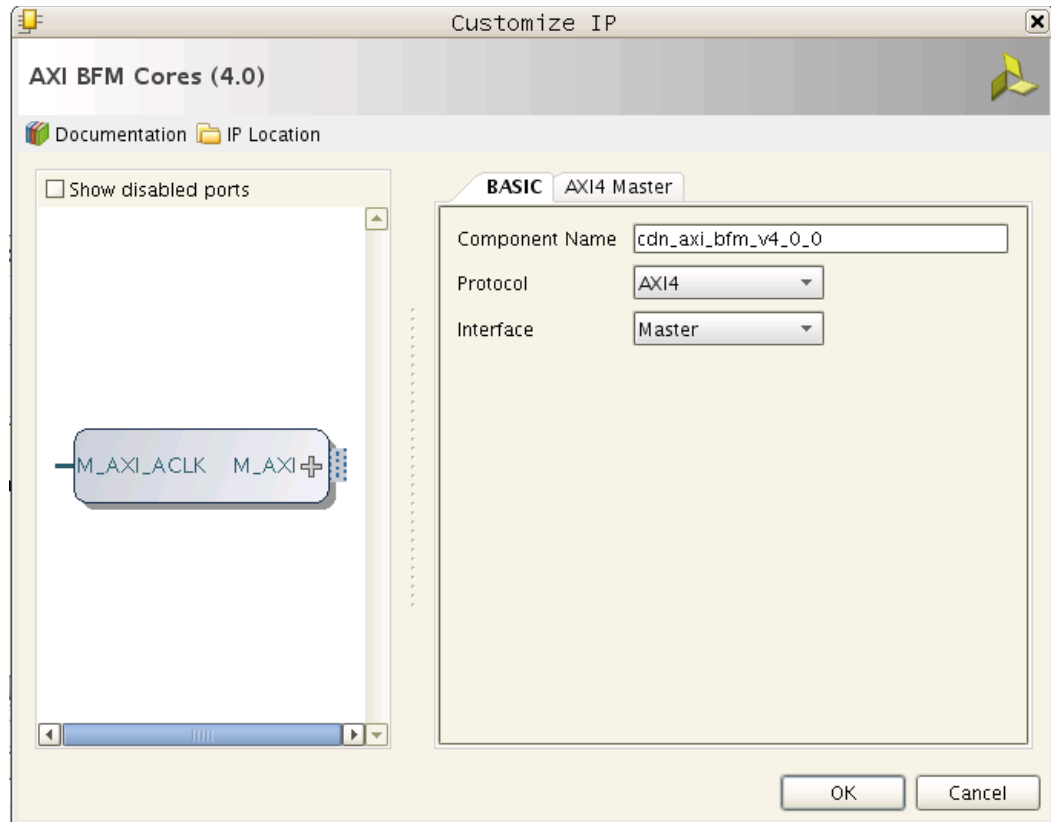


Figure 4-1: Vivado Customize IP Dialog Box – Basic Tab

Basic

- **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.

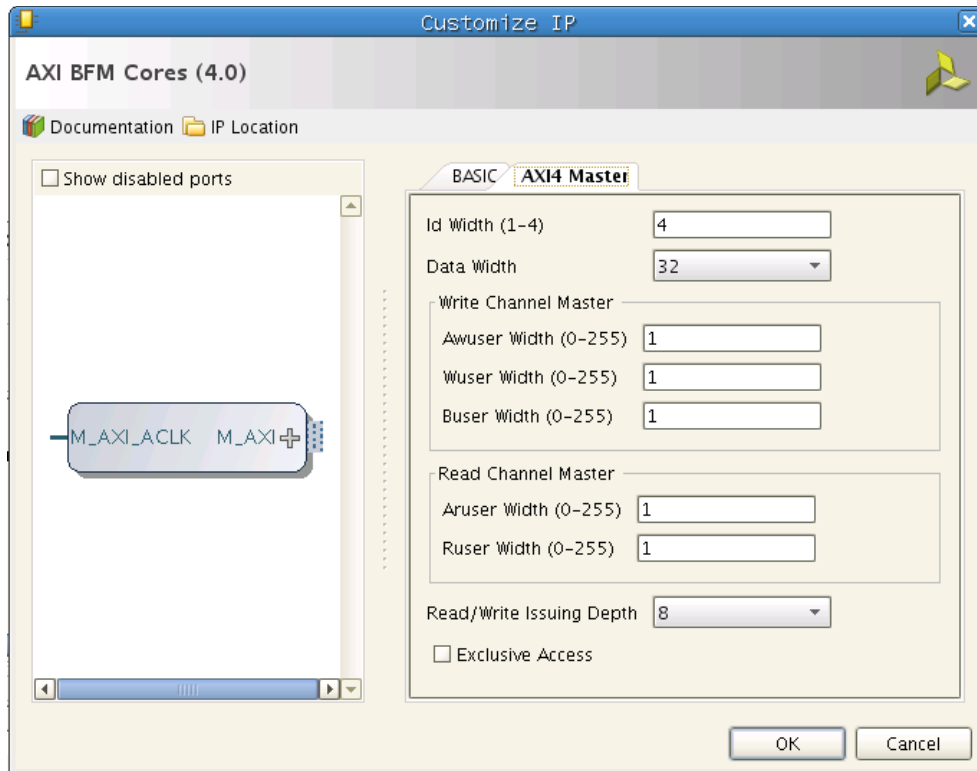


Figure 4-2: Vivado Customize IP Dialog Box – AXI4 Master Tab

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.
- **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 0 to 255 with default set to 1.
- **Wuser Width** – Range of 0 to 255 with default set to 1.
- **Buser Width** – Range of 0 to 255 with default set to 1.

Read Channel Master

- **Aruser Width** – Range of 0 to 255 with default set to 1.
- **Ruser Width** – Range of 0 to 255 with default set to 1.

Detailed Example Design

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

Example Design

This section describes the example test benches and example tests used to demonstrate the abilities of each AXI BFM pair. Example tests are delivered either in VHDL or Verilog based on the design entry while generating the core. These example designs are available in the AXI_BFM installation area. 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 is 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 AMBA® AXI protocol specification [Ref 6], Section 3.3, Dependencies between Channel Handshake Signals, 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 for Standalone RTL Design

The AXI BFM 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.

Demonstration Test Bench

AXI3 BFM Example Test Bench and Test

The Verilog example test bench and example test for the AXI3 BFM is shown in [Figure 5-1](#).

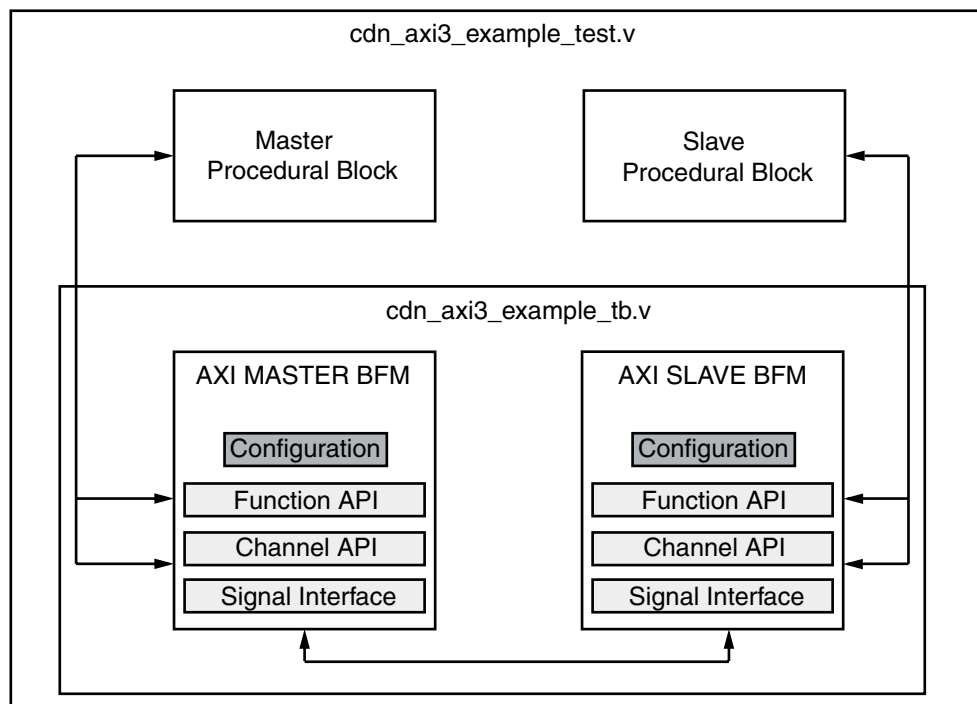


Figure 5-1: Verilog Example Test Bench and Test Case Structure

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:

1. Simple sequential write and read burst transfers example
2. Looped sequential write and read transfers example
3. Parallel write and read burst transfers example
4. Narrow write and read transfers example
5. Unaligned write and read transfers example

6. Narrow and unaligned write and read transfers example
7. Out of order write and read burst example
8. Write Bursts performed in two different ways; Data before address and data with address concurrently
9. Write data interleaving example
10. Read data interleaving example
11. Outstanding transactions example
12. Slave read and write bursts error response example
13. Write and read bursts with different length gaps between data transfers example
14. Write and Read bursts with different length gaps between channel transfers example
15. 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 5-1](#). 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:

1. Simple sequential write and read burst transfers example
2. Looped sequential write and read transfers example
3. Parallel write and read burst transfers example
4. Narrow write and read transfers example
5. Unaligned write and read transfers example
6. Narrow and unaligned write and read transfers example
7. Write Bursts performed with address and data channel transfers concurrently
8. Outstanding transactions example
9. Slave read and write bursts error response example
10. Write and read bursts with different length gaps between data transfers example
11. Write and Read bursts with different length gaps between channel transfers example
12. Write burst that is longer than the data it is sending example
13. 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 5-1](#). 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:

1. Simple sequential write and read burst transfers example
2. Looped sequential write and read transfers example
3. Parallel write and read burst transfers example
4. Write Bursts performed in two different ways; Data before address and data with address concurrently
5. Outstanding transactions example
6. Slave read and write bursts error response example
7. Write and Read bursts with different length gaps between channel transfers example
8. Unaligned write and read transfers example
9. 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 5-1](#). 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:

1. Simple master to slave transfer example
2. Looped master to slave transfers example
3. Simple master to slave packet example
4. Looped master to slave packet example
5. Ragged (less data at the end of the packet than can be supported) master to slave packet example
6. 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
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. The AXI BFM 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 – Memory Model Example

In most cases, the behavior of a master or slave is more complicated than simple transfer generation. For this reason, the AXI BFM API enables the end user 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. The first step is to 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. The second step is to 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. The third step is to 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. The last step to 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. The first step is to 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. The second step is to 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`.

Simulation

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.

Verification, Compliance, and Interoperability

The AXI BFM is compliant to AXI3, AXI4, AXI4-Lite, and AXI4-Stream protocols.

Migrating

This appendix describes migrating from older versions of the IP to the current IP release.

For information on migrating to the Vivado™ Design Suite, see UG911, *Vivado Design Suite Migration Methodology Guide* [\[Ref 2\]](#).

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

Debugging

This appendix includes details about resources available on the Xilinx Support website and debugging tools. In addition, this appendix provides a step-by-step debugging process to guide you through debugging the AXI BFM core.

The following topics are included in this appendix:

- [Finding Help on Xilinx.com](#)
- [Interface Debug](#)

Finding Help on Xilinx.com

To help in the design and debug process when using the AXI BFM, the [Xilinx Support web page](http://www.xilinx.com/support) (www.xilinx.com/support) contains key resources such as product documentation, release notes, answer records, information about known issues, and links for opening a Technical Support WebCase.

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 (www.xilinx.com/support) or by using the Xilinx Documentation Navigator.

Download the Xilinx Documentation Navigator from the Design Tools tab on the Downloads page (www.xilinx.com/download). For more information about this tool and the features available, open the online help after installation.

Known Issues

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 are listed below, and can also 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](#)

Contacting Technical Support

Xilinx provides technical support at www.xilinx.com/support for this LogiCORE™ IP product when used as described in the product documentation. Xilinx cannot guarantee timing, functionality, or support of product if implemented in devices that are not defined in the documentation, if customized beyond that allowed in the product documentation, or if changes are made to any section of the design labeled DO NOT MODIFY.

To contact Xilinx Technical Support:

1. Navigate to www.xilinx.com/support.
2. Open a WebCase by selecting the [WebCase](#) link located under Support Quick Links.

When opening a WebCase, include:

- Target FPGA including package and speed grade.
- All applicable Xilinx Design Tools and simulator software versions.
- Additional files based on the specific issue might also be required. See the relevant sections in this debug guide for guidelines about which file(s) to include with the WebCase.

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

Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see the Xilinx Support website at:

www.xilinx.com/support.

For a glossary of technical terms used in Xilinx documentation, see:

www.xilinx.com/company/terms.htm.

References

These documents provide supplemental material useful with this product guide:

1. *LogiCORE IP AXI Interconnect Product Guide* ([PG059](#))
2. Vivado™ Design Suite user documentation
3. *Vivado Design Suite Getting Started Guide* ([UG814](#))
4. *Vivado Design Suite User Guide, Designing with IP* ([UG896](#))
5. [AMBA AXI4-Stream Protocol Specification](#)
6. ARM® AMBA® AXI Protocol v2.0 Specification (ARM IHI 0022C)
<http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ih0022c/index.html>
7. *Cadence AXI UVC User Guide* (VIPP 9.2/VIPP 10.2 releases)

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
03/20/2013	1.0	Initial Xilinx release of the product guide and replaces DS824.

Notice of Disclaimer

The information disclosed to you hereunder (the "Materials") is provided solely for the selection and use of Xilinx products. To the maximum extent permitted by applicable law: (1) Materials are made available "AS IS" and with all faults, Xilinx hereby DISCLAIMS ALL WARRANTIES AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and (2) Xilinx shall not be liable (whether in contract or tort, including negligence, or under any other theory of liability) for any loss or damage of any kind or nature related to, arising under, or in connection with, the Materials (including your use of the Materials), including for any direct, indirect, special, incidental, or consequential loss or damage (including loss of data, profits, goodwill, or any type of loss or damage suffered as a result of any action brought by a third party) even if such damage or loss was reasonably foreseeable or Xilinx had been advised of the possibility of the same. Xilinx assumes no obligation to correct any errors contained in the Materials or to notify you of updates to the Materials or to product specifications. You may not reproduce, modify, distribute, or publicly display the Materials without prior written consent. Certain products are subject to the terms and conditions of the Limited Warranties which can be viewed at <http://www.xilinx.com/warranty.htm>; IP cores may be subject to warranty and support terms contained in a license issued to you by Xilinx. Xilinx products are not designed or intended to be fail-safe or for use in any application requiring fail-safe performance; you assume sole risk and liability for use of Xilinx products in Critical Applications: <http://www.xilinx.com/warranty.htm#critapps>.

© Copyright 2013 Xilinx, Inc. Xilinx, the Xilinx logo, Artix, ISE, Kintex, Spartan, Virtex, Vivado, Zynq, and other designated brands included herein are trademarks of Xilinx in the United States and other countries. All other trademarks are the property of their respective owners.