AXI Bus Functional Model v1.1

User Guide

UG783 December 14, 2010

XILINX®

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
12/14/2010	1.0	Initial Xilinx release.

Table of Contents

Chapter 5: Directory Structure

Chapter 6: AXI4 BFM Example Designs

Schedule of Figures

Schedule of Tables

Chapter 1: Overview

Chapter 2: Configuration Options

Chapter 3: Test Writing API

Chapter 4: Protocol Checking

Chapter 5: Directory Structure

Chapter 6: AXI4 BFM Example Designs

About This Guide

This document defines the AXI Bus Functional Model (BFM) solution, created for Xilinx by Cadence Design Systems. The AXI BFMs enable Xilinx customers to verify and simulate communication with AXI-based IP that is being developed. Complete verification of these interfaces and protocol compliance is outside the scope of the AXI BFM solution; for compliance testing and complete system-level verification of AXI interfaces, the Cadence AXI UVC can be used.

The AXI BFM solution is an optional product that is purchased separate from the ISE software. Licensing is handled through the standard Xilinx licensing scheme. A new license feature, XILINX_AXI_BFM, is needed in addition to the standard ISE license features. A license is checked out at simulation run time. While the Xilinx ISE software does not need to be running while the AXI BFM solution is in use, the AXI BFM will only operate on a computer that has the Xilinx software installed and licensed.

The BFM solution is encrypted using either the Verilog P1735 IEEE standard or a vendor-specific encryption scheme. To use the AXI BFM with Cadence IUS/IES simulator products, an export control regulation license feature is required. Contact your Cadence sales office for more information.

Guide Contents

This manual contains the following chapters:

- Chapter 1, Overview provides an overview of the AXI BFM solution.
- Chapter 2, Configuration Options describes the configuration options for each of the AXI BFMs.
- Chapter 3, Test Writing API demonstrates how to create the various types of AXI stimulus.
- Chapter 4, Protocol Checking describes the level of protocol checking performed by the AXI BFMs.
- Chapter 5, Directory Structure shows where the BFMs are located relative to the rest of the Xilinx software installation.
- Chapter 6, AXI4 BFM Example Designs includes examples showing how the BFMs are used in a stand-alone form.

Additional Resources

To find additional documentation, see the Xilinx website at:

www.xilinx.com/support/documentation/index.htm.

To search the Answer Database of silicon, software, and IP questions and answers, or to create a technical support WebCase, see the Xilinx website at:

www.xilinx.com/support.

Conventions

This document uses the conventions listed in this section. An example illustrates each convention.

Typographical

These typographical conventions are used in this document:

Convention	Meaning or Use	Example
`Courier font`	Messages, prompts, and program files that the system displays	`speed grade: - 100`
`Courier bold`	Literal commands that you enter in a syntactical statement	**`ngdbuild`** *design_name*
Helvetica bold	Commands that you select from a menu	**File → Open**
	Keyboard shortcuts	**Ctrl+C**
Italic font	Variables in a syntax statement for which you must supply values	**`ngdbuild`** *design_name*
	References to other manuals	See the *Command Line Tools User Guide* for more information.
	Emphasis in text	If a wire is drawn so that it overlaps the pin of a symbol, the two nets are *not* connected.
Square brackets [ ]	An optional entry or parameter. However, in bus specifications, such as **`bus[7:0]`**, they are required.	**`ngdbuild`** [*`option_name`*] *`design_name`*
Braces { }	A list of items from which you must choose one or more	**`lowpwr ={on|off}`**
Vertical bar |	Separates items in a list of choices	**`lowpwr ={on|off}`**

Online Document

These conventions are used in this document:

Convention	Meaning or Use	Example
Blue text	Cross-reference link to a location in the current document	See the section Additional Resources for details. Refer to Title Formats in Chapter 1 for details.
Blue, underlined text	Hyperlink to a website (URL)	Go to www.xilinx.com for the latest speed files.

XILINX®

Chapter 1

Overview

This chapter provides a high level, architectural overview of the general AXI BFM structure. It also shows how the AXI BFMs fit into an overall test environment.

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

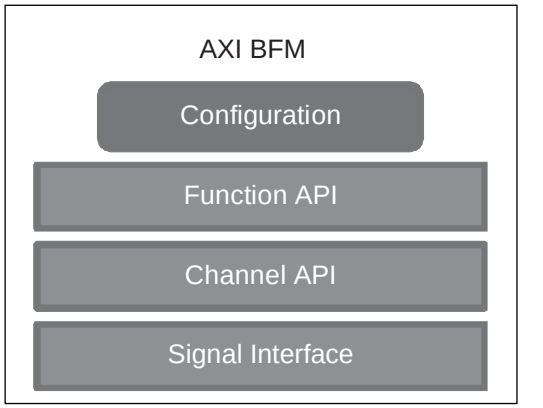

Figure 1-1: **AXI BFM Architecture**

All of the AXI BFMs consist of three main layers: the signal interface, the channel API and the 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 Chapter 3, 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 Chapter 3, 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.

One final but important piece 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 too (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 Chapter 2, Configuration Options.

The intended usage of the AXI BFM is shown in Figure 1-2.

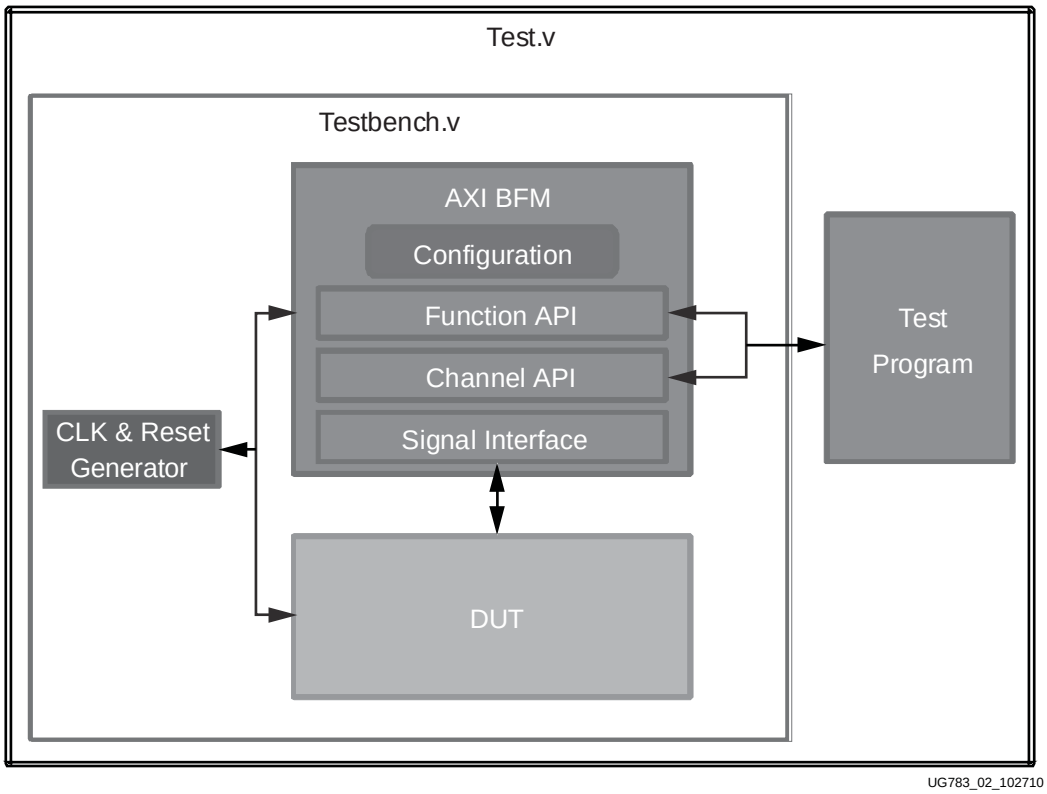

Figure 1-2: **AXI BFM Usage**

Figure 1-2 shows a single AXI BFM. However, the test bench can contain multiple instances of AXI BFMs. The DUT and the AXI BFMs are instantiated in a test bench that also contains a clock and reset generator. Then, the test writer instantiates the test bench into the test module and creates a test program using the BFM API layers. The test program would call API tasks either sequentially or concurrently using fork and join. See Chapter 6, AXI4 BFM Example Designs for practical examples of test programs and test bench setups.

References

1. ARM AMBA 4.0 AXI4 Protocol Specification, version 1.0, March 2010 http://www.arm.com/products/system-ip/amba/amba-open-specifications.php
2. Cadence AXI UVC User Guide (VIPP 9.2/VIPP 10.2 releases)

XILINX®

Chapter 2

Configuration Options

This chapter describes the configuration options for each of the AXI BFMs. 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 since 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 following task call should be made: **`set_channel_level_info(1)`**. For more information on the API for changing internal variables, see Chapter 3, Test Writing API.

AXI3 BFMs

The AXI3 BFMs modules and files are named as follows:

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

AXI3 Master BFM

Table 2-1 contains a list of parameters and configuration variables supported by the AXI3 Master BFM.

Table 2-1: **AXI3 Master BFM Parameters**

BFM Parameters	Description
NAME	String name for the master BFM. This is used in the messages coming from the BFMs. The default for the master BFM is “MASTER_0”.
DATA_BUS_WIDTH	Read and write data busses can be 8, 16, 32, 64, 128, 256, 512 or 1024 bits wide. Default is 32.
ADDRESS_BUS_WIDTH	Default is 32.
ID_BUS_WIDTH	Default is 4.

Table 2-1: **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 will be 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 will expect 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 will be 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 write bursts are called. Then write data interleaving will occur. 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 will always stop simulation.

Table 2-1: **AXI3 Master 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 will be printed, when set to zero no channel level information will be 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 will be printed, when set to zero no function level information will be printed. Default (1) means function level info messages are enabled.

AXI3 Slave BFM

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

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

BFM Parameters	Description
NAME	String name for the slave BFM. This is used in the messages coming from the BFMs. The default for the slave BFM is "SLAVE_0".
DATA_BUS_WIDTH	Read and write data busses can be 8, 16, 32, 64, 128, 256, 512 or 1024 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's 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 4096 = 4Kbytes. The default value is 4 bytes, meaning, one 32-bit entry.

Table 2-2: **AXI3 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 will be 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 will automatically respond to all transfers and will 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 INCR transfers 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. NOTE: 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 will be an EXOKAY to exclusive accesses. A value of 0 means the slave does not support this so a response of OKAY will be 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 will occur. The depth of this data interleaving depends on the number of parallel writes being performed. This configuration variable can be changed during simulation.

Table 2-2: **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 will always stop simulation.
CHANNEL_LEVEL_INFO	This configuration variable controls the printing of channel level information messages. When set to 1 info messages will be printed, when set to zero no channel level information will be 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 will be printed, when set to zero no function level information will be printed. The default (1) enables the function level info messages.

AXI4 BFMs

The AXI4 BFMs modules and files are named as follows:

- Full Master BFM
  - Module Name: cdn_axi4_master_bfm
  - File Name: `cdn_axi4_master_bfm.v`
- Full Slave BFM
  - Module Name: cdn_axi4_slave_bfm
  - File Name: `cdn_axi4_slave_bfm.v`
- Lite Master BFM
  - Module Name: cdn_axi4_lite_master_bfm
  - File Name: `cdn_axi4_lite_master_bfm.v`
- Lite Slave BFM
  - Module Name: cdn_axi4_lite_slave_bfm
  - File Name: `cdn_axi4_lite_slave_bfm.v`
- Streaming Master BFM
  - Module Name: cdn_axi4_streaming_master_bfm
  - File Name: `cdn_axi4_streaming_master_bfm.v`
- Streaming Slave BFM
  - Module Name: cdn_axi4_streaming_slave_bfm
  - File Name: `cdn_axi4_streaming_slave_bfm.v`

AXI4 Master BFM

Table 2-3 contains a list of parameters and configuration variables supported by the AXI4 Master BFM.

Table 2-3: **AXI4 Master BFM Parameters**

BFM Parameters	Description
NAME	String name for the master BFM. This is used in the messages coming from the BFMs. The default for the master BFM is “MASTER_0”.
DATA_BUS_WIDTH	Read and write data busses can be 8, 16, 32, 64, 128, 256, 512 or 1024 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.

Table 2-3: **AXI4 Master BFM Parameters**

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 will be 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 will expect 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 will be 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 write bursts are called, then AXI4 protocol will be violated as the BFM will attempt to perform 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 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 will always stop simulation.

Table 2-3: **AXI4 Master BFM Parameters**

BFM Parameters	Description
CHANNEL_LEVEL_INFO	This configuration variable controls the printing of channel level information messages. When set to 1 info messages will be printed, when set to zero no channel level information will be 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 will be printed, when set to zero no function level information will be printed. The default (1) enables the function level info messages.

AXI4 Slave BFM

Table 2-4 contains a list of parameters and configuration variables supported by the AXI4 Slave BFM.

Table 2-4: **AXI4 Slave BFM Parameters**

BFM Parameters	Description
NAME	String name for the slave BFM. This is used in the messages coming from the BFMs. The default for the slave BFM is “SLAVE_0”.
DATA_BUS_WIDTH	Read and write data busses can be 8, 16, 32, 64, 128, 256, 512 or 1024 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.
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’s 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 4096 = 4Kbytes. The default value is 4 bytes (one 32 bit entry).

Table 2-4: **AXI4 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 will be 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 will automatically respond to all transfers and will 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 i.e. narrow transfers are not supported and WRAP and INCR transfers 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. NOTE: 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 will be an EXOKAY to exclusive accesses. A value of 0 means the slave does not support this so a response of OKAY will be automatically generated in response to exclusive accesses. Default is 1.
READ_BURST_DATA_TRANSFER_GAP	The configuration variable controls the gap between the read data transfers that comprise a read data burst. This value is an integer number and is measured in clock cycles. Default is 0. NOTE: If this is set to a value greater than zero *and* concurrent read bursts are called, then AXI4 protocol will be violated as the BFM will attempt to perform data interleaving.

Table 2-4: **AXI4 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 1 stops the simulation on and error. This configuration variable can be changed during simulation for error testing. NOTE: This is *not* used for timeout errors; such errors will always stop simulation.
CHANNEL_LEVEL_INFO	This configuration variable controls the printing of channel level information messages. When set to 1 info messages will be printed, when set to zero no channel level information will be 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 will be printed, when set to zero no function level information will be printed. The default (1) enables the function level info messages.

AXI4-Lite Master BFM

Table 2-5 contains a list of parameters and configuration variables which are supported by the AXI4-Lite Master BFM.

Table 2-5: **AXI4-Lite Master BFM Parameters**

BFM Parameters	Description
NAME	String name for the master BFM. This is used in the messages coming from the BFMs. The default for the master BFM is "MASTER_0".
DATA_BUS_WIDTH	Read and write data busses 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 will be 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 will always stop simulation.

Table 2-5: **AXI4-Lite Master BFM Parameters**

BFM Parameters	Description
CHANNEL_LEVEL_INFO	This configuration variable controls the printing of channel level information messages. When set to 1 info messages will be printed, when set to zero no channel level information will be 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 will be printed, when set to zero no function level information will be printed. The default (1) enables the function level info messages.

AXI4-Lite Slave BFM

Table 2-6 contains a list of parameters and configuration variables which are supported by the AXI4-Lite Slave BFM.

Table 2-6: **AXI4-Lite Slave BFM Parameters**

BFM Parameters	Description
NAME	String name for the slave BFM. This is used in the messages coming from the BFMs. The default for the slave BFM is “SLAVE_0”.
DATA_BUS_WIDTH	Read and write data busses 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’s 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 4096 = 4Kbytes. The default value is 4 bytes. i.e. 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 will be handled by stalling until at least one of the outstanding transactions has finished. Default is 8.

Table 2-6: **AXI4-Lite Slave BFM Parameters *(Cont'd)***

BFM Parameters	Description
MEMORY_MODEL_MODE	The parameter puts the slave BFM into a simple memory model mode. This means that the slave BFM will automatically respond to all transfers and will 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 i.e. narrow transfers are not supported and WRAP and INCR transfers 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. NOTE: 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. 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.

Table 2-6: **AXI4-Lite Slave BFM Parameters *(Cont'd)***

BFM Parameters	Description
STOP_ON_ERROR	This configuration variable is used to enable/disable the stopping of the simulation on an error condition. The default value of one stops the simulation on an error. This configuration variable can be changed during simulation for error testing. NOTE: This is *not* used for timeout errors; such errors will always stop simulation.
CHANNEL_LEVEL_INFO	This configuration variable controls the printing of channel level information messages. When set to 1 info messages will be printed, when set to zero no channel level information will be 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 will be printed, when set to zero no function level information will be printed. The default (1) enables the function level info messages.

AXI4-Stream Master BFM

Table 2-7 contains a list of parameters and configuration variables which are supported by the AXI4-Stream Master BFM.

Table 2-7: **AXI4-Stream BFM Parameters**

BFM Parameters	Description
NAME	String name for the master BFM. This is used in the messages coming from the BFMs. The default for the master BFM is "MASTER_0".
DATA_BUS_WIDTH	Read and write data busses can 32 or 64 bits wide only. Default is 32.
ID_BUS_WIDTH	Default is 8.
DEST_BUS_WIDTH	Default is 4.
USER_BUS_WIDTH	Default is 8

Table 2-7: **AXI4-Stream BFM Parameters**

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 will be 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 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 will always stop simulation.
CHANNEL_LEVEL_INFO	This configuration variable controls the printing of channel level information messages. When set to 1, info messages will be printed, when set to zero no channel level information will be printed. The default (1) enables channel level info messages.

AXI4-Stream Slave BFM

Table 2-8 contains a list of parameters and configuration variables which are supported by the AXI4-Stream Slave BFM.

Table 2-8: **AXI4-Stream Slave BFM Parameters**

BFM Parameters	Description
NAME	String name for the slave BFM. This is used in the messages coming from the BFMs. The default for the slave BFM is “SLAVE_0”.
DATA_BUS_WIDTH	Read and write data busses can be 32 or 64 bits wide only. Default is 32.
ID_BUS_WIDTH	Default is 8.
DEST_BUS_WIDTH	Default is 4.

Table 2-8: **AXI4-Stream Slave BFM Parameters**

BFM Parameters	Description
USER_BUS_WIDTH	Default is 8
MAX_OUTSTANDING_TRANSACTIONS	This defines the maximum number of outstanding transactions. Any attempt to generate more traffic while this limit has been reached will be handled by stalling until at least one of the outstanding transactions has finished. Default is 8.
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 will always stop simulation.
CHANNEL_LEVEL_INFO	This configuration variable controls the printing of channel level information messages. When set to 1, info messages will be printed, when set to zero no channel level information will be printed. The default (1) enables the channel level info messages.

Chapter 3

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 1] . For example, LEN = 0 as an input means a burst of length 1.

Tasks and functions common to all BFMs are described in Table 3-1.

Table 3-1: **Utility API Tasks/Functions**

API Task/Function Name and Description	Inputs	Outputs
report_status This function can be called at the end of a test to report the final status of the associated BFM.	dummy_bit: The value of this input can be 1 or 0 and does not matter. It is only required because a Verilog function needs at least 1 input.	report_status: This is an integer number which is calculated as: report_status = error_count + warning_count + pending_transactions_count
report_config This task prints out the current configuration as set by the configuration parameters and variables. This task can be called at any time.	None	None
set_channel_level_info This function sets the CHANNEL_LEVEL_INFO internal control variable to the specified input value.	LEVEL: A bit input for the info level.	None
set_function_level_info This function sets the FUNCTION_LEVEL_INFO internal control variable to the specified input value.	LEVEL: A bit input for the info level.	None
set_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

Table 3-1: **Utility API Tasks/Functions *(Cont'd)***

API Task/Function Name and Description	Inputs	Outputs
set_write_response_gap This function sets the SLAVE WRTE_RESPONSE_GAP internal control variable to the specified input value.	TIMEOUT: An integer value measured in clock cycles.	None
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 WRTE_BURST_DATA_TRANSFER_GAP internal control variable to the specified input value:	TIMEOUT: An integer value measured in clock cycles.	None

AXI3 Master BFM Test Writing API

The channel level API for the AXI3 Master BFM is detailed in Table 3-2.

Table 3-2: **Channel Level API for AXI3 Master BFM**

API Task Name and Description	Inputs	Outputs
SEND_WRITE_ADDRESS Creates a write address channel transaction. This task returns after the write address has been acknowledged by the slave. This task emits a "write_address_transfer_complete" event upon completion.	ID: Write Address ID tag ADDR: Write Address LEN: Burst Length SIZE: Burst Size BURST:Burst Type LOCK: Lock Type CACHE: Cache Type PROT: Protection Type	None
SEND_WRITE_DATA Creates a single write data channel transaction. The ID tag should be the same as the write address ID tag it is associated with. The data should be the same size as the width of the data bus. This task returns after is has been acknowledged by the slave. The data input will be used as raw bus data i.e. no realignment for narrow or unaligned data. This task emits a "write_data_transfer_complete" event upon completion. NOTE: Should be called multiple times for a burst with correct control of the LAST flag	ID: Write ID tag STOBE: Strobe signals DATA: Data for transfer LAST: Last transfer flag	None

Table 3-2: **Channel Level API for AXI3 Master BFM *(Cont'd)***

API Task Name and Description	Inputs	Outputs
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.	ID: Read Address ID tag ADDR: Read Address LEN: Burst Length SIZE: Burst Size BURST: Burst Type LOCK: Lock Type CACHE: Cache Type PROT: Protection Type	None
RECEIVE_READ_DATA This task drives the RREADY signal and monitors the read data bus for read transfers coming from the slave that have the specified ID tag. It then returns the data associated with the transaction and the status of the last flag. The data output here is raw bus data i.e. no realignment for narrow or unaligned data. This task emits a "read_data_transfer_complete" event upon completion. NOTE: This would need to be called multiple times for a burst > 1.	ID: Read ID tag	DATA: Data transferred by the slave RESPONSE: The slave read response from the following: [OKAY, EXOKAY, SLVERR, DECERR] LAST: Last transfer flag
RECEIVE_WRITE_RESPONSE This task drives the BREADY signal and monitors the write response bus for write responses coming from the slave that have the specified ID tag. It then returns the response associated with the transaction. This task emits a "write_response_transfer_complete" event upon completion.	ID: Write ID tag	RESPONSE: The slave write response from the following: [OKAY, EXOKAY, SLVERR, DECERR]

Table 3-2: **Channel Level API for AXI3 Master BFM *(Cont'd)***

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

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

Table 3-3: **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 complete 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]

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

API Task Name and Description	Inputs	Outputs
WRITE_BURST_CONCURRENT This task does the same function as the WRITE_BURST task; however, it performs the write address and write data phases concurrently.	ID: Write ID tag ADDR: Write Address LEN: Burst Length SIZE: Burst Size BURST: Burst Type LOCK: Lock Type CACHE: Cache Type PROT: Protection Type DATA: Data to send DATASIZE: The size in bytes of the valid data contained in the input data vector	RESPONSE:The slave write response from the following: [OKAY, EXOKAY, SLVERR, DECERR]
WRITE_BURST_DATA_FIRST This task does the same function as the WRITE_BURST task; however, it sends the write data burst before sending the associated write address transfer on the write address channel.	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 3-4.

Table 3-4: **Channel Level API for AXI3 Slave BFM**

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

Table 3-4: **Channel Level API for AXI3 Slave BFM *(Cont'd)***

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

Table 3-4: **Channel Level API for AXI3 Slave BFM *(Cont'd)***

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

The function level API for the AXI3 Slave BFM is detailed in Table 3-5.

Table 3-5: **Function Level API for AXI3 Slave BFM**

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

Table 3-5: **Function Level API for AXI3 Slave BFM *(Cont'd)***

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

AXI4 Master BFM Test Writing API

The channel level API for the AXI4 Master BFM is detailed in Table 3-6.

Table 3-6: **Channel Level API for AXI4 Master BFM**

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

Table 3-6: **Channel Level API for AXI4 Master BFM *(Cont'd)***

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

Table 3-6: **Channel Level API for AXI4 Master BFM *(Cont'd)***

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

The function level API for the AXI4 Master BFM is detailed in Table 3-7.

Table 3-7: **Function Level API for AXI4 Master BFM**

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

Table 3-7: **Function Level API for AXI4 Master BFM *(Cont'd)***

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

AXI4 Slave BFM Test Writing API

The channel level API for the AXI4 Slave BFM is detailed in Table 3-8.

Table 3-8: **Channel Level API for AXI4 Slave BFM**

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

Table 3-8: **Channel Level API for AXI4 Slave BFM *(Cont'd)***

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

Table 3-8: **Channel Level API for AXI4 Slave BFM *(Cont'd)***

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

The function level API for the AXI4 Slave BFM is detailed in Table 3-9.

Table 3-9: **Function Level API for AXI4 Slave BFM**

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

Table 3-9: **Function Level API for AXI4 Slave BFM *(Cont'd)***

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

AXI4-Lite Master BFM Test Writing API

The channel level API for the AXI4-Lite Master BFM is detailed in Table 3-10.

Table 3-10: **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.	STOBE: Strobe signals DATA: Data for transfer	None
SEND_READ_ADDRESS Creates a read address channel transaction. This task returns after the read address has been acknowledged by the slave. 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 3-11.

Table 3-11: **Function Level API for AXI4-Lite Master BFM**

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

AXI4-Lite Slave BFM Test Writing API

The channel level API for the AXI4-Lite Slave BFM is detailed in Table 3-12.

Table 3-12: **Channel Level API for AXI4-Lite Slave BFM**

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

Table 3-12: **Channel Level API for AXI4-Lite Slave BFM *(Cont'd)***

API Task Name and Description	Inputs	Outputs
RECEIVE_READ_ADDRESS This task drives the ARREADY signal and monitors the read address bus for read address transfers coming from the master. It returns the data associated with the read address transaction. This task uses the SLAVE_ADDRESS and SLAVE_MEM_SIZE parameters to determine if the address is valid. This task emits a "read_address_transfer_complete" event upon completion.	ADDR: Read Address ADDRValid: Bit to indicate if the address input parameter is to be used. When set to 1 the ADDR is valid and used, when set to 0 it is ignored.	PROT: Protection Type SADDR: Sampled Read Address
RECEIVE_WRITE_DATA This task drives the WREADY signal and monitors the write data bus for write transfers coming from the master. It returns the data associated with the transaction. This task emits a "write_data_transfer_complete" event upon completion.	None	DATA: Data transferred from the master STRB:Strobe signals used to validate the data

The function level API for the AXI4-Lite Slave BFM is detailed in Table 3-13.

Table 3-13: **Function Level API for AXI4-Lite Slave BFM**

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

Table 3-13: **Function Level API for AXI4-Lite Slave BFM *(Cont'd)***

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

AXI4-Stream Master BFM Test Writing API

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

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

API Task Name and Description	Inputs	Outputs
SEND_TRANSFER Creates a single AXI4-Stream transfer. This task emits a "transfer_complete" event upon completion.	ID: Transfer ID Tag DEST:Transfer Destination DATA:Transfer Data STRB: Transfer Strobe Signals KEEP:Transfer Keep Signals LAST: Transfer Last Signal USER: Transfer User Signals	None

AXI4-Stream Slave BFM Test Writing API

The channel level API for the AXI4-Stream Slave BFM is detailed in Table 3-15.

Table 3-15: **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

XILINX®

Chapter 4

Protocol Checking

The purpose of the AXI BFMs is to verify connectivity and basic functionality of AXI masters and AXI slaves. A basic level of protocol checking is included with the AXI BFMs. For comprehensive protocol checking, the Cadence AXI UVC should be deployed [Ref 2].

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

- 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 BFMs, respectively, are checked to ensure they are valid to prevent protocol violations.
- Address ranges are checked in the Slave BFMs.

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

Common BFM Checkers

The AXI checkers that are implemented as Verilog tasks and are common to both the master and slave BFMs are described in Table 4-1.

Table 4-1: **Common BFM Checker Tasks**

Checker Task Name	Inputs	Description
check_burst_type	BURST_TYPE	Checks to see if the burst type value is valid.
check_burst_length	BURST_TYPE LENGTH LOCK_TYPE	Checks to see if the burst length value is valid given the burst type. NOTE: LOCK_TYPE input added for AXI4 only. In AXI4, exclusive accesses must be 16 beats or less in length. Also only INCR bursts can be greater than 16 beats in length.
check_burst_size	SIZE	Checks that the burst size is not greater than the data bus size.
check_lock_type	LOCK_TYPE	Checks if the lock type value is valid. NOTE: AXI4 reduces this to a single bit: Normal access = 0, Exclusive access =1.

Table 4-1: **Common BFM Checker Tasks *(Cont'd)***

Checker Task Name	Inputs	Description
check_cache_type	CACHE_TYPE	Checks if the cache type value is valid. NOTE: Different valid ranges for AXI4.
check_address	ADDRESS BURST_TYPE SIZE	Checks to see if the address is valid given the burst type and the transfer size. For example, a WRAP burst with an address which is not aligned to the transfer size is illegal.

The AXI 3 and AXI4 checkers that are implemented as tasks and are common to the master and slave BFMs are located in the following Verilog files:

- `cdn_axi3_bfm_checkers.v` - AXI3 common checking tasks
- `cdn_axi4_bfm_checkers.v` - AXI4 common checking tasks
- `cdn_axi4_lite_bfm_checkers.v` - AXI4-Lite common checking tasks
- `cdn_axi4_streaming_bfm_checkers.v` - AXI4-Stream common checking tasks

BFM Specific Checkers

Table 4-2 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 4-2: **BFM Specific Checker Tasks**

Checker Task Name	Inputs	Checker Location/s	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.

XILINX®

Chapter 5

Directory Structure

The following chapter describes the directory structure of the AXI BFM solution.

The AXI BFM solution directory (`ISE_Version_Number/ise_ds/ise`) includes the following folders:

- AXI_BFM
  - mti

    Supports ModelSim's native encryption
    - cdn_axi_bfm_vip
      - vpi_lib
      - hdl
      - examples
  - p1735

    Supports IEEE's Standard Verilog encryption supported by the Cadence simulator
    - cdn_axi_bfm_vip
      - vpi_lib
      - hdl
      - examples

cdn_axi_bfm_vip

The name of the AXI BFM verification IP package is “cdn_axi_bfm_vip” and its directory and file structure is illustrated below:

vpi_lib

The docs directory contains the following file:

- ug783_axi_bfm.pdf

hdl

The hdl directory contains all of the source code for the AXI BFMs, including the following files:

- cdn_axi3_master_bfm.v
- cdn_axi3_slave_bfm.v
- cdn_axi3_bfm_defines.v
- cdn_axi3_bfm_checkers.v
- cdn_axi4_master_bfm.v
- cdn_axi4_slave_bfm.v
- cdn_axi4_bfm_defines.v
- cdn_axi4_bfm_checkers.v
- cdn_axi4_lite_master_bfm.v
- cdn_axi4_lite_slave_bfm.v
- cdn_axi4_lite_bfm_defines.v
- cdn_axi4_lite_bfm_checkers.v
- cdn_axi4_streaming_master_bfm.v
- cdn_axi4_streaming_slave_bfm.v
- cdn_axi4_streaming_bfm_defines.v
- cdn_axi4_streaming_bfm_checkers.v

examples

The examples directory contains the following:

- An example test bench and test for each AXI BFM pair. Descriptions of these test benches and example tests are described in Chapter 6, AXI4 BFM Example Designs.
- A Makefile to run the example tests:
  - The make file should be called as follows:

    ```
    make TEST=<test_file_name.v> <target>
    ```

  - Targets:
    - clean - This option cleans all compiled files and log files
    - example_test_axi3 - Target for any AXI3 example test
    - example_test_axi4 - Target for any AXI4 example test
    - example_test_axi4_lite - Target for any AXI4-Lite example test

    - example_test_axi4_streaming - Target for any AXI4-Stream example test
  - All of the above targets (except clean) can have an "_i" appended to the end to start interactive mode (starts the test and tools in GUI mode).

All of the above targets (except clean) can have an "ms_" pre-pended to the start of the target to enable simulation with ModelSim

The examples directory includes the following files:

- Makefile
- cdn_axi_test_level_api.v
- cdn_axi3_example_tb.v
- cdn_axi3_example_test.v
- cdn_axi3_example_memory_model_test.v
- cdn_axi4_example_tb.v
- cdn_axi4_example_test.v
- cdn_axi4_example_memory_model_test.v
- cdn_axi4_lite_example_tb.v
- cdn_axi4_lite_example_test.v
- cdn_axi4_lite_example_memory_model_test.v
- cdn_axi4_streaming_example_tb.v
- cdn_axi4_streaming_example_test.v

Chapter 6

AXI4 BFM Example Designs

This chapter describes the example test benches and example tests used to demonstrate the abilities of each AXI BFM pair. 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.

AXI3 BFM Example Test Bench and Tests

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

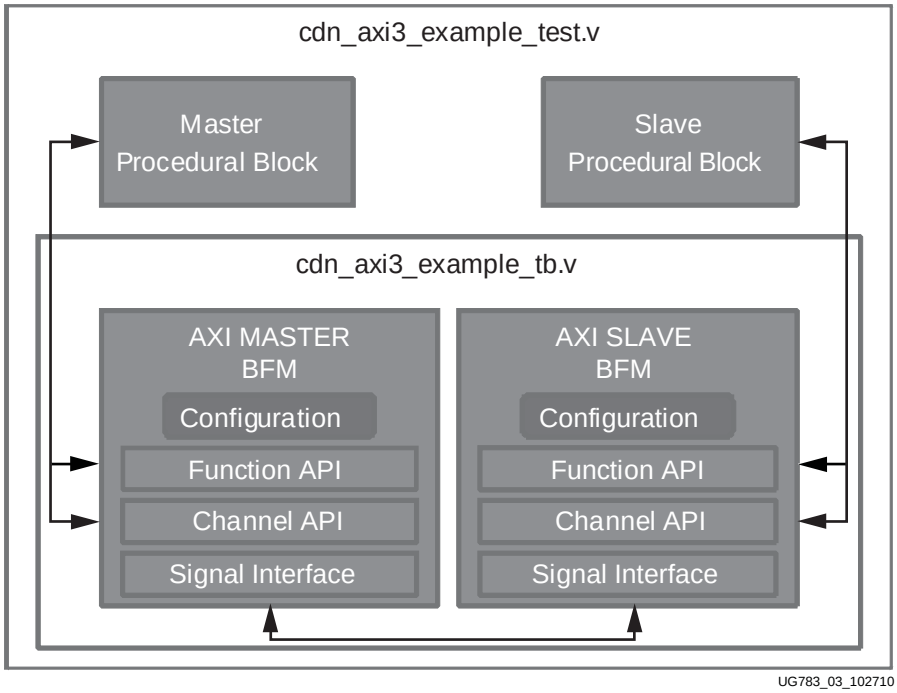

Figure 6-1: **Example Test Bench and Test Case Structure**

The example test bench has the master and slave BFMs 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 tests.

cdn_axi3_example_test.v

The example test (`example/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

cdn_axi3_example_memory_mode_test.v

The example test (`example/cdn_axi3_example_memory_mode_test.v`) contains the slave code to ensure that the slave BFM is configured as a 4 KB memory model. The master code in this test writes maximum length bursts into the memory and reads them back. It does this with two different sets of test values.

AXI4 BFM Example Test Bench and Tests

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

cdn_axi4_example_test.v

The example test (`example/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

cdn_axi4_example_memory_mode_test.v

The example test (`example/cdn_axi4_example_memory_mode_test.v`) contains the slave code to ensure that the slave BFM is configured as a 4 KB memory model. The master code in this test writes maximum length bursts into the memory and reads them back. It does this with two different sets of test values.

AXI4-Lite BFM Example Test Bench and Tests

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

cdn_axi4_lite_example_test.v

The example test (`example/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. Unaligned write and read transfers example
5. Write Bursts performed in two different ways; Data before address, and data with address concurrently
6. Outstanding transactions example
7. Slave read and write bursts error response example
8. Write and Read bursts with different length gaps between channel transfers example
9. Write burst that has valid data size less than the data bus width

cdn_axi4_lite_example_memory_mode_test.v

The example test (`example/cdn_axi4_lite_example_memory_mode_test.v`) contains the slave code to ensure that the slave BFM is configured as a 4 KB memory model. The master code in this test simply writes data transfers into the memory and reads them back. It does this with two different sets of test values.

AXI4-Stream BFM Example Test Bench and Tests

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

cdn_axi4_streaming_example_test.v

The example test (`example/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

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 BFMs. For example:

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

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

This for loop cycles through the following stimulus:

- Access Type: nFIXED, INCR
- Burst Size:k1_BYTE, 2_BYTES, 4_BYTES
- Burst Length:i1 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 BFMs 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 BFMs: 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 BFMs. This example shows the code used for the AXI3 Slave BFM memory model mode, starting with the write data path:

```
//-----------------------------------------------------------------//
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 above, it is possible to create the write data path 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 (NOTE: More code could be added here to support DECERR or SLVERR response types). Once 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 data path for a simple slave memory model:

```
//-----------------------------------------------------------------
// Read Path
//
-----------------------------------------------------------------alway
s @(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 above, it is possible to create the read data path 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 via 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.