Zynq-7000 Bus Functional Model

DS897 May 24, 2013 **Product Specification**

Introduction

The Zynq™-7000 Bus Functional Model (BFM) supports the functional simulation of Zynq-7000 based applications. It is targeted to enable the functional verification of Programmable Logic (PL) by mimicking the PS-PL interfaces and OCM/DDR memories of Processor System (PS) logic. This BFM is delivered as a package of encrypted Verilog modules. BFM operation is controlled by using a sequence of Verilog tasks contained in a Verilog-syntax file.

Features

- Pin compatible and Verilog Based simulation model.
- Supports all AXI interfaces.
  - AXI 3.0 compliant.
- Sparse memory model (for DDR) and a RAM model (for OCM).
- Verilog task-based API.
- Delivered in Vivado Design Suite.
- Blocking and non-blocking interrupt support
- Requires license to AXI BFM models.

LogiCORE IP Facts Table	
Core Specifics	
Supported Device Family(1)	Zynq™-7000
Supported User Interfaces	Master: M_AXI_GP Slave: S_AXI_GP, S_AXI_HP, S_AXI_ACP
Provided with Core	
Documentation	Product Specification
Design Files	Not Provided
Example Design	Verilog
Test Bench	Verilog
Constraints File	Not Provided
Simulation Model	Verilog
Supported S/W Driver	N/A
Tested Design Tools	
Design Entry Tools	Vivado™ Design Suite
Simulation(2)	Mentor Graphics Questa®SIM, Xilinx® Vivado Simulator/ISim v2013.1 Cadence IES Synopsys VCS and VCS MX
Support	
Provided by Xilinx @ www.xilinx.com/support	

1. For a complete listing of supported devices, see the Vivado IP Catalog.
2. Zynq-7000 BFM can be simulated with Cadence and Synopsys simulators outside of the Vivado IDE.

Applications

The Zynq-7000 BFM is intended to provide a simulation environment for the Zynq-7000 PS logic, typically replacing the processing_system7 block in a design. The Zynq-7000 BFM models the following:

- Transactions originating from PS masters through the AXI BFM master API calls
- Transactions terminating through the PS slaves to models of the OCM and DDR memories through interconnect models
- FCLK reset and clocking support
- Input interrupts to the PS from PL
- PS register map

Functional Description

Zynq-7000 BFM consists of four main layers. Figure 1 show the Zynq-7000 BFM architecture.

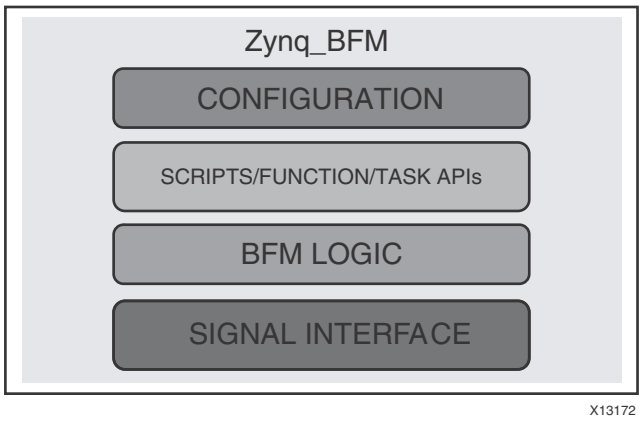

Figure 1: **Zynq-7000 BFM Architecture**

- Configuration

  Configuration is implemented using Verilog parameters and is used to configure the Zynq-7000 BFM. Some configuration must be done using configuration APIs.
- Function and Task APIs

  Verilog tasks and functions that help to set:
  - Datapath between processing system (PS) and programmable logic (PL) in memory mapped interfaces.
  - Control path between PS and PL in register interface.
  - Configure the traffic profiles for each ports.
- BFM Logic

  BFM logic has the PS-PL interface with supporting functionality that contains the AXI interfaces, sparse memory implementation, and the interconnect (arbiter) model as shown in Figure 2.
- Signal Interface

  The signal interface includes the typical Verilog input and output ports and associated signals.

Figure 2 shows the detailed architecture for the BFM logic.

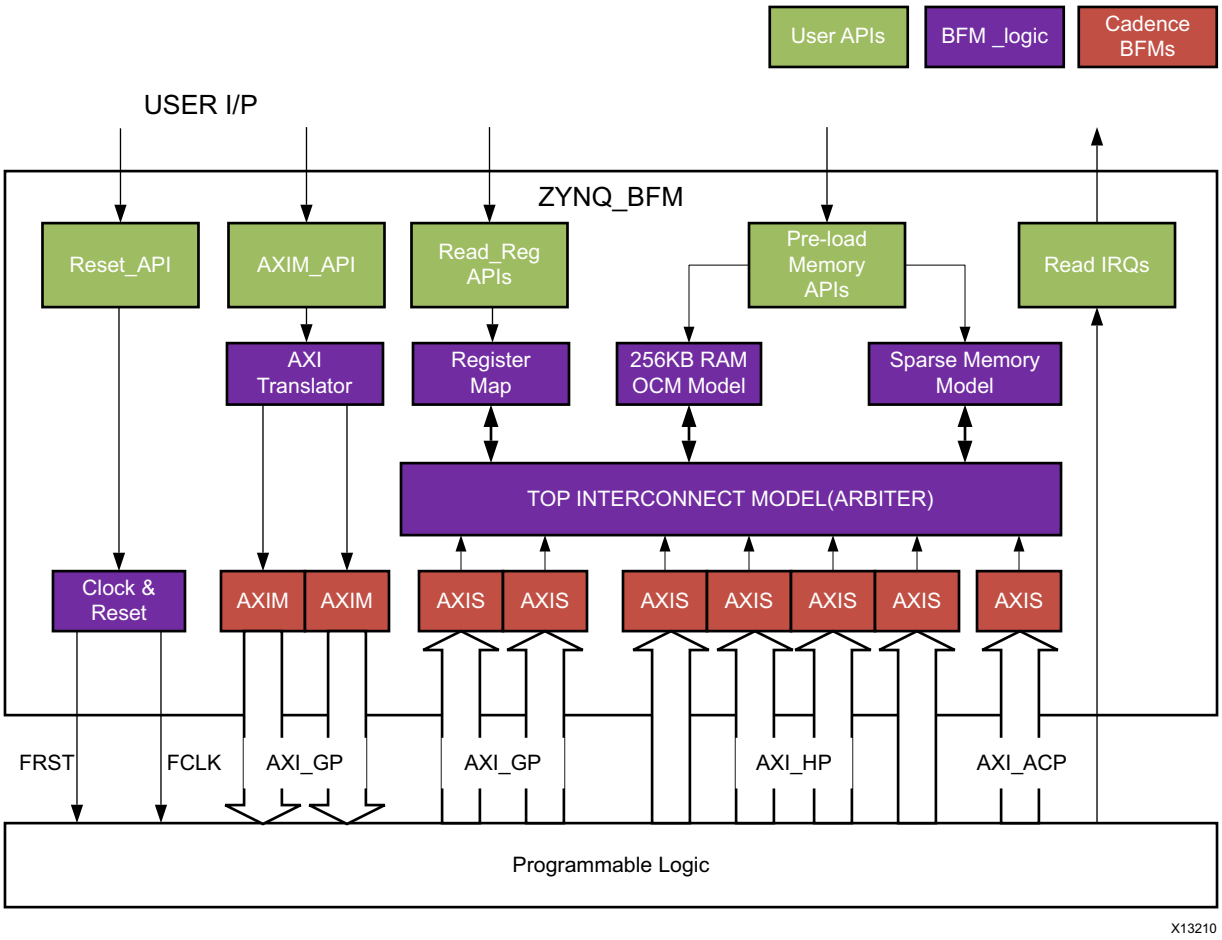

Figure 2: **Zynq-7000 BFM Architecture Details**

Figure 3 show the Zynq-7000 BFM test bench.

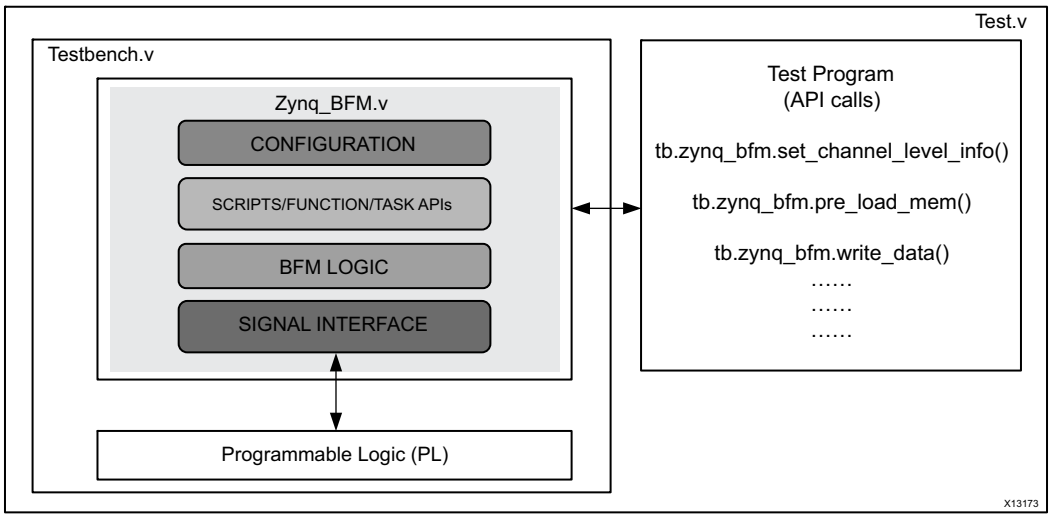

Figure 3: **Zynq-7000 BFM Test Bench**

Feature Details

The Zynq-7000 BFM is enabled with the following features:

- Pin compatible and Verilog based simulation model.
- Supports all PS AXI interfaces.
- AXI 3.0 compliant.
- 32/64–bit Data-width for AXI_HP, 32-bit for AXI_GP and 64-bit for AXI_ACP.
- ID width support as per the Zynq-7000 specification.
- Support for FIXED, INCR and WRAP transaction types.
- Support for all Zynq-7000 supported burst lengths and burst sizes.
- Protocol checking, provided by the AXI BFM models.
- Read/Write request capabilities
- System Address Decode for OCM/DDR transactions.
- System Address Decode for Register Map Read transactions (only default value of the registers can be read).
- Support for static remap for OCM address.
- Configurable latency for Read/Write responses.
- First-level arbitration scheme based on the priority indicated by the AXI QoS signals.
- Datapath connectivity between any AXI master in PL and the PS memories and register map.
- Parameters to enable and configure AXI Master and Slave ports.
- APIs to set the traffic profile and latencies for different AXI Master and Slave ports.
- Support for FPGA logic clock generation.
- Soft Reset Control for the PL.
- Sparse memory model (for DDR) and Block RAM (for OCM).
- API support to pre-load the memories, read/wait for the interrupts from PL, and checks for certain data pattern to be updated at certain memory location.
- All unused interface signals that output to the PL are tied to a valid value.
- Semantic checks on all other unused interface signals.
- An example design that demonstrates the usage of this BFM is available for reference.

Limitations

- Support for in-order transactions only.
- Exclusive Access transfers are not supported on any of the slave ports.
- Read/Write data interleaving is not supported.
- Write access to the Register Map is not supported.

Using Zynq-7000 BFM

This section details the configuration parameters and the APIs necessary for using the Zynq-7000 BFM. It also explains how to run the sample Verilog tests.

The interface models in the Zynq-7000 BFM are based on the AXI BFM models that are delivered with Vivado, and requires a license for use. See Licensing and Ordering Information for more information.

APIs

The APIs in Table 1 can be used to configure the BFM and develop a test program.

Table 1: **Zynq-7000 BFM APIs**

APIs	Inputs	Outputs
set_stop_on_error When set to default value '1', Stop the simulation on error.	**LEVEL**: A bit input for the info level.	None
set_channel_level_info When set to default value '1', channel level info for each AXI BFM is reported.	**Name**: M_AXI_GP0, M_AXI_GP1, S_ AXI_GP0, S_AXI_GP1, S_AXI_HP0, S_AXI_HP1, S_AXI_HP2, S_AXI_HP3, S_AXI_ACP or ALL **LEVEL**: A bit input for the info level.	None
set_function_level_info When set to default value '1', function level info for each AXI BFM is reported.	**Name**: M_AXI_GP0, M_AXI_GP1, S_ AXI_GP0, S_AXI_GP1, S_AXI_HP0, S_AXI_HP1, S_AXI_HP2, S_AXI_HP3, S_AXI_ACP or ALL **LEVEL**: A bit input for the info level.	None
set_debug_level_info When set to default value '1', debug level info for Zynq-7000 BFM is reported, else no info is reported.	**LEVEL**: A bit input for the info level.	None
set_arqos Set the ARQOS value to be used by Slave ports for first level arbitration scheme.	**Name**: S_ AXI_GP0, S_AXI_GP1, S_AXI_HP0, S_AXI_HP1, S_AXI_HP2, S_AXI_HP3, S_AXI_ACP **[3:0] val**: ARQoS value.	None
set_awqos Set the AWQOS value to be used by Slave ports for first level arbitration scheme.	**Name**: S_ AXI_GP0, S_AXI_GP1, S_AXI_HP0, S_AXI_HP1, S_AXI_HP2, S_AXI_HP3, S_AXI_ACP **[3:0] val**: AWQoS value.	None
fpga_soft_reset Issue/Generate soft reset for PL.	**[31:0] reset_ctrl** : 32-bit input indicating the reset o/p to be asserted for PL. (Details same as FPGA_RST_CTRL register defined in PS)	None
pre_load_mem_from_file Preload DDR/OCM with data from a file. Based on the address specified, the data will be loaded in DDR/OCM. **DDR**: Address must be 32-bit aligned. **OCM**: Address must be 32-bit aligned.	**[1023:0] file_name**: File name (max. 128 characters) **[31:0] start_addr**: Start Address from where DDR/OCM should be initialized with data from the file. **no_of_bytes**: Number of data bytes to be loaded	None

Table 1: **Zynq-7000 BFM APIs *(Cont'd)***

APIs	Inputs	Outputs
pre_load_mem Preload DDR/OCM with random_data/all_zeros/all_ones. Based on the address specified, the data will be loaded in DDR/OCM. DDR: Address must be 32-bit aligned. OCM: Address must be 32-bit aligned.	**[1:0] data_type**: Random, zeros or ones. 01- Zeros, 10 - Ones. **[31:0] start_addr**: Start Address from where DDR should be initialized with data from the file. **no_of_bytes**: Number of data bytes to be loaded	None
read_interrupt Users can use this API to poll the interrupt status at any given time. This will be a non-blocking task which will return immediately with the current status of the interrupt pins. 4 interrupt lines are currently supported.	None	**[3:0] irq_status**: Interrupts generated by PL.
wait_interrupt Users can use this API to wait for any of the interrupt to occur. This will be a blocking task and will return only any of the interrupt pin is asserted. 4 interrupt lines are currently supported.	None	**[3:0] irq_status**: Interrupts generated by PL.
wait_mem_update Users can use this API to poll for a specific location. This task is a blocking task and returns when some value is written in that location (either a specific value or a new value). If the value does not match the expected data pattern then a fatal error is issued and the simulation halts. Only One instance of this API should be used at a time.	**[31:0] addr**: 32-bit address (DDR/OCM) [**31:0] data_i**: expected data pattern	**[31:0] data_o**: data value that is updated in the memory
write_from_file Initiate a AXI write transaction on the GP master port . The write data will be used from the file. Burst type used is INCR. This will be a blocking task and will return only after the complete AXI WRITE transaction is done. Address has to 32-bit aligned.	**[1023:0] file_name** : File name that stores the write data (max. 128 characters). **[31:0] addr**: Write address **wr_size**: Number of data bytes to written	**RESPONSE**: The slave write response from the following: [OKAY, EXOKAY, SLVERR, DECERR]
read_to_file Initiate a AXI read transaction on the GP master port. The read data will be written to the file. Burst type used is INCR. This will be a blocking task and will return only after the complete AXI READ transaction is done. Address has to 32-bit aligned.	**[1023:0] file_name** : File name that stores the read data (max. 128 characters). **[31:0] addr**: Read address **rd_size**: Number of data bytes to be read	**RESPONSE**: The slave write response from the following: [OKAY, EXOKAY, SLVERR, DECERR]
write_data Initiate a AXI write transaction on the GP master port. This task should be used when the transfer size is less or equal to 128 bytes and the write data is provided as an argument to the task call. Burst type used is INCR. This will be a blocking task and will return only after the AXI write is completed via a write response (BRESP).	**[31:0] addr**: Write address **[7:0] wr_size**: Number of data bytes to be written **[1023:0] wr_data**: write data (max. 128 bytes).	**RESPONSE**: The slave write response from the following: [OKAY, EXOKAY, SLVERR, DECERR]

Table 1: **Zynq-7000 BFM APIs *(Cont'd)***

APIs	Inputs	Outputs
read_data Initiate a AXI read transaction on the master port. This task should be used when the transfer size is less or equal to 128 bytes and the read data is returned as an output to the task call. Burst type used is INCR. This will be a blocking task and will return only after the complete AXI READ transaction is done.	**[31:0] addr**: Write address **[7:0] rd_size**: Number of data bytes to be read	**DATA**: Valid data transferred by the Slave **RESPONSE**: The slave write response from the following: [OKAY, EXOKAY, SLVERR, DECERR]
write_burst Initiate a single AXI write burst transaction on the master port. This calls the AXI BFM API. This task returns when the complete write transaction is complete.	**ID**: Write ID tag **ADDR**: Write Address **LEN**: Burst Length **SIZE**: Burst Size **BURST**: Burst Type **LOCK**: Lock Type **CACHE**: Cache Type **PROT**: Protection Type **DATA**: Data to send **DATASIZE**: The size in bytes of the valid data contained in the input data vector.	**RESPONSE**: The slave write response from the following: [OKAY, EXOKAY, SLVERR, DECERR]
write_burst_concurrent Initiate a single AXI write burst transaction on the master port. This calls the AXI BFM API. This task performs write phase and address phases concurrently and returns when the complete write transaction is complete.	**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]
read_burst Initiate a single AXI Read burst transaction on the master port. This is hook to call the AXI BFM API. This task returns when the complete read transaction is complete.	**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**: Valid data transferred by the slave **RESPONSE**: This is a vector that is created by concatenating all slave read responses together
read_register Read Register.	**[31:0] addr**: Register address	**DATA**: Valid register read data
read_register_map Read a chunk of registers. 32 registers can be read at a time using this API.	**[31:0] addr**: Starting Register address **size**: Number of registers to be read.	**DATA**: Valid register read data

Table 1: **Zynq-7000 BFM APIs *(Cont'd)***

APIs	Inputs	Outputs
set_slave_profile Set the Slave Profile for each slave port. The latency to be used on all slave ports is set using this API.	**Name**: S_AXI_GP0 , S_AXI_GP1, S_AXI_HP0, S_AXI_HP1, S_AXI_HP2, S_AXI_HP3, S_AXI_ACP or ALL **[1:0] latency**: latency type **00**: Best case **01**: Average case **10**: Worst case **11**: Random	None
wait_reg_update This API can be used to wait for a particular address in PL to be updated with a specific pattern. Zynq-7000 BFM will issue an continuous AXI Read command with the specified address until the read data matches the expected data pattern. The time interval between the reads can be decided by the user. This will be a blocking task and will return only when the read data matches the expected data pattern or timeout. Only One API instance per GP Master port should be used at a time.	**[31:0] addr**: Register address **[31:0] data_i**: expected data pattern. **[31:0] mask_i**: Mask indicating the bits that are masked (1-indicates bit is masked from read/write). **time_interval**: number of clock cycles to wait in between reads. **time_out**: number of clock cycles to wait for the register update.	**[31:0] data_o**: data value that is updated in the register.

Integrating Zynq-7000 BFM

Zynq-7000 BFM must be instantiated in a common top file along with any desired PL logic with all necessary connections between the PL and the BFM created in this file.

Configuration Options

Table 2 shows the configuration options that are passed to the BFM through Verilog parameters.

Table 2: **Configuration Options Using Verilog Parameters**

BFM Parameter	Default Value	Description
C_FCLK_CLK0_FREQ	50	PL Clock Frequency in MHz for FCLK0
C_FCLK_CLK1_FREQ	50	PL Clock Frequency in MHz for FCLK1
C_FCLK_CLK2_FREQ	50	PL Clock Frequency in MHz for FCLK2
C_FCLK_CLK3_FREQ	50	PL Clock Frequency in MHz for FCLK3
C_HIGH_OCM_EN	0	When set to '1', enables the high address range for OCM.
C_USE_S_AXI_HP0	0	When set to '1', enables the S_AXI_HP0 Slave port
C_USE_S_AXI_HP1	0	When set to '1', enables the S_AXI_HP1 Slave port
C_USE_S_AXI_HP2	0	When set to '1', enables the S_AXI_HP2 Slave port
C_USE_S_AXI_HP3	0	When set to '1', enables the S_AXI_HP3 Slave port
C_S_AXI_HP0_DATA_WIDTH	32	Set the data-width for S_AXI_HP0 port
C_S_AXI_HP1_DATA_WIDTH	32	Set the data-width for S_AXI_HP1 port
C_S_AXI_HP2_DATA_WIDTH	32	Set the data-width for S_AXI_HP2 port
C_S_AXI_HP3_DATA_WIDTH	32	Set the data-width for S_AXI_HP3 port
C_USE_M_AXI_GP0	0	When set to '1', enables the M_AXI_GP0 Slave port
C_USE_M_AXI_GP1	0	When set to '1', enables the M_AXI_GP1 Slave port
C_USE_S_AXI_GP0	0	When set to '1', enables the S_AXI_GP0 Slave port
C_USE_S_AXI_GP1	0	When set to '1', enables the S_AXI_GP1 Slave port
C_USE_S_AXI_ACP	0	When set to '1', enables the S_AXI_ACP Slave port

Example Design

An example design and test bench named “zynq_bfm_example” is attached with AR 55345.

http://www.xilinx.com/support/answers/55345.htm

Support

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

Licensing and Ordering Information

The Zynq-7000 Bus Functional Model is provided under the terms of the Xilinx Core License Agreement. A full license for the model must be purchased and obtained from Xilinx. To access the full functionality of the core, visit the Zynq-7000 Bus Functional Model web page. Contact your local Xilinx sales representative. for pricing and availability of additional Xilinx modules and software. Information about additional Xilinx solutions is available on the Xilinx Intellectual Property.

Revision History

The following table shows the revision history for this document:

Date	Version	Description of Revisions
03/20/2013	1.0	Initial Xilinx release.
05/24/2013	1.1	Updated for Simulator support and Example Design section with link to AR.

Notice of Disclaimer