Cadence CXL/AXI Verilog Delivery Testbench

User Guide

31 August 2020

Cadence Design Systems, Inc. (Cadence), 2655 Seely Ave., San Jose, CA 95134, USA.

Contents

Chapter 1

Verilog Delivery Testbench Overview

Topics:

Introduction

The Cadence® PCIe® Verilog Delivery Testbench provides a simple Verilog-based test environment to enable you to generate and observe inbound and outbound traffic through a PCIe controller and PHY, if operating in a Subsystem mode.

It is a self-contained environment which does not required any VIP (*Verification IP*).

Note:

- The traffic and scenarios generated with this testbench are limited when compared to an environment utilising PCIe and ARM® AMBA® VIP.
- The verilog delivery testbench is not intended to verify the functionality of the controller, PHY, or the integration of the controller with the PHY.

This document defines verilog delivery testbench that enables you to run simple test case that demonstrates the Cadence® PCIe IP based features.

Simulator and LRM Compatibility

The testbench is tested on the following simulator version:

Simulator: Cadence® Xcelium® 19.03 (19.03.003)

Verilog LRM Version: Verilog-2005

Chapter 2

Verilog Delivery Testbench Architecture

Topics:

Architecture Overview

This section describes the Verilog Delivery Testbench common architecture which may be shared between variants of the Cadence PCIe IP. Figure 1: Hybrid subsystem testbench block diagram on page 9 shows the block diagram of Testbench Architecture for a PCIe Hybrid Subsystem.

The verilog delivery testbench is architected as a mirrored verilog delivery testbench solution, where the PCIe IP is instantiated as the DUT and is mirrored as the Testbench Reference BFM on the other end of the PCIe PIPE or Serial interface. Depending on the DUT configuration chosen either as PCIe EndPoint or PCIe Root Port, the Reference BFM is selected as the other. This section describes various DUT, Client interface and Register interfaces that are supported by the PCIe verilog delivery testbench.

The verilog delivery testbench demonstrates:

- The DUT integration to the testbench.
- Initialization and Enumeration.
- PCIe Link up.
- Speed-up operation.
- Basic traffic scenarios.

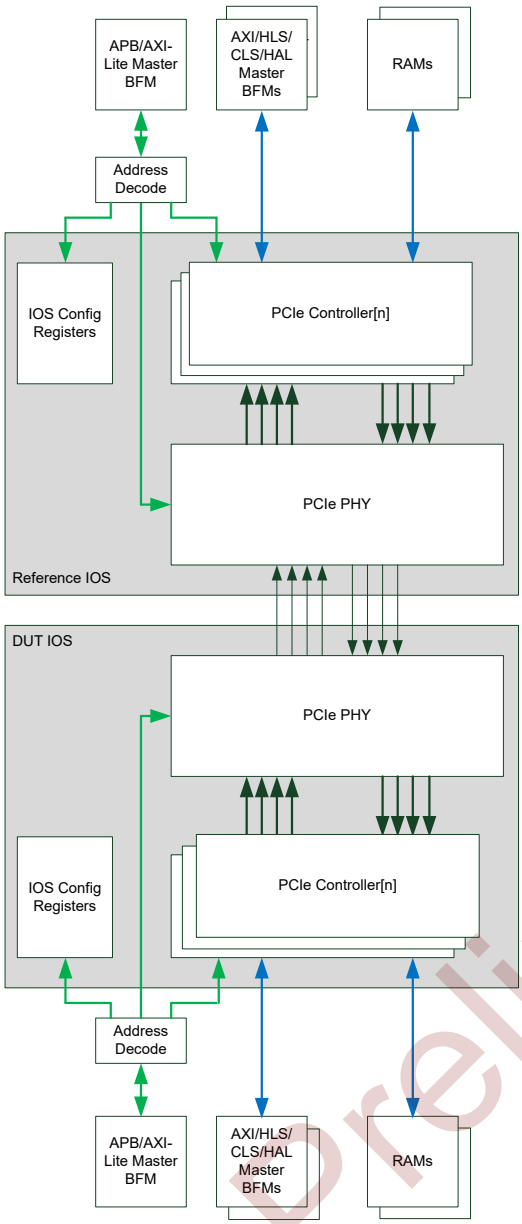

Figure 1: Hybrid subsystem testbench block diagram

Note: The figure is for representative purpose only, and the blocks are applicable based on the configuration selected.

Testbench Configuration

The testbench provides the following configuration and operation mode:

- **Serial mode**

  In this mode, the testbench instances two PCIe Subsystems, each having an Integrated Controller and PHY connected at the Serial link.

Client Interfaces

Verilog BFMs (*Bus Functional Models*) are connected to the application level interfaces of both the DUT and the Tesbench Reference BFM. The HLS and CLS BFMs connect to the Outbound and Inbound client interfaces of the PCIe Controller. The HLS and AXI BFMs connect to the Outbound and Inbound client interfaces of the PCIe Controller. When required by DT configuration, the testbench instances DUT SRAMs models and connects them to the relevant external SRAM interfaces of the PCIe Controller.

Register Interfaces

The testbench supports the following Register Configuration Mode of the PCIe Controller:

- APB.

The APB BFM connects to the register configuration ports for the PCIe Controller, the PHY and the configuration registers within the PCIe Subsystem.

Chapter 3

Getting Started

Topics:

Installating Testbench

This section describes how to install and use the testbench.

The testbench is shipped as either part of a full IP release or a standalone single file in the TAR/GZIP format.

- If provided as a full IP release, ensure that you follow the installation instructions provided with the release.
- If provided as a standalone release, ensure that you follow the steps below to install it to your system.
- If you have requested shipment on other media, ensure that you follow the instructions on the media label.

To install the testbench, execute the following steps:

1. `cp CDNS_PCIe_B2B_TB.tar.gz .`

   Copies the CDNS_PCIe_B2B_TB delivery pack file to your local directory.
2. `tar -zvxf CDNS_PCIe_B2B_TB.tar.gz`

   Un-tars the archive. On excuting this command, you will have the Testbench directory that contains all deliverables.

   Note: The testbench requires a Verilog simulator to be installed on your system and has been tested with only a subset of simulators.

Directory Structure

Figure 2: Verilog delivery testbench directory structure on page 13 shows the directory structure of the testbench.

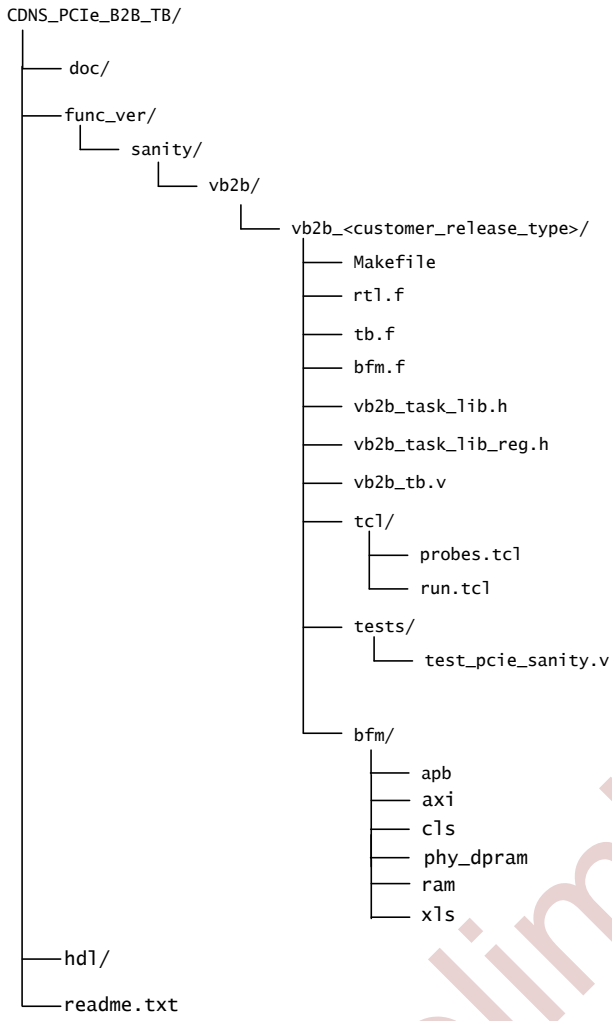

Figure 2: Verilog delivery testbench directory structure

Test Flow

The test is the top-level module and instances the testbench.

The test configures the static configuration inputs to the PCIe IP through hierarchical signal name assignments. Register programming and various Client Interface transfers are performed through abstracted testbench task calls.

A typical test flow is:

1. Sets static configuration values (speed, number of lanes).
2. Configures PHYs via Register Interface BFM.
3. De-asserts PIPE resets.
4. Waits for PCIe link to train.

Test Flow for PCIe

AXI

1. Configures AXI Outbound Sideband regions and addresses translation through AXI-Lite.
2. Initiates traffic through AXI writes or reads to the reference or DUT cores.

The testbench self-checks the success of Memory, IO or Config write operations by a subsequent read operation.

Test Flow for CXL

CLS

1. Configures the Alternate Protocol Negotiation ports.
2. Configures the CXL mode to enable the CXL mode during link training.
3. Once link is trained to L0 and at least 8GT/s, the HLS BFM sends CXL.io transactions to enumerate the CXL device and configure CXL mode of operation
4. CLS inbound and outbound interfaces are then activated and the CLS BFM sends CXL.cache and CXL.mem messages.

Chapter 4

Application Level Interface BFMs

Topics:

CLS BFM

The CLS BFM generates and sends the CXL messages on the Tx side and receive the CXL messages from the Rx side. The Tx side of BFM performs the following functions for generating all the different CXL messages:

- F_GET_CXL_CACHE_D2H_REQ
- F_GET_CXL_CACHE_D2H_RESP
- F_GET_CXL_CACHE_D2H_DATA
- F_GET_CXL_CACHE_H2D_REQ
- F_GET_CXL_CACHE_H2D_RESP
- F_GET_CXL_CACHE_H2D_DATA
- F_GET_CXL_CACHE_M2S_REQ
- F_GET_CXL_CACHE_M2S_RWD
- F_GET_CXL_CACHE_S2M_NDR
- F_GET_CXL_CACHE_S2M_DRS

These functions are used in the testcase to generate various CXL messages that can be sent to the DUT over the CLS interface. Once the messages are generated Tx BFM performs the following tasks to send the messages:

- SET_PACKETS
- ACTIVATE_LINK
- SEND_TRAFFIC

The Rx BFM performs tasks to set the credit information so that it can receive the messages from the DUT.

The POP_MONITOR_QUEUE task is present in both Tx and Rx BFMs to get the transmitted and received data respectively and check if both the data matches, and hence, it is used for checking the validity of the testcase.

AXI BFM

An AXI transaction is initiated by calling a task within the BFM with all the required AXI parameters.

Individual tasks are implemented for the DUT and REF BFMs and for read and write:

- RP_REG_WRITE
- RP_REG_READ
- EP_REG_WRITE
- EP_REG_READT

The example tests supplied with the testbench make use of these tasks.

The parameters passed to the write tasks are:

- AWID
- AWADDR
- AWLEN
- AWSIZE
- WDATA - for all beats in the burst
- datasize – integer indicating number of bytes WSTRB should be asserted for

The only parameter returned by the write tasks is BUSER.

The parameters passed to the read tasks are:

- ARID
- ARADDR
- ARLEN
- ARSIZE

The parameters returned by the read tasks are:

- RDATA – for all beats in the burst
- RRESP – for all beats in the burst

The task will consume time until the BFM returns the response for the access, and therefore stall the progress of the test sequence. To issue several accesses, the task calls should be located within a `fork..join` statement so that second task is independent on the completion of the first task.

Note: The AXI BFMs are connected directly to the PCIe Subsystems. There is no AXI address decoding implemented in the testbench.

Chapter 5

Register Interface BFMs

Topics:

APB BFM

An APB transaction is initiated by calling a task within the BFM with all the required APB parameters.

The parameters passed for a write are:

- PADDR
- PWDATA – 32-bit

The parameter returned from a write is:

- APB_RESP_OK

The parameter passed for a read is:

- PADDR

The parameters returned from a read are:

- PRDATA – 32-bit
- APB_RESP_OK

The PCIe Subsystem does not return PSLVERR.

Individual tasks are implemented for the DUT and REF BFMs and for read and write:

- REF_APB_WRITE
- REF_ APB _READ
- DUT_ APB _WRITE
- DUT_ APB _WRITE

The example tests supplied with the testbench make use of these tasks.

APB does not have separate request and response transactions and has a common address bus for reads and writes. Therefore, there is no benefit in placing multiple APB task calls for the same APB BFM within a `fork..join` statement.

Chapter 6

SRAMs

Topics:

PCIe SRAMs

AXI

An AXI SRAM is connected as AXI slave to the AXI master port of the PCIe Subsystem. The internal memory is dual-port and therefore can perform simultaneous read and write operations.

CXL SRAMs

CLS

The CXL SRAMs are used for the DUT inbound CXL.mem and CXL.cache transaction layer buffers. or for the retry buffer in case of any link level errors which may require to resend the transactions. The SRAMs used are dual-port RAMs. The SRAM is used in the CXL mode for the following buffers:

- Cache Request Buffer.
- Cache Response Buffer.
- Cache Data-Data Header Buffer.
- Mem Request Response Buffer.
- Mem Data-Data Header Buffer.
- Cache Mem Byte Enable Buffer.
- CXL Retry Buffer.

Chapter 7

Sanity Test Implementation

Topics:

Checker Tasks

The testbench implements the following checker tasks which the example tests make use of:

- check_bresponse_value
  - Accepts single bit BRESP signal.
  - Displays error message and sets fail flag if not zero.
- check_rresponse_value
  - Accepts multi-bit RRESP signal.
  - Displays error message and sets fail flag if any bit is not zero.

Sanity Test

The Sanity test is provided with the Cadence PCIe Verilog Delivery Testbench.

Objective:

To verify the ability of PCIe controller to generate memory transactions.

Test Description:

Sanity test demonstrates:

1. Initialization and Enumeration.
2. PCIe Link up.
3. Speed-up operation.
4. Basic Traffic scenarios.

Sanity Test Implementation for AXI

1. Straps DUT as EP and REF as RP by setting ref_mode_select=1. Trains link to Gen5.
2. Configures RP AXI Regions for Configuration and Memory access.
3. Configures EP AXI Regions for Memory access.
4. Performs config writes and reads from RP to EP.
5. Sends some Basic Memory transactions from RP to EP and EP to RP.

Sanity Test Implementation for CXL

1. Straps DUT as EP and REF as RP.
2. Drives the APN related signals to enable the CXL mode by using the modified TS1 and modified TS2
3. Trains the CLS link to Gen5
4. Activates the Rx CLS link and sets available CLS credits using the tasks in CLS Rx BFMs.
5. Accesses the functions in the CLS Tx BFM to generate the CXL messages and using the tasks in the CLS Tx BFM activates the link and sends the CXL messages.
6. Sends some D2H and S2M transactions from EP to RP.
7. Sends some H2D and M2S transactions from RP to EP.

Self-checks

1. Read and write response check.

2. Read transaction follows the write transaction on the same address and read data is compared.

Chapter 8

Simulation on a Standalone Package

Topics:

Running the Testbench

This section describes how to run the Cadence PCIe Verilog Delivery Testbench using the flexible Makefile. It is recommended to follow this approach for a standalone package delivery. For a full IP package, refer to the *Release Notes*.

Note: It is assumed that the Cadence Xcelium has been installed and set up with all relevant environment variables for these tools configured in your current shell.

As a minimum requirement, the "`which xrun`" command must return a valid path to the required version of the Cadence® Xcelium™ tool installation. Refer to Simulator and LRM Compatibility on page 6 the tool details.

Test Completion Status Pass/Fail

The Cadence PCIe Verilog Delivery Testbench uses the self-checking mechanism to report a test pass or fail. The Testbench reports a pass/fail status as a text banner at the end of the test. The banners are printed as a single display message as shown below.

Message for test pass:

```
----------------------------------------------------------------------------------------------------

CADENCE PCIE VERILOG DELIVERY TESTBENCH: PASS

----------------------------------------------------------------------------------------------------
```

Message for test fail:

```
----------------------------------------------------------------------------------------------------

CADENCE PCIE VERILOG DELIVERY TESTBENCH: FAIL

----------------------------------------------------------------------------------------------------
```

Note: Other error types, such as assertion errors or tool-specific errors are not reported as errors or fatal types. In such cases, a grep of the log file for such errors is advised. Under such conditions the test completion status banner may be incorrect.

Limitations

The limitations of the Verilog Delivery Testbench architecture are:

- **AXI SRAM**
  - An SRAM is used as the target device for inbound memory and IO accesses. This completes all accesses with:
    - Minimal wait states.
    - No reordering.
    - No errors.

    Therefore, the PCIe cores will never perform completer aborts or out of order completions for memory or IO accesses.
  - Parity driving logic is not supported.
- **AXI**
  - Inbound read requests are initiated through an outbound AXI read at the link partner. AXI has no byte strobes for read accesses and only supplies the address of the first byte in the transfer. Therefore, in a multi-beat

transfer the second and subsequent beats read the full data stripe. Inbound read requests which occupy more than 1 AXI stripe in initiating BFM will therefore always have all byte selects asserted for the final DWord.

- The PCIe core will split an inbound read request into multiple AXI read requests and will return the data as split completions when the read request exceeds the maximum AXI burst size. However, the core limits outbound read requests to those which meet the PCIe maximum read request size and maps each AXI read request to one PCIe read request. Therefore, all PCIe read requests are compatible with a single AXI initiator read request. Split completions can only occur if the initiator AXI width is larger than the completer AXI width.
- The AXI transaction output from the PCIe controller to the testbench will therefore receive low wait state response and will never receive a SLVERR or DECERR HRESP/BRESP status.

Chapter 9

Change Log

Topics:

Differences between the Released Versions

The following table shows the difference between the released versions of the Cadence PCIe Verilog Delivery Testbench User Guide.

Version	Release Date	Comments
0.7	31 August 2020	Initial Release