cadence

Cadence CDN_AXI VIP User Guide

Product Version 11.3

May 2013

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

Table of Contents

List of Figures

List of Tables

List of Examples

Preface

The Cadence CDN_AXI VIP User Guide describes the CDN_AXI VIP based on the PureSpec™ API.

The CDN_AXI VIP provides the solution for verifying compliance and compatibility of the protocol. It includes highly configurable and flexible simulation models of all the protocol layers, devices, and transaction types. It also supports integration and traffic generation in all popular verification environments. The CDN_AXI VIP is built on top of the MMAV™ architecture to ensure high quality, high performance, and seamless EDA integration.

Using CDN_AXI VIP, you can generate tests quickly and efficiently to ensure that your design under test (DUT) is compatible with the other components that may be used in the end system. The extensive protocol checks built into the CDN_AXI VIP help you verify your design by generating warnings and errors when the protocol violations occur.

Note

The CDN_AXI VIP is supported only when installed from a VIPCAT release.

1. About This Manual

This manual describes the CDN_AXI VIP interfaces and configuration.

2. Terminology

Included below is a set of definitions of terms used throughout this document.

$CDN_VIP_ROOT	An environment variable that contains the path of the VIP installation.
Design Under Test (DUT)	This is the device being verified. Sometimes the term design under verification (DUV) is used.
Active Mode	In this mode, the CDN_AXI VIP interface model acts as an actual device in your verification environment, both receiving and generating transactions.
Passive Mode	Connected directly to the DUT, the CDN_AXI VIP model in this mode does not generate transactions, but verifies both incoming and outgoing transactions from the DUT.
SOMA	Specification of Model Architecture. SOMA is a Cadence-standard format to parameterize the model for user-specific verification needs.
Bus Functional Model (BFM)	Verification software that emulates a given device or protocol.

Data-Driven Verification API (DDV API)	The Data-Driven Verification API (DDV API) is an extension to the simulation environment offered using the Memory Model Portfolio and PureSpec Verification IP software. The DDV API can be used to integrate applications within the simulation process.
MMAV	Memory Modeler Advanced Verification is the industry-standard solution for memory simulation and system verification. MMAV dramatically enhances verification by enabling observations and operations on system-level data transactions during simulation. This "data-driven verification" approach is key to optimizing regressions and accelerating your overall verification process.

3. Customer Support

If you have a problem using this product or the documentation, you can submit a customer Support Request (SR) to Cadence Support. When you file a customer support request, you should provide as much detailed information as possible with regards to the problem you have encountered along with a tracefile of your simulation.

To obtain a tracefile of your simulation, you must run your simulation with the following configuration options set in your `.denailrc` file:

- `Historyfile denali.his`
- `Historydebug on`
- `Tracefile denali.trc`

Note

You can use different naming convention for the files, but keep the same file extension types.

3.1. Service Requests

SRs are your way of giving feedback, asking questions, getting solutions, and reporting problems. Unless told otherwise, Cadence support staff will respond to your service request. If Cadence Support cannot answer your question, Cadence research and development personnel will get involved. It is important to specify the severity level of the service request as accurately as possible. There are three levels of severity:

- Critical — You cannot proceed without a solution to the issue.
- Important — You can proceed, but you need a solution to the issue.
- Minor — You prefer to have a solution, but you can wait for it.

Note

You can request support to increase the severity level of an issue. Therefore, do not use Critical unless immediate resolution of an issue is absolutely necessary and urgently required.

3.2. Using Cadence Online Support

Cadence encourages you to submit requests using Cadence Online Support. With Cadence Online Support you can also track your open service requests.

To use Cadence Online Support to submit a service request:

1. If you do not yet have a Cadence Online Support account, go to http://support.cadence.com and click *Register Now* under the *New User* heading.

   You must provide a valid HostID for any VIP product. The `HostID` is contained in the `SERVER` line of your VPA product license file.

   ### Note

   If you already have a Cadence Online Support account, then you only need to update your Cadence Online Support preferences to include a valid `HostID` for a VIP product.

2. Log in to Cadence Online Support, and on the upper left side of the page click *Create Service Request* under the *My Service Requests* heading.

   A form is presented for submission of your service request. Select *Verification IP* in the *Product list* box and click *Continue*. Follow the online instructions to complete the Service Request.

3.2.1. Creating Group Privileges in Cadence Online Support

Sometimes it is beneficial to view the service requests of others on your project.

To create group privileges in Cadence Online Support:

1. Open a Cadence Online Support service request by clicking *Create Service Request* under the *My Service Requests* heading.

2. Select *Verification IP* in the *Product list* box and click *Continue*.

3. Fill in the required fields in the form presented.

   Explain in the *Stated Problem* text box that you want to create a group of users.

4. In the *People to notify* upon SR creation field, include the email addresses of the users you want to have group privileges.

Note

Each person receiving group privileges must have a Cadence Online Support account.

5. Click Submit SR to complete the Service Request.

Visit http://www.cadence.com/support/Pages/default.aspx to learn more about Cadence Global Customer Support and the support offerings we provide. For more details about our support process, visit http://www.cadence.com/support/Pages/support_process.aspx

4. Related Documentation

Besides the information in this document about the CDN_AXI VIP, the following related information is available:

Table 1. Release-Related Documentation

To Find Out About ...	Look In ...
Release compatibility, installation, and configuration	The “Release Information” chapter of the *VIP Catalog Release Information* document, which is available from downloads.cadence.com, in your `$CDN_VIP_ROOT/doc` directory, or in Cadence Help.
What's new	The “What’s New in Verification IP” chapter of the *VIP Catalog Release Information* document, which is available from downloads.cadence.com, in your `$CDN_VIP_ROOT/doc` directory, or in Cadence Help.
Known limitations, problems, and solutions or fixes for them	The “Limitations and Workarounds” and “Known Problems and Solutions” chapters of the *VIP Catalog Release Information* document, which is available from downloads.cadence.com, in your `$CDN_VIP_ROOT/doc` directory, or in Cadence Help.
Fixed CCRs	The “Fixed CCRs” chapter of the *VIP Catalog Release Information* document, which is available from downloads.cadence.com, in your `$CDN_VIP_ROOT/doc` directory, or in Cadence Help.

To start Cadence Help, use the following command:

```
$CDN_VIP_ROOT/tools/bin/cdnshelp &
```

API Reference - Each release also includes API reference documents that are automatically generated with information about structs, fields, methods, and events. These documents can be very helpful

for debugging. One API reference is created each for UVM and OVM methodologies. To read these references, open the `index.html` files in the following locations with your Web browser:

- `$CDN_VIP_ROOT/doc/cdn_axi/axi_uvm_sv_ref/index.html`
- `$CDN_VIP_ROOT/doc/cdn_axi/axi_ovm_sv_ref/index.html`

Chapter 1. General VIP Concepts

The VIP is a full-timing, bus-functional model that supports many protocols. Using Cadence VIP you can quickly create a verification hierarchy that fits your design requirements.

You can perform comprehensive protocol checking by enabling or disabling the individual rules. The VIP issues callbacks to your testbench with error information when a protocol violation is detected. Model callbacks are also available at multiple points along the transaction flow through the model to aid scoreboarding or reactive testing.

Each protocol's functionality is configurable. The specification of model architecture (SOMA) configuration file describes the VIP modeling rules and properties of each device. You can use PureView, a graphical tool, to generate both the SOMA file and the instantiation interface for each device in your test configuration. The VIP architecture supports a wide range of commercial simulators and testbench platforms. You can use almost any commercial simulator, write your test cases in almost any language, and then use the VIP APIs to hook up your test cases. Callbacks allow you to take backdoor peeks into your test data as it flows through your test configuration, and to modify transaction at selected points during this flow. For example, you can introduce errors or discard the transaction. Finally, the model’s output includes various transaction history logs that show the flow of data transactions through your configuration.

1.1. VIP Verification Flow

The following diagram illustrates how to verify your design with the VIP:

Figure 1.1. The VIP Verification Flow

The VIP is typically utilized and linked in an event-driven or cycle-accurate logical simulation environment.

To perform VIP verification:

1. **Create a SOMA file using PureView**. A protocol model is characterized with a SOMA file. This SOMA file captures a complete device behavior, including device type, device capabilities, pin interface, and so on. Every component in the system/testbench hierarchy can be described by SOMA. You can create a SOMA file using PureView GUI tool. For details on how to use this tool, refer to Chapter 4, *PureView Graphical Tool*.

2. **Generate the instantiation interface using PureView**. Select the proper simulation HDL language format.

   Note

   Currently, only SystemVerilog is supported.

   Save the generated instantiation interface, which corresponds to the configuration and has the matching pin interface.

3. **Instantiate the generated interface in your testbench**. This entity becomes an instance in the simulation. Depending on the configuration, this instance can receive and generate traffic, or act like a monitor (only receiving traffic) on the DUT.

   You can find more details on how to instantiate the CDN_AXI VIP model in the chapter Chapter 6, *CDN_AXI VIP Model Operation*.

4. **Link the VIP (library) with the simulator**. The VIP is supplied in the form of a binary object file. By linking this library, the VIP instance communicates with simulator program through the API function calls. Note that these are in the same UNIX process.

5. **Simulate**. At the beginning of simulation time 0, the CDN_AXI VIP model instance locates and loads the SOMA file for this configuration. As simulation time proceeds and signals change values, the model updates its state machine and takes an appropriate action based on the current state. The appropriate actions include transmitting transactions/symbols, reporting error conditions, and providing the testbench with event callbacks.

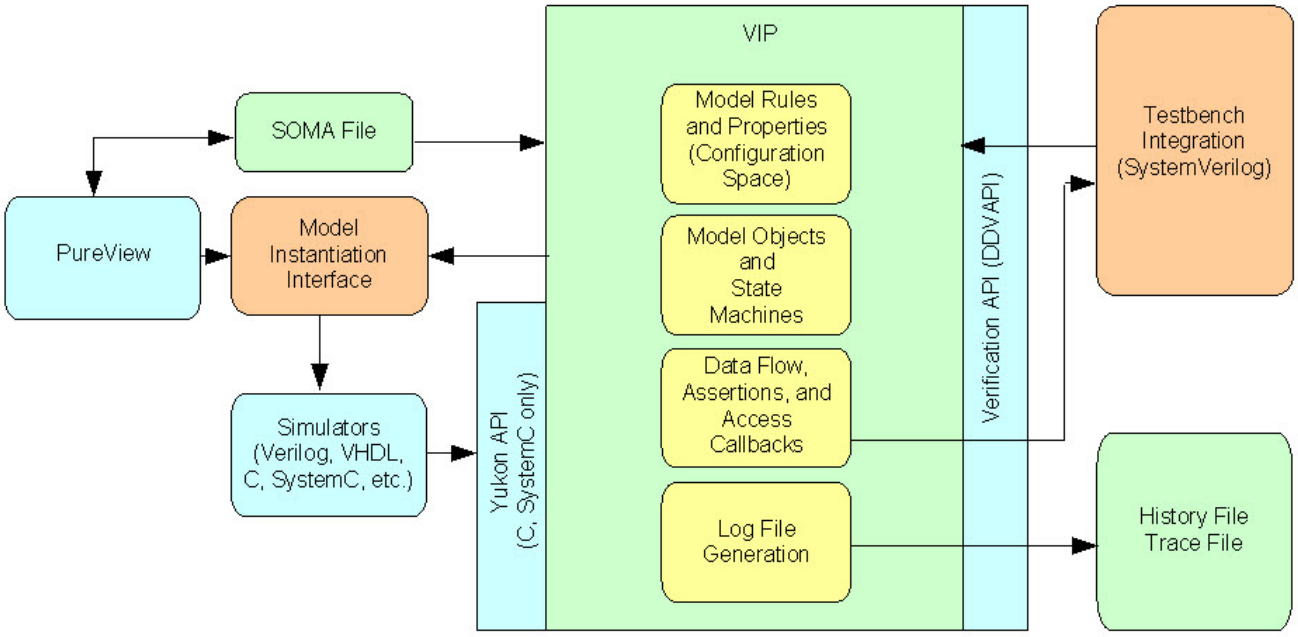
   Refer to Chapter 8, *CDN_AXI VIP Simulation* for more details.

1.2. Architecture

The VIP is based on a hybrid C and **e** implementation, an architectural approach that combines the most comprehensive protocol checking with the highest performance. Additionally, the CDN_AXI VIP model is highly configurable based on the SOMA file and run-time configuration controls.

The following figure illustrates the general architecture of the VIP.

Figure 1.2. The VIP Architecture



Note

Currently, the Yukon Interface (for C and SystemC) is not supported for the CDN_AXI VIP.

The key blocks in the VIP architecture are described in the following sections.

1.2.1. PLI, VHPI, and so on

For Verilog simulators, a thin PLI version 1.0 is applied to establish the integration with the VIP. For VHDL simulators, either FLI, FMI, or VHPI is used for the integration.

1.2.2. SOMA File

Each device in your model has its own specification of modeling architecture (SOMA) parameter file describing its attributes. The SOMA file parameters are read at 0 simulation time.

1.2.3. Configuration Options via SOMA

The VIP model is highly configurable via the SOMA parameter file. In addition to supporting all the configurable options in the CDN_AXI specifications, implementation-specific options and testability options are also covered.

1.2.4. Protocol Rules and Checks

One of the most important capabilities of the VIP is the extensive and complete checking of protocol rules. All aspects of the given protocol's specification are covered by the VIP checks. Hundreds of

run-time checks are built into the VIP and are tested during the simulation to catch any potential protocol violations.

1.2.5. Configuration Space

The VIP models the complete configuration space as:

- **Protocol Configuration Space** - The VIP defines all the protocol-specific configuration register spaces, which can be accessed via the testbench in addition to the protocol-specific access methods.
- **VIP Configuration Space** - Apart from the protocol-specific configuration register space, the VIP also defines a configuration space to allow for more granular testbench control of the VIP model.

For details, refer to Chapter 6, *CDN_AXI VIP Model Operation*.

1.2.6. Model Objects and State Machines

The VIP models closely reflect the state machines and structures described in the protocol's specifications. Within each model, various queues and state machines are implemented, and can be manipulated or accessed through the testbench routines.

1.2.7. Callbacks

Within the VIP model, there are a number of key data structures, memories, and transaction activities. The accesses or events trigger a callback to the testbench code with the relevant information. You can choose your callback point from the list of all possible callback points.

1.2.8. Verification Simulation Interface

The VIP verification simulation interface is a key testbench interface for monitoring the CDN_AXI VIP model state, controlling model behavior, and generating and modifying traffic. This is the DDVAPI that provides the backdoor interface to all testbench environments. This enables the CDN_AXI VIP to be used in any sophisticated testbench environment or methodology and includes the ability to develop truly directed, constrained-random, or fully random self-checking test environments. This universal capability is available to any C, Verilog, VHDL, TCL, VERA, **e**, SystemC, or SystemVerilog environment.

1.3. Supported Operating Modes

The VIP model has the following main mode(s) of operation:

- Active Mode
- Passive Mode

In active mode, the model behaves like a component in the system. It can be used to either initiate or respond to the protocol traffic and to verify that the traffic from the design under test (DUT) conforms to the specification.

In passive mode, the model acts like a shadow reference model of the DUT and checks the DUT. However, it does not generate any traffic.

You must specify the operating mode separately for each model, using the PureView interface to generate the SOMA and HDL files. Every model in your configuration except for the monitor should be set to Active mode. The monitor, which is an exact copy of your DUT, is the only model that should be set to function in passive mode.

1.3.1. Active Mode

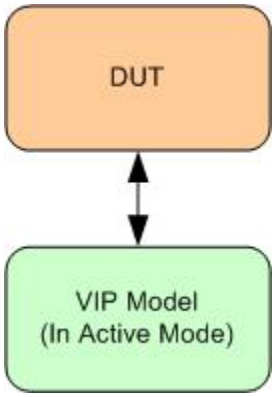
In active mode, the VIP model acts as a bus functional model (BFM) with protocol checking. The model characteristics are described in the SOMA specification.

The VIP verification functions can be used to generate transactions from the BFM to the DUT. The verification functions also provide backdoor access from the testbench for initializing and checking the active mode model register and memory spaces.

To use the VIP model in active mode, you need to have a SOMA file that describes the model and have the active mode parameter set. Connect the DUT pins to the appropriate VIP model connections.

For details on how to specify the model to work in the active mode using PureView, refer to Chapter 4, *PureView Graphical Tool*.

Figure 1.3. The VIP Model in Active Mode

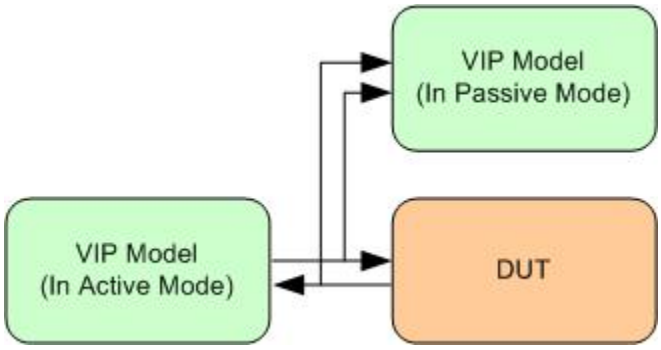


1.3.2. Passive Mode

The model in passive mode is a protocol checker enforced on the design under test (DUT). The monitor functions in a passive mode because it does not drive any data on the bus.

To specify that the model functions in passive mode, you need a SOMA file configured to run in passive mode. Connect the DUT pins to the appropriate VIP model connections. The SOMA configuration should exactly match the DUT so that it can monitor the DUT precisely. The VIP model in passive mode (monitor) records the inputs to the DUT, and verifies correctness by tracking the DUT's state, based upon what has been received, and checks that the DUT transmits appropriately for a given state.

Figure 1.4. The VIP Model in Passive Mode



Chapter 2. CDN_AXI VIP Product Overview

2.1. Features

CDN_AXI VIP enables you to ensure compliance to the CDN_AXI specifications. You can test all possible configurations of CDN_AXI devices, as well as monitor them using CDN_AXI VIP. Furthermore, easy integration a variety of design and verification tool flows gives you a test methodology that fits into with your development and testing cycles.

Key features of CDN_AXI VIP include:

- CDN_AXI specification modeling on two levels: transfer and transaction/burst
- Two main device types:
  - Master -- issues read and write transactions.
  - Slave -- responds to read/write transactions and can also act as a dummy interconnect.
- Model usage in either active mode or passive mode.

  Passive mode provides the DUT shadow reference model that compares DUT responses to expected responses.
- Independent state machines for pin-level valid/ready signal protocol for each channels like:
  - read address
  - write address
  - read data
  - write data
  - write response
- Transfer-level state machines that monitor the latency and consistency of read and write transaction sequences of transfers. For example, `WriteAddress`, multiple `WriteTransfers`, and `WriteResponse`.
- Automatic segmentation of a high-level read and write into a sequence of transfers. For example, `WriteAddress`, multiple `WriteTransfers`, and `WriteResponse.`
- Automatic retrieval of transfers from pin changes and high-level transactions from transfers.
- Multiple interleaving transactions.

- Pipelined operation.
- Out-of-order transaction support.
- Automatic reordering transactions for memory coherency for maximizing performance
- Customizable address and data width.
- Automatic write strobe generation for non-aligned transactions.
- Exclusive access support.
- Low-power interface support.
- Customizable error reporting.

When the CDN_AXI VIP model functions in active mode (Bus Functional Mode), it interacts with complementary devices, and issues and responds to the command. When the CDN_AXI VIP model functions in passive mode (Monitor), it attaches itself to the DUT's interfaces, monitors the incoming and outgoing traffic, and flags any deviations from the specification of the DUT. All checks are configurable and can be disabled. CDN_AXI VIP provides powerful callback capabilities for user-defined testbenches to enable full control over all transactions within the CDN_AXI model.

2.2. Model Types

There are several CDN_AXI VIP models. You can use each of these device types either in an active mode (BFM) or passive mode (Monitor).

2.2.1. Active Model of a Device with a Single Master Port

You can use the active model of a device with a single master port to:

- Drive high-level read and write transactions to a slave device under test (Device Under Test)

2.2.2. Active Model of a Device with a Single Slave Port

You can use the active model of a device with a single slave port to:

- Convert signals from a DUT master into high-level transactions
- Generate automatic responses
- Transmit these responses back to the master DUT

2.2.3. Passive Model of a Device with a Single Slave Port

You can instantiate the CDN_AXI VIP passive model as a shadow model of the DUT. To do this, prepare the SOMA file and fill the model's memory so that the model behaves like a DUT.

All pins of the CDN_AXI VIP passive model are input pins. The slave monitor receives transactions from the master driving DUT, generates the expected response, and compares this response with the actual response from the DUT received by the monitor through the wires.

The monitor generates all error messages of the CDN_AXI VIP active model (transaction checking and timeouts). In addition, the monitor generates errors about the transaction mismatch with the expected transaction, indicating that some expected transaction is missing, and that the DUT slave response is unexpected.

Chapter 3. Getting Started

3.1. Integration Checklist

This section provides step-by-step instructions on setting up your environment to simulate the VIP followed by setting up (and simulating) an example provided in the release. Skip the steps that are not relevant for you.

1. Download the latest VIPCAT version.

   Refer to the *README_VIPCAT<##>_Release_Info.pdf* document on downloads.cadence.com for details on how to install the release, the VIP availability matrixes, platform support, release dependencies, and licensing information.

2. Set up your environment to simulate the VIP by executing the scripts provided with the VIPCAT release.

   a. Set `CDN_VIP_ROOT` to point to the root directory of the VIPCAT installation.

   b. Execute `$CDN_VIP_ROOT/bin/cdn_vip_setup_env` with the appropriate arguments. This generates a shell script with the necessary commands to set up your environment. The generated shell script is named `cdn_vip_env_<...>.[c]sh`.

      Note

      - `cdn_vip_setup_env -h` provides information on the arguments to this script.
      - The simulator executable (ncsim, vcs, or vlog) must be available in your path (or the simulator installation directory must be explicitly specified with the `-sim_root` option to `cdn_vip_setup_env`).
      - For IUS users only: You must specify the `-install` and `-cdn_vip_lib` options for `cdn_vip_setup_env`. The argument for the `-cdn_vip_lib` option must be a user-writeable directory. `cdn_vip_setup_env` compiles the libraries necessary to run the VIP with IUS into this directory.
      - If your environment is based on C shell (csh, tcsh, etc.), the `-csh` option must be specified for `cdn_vip_setup_env`. By default, commands to set up the environment are generated for Bourne shell (sh).

   c. Execute the generated script.

      C-shell users: `source cdn_vip_env_<...>.csh`
      Bourne-shell users: `. cdn_vip_env_<...>.sh`

3. Set up and run the example provided in the release.

4. For details on setting up and running the example for each simulator, refer to the following:

   - UVM

     - NCSIM (Section 10.3.2.1, “*NCSIM*”)

     - VCS (Section 10.3.2.2, “*VCS*”)

     - MTI (Section 10.3.2.3, “*MTI*”)

   - SV non UVM

     - NCSIM (Section 10.1.2, “*NCSIM*”)

     - VCS (Section 10.1.3, “*VCS*”)

     - MTI (Section 10.1.4, “*MTI*”)

5. Create a SOMA and an HDL instantiation interface for each component type. A SOMA file represents the component configuration.

   Refer to Section 3.2, “*Creating a SOMA File and an HDL Instantiation Interface for CDN_AXI VIP*”. In addition, you can refer to Chapter 4, *PureView Graphical Tool* to learn about configuring SOMA parameters using the GUI. And also Section 5.2, “*SOMA*” for SOMA parameter details.

6. Create a testbench for your environment.

   a. Create the testbench file. Refer to Section 3.3, “*Creating a Testbench*”.

   b. Connect the components together. Refer to Section 3.4, “*Connecting the Components*”.

   c. Configure the components. Refer to Section 3.5, “*Configuring the Components*”.

7. Create setup, compilation, and run scripts for your environment. Refer to Section 10.1.1, “*VIP Scripts*”. If your simulator is NCSIM, you can use the **irun** invocation tool to handle the details of incorporating VIP models. For details, refer to Section 10.1.2.3, “*Using irun*”

8. Adjust the memory segments according to the design (check that the segments definition of agents in the same interface matches). Refer to Section 6.1.2.3, “*Defining Memory Segments*”.

9. Write your first test. Refer to Section 3.6, “*Writing a Simple Test Scenario*”.

   Once you have written your first test, you can start generating transactions and test the verification flow. Refer to Section 6.3, “*Transactions*”, and Section 9.1, “*Test Flow*”, respectively. For details on how to debug, refer to Section 8.5, “*Debugging the Simulation*”.

10. Write more advanced tests. Refer to Chapter 9, *CDN_AXI Verification Test Scenarios*.

a. Add specific constraints to the transaction. Make sure you limit the `IdTag` and `address` according to the corresponding signals size. See the `class myTransaction` definition in the example of section Section 3.6, "*Writing a Simple Test Scenario*".

   You can find more information about the VIP SV constraint in Section 9.4, "*Built-in SV Constraints*".

b. For better control over the model, use the VIP callbacks. For details on how to use these callbacks, refer to Chapter 7, *CDN_AXI VIP Callbacks*.

11. If you encounter any issues, first check Chapter 11, *Frequently Asked Questions* and Chapter 12, *Troubleshooting*.

3.2. Creating a SOMA File and an HDL Instantiation Interface for CDN_AXI VIP

A SOMA and an HDL instantiation interface must be created for each component type in the simulation.

Note

A SOMA file and an HDL instantiation interface are needed for each type and not for each instance.

The SOMA file is an XML file that can be edited using the PureView GUI tool. This file defines a specific component type configuration; you should configure it to match a DUT port. The HDL instantiation interface is a **.v* file created using the PureView GUI tool.

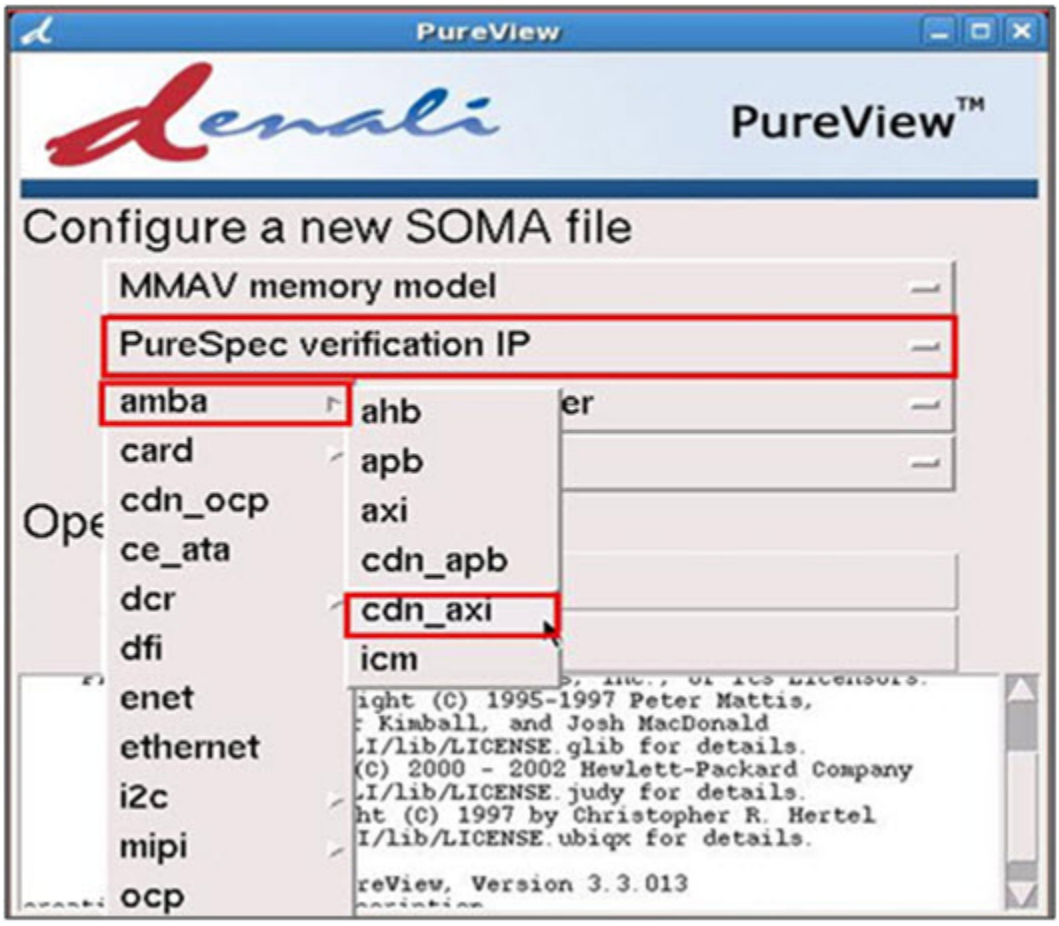
Every HDL instantiation interface has a corresponding SOMA file associated with it. The instantiation interface contains a module with the interface between the testbench and the VIP.

In this example, we show how to create a SOMA file and HDL instantiation interface for two component types:

- Active master
- Active slave

1. Open PureView.
2. Choose **PureSpec Verification IP** >> **amba** >> **cdn_axi** as shown in the figure below:

Figure 3.1. PureView Opening Window

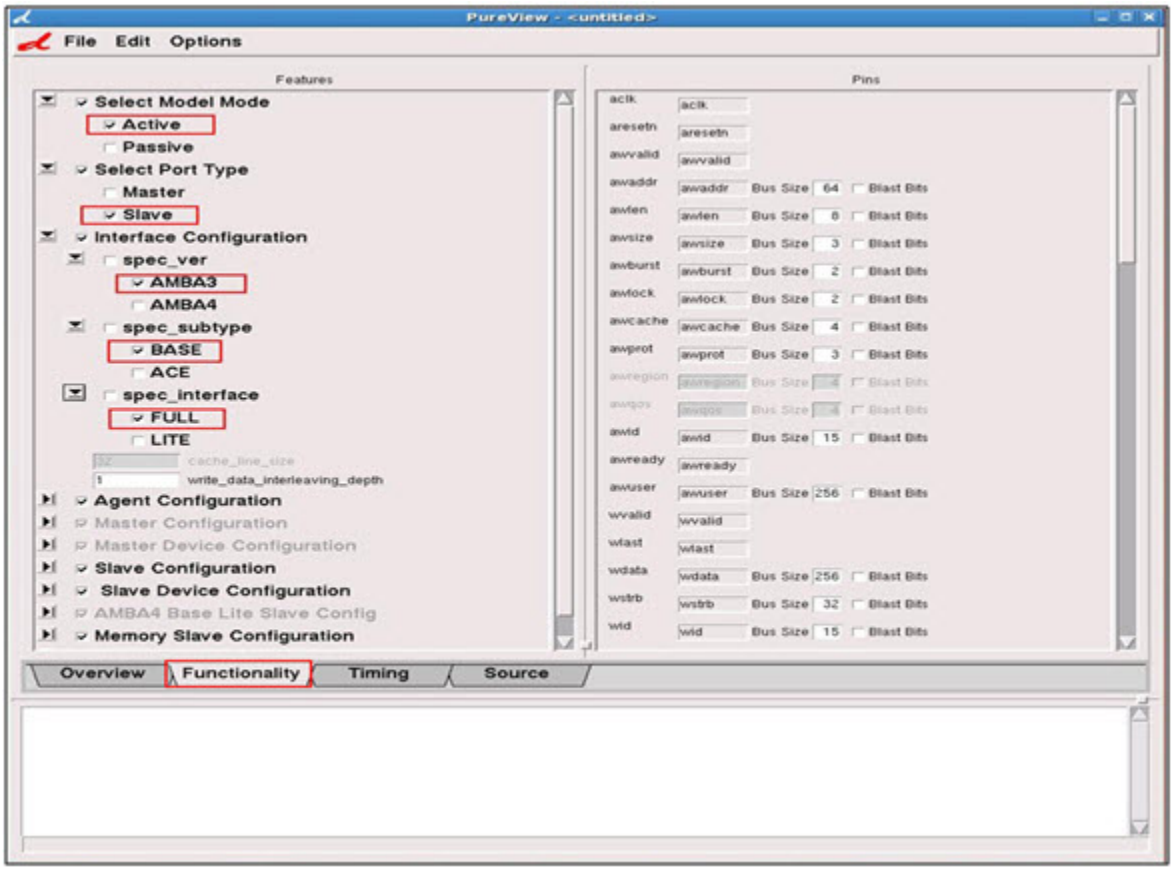


3. The PureView window opens.

4. Choose the **Functionality** tab at the bottom of the window to view the default configuration set for AXI Slave.

5. Modify the default configuration parameters to AXI3 as shown below in the **Features** pane.

Figure 3.2. PureView - Functionality Tab



6. **Select Model Mode** >> **Active.**

7. **Select Port Type** >> **Slave.**

8. **Interface Configuration** >> **spec_ver** >> **AMBA3.**

9. **Interface Configuration** >> **spec_subtype** >> **BASE.**

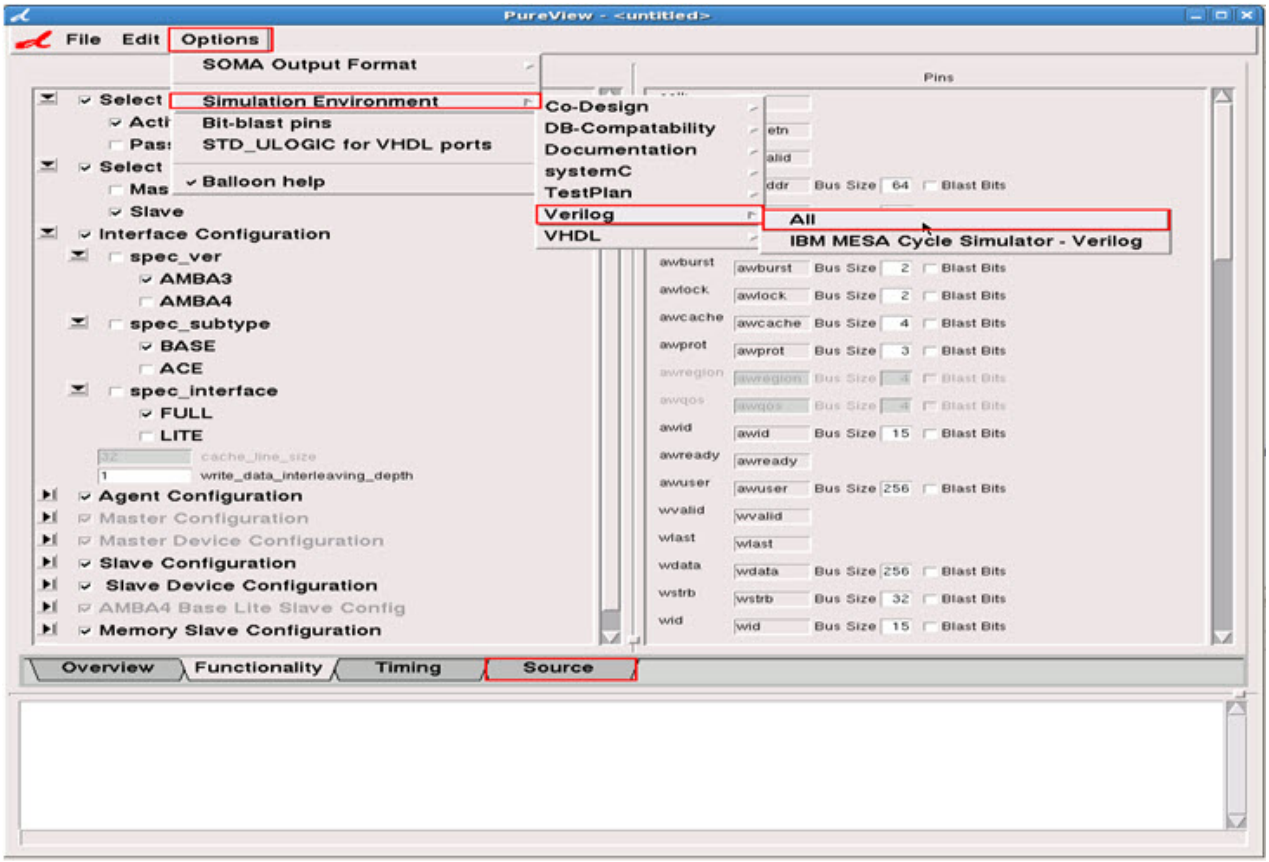
10. **Interface Configuration->spec_interface-> FULL.**

Once all the above parameters are set, you can control other configuration information, including `signals names` and `width` in the **Pins** pane.

1. Navigate using **File** >> **Save as** to save the agent configuration file as an `activeSlave.soma` file.

2. Navigate using **Options** >> **Simulation Environment** >> **Verilog** >> **All** to save the interface as shown in the figure below.

Figure 3.3. PureView - Source Tab



Note

Select the **Source** tab to review the file that is created when you select **Save Source as**. You can edit the default module name, which is the same as the SOMA file. At the bottom of the file, you can also view the module created with a path to the associated SOMA file. By default, this is an absolute path. You can edit it to be a relative path using environment variables.

3. Repeat the previous steps for each component type needed in your environment. In this case, create *ActiveMaster.soma* (by choosing **Select Port Type** >> **Master**) and *ActiveMaster.v* files.

Note

Do not reopen *PureView*, because the default name is that of the last saved file. Do not overwrite the previous files, and remember to type the new file name each time.

At this stage, you should have a **.soma file* and a **.v* file for each component type in your system, as shown in the example:

- `ActiveSlave.soma` and `ActiveSlave.v`
- `ActiveMaster.soma` and `ActiveMaster.v`

3.3. Creating a Testbench

1. Create a testbench file `tb.sv` and declare the top module in it.

```
module axi_top;
endmodule
```

3.4. Connecting the Components

1. Add all the interface signals to be connected in the testbench to the top module you declared.
   - Clock and reset should be **reg**
   - All of the other signals should be **wire**
   - **Tip:** Open one of the HDL instantiation interface files (for example, activeSlave.v). Copy the declaration of the interface signals and change them to wire, using find and replace.

     Expected *tb.sv*

```
reg aclk;
    reg aresetn;
    wire awvalid;
    wire [31:0] awaddr;
    wire [7:0] awlen;
    wire [2:0] awsize;
    wire [1:0] awburst;
    wire [1:0] awlock;
    wire [3:0] awcache;
    wire [2:0] awprot;
    wire [3:0] awid;
    wire awready;


    wire [31:0] awuser;
    wire wvalid;
    wire wlast;
    wire [31:0] wdata;
    wire [3:0] wstrb;
    wire [3:0] wid;
    wire wready;


    wire [31:0] wuser;
    wire bvalid;


    wire [1:0] bresp;


    wire [3:0] bid;


    wire bready;
```

```
    wire [31:0] buser;


    wire arvalid;
    wire [31:0] araddr;
    wire [7:0] arlen;
    wire [2:0] arsize;
    wire [1:0] arburst;
    wire [1:0] arlock;
    wire [3:0] arcache;
    wire [2:0] arprot;
    wire [3:0] arid;
    wire arready;

    wire [31:0] aruser;
    wire rvalid;

    wire rlast;

    wire [31:0] rdata;

    wire [1:0] rresp;

    wire [3:0] rid;

    wire rready;

    wire [31:0] ruser;
```

Existing *activeSlave.v*

```
input aclk;
    input aresetn;
    input awvalid;
    input [31:0] awaddr;
    input [7:0] awlen;
    input [2:0] awsize;
    input [1:0] awburst;
    input [1:0] awlock;
    input [3:0] awcache;
    input [2:0] awprot;
    input [3:0] awid;
    output awready;
      reg den_awready;
      assign awready = den_awready;
    input [31:0] awuser;
    input wvalid;
    input wlast;
    input [31:0] wdata;
    input [3:0] wstrb;
    input [3:0] wid;
    output wready;
      reg den_wready;
      assign wready = den_wready;
    input [31:0] wuser;
    output bvalid;
      reg den_bvalid;
      assign bvalid = den_bvalid;
    output [1:0] bresp;
      reg [1:0] den_bresp;
      assign bresp = den_bresp;
    output [3:0] bid;
      reg [3:0] den_bid;
      assign bid = den_bid;
    input bready;
    output [31:0] buser;
```

```
      reg [31:0] den_buser;
      assign buser = den_buser;
    input arvalid;
    input [31:0] araddr;
    input [7:0] arlen;
    input [2:0] arsize;
    input [1:0] arburst;
    input [1:0] arlock;
    input [3:0] arcache;
    input [2:0] arprot;
    input [3:0] arid;
    output aready;
      reg den_aready;
      assign aready = den_aready;
    input [31:0] aruser;
    output rvalid;
      reg den_rvalid;
      assign rvalid = den_rvalid;
    output rlast;
      reg den_rlast;
      assign rlast = den_rlast;
    output [31:0] rdata;
      reg [31:0] den_rdata;
      assign rdata = den_rdata;
    output [1:0] rresp;
      reg [1:0] den_rresp;
      assign rresp = den_rresp;
    output [3:0] rid;
      reg [3:0] den_rid;
      assign rid = den_rid;
    input rready;
    output [31:0] ruser;
      reg [31:0] den_ruser;
      assign ruser = den_ruser
```

2. Add clock block and reset control. For example:

```
Initial
begin
     aclk = 1'b1;
     aresetn = 1'b1;
     #100
     aresetn = 1'b0;
     #500
     aresetn = 1'b1;
end
always #50 aclk = ~aclk;
```

3. Instantiate the code in the top module scope. The module name was determined when you saved the file. You can set a different name in the **Source** tab in *PureView*.

   Tip: Add the declaration for the instantiation interface, for example, `activeMaster activeMasterDevice()`. Also copy the signals for connection.

   Expected *tb.sv*

```
activeMaster activeMasteDevice(
    aclk,
    aresetn,
    awvalid,
    awaddr,
    awlen,
    awsize,
```

```
    awburst,
    awlock,
    awcache,
    awprot,
    awid,
    awready,
    awuser,
    wvalid,
    wlast,
    wdata,
    wstrb,
    wid,
    wready,
    wuser,
    bvalid,
    bresp,
    bid,
    bready,
    buser,
    arvalid,
    araddr,
    arlen,
    arsize,
    arburst,
    arlock,
    arcache,
    arprot,
    arid,
    arready,
    aruser,
    rvalid,
    rlast,
    rdata,
    rresp,
    rid,
    rready,
    ruser);
```

Existing *activeMaster.v*

```
module activeMaster(
    aclk,
    aresetn,
    awvalid,
    awaddr,
    awlen,
    awsize,
    awburst,
    awlock,
    awcache,
    awprot,
    awid,
    awready,
    awuser,
    wvalid,
    wlast,
    wdata,
    wstrb,
    wid,
    wready,
    wuser,
    bvalid,
    bresp,
    bid,
    bready,
    buser,
```

```
   arvalid,
   araddr,
   arlen,
   arsize,
   arburst,
   arlock,
   arcache,
   arprot,
   arid,
   arready,
   aruser,
   rvalid,
   rlast,
   rdata,
   rresp,
   rid,
   rready,
   ruser);
```

4. Repeat the process for all components.
5. *tb.sv* should now contain:
   - Testbench signals
   - Clock block and reset control
   - Instantiation code for each component bound to testbench signals. This is what we have so far:

```
module axi_top;

  reg aclk;
  reg aresetn;
  wire awvalid;
  wire [31:0] awaddr;
  wire [7:0] awlen;
  wire [2:0] awsize;
  wire [1:0] awburst;
  wire [1:0] awlock;
  wire [3:0] awcache;
  wire [2:0] awprot;
   wire [3:0] awid;
   wire awready;

   wire [31:0] awuser;
   wire wvalid;
   wire wlast;
   wire [31:0] wdata;
   wire [3:0] wstrb;
   wire [3:0] wid;
   wire wready;

   wire [31:0] wuser;
   wire bvalid;

   wire [1:0] bresp;

   wire [3:0] bid;

   wire bready;
   wire [31:0] buser;

   wire arvalid;
   wire [31:0] araddr;
```

```
    wire [7:0] arlen;
    wire [2:0] arsize;
    wire [1:0] arburst;
    wire [1:0] arlock;
    wire [3:0] arcache;
    wire [2:0] arprot;
    wire [3:0] arid;
    wire arready;

    wire [31:0] aruser;
    wire rvalid;

    wire rlast;

    wire [31:0] rdata;

    wire [1:0] rresp;

    wire [3:0] rid;

    wire rready;
    wire [31:0] ruser;


activeMaster activeMasterDevice(aclk, aresetn, awvalid, awaddr, awlen, awsize, awburst, awlock,
 awcache, awprot, awid, awready, awuser,  wvalid, wlast, wdata, wstrb, wid, wready, wuser,
 bvalid, bresp, bid, bready, buser, arvalid, araddr, arlen, arsize, arburst, arlock, arcache,
 arprot, arid, arready, aruser, rvalid, rlast, rdata,  rresp, rid, rready, ruser);

activeSlave activeSlaveDevice(aclk, aresetn, awvalid, awaddr, awlen, awsize, awburst, awlock,
 awcache, awprot, awid, awready, awuser,  wvalid, wlast, wdata, wstrb, wid, wready, wuser,
 bvalid, bresp, bid, bready, buser, arvalid, araddr, arlen, arsize, arburst, arlock, arcache,
 arprot, arid, arready, aruser, rvalid, rlast, rdata,  rresp, rid, rready, ruser);

initial
 begin
     aclk = 1'b1;
     aresetn = 1'b1;
     #100
     aresetn = 1'b0;
     #500
     aresetn = 1'b1;
end
 always #50 aclk = ~aclk;
endmodule
```

3.5. Configuring the Components

1. For each component, do the following:

   - Import the CDN_AXI VIP package `denaliSvCdn_axi`, which includes the CDN_AXI VIP instance `denaliCdn_axiInstance`.
   - Import the `denaliSvMem` package, which includes utilities for reads/writes, callbacks, memory transactions, SOMA parameter value access, TCL command evaluation, and so on.

     Recommendation: Testbench configuration of an instance is largely based on reading and writing to models such as the memory model, control registers model, and so on. Each of these models has an instance ID associated with it; therefore, you should write code to instantiate

these models in your testbench and implement methods to read and write to them. See Section 6.1.1, "*Instantiating the Models in the Testbench*".

The syntactic patterns for the most-used methods are readily available at `$DENALI/example/cdn_axi/denaliAxiUserInstance.sv` for AXI. If you choose to include this file in your testbench and use the available methods, make sure that the Instance name is `axiInstanceTemplate` (extending `denaliCdn_axiInstance`).

Use `axiInstanceTemplate` in this example; therefore, the code will look as shown below:

```
package myPackage;
 import DenaliSvCdn_axi::*;
 import DenaliSvMem::*;
`include "denaliAxiUserInstance.sv"
endpackage

module axi_top;

    import myPackage::*;
    import DenaliSvCdn_axi::*;
    import DenaliSvMem::*;


    axiInstanceTemplate activeMasterInstance;
    axiInstanceTemplate activeSlaveInstance;

    initial
    begin
        activeMasterInstance = new ("axi_top.activeMasterDevice");
        activeSlaveInstance = new ("axi_top.activeSlaveDevice");
    end
endmodule
```

2. Add the actual configuration - the minimum is to configure the memory space.

```
package myPackage;
 import DenaliSvCdn_axi::*;
 import DenaliSvMem::*;
`include "denaliAxiUserInstance.sv"
endpackage

module axi_top;

 import myPackage::*;
import DenaliSvCdn_axi::*;
import DenaliSvMem::*;


    axiInstanceTemplate activeMasterInstance;
    axiInstanceTemplate activeSlaveInstance;

    …

    initial
    begin

        activeMasterInstance = new ("axi_top.activeMasterWrapper");
        activeSlaveInstance = new ("axi_top.activeSlaveWrapper");

// mapMemorySegment is declared in the file denaliAxiUserInstance.sv

        activeMasterInstance.mapMemorySegment(64'h0,64'h3FFF);
        activeSlaveInstance.mapMemorySegment(64'h0,64'h3FFF);
```

```
    end
endmodule
```

3.6. Writing a Simple Test Scenario

This section shows how to write and send transactions. For more test scenarios, refer to Chapter 9, *CDN_AXI Verification Test Scenarios*.

Note

Cadence recommends not to use `denaliCdn_axiTransaction` as it is; you must extend it and add your DUT-specific constraints. See Section 6.3.5, "*Transaction Types and Fields*"

```
package myPackage;
import DenaliSvCdn_axi::*;
import DenaliSvMem::*;
`include "denaliAxiUserInstance.sv"
class myTransaction extends denaliCdn_axiTransaction;

 function new();
  super.new();
  // these fields must be set in addition to the SOMA setting
  this.SpecVer = DENALI_CDN_AXI_SPECVERSION_AMBA3;
  this.SpecSubtype = DENALI_CDN_AXI_SPECSUBTYPE_BASE;
  this.SpecInterface = DENALI_CDN_AXI_SPECINTERFACE_FULL;
 endfunction

 constraint user_dut_information {
   (StartAddress >= 'h0) && (StartAddress <= 'h1FF);
  BurstMaxSize == DENALI_CDN_AXI_TRANSFERSIZE_WORD;
  IdTag < (1 << 3);
  Cacheable == DENALI_CDN_AXI_CACHEMODE_NON_CACHEABLE;
  ModelGeneration == 0;
 }
endclass
endpackage
module axi_top;
import myPackage::*;
import DenaliSvCdn_axi::*;
import DenaliSvMem::*;
axiInstanceTemplate activeMasterInstance;
axiInstanceTemplate activeSlaveInstance;

 reg aclk;
reg aresetn;
wire awvalid;
wire [31:0] awaddr;
wire [7:0] awlen;
wire [2:0] awsize;
wire [1:0] awburst;
wire [1:0] awlock;
wire [3:0] awcache;
wire [2:0] awprot;
wire [3:0] awid;
wire awready;

wire [31:0] awuser;
wire wvalid;
wire wlast;
wire [31:0] wdata;
wire [3:0] wstrb;
```

```
wire [3:0] wid;
wire wready;

wire [31:0] wuser;
wire bvalid;

wire [1:0] bresp;
wire [3:0] bid;

wire bready;
wire [31:0] buser;

wire arvalid;
wire [31:0] araddr;
wire [7:0] arlen;
wire [2:0] arsize;
wire [1:0] arburst;
wire [1:0] arlock;
wire [3:0] arcache;
wire [2:0] arprot;
wire [3:0] arid;
wire arready;

wire [31:0] aruser;
wire rvalid;

wire rlast;
wire [31:0] rdata;

wire [1:0] rresp;
wire [3:0] rid;

wire rready;
wire [31:0] ruser;

activeMaster activeMasterDevice(aclk, aresetn, awvalid, awaddr, awlen, awsize, awburst, awlock,
 awcache, awprot, awid, awready, awuser,  wvalid, wlast, wdata, wstrb, wid, wready, wuser, bvalid,
 bresp, bid, bready, buser, arvalid, araddr, arlen, arsize, arburst, arlock, arcache, arprot, arid,
 arready, aruser, rvalid, rlast, rdata,  rresp, rid, rready, ruser);

activeSlave activeSlaveDevice(aclk, aresetn, awvalid, awaddr, awlen, awsize, awburst, awlock, awcache,
 awprot, awid, awready, awuser,  wvalid, wlast, wdata, wstrb, wid, wready, wuser, bvalid, bresp,
 bid, bready, buser, arvalid, araddr, arlen, arsize, arburst, arlock, arcache, arprot, arid, arready,
 aruser, rvalid, rlast, rdata,  rresp, rid, rready, ruser);

initial
 begin
aclk = 1'b1;
 aresetn = 1'b1;
 #100
aresetn = 1'b0;
#500
aresetn = 1'b1;
end
 always #50 aclk = ~aclk;

myTransaction masterBurst; // transaction with dut specific constraints

integer status;

initial
begin
activeMasterInstance = new ("axi_top.activeMasterDevice");
         activeSlaveInstance = new ("axi_top.activeSlaveDevice");

// mapMemorySegment is declared in the file denaliAxiUserInstance.sv
         activeMasterInstance.mapMemorySegment(64'h0,64'h3FFF);
```

```
        activeSlaveInstance.mapMemorySegment(64'h0,64'h3FFF);

masterBurst = new();

 activeMasterInstance.regWrite(DENALI_CDN_AXI_REG_Verbosity,DENALI_CDN_AXI_MESSAGEVERBOSITY_MEDIUM);

 for (int ii=0; ii<1000; ii++) begin
assert(masterBurst.randomize() with {
Type == (ii % 2 == 0 ? DENALI_CDN_AXI_TR_Read : DENALI_CDN_AXI_TR_Write);
                           Direction == (ii % 2 == 0 ? DENALI_CDN_AXI_DIRECTION_READ :
 DENALI_CDN_AXI_DIRECTION_WRITE); Cacheable == DENALI_CDN_AXI_CACHEMODE_NON_CACHEABLE;
Access == DENALI_CDN_AXI_ACCESS_NORMAL;
Length == 4;
StartAddress < 'h400;
Kind == DENALI_CDN_AXI_BURSTKIND_INCR;
Data.size() == 16;
foreach (Data[i])
Data[i] == i;
             Size == DENALI_CDN_AXI_TRANSFERSIZE_WORD;
             IdTag == 0;
}) else $fatal;
status = activeMasterInstance.transAdd(masterBurst, DENALI_CDN_AXI_QUEUE_Burst);

                  #1000;

                end

        $finish;
end
endmodule
```

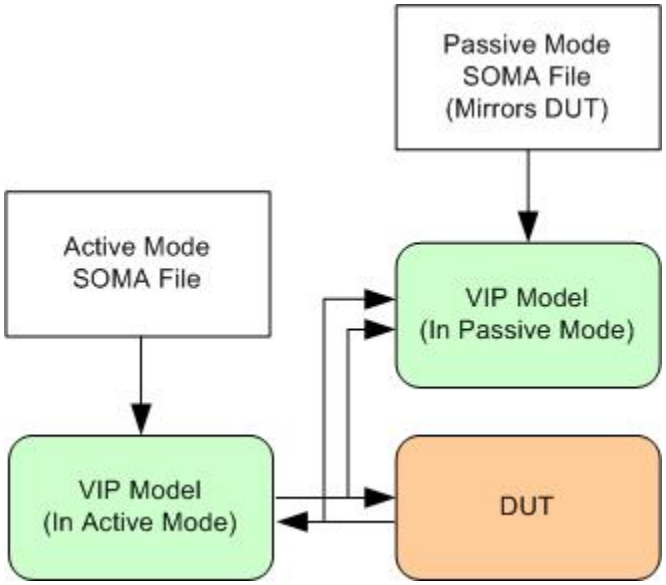
Chapter 4. PureView Graphical Tool

The protocol model is characterized with a Specification of Modeling Architecture (SOMA) file. This file captures a complete model behavior, including model type, model capabilities, pin interface, and so on.

The PureView graphical user interface enables you to configure a SOMA file and generate an instantiation interface file. You can use PureView to set most of the model's configurable features, such as the CDN_AXI VIP model mode, interface type, and so on.

To use CDN_AXI VIP, you need to generate two SOMA files. One for the passive mode model (this SOMA file mirrors your DUT's configuration) and one for the active mode model. The active mode model SOMA file should be configured to mirror an appropriate target to communicate with using the desired protocol. You can use PureView to create these two SOMA files.

Figure 4.1. SOMA Files for the VIP Model



To instantiate any CDN_AXI VIP model, you must first configure a SOMA file and generate an instantiation interface using PureView.

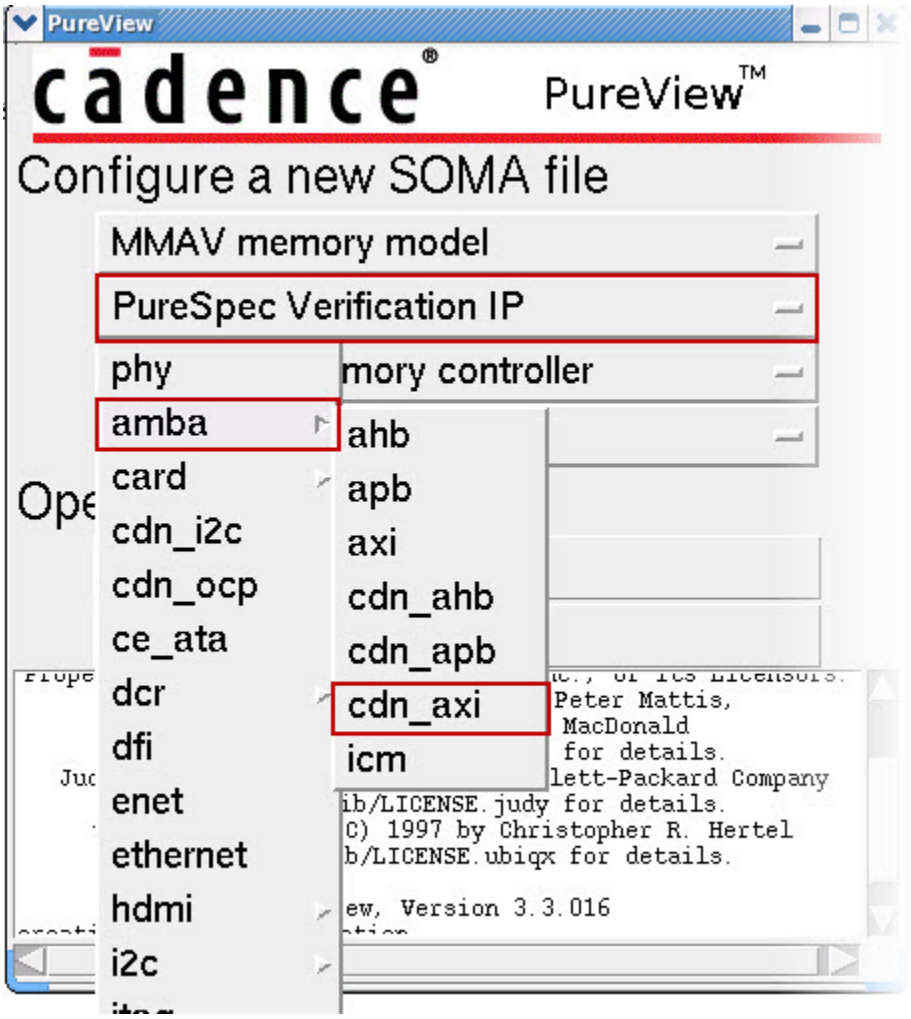
4.1. Creating a New SOMA File or Reconfiguring An Existing SOMA Setting

To create a new SOMA file or to reconfigure an existing SOMA setting:

1. Invoke PureView from your working directory:

   % pureview

Figure 4.2. PureView Opening Window



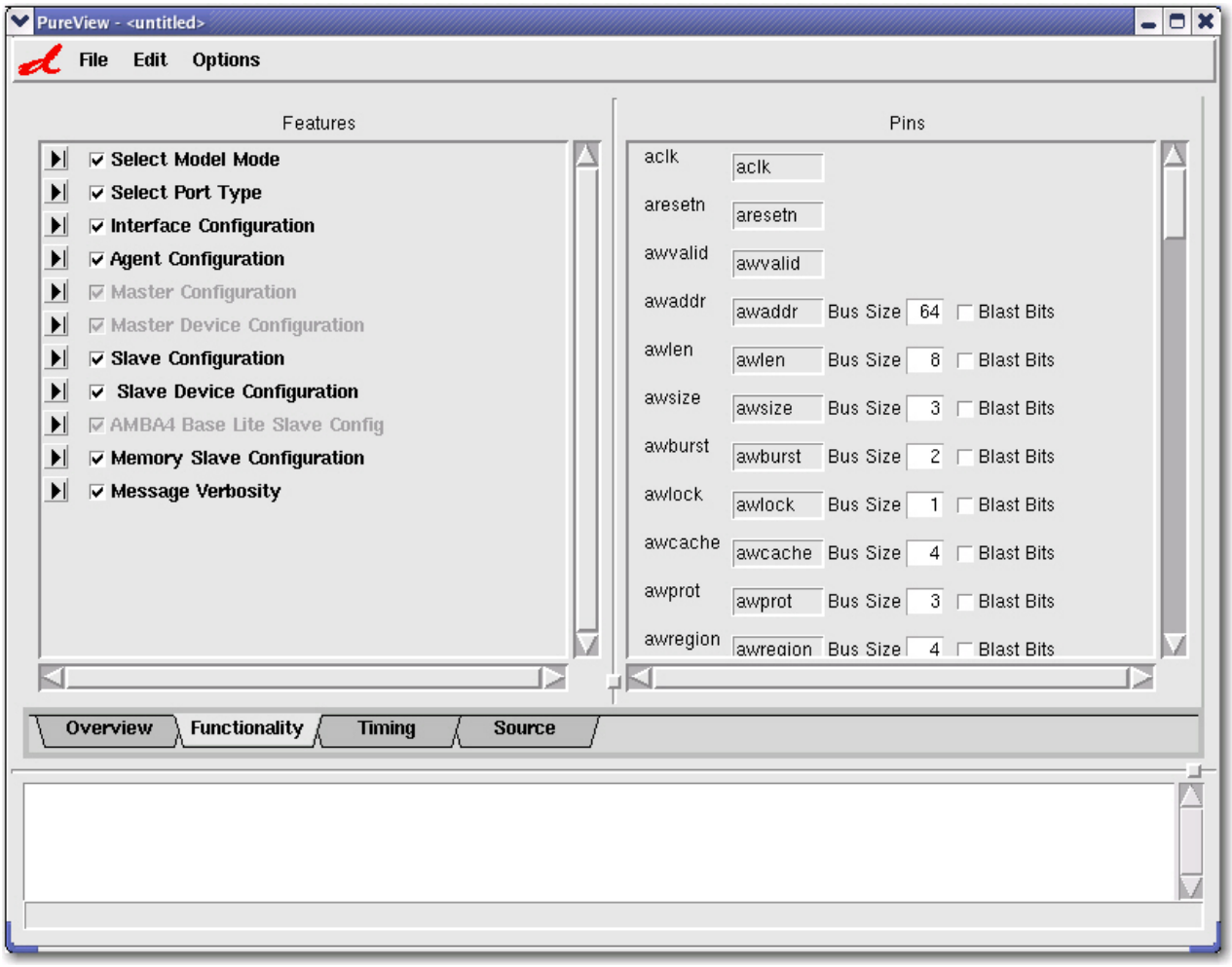
2. From the drop down list, either **Configure a new SOMA file** or **Open an existing file**.

3. Click on the **Functionality** tab. The following window appears.

Note

Here **Device Mode** refers to the *Active Mode* and **Monitor Mode** refers to the *Passive Mode*.

Figure 4.3. PureView - Functionality Tab



4. Select or de-select SOMA features as per your requirements.
5. Verify the SOMA file settings using **File > Check SOMA**.
6. Select **File > Save As** to save the SOMA file. You can save this file with an extension as either `.spc` or `.soma`.

When you have successfully saved the SOMA file, you need to create an instantiation interface for the device suitable for your particular simulation environment.

4.2. Creating an HDL Instantiation Interface

To create an instantiation interface:

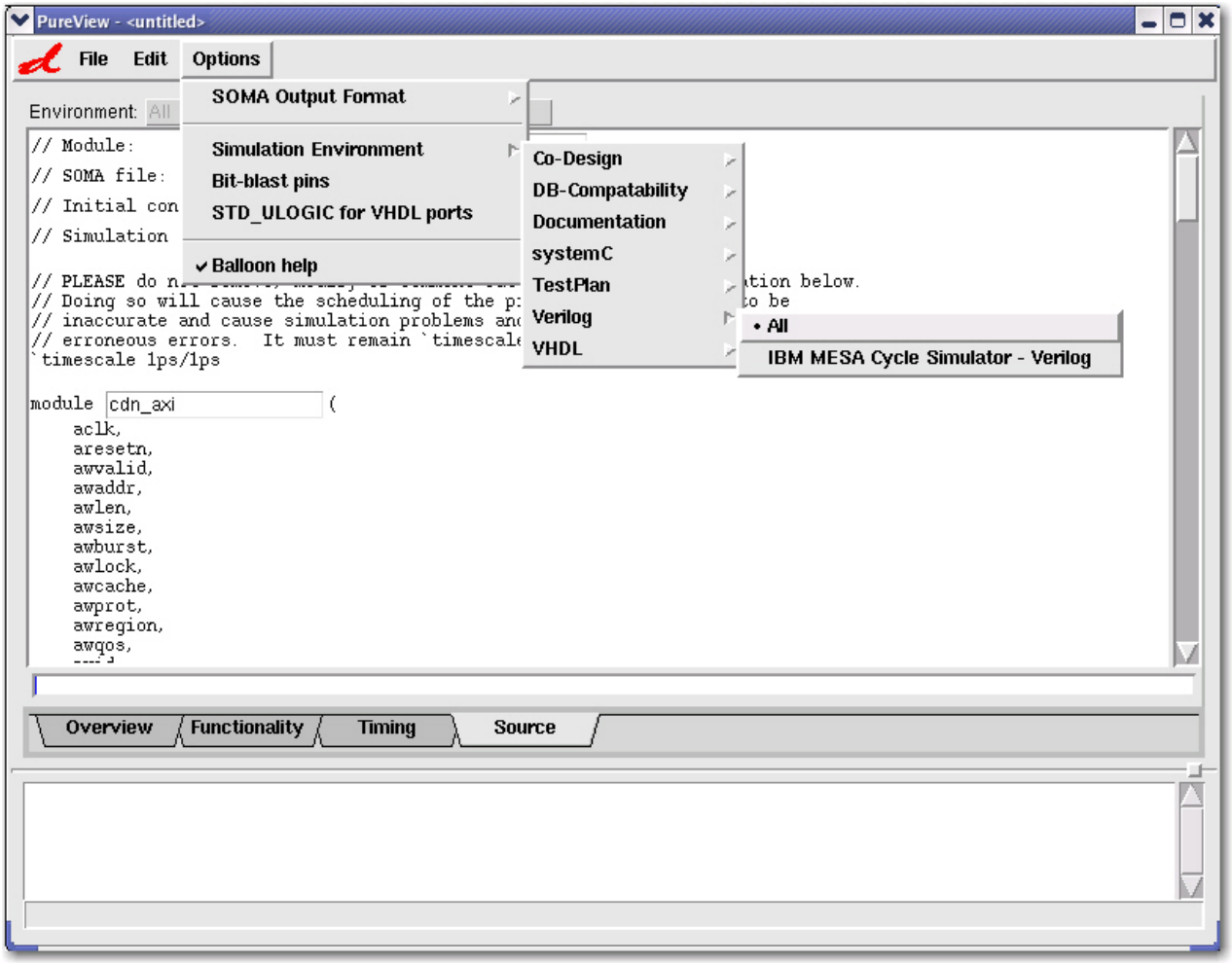
1. Select the **Source** tab in the PureView user interface.

   The **Source** window is initially blank.

2. Select **Options > Simulation Environment**.

   Though the user interface has three choices for HDL (**Verilog**, **VHDL**, or **SystemC**) **please note** that only **SystemVerilog** is currently supported. Then, proceed to select the actual simulator you want to use. The actual code for the HDL simulator appears in the **Source** window. Note that some of the fields can be edited. These fields have been preloaded with an appropriate data from the SOMA file you just saved, and do not need to be modified. If you choose to modify the location of the `.soma` file or the initial contents of SOMA file, select the ... Browse button next to these fields, at which point, the file selection window appears. Use this window to select an alternate file.

Figure 4.4. PureView - Source Tab



3. Select **File > Save Source As** to write the instantiation interface to a file so it can be instantiated into your design.

   A file selection window appears in which you can save your file.

Chapter 5. CDN_AXI VIP Model Configuration

The CDN_AXI VIP model is configurable via the specification of modeling architecture (SOMA) parameter file and the `.denalirc` file. The SOMA parameters are used to configure individual models and the `.denalirc` parameters are used for the run-time options. In addition to the configurations implied by CDN_AXI specifications, the CDN_AXI VIP also provides implementation-specific and testability options.

5.1. .denalirc

The `.denalirc` file is a simple text file containing keyword-value pairs used to store your CDN_AXI VIP model-specific settings.

A default `.denalirc` file is located in `$DENALI/.denalirc`. You can create up to four `.denalirc` files to control simulation or to set basic logging and simulation parameters. See Chapter 8, *CDN_AXI VIP Simulation* and Section 8.2, "*Controlling Model Behavior*" for further information on simulation-specific settings and the locations of `.denalirc` files.

5.1.1. General .denalirc Options

The following section details the `.denalirc` parameters that are not protocol-specific.

Table 5.1. General .denalirc Parameters

Name	Values	Default
	Description	
Historyfile	Filename	None
	The filename where the simulation history information will be stored. You can specify the `-gzip` option if you want to gzip the files during the simulation run. By default, the model does not zip the files. `Historyfile -gzip denali.his.gz` When you create the history file using the `-gzip` option, Cadence recommends that you specify the filename with extension `.gz`. If you do not specify the `.gz` extension, the model still creates a legal zip file. However, you need to add the `.gz` extension later (using UNIX command `mv`) to unzip successfully using the *gunzip* utility. The operating system IO file size (2 GB) limitation also applies on the gzipped version.	
HistoryDebug	On/Off	Off
	Enables more detailed history messages.	
Tracefile	Filename	None

Name	Values	Default
	Description	
	Enables the Tracefile. You can specify the `-gzip` option if you want to gzip the files during the simulation run. By default, the model does not zip the files. `Tracefile -gzip denali.trc.gz` When you create the trace file using the `-gzip` option, Cadence recommends that you specify the filename with extension `.gz`. If you do not specify the filename extension, the model still creates a legal zip file. However, you need to add the `.gz` extension later (using UNIX command `mv`) to unzip successfully using the *gunzip*utility. The operating system IO file size (2 GB) limitation also applies on the gzipped version.	
HistoryInSimLog	0/1	0
	Redirects the history file to your simulation log versus the filename mentioned in the `Historyfile`	
HistoryPattern	Pattern string	All instances
	Adds only information about the specified `pattern` to the history file. This parameter is used to limit the amount of information dumped into the history file.	
TracePattern	Pattern string	All instances
	Adds only information about the specified `pattern` to the trace file. This parameter is used to limit the amount of information dumped into the trace file.	
ReportTimeUnits	Integer, us, ns, ps, or fs	Integer
	Displays values in the specified format in the history and trace files. By default, the time units are reported in the largest unit for which the numeric part will be an integer. For example, to specify the time units in picoseconds, use the following: `ReportTimeUnits ps`	
InitMessages	On/Off	On
	Turns on/off messages pertaining to the instances in the design.	
LicenseQueueTimeout	Time value	0
	Specifies the number of minutes to wait before exiting, if the license is not available.	
LicenseQueueRetryDelay	Time value	0
	Specifies the number of seconds to wait before pinging for available licenses.	
MaxHeapSize	Integer	0

Name	Values	Default
	Description	
	Some models control the maximum heap size that a simulation process can use. You can override the default heap size by setting the MaxHeapSize variable with a non-zero integer for the size (in MegaBytes). The value zero indicates that the default setting should be used. For 32_bits, the max size should not exceed 3072 (MB), which is the default.	
EnableCoverage	On/Off	An instance that is not covered by an enable will have the default behavior for that kind of model.
	Some models allow you to turn coverage on with a variable. You can use the EnableCoverage variable to override the default behavior of one or more instances. An instance that is not covered by an enable will have the default behavior for that kind of model. Patterns identify the set of instances that you are enabling. They use shell glob syntax (*,",[]). For example: *[Pp]assive* or top.Active*. Here * specifies all instances.	
DisableCoverage	On/Off	An instance that is not covered by a disable will have the default behavior for that kind of model.
	Some models allow you to turn coverage off with a variable. You can use the DisableCoverage variable to override the default behavior of one or more instances. An instance that is not covered by a disable will have the default behavior for that kind of model. Patterns identify the set of instances that you are disabling. They use shell glob syntax (*,",[]). For example: *[Pp]assive* or top.Active*. Here * specifies all instances.	
EnableProtocolChecks	On/Off	An instance that is not covered by an enable will have the default behavior for that kind of model.
	Some models allow you to turn protocol checks on with a variable. You can use EnableProtocolChecks to override the default behavior of one or more instances. An instance that is not covered by an enable will have the default behavior for that kind of model. Patterns identify the set of instances that you are enabling. They use shell glob syntax (*,",[]). For example: *[Pp]assive* or top.Active*. Here * specifies all instances.	
DisableProtocolChecks	On/Off	An instance that is not covered by a disable will have the default behavior for that kind of model.

Name	Values	Default
	Description	
	Some models allow you to turn protocol checks off with a variable. You can use DisableProtocolChecks to override the default behavior of one or more instances. An instance that is not covered by a disable will have the default behavior for that kind of model. Patterns identify the set of instances that you are disabling. They use shell glob syntax (*,'',[]). For example: *[Pp]assive* or top.Active*. Here * specifies all instances.	

Note

The CDN_AXI VIP model does not support save or restore features during simulation.

5.2. SOMA

The CDN_AXI VIP model has its own SOMA parameter file that describes its attributes.

5.2.1. Details / Table

The following table lists the SOMA parameters available to configure the CDN_AXI VIP model. You can change most of the parameters during run time using the CDN_AXI VIP registers.

Table 5.2. SOMA Features

Name	Description
`DeviceMode`	Active The model models an actual device. The default value is `1`
`MonitorMode`	Passive The model models a passive device, monitoring The default value is `0`
`AcceleratedMode`	Accelerated VIP (AVIP) The model can run in simulation or emulation The default value is `0`
`Master`	Master Models behavior of AXI Master The default value is `0`
`Slave`	Slave Models behavior of AXI Slave

Name	Description
	The default value is `1`
`vr_axi_interface`	Interface Configuration Interface between a master and a slave (each or both of them can be interconnect ports). Contains configuration fields that are common to a master-slave pair. The default value is `1`
`spec_ver`	spec_ver Version of AXI Specification The default value is `0`
`spec_ver_AMBA3`	AMBA3 The default value is `0`
`spec_ver_AMBA4`	AMBA4 The default value is `1`
`spec_subtype`	spec_subtype The spec subtype The default value is `0`
`spec_subtype_BASE`	BASE The default value is `0`
`spec_subtype_ACE`	ACE The default value is `1`
`spec_interface`	spec_interface The spec interface (FULL or LITE) The default value is `0`
`spec_interface_FULL`	FULL The default value is `1`
`spec_interface_LITE`	LITE The default value is `0`
`cache_line_size`	cache_line_size The cache line size in bytes. Pertains to AMBA4 ACE Specification The default value is `32`

Name	Description
write_data_interleaving_depth	write_data_interleaving_depth The maximum allowed number of different write transfers with different IDs on the write data bus. Note: This field is only relevant for AXI3. In AXI4 (or higher), where there is no WID signal, this field is always 1. The default value is 1
vr_axi_agent_config	Agent Configuration Base unit for all agents configuration. The default value is 1
do_signal_check_only_when_valid	do_signal_check_only_when_valid Whether signal checking be done only when valid signal high or at all times. default: FALSE The default value is 0
do_basic_deadlock_check	do_basic_deadlock_check Enable the deadlock check, i.e. count how many cycles have passed without AWVALID and ARVALID signals changing their state in Write Address and Read Address channels correspondingly. default: FALSE The default value is 0
no_changes_in_address_channels_limit	no_changes_in_address_channels_limit If do_basic_deadlock_check is TRUE, this entry determines the limit for the deadlock control counter. default: 50 The default value is 50
random_data_when_not_valid	random_data_when_not_valid Whether to drive random data on relevant channels when the valid signal is low. Applies only to ACTIVE agents. The default value is 0
update_cache_for_error_burst_response	update_cache_for_error_burst_response Setting this field to true will update the master cache (allocation/ eviction) even if the burst was not ended ok according to the method 'is_ended_ok()'. The master will update the cache regardless of the transfers responses (rresp,bresp,crresp). Pertains to AMBA4 ACE Specification. Default value - TRUE The default value is 1

Name	Description
using_snoop_filter	using_snoop_filter Adjust the agent's behavior - whether it is connected to a snoop filter or not. Pertains to AMBA4 ACE Specification. Default value - TRUE The default value is 1
ace_lite_dvm_support	ace_lite_dvm_support Agents can support DVM. This does not appear in the spec but is implemented in CCI400 and required by customers. Pertains to AMBA4 ACE Specification The default value is 0
vr_axi_master_config	Master Configuration Master configuration unit. The default value is 1
max_write_bursts_behavior	max_write_bursts_behavior Determines master behavior when the number of transmitted WRITE bursts reaches 'write_data_interleaving_depth'. CONTINUE_TO_SEND : Continue to send new bursts address-phases only. STOP_WRITE : Stop sending WRITE bursts. STOP_ALL : Stop sending both READ and WRITE bursts. The default value is 0
max_write_bursts_behavior_CONTINUE_TO_SEND	CONTINUE_TO_SEND The default value is 1
max_write_bursts_behavior_STOP_WRITE	STOP_WRITE The default value is 0
max_write_bursts_behavior_STOP_ALL	STOP_ALL The default value is 0
read_issuing_capability	read_issuing_capability This is the maximum number of outstanding read transactions that the master can send The default value is 5
write_issuing_capability	write_issuing_capability This is the maximum number of outstanding write transactions that the master can send

Name	Description
	The default value is `5`
`stall_addr_after_max_read_issuing_capability`	stall_addr_after_max_read_issuing_capability TRUE: When the number of READ pending bursts equals 'read_issuing_capability' - stall the read address channel (ARVALID low). FALSE:(DEFAULT) Can continue to issue new read bursts address-phases. The default value is `1`
`stall_addr_after_max_write_issuing_capability`	stall_addr_after_max_write_issuing_capability TRUE: When the number of WRITE pending bursts equals 'write_issuing_capability' - stall the write address channel (AWVALID low). FALSE:(DEFAULT) Can continue to issue new write bursts address-phases. The default value is `1`
`consider_wstrb_in_bad_signal_checks`	consider_wstrb_in_bad_signal_checks Take into consideration the strobe value when running validity check on the WDATA signal. Default : FALSE The default value is `0`
`using_speculative_fetch`	using_speculative_fetch speculative fetch is a state when the master generate read bursts Although the data is in the cache. Pertains to AMBA4 ACE Specification. The default value is `0`
`all_legal_states`	all_legal_states According to the Spec, there are legal states for sending new read/write snoop commands TRUE - sending new read/write snoop commands for every legal state in the cache FALSE - sending new read/write snoop commands only for expected states in the cache. Pertains to AMBA4 ACE Specification. The default value is `0`
`ACTIVEactive_passivevr_axi_master_config`	Master Device Configuration Master configuration unit. The default value is `1`
`master_gen_auto_resp`	master_gen_auto_resp

Name	Description
	By default ACE Master agent auto-generates responses to received SNOOP bursts. If this behavior is disabled, the user would be responsible for manually creating master's responses to incoming SNOOP bursts. For AMBA4 ACE and snooping only The default value is `1`
`ordering_algorithm`	ordering_algorithm The type of ordering algorithm to use for WRITE transfers. The default value is `0`
`ordering_algorithm_NORMAL_ORDER`	NORMAL_ORDER The default value is `1`
`ordering_algorithm_ROUND_ROBIN`	ROUND_ROBIN The default value is `0`
`ordering_algorithm_FULL_RANDOM`	FULL_RANDOM The default value is `0`
`vr_axi_slave_config`	Slave Configuration Slave configuration unit. The default value is `1`
`use_memory`	use_memory When set: In Device mode: Slave will use the sparse memory model for storying/retrieving data. In Monitor mode: Slave will use the sparse memory model for DUT shadowing and data consistency checking. When Unset: In Device mode: Slave will always randomize new READ data even if this location was previously accessed. In Monitor mode: Slave will not write new data to the memory and will not check READ response consistency. The default value is `1`
`read_data_reordering_depth`	read_data_reordering_depth Determines the maximum allowed number of different read transfers with different IDs on the read data bus. The default value is `5`
`read_acceptance_capability`	read_acceptance_capability

Name	Description
	Determines the maximum number of different read bursts accepted (arvalid+arready high) on the read address channel. The default value is `5`
`write_acceptance_capability`	write_acceptance_capability Determines the maximum number of different write bursts accepted (awvalid+awready high) on the write address channel. The default value is `5`
`stall_addr_after_max_write_depth`	stall_addr_after_max_write_depth TRUE: When the number of WRITE pending bursts equals 'write_data_interleaving_depth' - stall the write address channel (AWREADY low). FALSE: Can continue to accept new bursts address-phases. The default value is `0`
`stall_addr_after_max_read_depth`	stall_addr_after_max_read_depth TRUE: When the number of READ pending bursts equals 'read_data_reordering_depth' - stall the read address channel (ARREADY low). FALSE: Can continue to accept new bursts address-phases. The default value is `0`
`stall_addr_after_max_read_acceptance_capability`	stall_addr_after_max_read_acceptance_capability TRUE: When the number of READ pending bursts equals 'read_acceptance_capability' - stall the read address channel (ARREADY low). FALSE:(DEFAULT) Can continue to accept new bursts address-phases. The default value is `1`
`stall_addr_after_max_write_acceptance_capability`	stall_addr_after_max_write_acceptance_capability TRUE: When the number of WRITE pending bursts equals 'write_acceptance_capability' - stall the write address channel (AWREADY low). FALSE:(DEFAULT) Can continue to accept new bursts address-phases. The default value is `1`
`update_exclusive_state_at_last_tr`	update_exclusive_state_at_last_tr

Name	Description
	This flag indicates whether exclusive status is determined when the last transfer is received or when write resp is sent (Default: FALSE) The default value is `0`
`stop_mon_ex_addr_on_unsuccessful_wr`	stop_mon_ex_addr_on_unsuccessful_wr This flag indicates whether the slave should stop monitoring an exclusive address after an unsuccessfully write attempt. When TRUE - slave will respond with OKAY to an EXCLUSIVE WRITE whenever there was a WRITE burst to an overlapping address (no matter what was the response to that write burst) When FALSE - slave will responsd with OKAY to an EXCLUSIVE WRITE whenever there is a WRITE burst to an overlapping address that was successful (OKAY) and with EXOKAY response after an unsuccessful write burst (SLVERR) (Default: FALSE) The default value is `0`
`disable_memory_update_on_write_burst`	disable_memory_update_on_write_burst Whether to prevent the ACTIVE slave bfm or the PASSIVE slave monitor from updating the memory. The default value is `0`
`snoop_issuing_capability`	snoop_issuing_capability This is the maximum number of active snoop transactions that the slave interface can send. Pertains to AMBA4 ACE Specification. The default value is `1`
`ACTIVEactive_passivevr_axi_slave_config`	Slave Device Configuration Slave configuration unit. The default value is `1`
`read_ordering_algorithm`	read_ordering_algorithm The type of ordering algorithm to use for READ transfer-responses. The default value is `0`
`read_ordering_algorithm_NORMAL_ORDER`	NORMAL_ORDER The default value is `1`
`read_ordering_algorithm_ROUND_ROBIN`	ROUND_ROBIN

Name	Description
	The default value is `0`
`read_ordering_algorithm_RANDOM_NO_INTERLEAVING`	RANDOM_NO_INTERLEAVING The default value is `0`
`read_ordering_algorithm_FULL_RANDOM`	FULL_RANDOM The default value is `0`
`write_resp_ordering_algorithm`	write_resp_ordering_algorithm The type of ordering algorithm to use for WRITE burst-responses. The default value is `0`
`write_resp_ordering_algorithm_NORMAL_ORDER`	NORMAL_ORDER The default value is `1`
`write_resp_ordering_algorithm_ROUND_ROBIN`	ROUND_ROBIN The default value is `0`
`write_resp_ordering_algorithm_DATA_PHASES_END_TIMES`	DATA_PHASES_END_TIMES The default value is `0`
`write_resp_ordering_algorithm_FULL_RANDOM`	FULL_RANDOM The default value is `0`
`gen_auto_resp`	gen_auto_resp By default Slave agent auto-generates responses to received bursts. If this behavior is disabled, the user would be responsible for manually creating Slave responses to incoming bursts. The default value is `1`
`LITEspec_interface BASEspec_subtypeAMBA4spec_vervr_axi_slave_config`	AMBA4 Base Lite Slave Config Slave configuration unit. The default value is `1`
`write_strobe_usage`	write_strobe_usage In AMBA4 Lite there are three options for the Slave to work with WSTRB: 1. FULL USE of wstrb. 2. IGNORE wstrb value and treat all write accesses as being the full data bus width. 3. Detect wstrb COMBINITATION. This option is not supported and would provide error response. The default value is `0`

Name	Description
write_strobe_usage_FULL_USE	FULL_USE The default value is 1
TRUEuse_memoryvr_axi_slave_config	Memory Slave Configuration Slave configuration unit. The default value is 1
slverr_write_cause_invalid_value	slverr_write_cause_invalid_value When set, invalidates slave memory content for the write burst which results in SLVERR response. The default value is 1
no_consistency_check_if_overlap	no_consistency_check_if_overlap When set, disables the memory consistency checks for read bursts when a write burst to the same address is in progress. The default value is 1
update_memory_for_error_snoop_response	update_memory_for_error_snoop_response Setting this field to true will update the slave's memory on snoops even if the snoop response was error. The slave will update the memory regardless of the snoop responses (crresp). Pertains to AMBA4 ACE Specification. Default value - FALSE The default value is 0
vr_axi_agent_monitor	Performance Configuration The default value is 1
coverage_performance_level	coverage_performance_level When set to HIGH, the monitor stops collecting coverage for the channel_info coverage group. This improves performance, because the channel_info coverage group collects coverage each cycle. (The default is NORMAL) The default value is 0
coverage_performance_level_HIGH	HIGH The default value is 0
coverage_performance_level_NORMAL	NORMAL The default value is 1
checks_performance_level	checks_performance_level

Name	Description
	When set to HIGH, the monitor stops perform checking for checker 100,190 (for master) and 200,240 (for slave) This improves performance, because these checks are performed every cycle. The default value is `0`
`checks_performance_level_HIGH`	HIGH The default value is `0`
`checks_performance_level_NORMAL`	NORMAL The default value is `1`

5.2.2. SOMA Configurability vs. Model Register Space

The following table lists the SOMA parameters that can be modified at run time using the CDN_AXI VIP registers. The table shows the SOMA parameter name and its corresponding run-time counterpart register.

Table 5.3. SOMA Parameters vs. Register Names

SOMA Parameter Name	Register Name
write_data_interleaving_depth	DENALI_CDN_AXI_REG_WriteData InterleavingDepth
do_signal_check_only_when_valid	DENALI_CDN_AXI_REG_DoSignalCheckOnly WhenValid
do_basic_deadlock_check	DENALI_CDN_AXI_REG_DoBasicDeadlock Check
no_changes_in_address_channels_limit	DENALI_CDN_AXI_REG_NoChangesIn AddressChannelsLimit
update_cache_for_error_burst_response	DENALI_CDN_AXI_REG_UpdateCacheFor ErrorBurstResponse
using_snoop_filter	DENALI_CDN_AXI_REG_UsingSnoopFilter
max_write_bursts_behavior	DENALI_CDN_AXI_REG_MaxWriteBursts Behavior
read_issuing_capability	DENALI_CDN_AXI_REG_ReadIssuing Capability
write_issuing_capability	DENALI_CDN_AXI_REG_WriteIssuing Capability
stall_addr_after_max_read_issuing_capability	DENALI_CDN_AXI_REG_StallAddrAfterMax ReadIssuingCapability

SOMA Parameter Name	Register Name
stall_addr_after_max_write_issuing_capability	DENALI_CDN_AXI_REG_StallAddrAfterMax WriteIssuingCapability
consider_wstrb_in_bad_signal_checks	DENALI_CDN_AXI_REG_ConsiderWstrbInBad SignalChecks
using_speculative_fetch	DENALI_CDN_AXI_REG_UsingSpeculative Fetch
all_legal_states	DENALI_CDN_AXI_REG_AllLegalStates
ordering_algorithm	DENALI_CDN_AXI_REG_OrderingAlgorithm
use_memory	DENALI_CDN_AXI_REG_UseMemory
read_data_reordering_depth	DENALI_CDN_AXI_REG_ReadDataReordering Depth
read_acceptance_capability	DENALI_CDN_AXI_REG_ReadAcceptance Capability
write_acceptance_capability	DENALI_CDN_AXI_REG_WriteAcceptance Capability
stall_addr_after_max_write_depth	DENALI_CDN_AXI_REG_StallAddrAfterMax WriteDepth
stall_addr_after_max_read_depth	DENALI_CDN_AXI_REG_StallAddrAfterMax ReadDepth
stall_addr_after_max_read_acceptance_capability	DENALI_CDN_AXI_REG_StallAddrAfterMax ReadAcceptanceCapability
stall_addr_after_max_write_acceptance_ capability	DENALI_CDN_AXI_REG_StallAddrAfterMax WriteAcceptanceCapability
update_exclusive_state_at_last_tr	DENALI_CDN_AXI_REG_UpdateExclusive StateAtLastTr
stop_mon_ex_addr_on_unsuccessful_wr	DENALI_CDN_AXI_REG_StopMonExAddrOn UnsuccessfulWr
disable_memory_update_on_write_burst	DENALI_CDN_AXI_REG_DisableMemory UpdateOnWriteBurst
snoop_issuing_capability	DENALI_CDN_AXI_REG_SnoopIssuing Capability
read_ordering_algorithm	DENALI_CDN_AXI_REG_ReadOrdering Algorithm
write_resp_ordering_algorithm	DENALI_CDN_AXI_REG_WriteRespOrdering Algorithm
gen_auto_resp	DENALI_CDN_AXI_REG_GenAutoResp

SOMA Parameter Name	Register Name
slverr_write_cause_invalid_value	DENALI_CDN_AXI_REG_SlverrWriteCause InvalidValue
no_consistency_check_if_overlap	DENALI_CDN_AXI_REG_NoConsistency CheckIfOverlap
update_memory_for_error_snoop_response	DENALI_CDN_AXI_REG_UpdateMemoryFor ErrorSnoopResponse

5.3. Registers

You can use model registers to dynamically control the simulation by writing appropriate values to them. The following table shows a complete list of the registers available in the CDN_AXI VIP model.

Table 5.4. Summary of Registers

Register	Description
`DENALI_CDN_AXI_REG_ErrCtrl`	Control over behavior when error
`DENALI_CDN_AXI_REG_RegionType`	Type of the memory domain, one of DENALI_CDN_AXI_DOMAIN_INNER, DENALI_CDN_AXI_DOMAIN_OUTER, DENALI_CDN_AXI_DOMAIN_NON_SHAREABLE or DENALI_CDN_AXI_DOMAIN_SYSTEM. Writing to this register tells the model to set a new memory segment with parameters defined in registers RegionAddressFromHigh, RegionAddressFromLow, RegionAddressToHigh, RegionAddressToLow.
`DENALI_CDN_AXI_REG_RegionAddressFromHigh`	High part of the start memory address of a memory region with a type defined in register RegionType. (Available also in AVIP mode)
`DENALI_CDN_AXI_REG_RegionAddressFromLow`	Low part of the start memory address of a memory region with a type defined in register RegionType. (Available also in AVIP mode)
`DENALI_CDN_AXI_REG_RegionAddressToHigh`	High part of the end memory address of a memory region with a type defined in register RegionType. (Available also in AVIP mode)
`DENALI_CDN_AXI_REG_RegionAddressToLow`	Low part of the end memory address of a memory region with a type defined in register RegionType. (Available also in AVIP mode)
`DENALI_CDN_AXI_REG_ErrorID`	Error ID enum value to which the severity will be set. This register was deprecated. Please use the new Err Ctrl register.

Register	Description
DENALI_CDN_AXI_REG_Error Severity	Set Severity to the Error ID enum value in the ErrorID register. This register was deprecated. Please use the new ErrCtrl register. (Available also in AVIP mode)
DENALI_CDN_AXI_REG_Reset SignalsSimStart	Module to reset signals at simulation start. When set to 1, the active components set the output values of their signals to known values at simulation's time zero. Otherwise, the signals values will be unknown until reset happens. (Available also in AVIP mode)
DENALI_CDN_AXI_REG_Retrieve Item	Available when using ICM - Is activated when user want to retrieve an item associated with ID
DENALI_CDN_AXI_REG_ModuleId	Available when using ICM - Returns a unique identifier of the instance in the tb to be used when connecting to the ICM.
DENALI_CDN_AXI_REG_Cms ControllerName	Write to this register the type of CMS controller to be changed. Pay attention- for this action you will need to have a CMS license.
DENALI_CDN_AXI_REG_EnableCms Controller	Enable or disbale the CMS controller value specified in the CMS controller name register.
DENALI_CDN_AXI_REG_GenCache AwareBursts	When True (Default), this will cause the bursts generated by the master to consider the cache state and set the value for AxSNOOP accordingly. When False, this will allow generating all values for the AxSNOOP regadless of the current cache state.
DENALI_CDN_AXI_REG_UserChoose NextCacheState	For ACE FULL Active master agent only. This register enables the user to override the cache line state in the scoop of a cache memory update callback. If a value is written to this register inside the WriteCbF() function it will be used as the next cache line state.
DENALI_CDN_AXI_REG_IsBounded CacheModel	For ACE FULL Active master agent only. Enable or disbale the cache model to be bounded by size which may require some lines evict before a new line can be allocated. Default value - FALSE, which means the cache model has an infinite size.
DENALI_CDN_AXI_REG_Bounded CacheModelSize	For ACE FULL Active master agent only. When 'is_ bounded_cache_model' is TRUE, determine the size of the cache model.
DENALI_CDN_AXI_REG_Bounded CacheModelNWay	For ACE FULL Active master agent only. When 'is_ bounded_cache_model' is TRUE and bounded_cache_ model_size is set, this sets the cache model as an N-

Register	Description
	ways cache. The Value of N must be a factor of two (2,4,8,16,etc.)
DENALI_CDN_AXI_REG_CanStartShareableExclusiveStore	For ACE FULL shareable exclusive transaction, used to ask if an exclusive store transaction can start to a cache line address when writing to this register, the aligned cache line address divided by cache line size is given when reading from the register, if returns 0 - shouldn't send transaction if returns 1 - can send transaction
DENALI_CDN_AXI_REG_InitialMemoryValue	For Active slaves agent only. Controls the value returned from unwritten memory locations. If the register has not been written to, return a random byte value. If the register contains data, return that data instead.
DENALI_CDN_AXI_REG_CacheApiStateForAddress	Used by cache API methods. For ACE FULL, this register controls which cache lines will be returned when reading the address_low and address_high registers. All cache lines that will be returned in cache_api_address_low/high will be in this cache state.
DENALI_CDN_AXI_REG_CacheApiTagForAddress	Used by cache API methods. For ACE FULL, this register controls which cache lines will be returned when reading the address_low and address_high registers. When not set to MAX_INT, all cache lines that will be returned in cache_api_address_low/high will be from this specific tag in the cache.
DENALI_CDN_AXI_REG_CacheApiAddressLow	Used by cache API methods. For ACE FULL, reading this register will return the low half of a cache line address that meets the critetias set by the "cache_api_..._for_address" registers.
DENALI_CDN_AXI_REG_CacheApiAddressHigh	Used by cache API methods. For ACE FULL, reading this register will return the higher half of a cache line address that meets the critetias set by the "cache_api_..._for_address" registers.
DENALI_CDN_AXI_REG_CacheApiListSize	Used by cache API methods. For ACE FULL, this register controls how many items should be returned by the API. This register is also used to return how many items were found by the query (in times when not enought items were found that meets the query).
DENALI_CDN_AXI_REG_CacheApiMaxTag	Used by cache API methods. For ACE FULL, reading this register will return the cache tag which is the most full.

Register	Description
DENALI_CDN_AXI_REG_CacheApiReset	Used by cache API methods. For ACE FULL, this register resets all the other "cache_api_..." registers. Should be used before issuing a new request by the cache API.
DENALI_CDN_AXI_REG_EnableTracker	Enable/disable the tracker Disabled by default.
DENALI_CDN_AXI_REG_DisableWriteAddrChannel	For Active agents only. Disable the driving of AW protocol channel. When set to FASLE (Default), The write address channel behave according to the test sequence and data items. When set to TRUE, force the deassetion of AWVALID (for active master) or AWREADY (for active slave) signal.
DENALI_CDN_AXI_REG_DisableReadAddrChannel	For Active agents only. Disable the driving of AR protocol channel. When set to FASLE (Default), The read address channel behave according to the test sequence and data items. When set to TRUE, force the deassetion of ARVALID (for active master) or ARREADY (for active slave) signal.
DENALI_CDN_AXI_REG_DisableWriteDataChannel	For Active agents only. Disable the driving of W protocol channel. When set to FASLE (Default), The write data channel behave according to the test sequence and data items. When set to TRUE, force the deassetion of WVALID (for active master) or WREADY (for active slave) signal.
DENALI_CDN_AXI_REG_DisableReadDataChannel	For Active agents only. Disable the driving of R protocol channel. When set to FASLE (Default), The read data channel behave according to the test sequence and data items. When set to TRUE, force the deassetion of RVALID (for active slave) or RREADY (for active master) signal.
DENALI_CDN_AXI_REG_DisableWriteRespChannel	For Active agents only. Disable the driving of B protocol channel. When set to FASLE (Default), The write response channel behave according to the test sequence and data items. When set to TRUE, force the deassetion of BVALID (for active slave) or BREADY (for active master) signal.
DENALI_CDN_AXI_REG_DisableSnoopAddrChannel	For Active agents only. Disable the driving of AC protocol channel. When set to FASLE (Default), The snoop address channel behave according to the test sequence and data items. When set to TRUE, force

Register	Description
	the deassetion of ACVALID (for active slave) or ACREADY (for active master) signal.
DENALI_CDN_AXI_REG_Disable SnoopRespChannel	For Active agents only. Disable the driving of CR protocol channel. When set to FASLE (Default), The snoop response channel behave according to the test sequence and data items. When set to TRUE, force the deassetion of CRVALID (for active master) or CRREADY (for active slave) signal.
DENALI_CDN_AXI_REG_Disable SnoopDataChannel	For Active agents only. Disable the driving of CD protocol channel. When set to FASLE (Default), The snoop data channel behave according to the test sequence and data items. When set to TRUE, force the deassetion of CDVALID (for active master) or CDREADY (for active slave) signal.
DENALI_CDN_AXI_REG_Awready ValueAfterReset	Set the default value of AWREADY after reset. Default value - TRUE.
DENALI_CDN_AXI_REG_Wready ValueAfterReset	Set the default value of WREADY after reset. Default value - TRUE.
DENALI_CDN_AXI_REG_Bready ValueAfterReset	Set the default value of BREADY after reset. Default value - TRUE.
DENALI_CDN_AXI_REG_Arready ValueAfterReset	Set the default value of ARREADY after reset. Default value - TRUE.
DENALI_CDN_AXI_REG_Rready ValueAfterReset	Set the default value of RREADY after reset. Default value - TRUE.
DENALI_CDN_AXI_REG_Acready ValueAfterReset	Set the default value of ACREADY after reset. Default value - TRUE.
DENALI_CDN_AXI_REG_Crready ValueAfterReset	Set the default value of CRREADY after reset. Default value - TRUE.
DENALI_CDN_AXI_REG_Cdready ValueAfterReset	Set the default value of CDREADY after reset. Default value - TRUE.
DENALI_CDN_AXI_REG_ReadData Width	Read data bus width in bits. Default: VR_AXI_MAX_DATA_WIDTH. (Available also in AVIP mode)
DENALI_CDN_AXI_REG_WriteData Width	Write data bus width in bits. Default: VR_AXI_MAX_DATA_WIDTH. (Available also in AVIP mode)
DENALI_CDN_AXI_REG_AddrWidth	Address bus width in bits. Default: VR_AXI_MAX_ADDR_WIDTH.
DENALI_CDN_AXI_REG_IdWidth	ID tag width in bits. Default: VR_AXI_MAX_ID_WIDTH.

Register	Description
DENALI_CDN_AXI_REG_WriteDataInterleavingDepth	The maximum allowed number of different write transfers with different IDs on the write data bus. Note: This field is only relevant for AXI3. In AXI4 (or higher), where there is no WID signal, this field is always 1.
DENALI_CDN_AXI_REG_EarlyResponse	Indicates early_response support EARLY_REPONSE_NOT_SUPPORTED: default value, write response will be sent after the last write transfer EARLY_RESPONSE_SUPPORTED: write response is sent regardless of the write transfers
DENALI_CDN_AXI_REG_SnoopDataWidth	Snoop data bus width in bits. Default: VR_AXI_MAX_SNOOP_DATA_WIDTH.
DENALI_CDN_AXI_REG_HasTrRecording	When this field is true, the agent record transactions
DENALI_CDN_AXI_REG_DoSignalCheckOnlyWhenValid	Whether signal checking be done only when valid signal high or at all times. default: FALSE
DENALI_CDN_AXI_REG_DoBasicDeadlockCheck	Enable the deadlock check, i.e. count how many cycles have passed without AWVALID and ARVALID signals changing their state in Write Address and Read Address channels correspondingly. default: FALSE
DENALI_CDN_AXI_REG_NoChangesInAddressChannelsLimit	If do_basic_deadlock_check is TRUE, this entry determines the limit for the deadlock control counter. default: 50
DENALI_CDN_AXI_REG_ConsiderAlignmentInDataSignalChecks	Take into consideration the address and size alignment when running validity check on the transfer's DATA signal (WDATA/RDATA). Default : FALSE
DENALI_CDN_AXI_REG_WriteResponseTimeout	The max number of cycles for the ACTIVE Master and monitors to wait for Slave device write response phase. The number of cycles are calculated from the beginning of the address phase (AWVALID) until the beginning of the write response phase (BVALID) for the same transaction. The timeout is being checked only after the write data phase has ended. Default value : zero, meaning no timeout.
DENALI_CDN_AXI_REG_ReadResponseTimeout	The max number of cycles for the ACTIVE Master and monitors to wait for Slave device read data phase to end. The number of cycles are calculated from the beginning of the address phase (ARVALID) until the transmission of last read transfer (RLAST) for the same transaction. Default value : zero, meaning no timeout.

Register	Description
DENALI_CDN_AXI_REG_Update CacheForErrorBurstResponse	Setting this field to true will update the master cache (allocation/ eviction) even if the burst was not ended ok according to the method 'is_ended_ok()'. The master will update the cache regardless of the transfers responses (rresp,bresp,crresp). Pertains to AMBA4 ACE Specification. Default value - TRUE
DENALI_CDN_AXI_REG_UsingSnoop Filter	Adjust the agent's behavior - whether it is connected to a snoop filter or not. Pertains to AMBA4 ACE Specification. Default value - TRUE
DENALI_CDN_AXI_REG_Auto CompleteDvmSync	Master: Setting this field to true will cause the master to generate DVM COMPLETE read transaction for each snoop DVM message of type SYNC it receives. Slave: Setting this field to true will cause the slave to generate DVM COMPLETE snoop transaction for each DVM message of type SYNC it receives.
DENALI_CDN_AXI_REG_DvmSync CompletionTimeout	Setting this field determines the maximum allowed number of clock cycles before a DVM complete transactions is expected to be sent after a DVM sync transaction is received. (Default: 500 clock cycles)
DENALI_CDN_AXI_REG_MaxWrite BurstsBehavior	Determines master behavior when the number of transmitted WRITE bursts reaches 'write_data_ interleaving_depth'. CONTINUE_TO_SEND : Continue to send new bursts address-phases only. STOP_WRITE : Stop sending WRITE bursts. STOP_ ALL : Stop sending both READ and WRITE bursts.
DENALI_CDN_AXI_REG_Read IssuingCapability	This is the maximum number of outstanding read transactions that the master can send
DENALI_CDN_AXI_REG_Write IssuingCapability	This is the maximum number of outstanding write transactions that the master can send
DENALI_CDN_AXI_REG_StallAddr AfterMaxReadIssuingCapability	TRUE: When the number of READ pending bursts equals 'read_issuing_capability' - stall the read address channel (ARVALID low). FALSE:(DEFAULT) Can continue to issue new read bursts address-phases.
DENALI_CDN_AXI_REG_Stall AddrAfterMaxWriteIssuing Capability	TRUE: When the number of WRITE pending bursts equals 'write_issuing_capability' - stall the write address channel (AWVALID low). FALSE: (DEFAULT) Can continue to issue new write bursts address-phases.
DENALI_CDN_AXI_REG_Consider WstrbInBadSignalChecks	Take into consideration the strobe value when running validity check on the WDATA signal. Default : FALSE

Register	Description
DENALI_CDN_AXI_REG_StallAddrWithActiveBurstsId	TRUE: stall the transmission of a new transaction's address phase (READ or WRITE) if its id is the same as the id of any active transaction on the bus. FALSE: (DEFAULT) Do not stall a transaction based on its id.
DENALI_CDN_AXI_REG_UsingSpeculativeFetch	speculative fetch is a state when the master generate read bursts Although the data is in the cache. Pertains to AMBA4 ACE Specification.
DENALI_CDN_AXI_REG_AllLegalStates	According to the Spec, there are legal states for sending new read/write snoop commands TRUE - sending new read/write snoop commands for every legal state in the cache FALSE - sending new read/write snoop commands only for expected states in the cache. Pertains to AMBA4 ACE Specification.
DENALI_CDN_AXI_REG_UsingCacheRandomization	internal randomizing the cache. Default: FALSE
DENALI_CDN_AXI_REG_OrderingAlgorithm	The type of ordering algorithm to use for WRITE transfers.
DENALI_CDN_AXI_REG_UseMemory	When set: In Device mode: Slave will use the sparse memory model for storying/retrieving data. In Monitor mode: Slave will use the sparse memory model for DUT shadowing and data consistency checking. When Unset: In Device mode: Slave will always randomize new READ data even if this location was previously accessed. In Monitor mode: Slave will not write new data to the memory and will not check READ response consistency.
DENALI_CDN_AXI_REG_ReadDataReorderingDepth	Determines the maximum allowed number of different read transfers with different IDs on the read data bus.
DENALI_CDN_AXI_REG_ReadAcceptanceCapability	Determines the maximum number of different read bursts accepted (arvalid+arready high) on the read address channel.
DENALI_CDN_AXI_REG_WriteAcceptanceCapability	Determines the maximum number of different write bursts accepted (awvalid+awready high) on the write address channel.
DENALI_CDN_AXI_REG_StallAddrAfterMaxWriteDepth	TRUE: When the number of WRITE pending bursts equals 'write_data_interleaving_depth' - stall the write address channel (AWREADY low). FALSE: Can continue to accept new bursts address-phases.
DENALI_CDN_AXI_REG_StallAddrAfterMaxReadDepth	TRUE: When the number of READ pending bursts equals 'read_data_reordering_depth' - stall the read

Register	Description
	address channel (ARREADY low). FALSE: Can continue to accept new bursts address-phases.
DENALI_CDN_AXI_REG_Stall AddrAfterMaxReadAcceptance Capability	TRUE: When the number of READ pending bursts equals 'read_acceptance_capability' - stall the read address channel (ARREADY low). FALSE: (DEFAULT) Can continue to accept new bursts address-phases.
DENALI_CDN_AXI_REG_Stall AddrAfterMaxWriteAcceptance Capability	TRUE: When the number of WRITE pending bursts equals 'write_acceptance_capability' - stall the write address channel (AWREADY low). FALSE: (DEFAULT) Can continue to accept new bursts address-phases.
DENALI_CDN_AXI_REG_Update ExclusiveStateAtLastTr	This flag indicates whether exclusive status is determined when the last transfer is received or when write resp is sent (Default: FALSE)
DENALI_CDN_AXI_REG_StopMonEx AddrOnUnsuccessfulWr	This flag indicates whether the slave should stop monitoring an exclusive address after an unsuccessfully write attempt. When TRUE - slave will respond with OKAY to an EXCLUSIVE WRITE whenever there was a WRITE burst to an overlapping address (no matter what was the response to that write burst) When FALSE - slave will responsd with OKAY to an EXCLUSIVE WRITE whenever there is a WRITE burst to an overlapping address that was successful (OKAY) and with EXOKAY response after an unsuccessful write burst (SLVERR) (Default: FALSE)
DENALI_CDN_AXI_REG_Disable MemoryUpdateOnWriteBurst	Whether to prevent the ACTIVE slave bfm or the PASSIVE slave monitor from updating the memory.
DENALI_CDN_AXI_REG_Snoop IssuingCapability	This is the maximum number of active snoop transactions that the slave interface can send. Pertains to AMBA4 ACE Specification.
DENALI_CDN_AXI_REG_Read OrderingAlgorithm	The type of ordering algorithm to use for READ transfer-responses.
DENALI_CDN_AXI_REG_WriteResp OrderingAlgorithm	The type of ordering algorithm to use for WRITE burst-responses.
DENALI_CDN_AXI_REG_GenAuto Resp	By default Slave agent auto-generates responses to received bursts. If this behavior is disabled, the user would be responsible for manually creating Slave responses to incoming bursts.

Register	Description
DENALI_CDN_AXI_REG_Calculate DynamicDelay	When calculate_dynamic_delay is TRUE, the delay for RVALID will be calculated relative to the end of the burst address phase When set to FALSE, the RVALID delay is calculated relative to the first cycle the transfer can be put on the bus
DENALI_CDN_AXI_REG_Slverr WriteCauseInvalidValue	When set, invalidates slave memory content for the write burst which results in SLVERR response.
DENALI_CDN_AXI_REG_No ConsistencyCheckIfOverlap	When set, disables the memory consistency checks for read bursts when a write burst to the same address is in progress.
DENALI_CDN_AXI_REG_Update MemoryForErrorSnoopResponse	Setting this field to true will update the slave's memory on snoops even if the snoop response was error. The slave will update the memory regardless of the snoop responses (crresp). Pertains to AMBA4 ACE Specification. Default value - FALSE
DENALI_CDN_AXI_REG_AvipBatch Mode	Only for accelerated VIP (AVIP). Set the AVIP to Batching (TRUE) or Reactive (FALSE) mode. Default value - FALSE
DENALI_CDN_AXI_REG_AvipWait Clks	Only for accelerated VIP (AVIP). Start a timer in HW to count N clocks.
DENALI_CDN_AXI_REG_AvipFlush Buffers	Only for accelerated VIP (AVIP) mode. Flush AVIP buffers. (relevant in Batching mode)
DENALI_CDN_AXI_REG_AvipCan Send	Only for accelerated VIP (AVIP) mode. Query AVIP status. Return TRUE if AVIP can send a transaction.
DENALI_CDN_AXI_REG_AvipOkTo Send	Only for accelerated VIP mode (AVIP) mode. AVIP notify user that it can send a transaction.
DENALI_CDN_AXI_REG_AvipNo PendingTrans	Only for accelerated VIP (AVIP) mode.
DENALI_CDN_AXI_REG_Avip ControlKind	Only for accelerated VIP (AVIP) mode. Set an AVIP control command. Must be followed with another register access and pass the control ID and VAL parameters.
DENALI_CDN_AXI_REG_Avip ControlId	Only for accelerated VIP (AVIP) mode. Set an AVIP control ID.
DENALI_CDN_AXI_REG_Avip ControlVal	Only for accelerated VIP (AVIP) mode. Set an AVIP control value.
DENALI_CDN_AXI_REG_Verbosity	

Chapter 6. CDN_AXI VIP Model Operation

6.1. Model Instances

CDN_AXI VIP saves configuration information in memory spaces associated with each instance. When the model loads in the simulator, the CDN_AXI VIP model creates several internal memories for each CDN_AXI VIP instance. These internal memories hold the model, configuration, and registers. For details, see *Section 5.3, "Registers"*

6.1.1. Instantiating the Models in the Testbench

CDN_AXI VIP has a simple interface to access model, model memory space, model control registers, transactions, and so on. Each of these objects has an instance ID associated with it. Before issuing the transactions or setting up callbacks, write code to instantiate these models in your testbench.

Note

The syntactic patterns for the most used methods relating to accessing these models are readily available at `$DENALI/example/cdn_axi/denaliAxiUserInstance.sv` for AXI.

The following kinds of memory models are associated with each instance:

- Registers -- Used to handle the instance register space.

Refer to the following example to initiate all three memory models in SystemVerilog:

```
// A denaliCdn_axiInstance should be instantiated per agent
// You should extend it and create your own, to implement callbacks
class cdn_axiInstance extends denaliCdn_axiInstance;

      denaliMemInstance regInst ; // Handle to the register-space
      denaliMemInstance memory ; // Handle to main memory

      function new(string instName);
          super.new(instName);
          regInst = new( { instName, "(registers)" } );
          memory = new( { instName, "(memory)" } );
      endfunction

      ...
endclass
```

6.1.2. Model Control Registers and Storage Memory

The CDN_AXI VIP includes several memories, including port-specific model control register arrays and the storage memory. You can access these by reading, writing, and setting up callbacks, as described below.

6.1.2.1. Accessing Control Registers

You can access control registers using the `regWrite()` and `readReg()` functions.

6.1.2.1.1. regWrite()

Refer to the following example to write the control registers:

```
   virtual function void regWrite( denaliCdn_axiRegNumT addr, reg [31:0] data );
       denaliMemTransaction trans = new();
       trans.Address = addr ;
       trans.Data = new[4];
       trans.Data[0] = data[31:24];
       trans.Data[1] = data[23:16];
       trans.Data[2] = data[15:08];
       trans.Data[3] = data[07:00];
       void'( regInst.write( trans ) );
   endfunction
```

Refer to the following example to change the value of the `DENALI_CDN_AXI_REG_WriteIssuingCapability` register:

```
   initial
   begin
     activeMaster.regWrite(DENALI_CDN_AXI_REG_WriteIssuingCapability, 10 );
```

After this call, the write issuing-capability is changed to 10. By default, the value of the write-issuing capability is read from the SOMA file.

6.1.2.1.2. readReg()

This function has one parameter - register address.

The following shows how to read from a control register:

```
   virtual function integer readReg( denaliCdn_axiRegNumT addr);
       denaliMemTransaction trans = new();
       trans.Address = addr ;
       void'( regInst.read( trans ) );
       readReg[31:24] = trans.Data[0];
       readReg[23:16] = trans.Data[1];
       readReg[15:08] = trans.Data[2];
       readReg[07:00] = trans.Data[3];
   endfunction
```

To use the register addresses in SystemVerilog, your testbench module must include the `denaliCdn_axiTypes.svh` file supplied with the CDN_AXI VIP model. You can find this file under `$DENALI/ddvapi/sv`.

6.1.2.2. Backdoor Reads and Writes to Slave Memory

You can perform backdoor reads and writes to slave memory space using `memRead()` and `memWrite()` functions.

To perform backdoor reads and writes, refer to the following examples:

```
class cdn_axiInstance extends denaliCdn_axiInstance;

   ...
```

```
    virtual function reg [7:0] memRead( reg [63:0] addr);
        denaliMemTransaction trans = new();
        trans.Address = addr ;
        void'( memory.read( trans ) );
        memRead[7:0] = trans.Data[0];
    endfunction

    virtual function void memWrite( denaliCdn_axiRegNumT addr, reg [7:0] data );
        denaliMemTransaction trans = new();
        trans.Address = addr ;
        trans.Data = new[1];
        trans.Data[0] = data[7:0];
        void'( memory.write( trans ) );
    endfunction

endclass
```

To perform backdoor writing to slave main memory at address 0x1000, use the following:

```
    Initial
    begin
      activeslave.memWrite('h1000,8b’10101010);
      #1000;
```

6.1.2.2.1. Pre-loading Memory from an External Data File

An agent memory (for example, as Active Slave memory) can be pre-loaded from an external data file, using the `mmload` command. You cannot load undefined values into an agent memory.

To pre-load memory from an external data file:

1. Add the following code to your testbench:

```
integer success;

initial begin

     // Pre-load Active Slave memory from an external file using $mmload
     $display("Pre-loading Active Slave's memory from data file");
     success = $mmload("pstest.axiActiveSlaveDevice(memory)","../mr.data");

end
```

> **Note**
>
> The data file location is relative to the directory where your test is run. You can use environment variables in the path, if needed.

2. Create the data file and populate it with data/address mapping. Both address and data are expressed in Hex radix. For example:

```
0/  A;
1/  B;
2/  C;
3/  D;
4/  E;
5/  F;
6/  6;
7/  7;
```

The memory data is loaded on a byte per address location basis. You can load data for the entire contiguous address space using `<start_addr>:<end_addr>/<data>;` notation. (See the *Cadence Memory Model Portfolio User Guide* for more information.)

3. You can confirm that the memory content has been updated by searching your log file for a message similar to the following:

```
*Denali* Memory contents loaded from file "../mr.data" into instance
 "pstest.axiActiveSlaveDevice(memory)".
```

Note

X (undefined) values cannot be loaded to an agent memory.

6.1.2.3. Defining Memory Segments

The register model lets you define memory regions with different domains. For example, attribute `DENALI_CDN_AXI_REGION_INNER` means that this memory segment is of INNER domain.

To define memory segments, refer to the following example:

```
function void mapMemorySegment(reg [63:0] fromAddress, reg [63:0] toAddress, denaliCdn_axiDomainT
domain);
       regWrite( DENALI_CDN_AXI_REG_RegionAddressFromHigh,fromAddress[63:32]);
       regWrite( DENALI_CDN_AXI_REG_RegionAddressFromLow, fromAddress[31:0]);
       regWrite( DENALI_CDN_AXI_REG_RegionAddressToHigh ,toAddress[63:32]);
       regWrite( DENALI_CDN_AXI_REG_RegionAddressToLow ,toAddress[31:0]);
       regWrite( DENALI_CDN_AXI_REG_RegionType ,domain);
   endfunction
```

Note

For AXI, no domain parameter is needed because the domain will always be `DENALI_CDN_AXI_DOMAIN_NON_SHAREABLE` .

The following example shows how to define a memory segment in the range [0..0xFFF]:

```
  axiActiveMaster.mapMemorySegment(64'h0,64'hFFF,DENALI_CDN_AXI_DOMAIN_NON_SHAREABLE);
```

Note

The regWrite function is defined in the example Section 6.1.2, *"Model Control Registers and Storage Memory"*.

6.2. Instantiation / Elaboration

To instantiate any CDN_AXI VIP model, you must first configure a SOMA file and generate a Verilog instantiation interface using the PureView user interface. For detailed information on the configuration parameters of the CDN_AXI VIP SOMA files, see Section 5.2, *"SOMA"*. To create an HDL instantiation interface, see Section 3.2, *"Creating a SOMA File and an HDL Instantiation Interface for CDN_AXI VIP"*

The following example uses the SystemVerilog code to show how the model gets instantiated:

```
cdn_axi_master master
(
ACLK, ARESETN, AWVALID, AWADDR, AWLEN, AWSIZE, AWBURST,
AWLOCK, AWCACHE, AWPROT, AWID, AWREADY,
AWUSER, WVALID, WLAST, WDATA, WSTRB, WID, WREADY, WUSER,
BVALID, BRESP, BID, BREADY, BUSER, ARVALID, ARADDR, ARLEN,
ARSIZE, ARBURST, ARLOCK, ARCACHE, ARPROT, ARID, ARREADY,
 ARUSER, RVALID, RLAST, RDATA, RRESP,
RID, RREADY, RUSER
);
defparam master.interface_soma = "/home/cdn_axi/example/master.soma";
```

The module `cdn_axi_master` is defined in the Verilog instantiation interface, which is automatically generated from a SOMA file. For details, refer to Section 5.2, *"SOMA"*. The parameter `interface_soma` must point to the SOMA file path.

6.3. Transactions

After completing the setup, you can start generating transactions and test various verification scenarios. This section gives a brief overview of how the CDN_AXI VIP represents transactions, followed by a detailed description of the interface you can use to generate transactions.

6.3.1. Basic AXI Transaction Information

6.3.1.1. Frequently Used AXI Transaction Fields

Use the following files for test writing and debugging:

- `The <VIPCAT_Inst_Dir>/tools/denali/ddvapi/sv/denaliCdn_axi.sv` file contains definitions of `denaliCdn_axiTransaction` (data item, including all transaction fields) and `denaliCdn_axiInstance` (VIP proxy) classes, as well as large number of constraint blocks used in SV generation. These blocks can be disabled if default behavior is not desired. See Section 9.4, *"Built-in SV Constraints"*.
- `The <VIPCAT_Inst_Dir>/tools/denali/ddvapi/sv/denaliCdn_axiTypes.svh` file contains all enumeration types defined in the VIP.
- `The <VIPCAT_Inst_Dir>/tools/denali/ddvapi/sv/denaliCdn_axiErrTable.svh` file contains all error related enumeration types defined in the VIP.

  The following fields are the frequently used fields in test writing, callback writing, and debugging. You can see the full list in Section 6.3.5, *"Transaction Types and Fields"*.

  Direction - Read or Write

  The following enums are defined in `the denaliCdn_axiTypes.svh` file:

```
typedef enum {

DENALI_CDN_AXI_DIRECTION_UNSET = 0,
```

```
DENALI_CDN_AXI_DIRECTION_READ = 1,
DENALI_CDN_AXI_DIRECTION_WRITE = 2

} denaliCdn_axiDirectionT;
```

Length - The number of transfers in the burst (equals to `ALEN` +) Unsigned integer

Size - Transfer size. Corresponds to `ARSIZE/AWSIZE`.

The following enums are defined in `the denaliCdn_axiTypes.svh` file:

```
typedef enum {
  DENALI_CDN_AXI_TRANSFERSIZE_UNSET = 0,
  DENALI_CDN_AXI_TRANSFERSIZE_BYTE = 1,
  DENALI_CDN_AXI_TRANSFERSIZE_HALFWORD = 2,
  DENALI_CDN_AXI_TRANSFERSIZE_WORD = 3,
  DENALI_CDN_AXI_TRANSFERSIZE_TWO_WORDS = 4,
  DENALI_CDN_AXI_TRANSFERSIZE_FOUR_WORDS = 5,
  DENALI_CDN_AXI_TRANSFERSIZE_EIGHT_WORDS = 6,
  DENALI_CDN_AXI_TRANSFERSIZE_SIXTEEN_WORDS = 7,
  DENALI_CDN_AXI_TRANSFERSIZE_K_BITS = 8
} denaliCdn_axiTransferSizeT;
```

BurstMaxSize. Similar to *Size* but defines the *upper Size* value that can be used during SV transaction randomization.

IdTag - Burst ID tag. Corresponds to `AWID/ARID/WID/RID/BID`. Unsigned integer.

Data - Each entry in the array is one byte of the data used in transfers. For write transfers, the array includes write data. For read transfers, when the transaction is finished, the array includes all the read data received from transfers.

Kind - `FIXED`, `INCR` or `WRAP`. Corresponds to `ARBURST/AWBURST`.

The following enums are defined in `denaliCdn_axiTypes.svh` file:

```
typedef enum {
  DENALI_CDN_AXI_BURSTKIND_UNSET = 0,
  DENALI_CDN_AXI_BURSTKIND_FIXED = 1,
  DENALI_CDN_AXI_BURSTKIND_INCR = 2,
  DENALI_CDN_AXI_BURSTKIND_WRAP = 3,
  DENALI_CDN_AXI_BURSTKIND_RESERVED = 4
} denaliCdn_axiBurstKindT;
```

StartAddress - First address in the burst. Unsigned integer.

Access - `NORMAL, EXCLUSIVE` or `LOCKED` access. Corresponds to `ARLOCK/AWLOCK`.

The following enums are defined in the `denaliCdn_axiTypes.svh` file:

```
typedef enum {
  DENALI_CDN_AXI_ACCESS_UNSET = 0,
  DENALI_CDN_AXI_ACCESS_NORMAL = 1,
  DENALI_CDN_AXI_ACCESS_EXCLUSIVE = 2,
  DENALI_CDN_AXI_ACCESS_LOCKED = 3,
  DENALI_CDN_AXI_ACCESS_RESERVED = 4
} denaliCdn_axiAccessT;
```

Privileged - NORMAL or PRIVILEGED access. Corresponds to ARPROT [0] / AWPROT [0].

The following enums are defined in the denaliCdn_axiTypes.svh file:

```
typedef enum {
  DENALI_CDN_AXI_PRIVILEGEDMODE_UNSET = 0,
  DENALI_CDN_AXI_PRIVILEGEDMODE_NORMAL = 1,
  DENALI_CDN_AXI_PRIVILEGEDMODE_PRIVILEGED = 2
} denaliCdn_axiPrivilegedModeT;
```

Secure - SECURE or NON-SECURE access. Corresponds to ARPROT [1] / AWPROT [1].

The following enums are defined in the denaliCdn_axiTypes.svh file:

```
typedef enum {
  DENALI_CDN_AXI_SECUREMODE_UNSET = 0,
  DENALI_CDN_AXI_SECUREMODE_SECURE = 1,
  DENALI_CDN_AXI_SECUREMODE_NONSECURE = 2
} denaliCdn_axiSecureModeT;
```

DataInstr - Data or Instruction access. Corresponds to ARPROT [2] / AWPROT [2].

The following enums are defined in the denaliCdn_axiTypes.svh file:

```
typedef enum {
  DENALI_CDN_AXI_FETCHKIND_UNSET = 0,
  DENALI_CDN_AXI_FETCHKIND_DATA = 1,
  DENALI_CDN_AXI_FETCHKIND_INSTRUCTION = 2
} denaliCdn_axiFetchKindT;
```

Bufferable - NON_BUFFERABLE or BUFFERABLE. Corresponds to ARCACHE[0] / AWCACHE[0].

The following enums are defined in the denaliCdn_axiTypes.svh file:

```
typedef enum {
  DENALI_CDN_AXI_BUFFERMODE_UNSET = 0,
  DENALI_CDN_AXI_BUFFERMODE_NON_BUFFERABLE = 1,
  DENALI_CDN_AXI_BUFFERMODE_BUFFERABLE = 2
} denaliCdn_axiBufferModeT;
```

Cacheable - NON_CACHEABLE or CACHEABLE. Corresponds to ARCACHE[1] / AWCACHE[1].

The following enums are defined in the denaliCdn_axiTypes.svh file:

```
typedef enum {
  DENALI_CDN_AXI_CACHEMODE_UNSET = 0,
  DENALI_CDN_AXI_CACHEMODE_NON_CACHEABLE = 1,
  DENALI_CDN_AXI_CACHEMODE_CACHEABLE = 2
} denaliCdn_axiCacheModeT;
```

ReadAllocate - NO_READ_ALLOCATE or READ_ALLOCATE. Corresponds to ARCACHE[2] / AWCACHE[2].

The following enums are defined in the `denaliCdn_axiTypes.svh` file:

```
typedef enum {
  DENALI_CDN_AXI_READALLOCATE_UNSET = 0,
  DENALI_CDN_AXI_READALLOCATE_NO_READ_ALLOCATE = 1,
  DENALI_CDN_AXI_READALLOCATE_READ_ALLOCATE = 2
} denaliCdn_axiReadAllocateT;
```

WriteAllocate - `NO_WRITE_ALLOCATE` or `WRITE_ALLOCATE`. Corresponds to `ARCACHE[3] / AWCACHE[3]`.

The following enums are defined in the `denaliCdn_axiTypes.svh` file:

```
typedef enum {
  DENALI_CDN_AXI_WRITEALLOCATE_UNSET = 0,
  DENALI_CDN_AXI_WRITEALLOCATE_NO_WRITE_ALLOCATE = 1,
  DENALI_CDN_AXI_WRITEALLOCATE_WRITE_ALLOCATE = 2
} denaliCdn_axiWriteAllocateT;
```

6.3.1.2. Using the ModelGeneration Field

The `ModelGeneration` field is a flag that indicates to the internal core to complete certain transaction fields that have not been generated in the SV environment. That is, the fields that remained `UNSET` in SV.

Some examples are as follows:

- If you assign only `Direction` and `StartAddress` fields in SV code and do not randomize your transaction in SV, then only `Direction` and `StartAddress` will be set to certain values and rest of the fields, even the `ModelGeneration`, will remain `UNSET`. When this transaction arrives to the *e* core, all the remaining transaction fields besides `Direction` and `StartAddress` will get generated properly in *e*, according to built-in *e* constraints.

- If you randomize your transaction in SV (for example, by using `randomize()` or `` `uvm_do() ``), then all transaction fields will be assigned in SV according to built-in SV constraints included in `the denaliCdn_axi.sv` file. Then the `ModelGeneration` field will automatically be set to *0* (according to `fast_di_const` constraint block in the same file), indicating to the internal core that there is no need for internal generation because everything was already generated in the SV code.

- If you randomize your transaction in SV but intercept it later and assign any of its fields to `UNSET`, you must set the `ModelGeneration` field to *1* in order to notify the internal core that it must internally generate `UNSET` fields, so that all transaction fields get generated.

Example 6.1. ModelGeneration Usage

In this example, we randomize the `StartAddress` and `Domain` fields, and assign them to the burst.

Create a class that holds the variables to randomize.

```
class my_vars;
    rand reg [63:0] StartAddress;
    rand denaliCdn_axiDomainT Domain;
   …
endclass
```

In your test, instantiate the class, randomize the class, and assign it*:*

```
my_vars random_vars;
   …
   initial
   begin
       random_vars = new();
       …
       void'(random_vars.randomize() with {
           StartAddress inside {['h1000:'h1FFF]};
           Domain == DENALI_CDN_AXI_DOMAIN_INNER;s]
       });
        …
       masterBurst.StartAddress = random_vars.StartAddress;
       masterBurst.Domain = random_vars.Domain;
       masterBurst.ModelGeneration = 1;
       …
       status = aceMaster.transAdd(masterBurst, DENALI_CDN_AXI_QUEUE_Burst);
    end
```

6.3.1.3. Using Slave Memory

- *`denaliMemInstance`* class is used for accessing register, memory, spaces. It is instantiated inside active or passive agents.
- **Slave Memory:** by default, the entire memory content is randomized.
- Memory can be overridden using `backdoor` access.

1. Memory handling functions, which include

```
virtual function reg [7:0] memRead( reg [63:0] addr); <new line>virtual function void
 memWrite( denaliCdn_axiRegNumT addr, reg [7:0] data );
```

6.3.1.4. Defining Transactions for Various Flavors of AXI Specs

The same `denaliCdn_axiTransaction` data item class is used for `AXI3` and `AXI4`.

Follow the instructions given below to define a transaction:

- For either `AXI3` or `AXI4` mode, `CDN_ACE` should NOT be defined.

Define `Spec Version`, `Spec Subtype`, and `Spec Interface` fields for your transactions in addition to the **SOMA** settings.

- In AXI mode, these fields are set to standard AXI3 spec by default.

Cadence recommends that you define `myBaseTransaction`, which is derived from `denaliCdn_axiTransaction` and uses `myBaseTransaction` in the sequences.

Following are the examples for `AXI4` transactions:

Example 6.2. Defining transactions for AXI4

```
class myAxiBaseTransaction extends denaliCdn_axiTransaction;
    `uvm_object_utils(myAxiBaseTransaction)

     function new();
            super.new();
              // Set transaction to AXI4
              this.SpecVer = DENALI_CDN_AXI_SPECVERSION_AMBA4;
              this.SpecSubtype = DENALI_CDN_AXI_SPECSUBTYPE_BASE;
              this.SpecInterface = DENALI_CDN_AXI_SPECINTERFACE_FULL;
    endfunction


    constraint user_dut_information {
        BurstMaxSize == DENALI_CDN_AXI_TRANSFERSIZE_EIGHT_WORDS;
    }
endclass : myAxiBaseTransaction
```

6.3.1.5. Using Clarification on the IsDiscarded Field

The `IsDiscarded` field is a flag that the VIP model uses to indicate that a transaction added to the transmission queue (using `transAdd` method) was discarded. The VIP model discards transactions when it is impossible to create a legal protocol transaction from the specified information.

The following are the examples of such cases:

- A combination of randomization and invalid constraints added by the user.
- A combination of randomization and original constraint blocks turned off.
- No use of randomization, and insufficient information set by user. For more information, see Section 6.3.1.2, *"Using the ModelGeneration Field"* for violation of the protocol rules.

When a transaction is discarded:

1. A short message will appear in the log, depending on the verbosity selected. For more information on verbosity, search for `DENALI_CDN_AXI_REG_Verbosity`.
2. An error callback will get triggered.
3. To debug the failure, enable the error callback, print and examine the transaction received in the callback

6.3.2. Transaction Generation

To send a transaction, instantiate the module `denaliCdn_axiTransaction`. You can assign module fields and call task `transAdd()` located in the `denaliCdn_axiInstance` module in the `$DENALI/ddvapi/sv/denaliCdn_axi.sv` file.

Note

Cadence recommends not to use the `denaliCdn_axiTransaction` module as it is. You must extend it and add DUT-specific constraints.

Following is an example:

```
class myTransaction extends denaliCdn_axiTransaction;
    constraint user_dut_information {
        BurstMaxSize == DENALI_CDN_AXI_TRANSFERSIZE_WORD; // For data bus size of 32 bits
        IdTag < (1 << 8); // For id signals of 8 bits
    ...
    }
endclass
```

The following example shows how to send a read transaction where few of the fields are constrained:

```
import DenaliSvCdn_axi::*;
import DenaliSvMem::*;

module testbench;

...
cdn_axiInstance activeMaster;
myTransaction burst;

initial
   begin

     activeMaster = new ("testbench.master");
     burst = new();
     #1000;
     burst.Direction  = DENALI_CDN_AXI_DIRECTION_READ;
     burst.StartAddress = 'h100;
     burst.Size  = DENALI_CDN_AXI_TRANSFERSIZE_WORD;
     Burst.Length = 4;

     status = activeMaster.transAdd(burst, DENALI_CDN_AXI_QUEUE_Burst);
```

You can also randomize all fields in SystemVerilog using the `randomize()` task, as shown below:

```
initial
   begin

     activeMaster = new ("testbench.master");
     burst = new();
     #1000;
     assert( burst.randomize() with {
           Direction       = DENALI_CDN_AXI_DIRECTION_READ;
           StartAddress = 'h100;
           Size = DENALI_CDN_AXI_TRANSFERSIZE_WORD;
           Length       = 4;
     });
     status = activeMaster.transAdd(burst, DENALI_CDN_AXI_QUEUE_Burst);
```

6.3.3. Transaction Flow

The master and slave devices maintain a scoreboard of outstanding transactions with various associated data about their timing, ordering, and so on.

You can modify various transactions' data during callback `BeforeSend` but not `Started` and `Ended` callbacks.

6.3.3.1. Master – Sending a Write

The basic transaction flow step includes:

1. The master gets a high-level `Write` transaction from the user queue.
2. During the callback `BeforeSend`, you can modify the transaction's data.
3. The `Write` transaction is split into two transactions: `WriteAddress` and `WriteData`.

   The `WriteData` transaction is not very useful; it exists solely for consistency with the `ReadAddress` / `ReadData` pair.
4. Callback `BeforeSendAddress`.
5. The `WriteAddress` transaction is transmitted through the wires to the slave.
6. Callback `BeforeSendTransfer` for each `WriteTransfer` transaction.
7. The `WriteData` transactions are transmitted through the wires to the slave.
8. Callback `EndedTransfer` for each `WriteTransfer` transaction.

6.3.3.2. Slave – Receiving Write Transactions and Sending WriteResponse

The basic transaction flow step includes:

1. The slave gets the `WriteAddress` and all of the `WriteData` transactions from its input wires.
2. Callback started for `WriteAddress` and each of the `WriteData` transactions.
3. Callback ended `WriteAddress` and each of the `WriteData` transactions.
4. Slave creates and sends the `WriteResponse` transaction.
5. Callback `BeforeSendResponse`.
6. The `WriteResponse` transaction is transmitted through the wires to the master.
7. Callback `EndedResponse` for the `WriteResponse` transaction.

6.3.3.3. Master – Receiving WriteResponse

The basic transaction flow step includes:

1. The master gets `WriteResponse` from its input wires.
2. Callback `StartedResponse` for the `WriteResponse` transaction.
3. Callback `EndedResponse` for the `WriteResponse` transaction.
4. The write transaction process is completed.

6.3.3.4. Master – Sending ReadAddress

The basic transaction flow step includes:

1. The master gets a high-level `Read` transaction from the user queue.
2. Callback `BeforeSend`. During this callback, you can modify the transaction's control data.
3. Callback `BeforeSendAddress`.
4. The `ReadAddress` transaction is transmitted through the wires to the slave.

6.3.3.5. Slave – Receiving ReadAddress and Sending ReadData

The basic transaction flow step includes:

1. The slave gets `ReadAddress` from its input wires.
2. Callback `StartedAddress`.
3. The slave creates a high-level `ReadData` transaction.
4. Callback `BeforeSendTransfer` for the `ReadData` transactions.
5. Callback `BeforeSendTransfer` for each of the `ReadTransfer` transactions.
6. All `ReadTransfer` transactions are transmitted through the wires to the master.
7. Callback `EndedTransfer` for each of the `ReadTransfer` transactions.

6.3.3.6. Master – Receiving ReadData

The basic transaction flow step includes:

1. The master gets all `ReadTransfer` transactions from its input wires.
2. Callback `StartedTransfer` for each of the `ReadTransfer` transactions.
3. Callback `EndedTransfer` for each of the `ReadTransfer` transactions.

4. The `Read` transaction process is done.

6.3.4. Controlling Delays on AXI Channels

This section explains how to constrain each of the delays for the different transaction types using specific fields of `denaliCdn_axiTransaction` class.

The delays can be set at two points:

- Before inserting a transaction to the relevant queue using `transAdd()` (either by doing randomize or set the fields procedurally).
- Settable in the relevant callback using `transSet()` for the relevant transaction types as listed below.

6.3.4.1. Controlling VALID Signals

6.3.4.1.1. Controlling AWVALID and ARVALID Master Delay

The master delay on the address channels is calculated as the number of cycles, from the cycle the BFM first attempts to send the burst until asserting `AWVALID` or `ARVALID` and putting the burst's control information on the address channel.

To control the master delay on the address channels:

- Constrain the `TransmitDelay` field of `denaliCdn_axiTransaction`.
- Disable a built-in `TransmitDelay_const` constraint block, which constrains `TransmitDelay` to zero.

Example 6.3. Controlling AxVALID Signal Delay

```
masterBurst = new();
masterBurst.TransmitDelay_const.constraint_mode(0);
masterBurst.TransmitDelay = 10;
status = activeMaster.transAdd(masterBurst, DENALI_CDN_AXI_QUEUE_Burst);
```

Control through callback: This delay can also be set using `transSet()` inside the callback `DENALI_CDN_AXI_CB_BeforeSend` with transaction types `DENALI_CDN_AXI_TR_Write / DENALI_CDN_AXI_TR_Read`

6.3.4.1.2. Controlling WVALID Master Delay

The delay for write transfers on the data channel is the number of cycles from the first cycle that the write transfer is ready and chosen until the assertion of `WVALID`.

To control the master delay on the write data channel, constrain the `TransfersChannelDelay` field of `denaliCdn_axiTransaction`.

Example 6.4. Controlling WVALID signal delay

```
masterBurst = new();
masterBurst.Direction = DENALI_CDN_AXI_DIRECTION_WRITE;
masterBurst.Length = 4;
masterBurst.TransfersChannelDelay = new [4];
for (int i=0; i< masterBurst.TransfersChannelDelay.size(); i++) begin
masterBurst.TransfersChannelDelay[i] = i;
end
status = activeMaster.transAdd(masterBurst, DENALI_CDN_AXI_QUEUE_Burst);
```

Control through callback: This delay can also be set using `transSet()` inside the callback `DENALI_CDN_AXI_CB_BeforeSend` with transaction type `DENALI_CDN_AXI_TR_Write`.

6.3.4.1.3. Controlling RVALID Slave Delay

The delay for read transfers on the data channel is the number of cycles from the first cycle that the read transfer response is ready and chosen until the assertion of `RVALID`.

To control the master delay on the write data channel, constrain the `TransfersChannelDelay` field of `denaliCdn_axiTransaction`.

Example 6.5. Controlling RVALID Signal Delay

```
slaveResp = new();
slaveResp.Length = 4;
slaveResp.Direction = DENALI_CDN_AXI_DIRECTION_READ;
slaveResp.TransfersChannelDelay = new [4];
for (int i=0; i< slaveResp.TransfersChannelDelay.size(); i++) begin
slaveResp.TransfersChannelDelay[i] = i;
end
status = activeSlave.transAdd(slaveResp, DENALI_CDN_AXI_QUEUE_Burst);
```

Control through callback: This delay can also be set using `transSet()` inside the callback `DENALI_CDN_AXI_CB_BeforeSendResponse` with transaction type `DENALI_CDN_AXI_TR_ReadData`.

6.3.4.1.4. Controlling BVALID Slave Delay

The slave delay on the write response channel is calculated as the number of cycles from the first cycle that the burst can be sent on the write response channel until the assertion of `BVALID`.

To control the `BVALID` delay on the write response channel, constrain the `ChannelDelay` fields of `denaliCdn_axiTransaction`.

Example 6.6. Controlling BVALID Signal Delay

```
slaveResp = new();
slaveResp. Direction = DENALI_CDN_AXI_DIRECTION_WRITE;
slaveResp.ChannelDelay = 10;
status = activeSlave.transAdd(slaveResp, DENALI_CDN_AXI_QUEUE_Burst);
```

Control through callback: This delay can also be set using `transSet()` inside the callback `DENALI_CDN_AXI_CB_BeforeSendResponse` with transaction type `DENALI_CDN_AXI_TR_WriteResponse.`

6.3.4.2. Controlling READY Signals

For each ready signal in each channel, there are two control fields:

- Control
- Delay

You can control the behavior of the ready signals using those control fields. The control field is of type `denaliCdn_axiReadyControlT`.

The following table describes the possible values:

Table 6.1. Values of denaliCdn_axiReadyControlT

Value	Description
DENALI_CDN_AXI_READYCONTROL_DELAYED_ASSERTION	The ready signal will be deasserted for the number of cycles specified by the corresponding delay field.
DENALI_CDN_AXI_READYCONTROL_OSCILLATING	The ready signal will be changed from low to high. The corresponding delay field represents the time of half a cycle. This option lets you de-assert the ready signal when the valid signal was not asserted.
DENALI_CDN_AXI_READYCONTROL_STALL_UNTIL_VALID	The ready signal will be de-asserted until the valid signal is asserted.
DENALI_CDN_AXI_READYCONTROL_STALL_UNTIL_VALID_AND_DELAY	The ready signal will be stalled until the channels valid signal will be asserted, after which it will wait according to the delay.
DENALI_CDN_AXI_READYCONTROL_STALL_UNTIL_VALID_AND_OTHER_PHASE_STARTED	Relevant only for the write address and write channels. The ready signal will be de-asserted until the valid signal is asserted and the burst was started on the other channel. (That is, the burst was started on the write channel if you control the AWREADY signal or the write address phase was started if you control the WREADYsignal.)
DENALI_CDN_AXI_READYCONTROL_STALL_UNTIL_OTHER_DIRECTION_CHANNEL_VALID	Relevant only for the write address and write channels. The ready signal will be de-asserted until the valid signal is asserted in both channels (that is, AWVALID is asserted if you control the WREADY signal or WVALID if you control the AWREADYsignal).
DENALI_CDN_AXI_READYCONTROL_STALL_UNTIL_BOTH_CHANNELS_VALID	Relevant only for the write address and write channels. The ready signal will be de-asserted

Value	Description
	until the valid signal is asserted in both channels (that is, both in the write address and the write channels).
DENALI_CDN_AXI_READYCONTROL_STALL_UNTIL_VALID_AND_OTHER_PHASE_STARTED_WITH_DELAY	The ready signal will be stalled until the channel's valid signal is asserted only if the other phase (data/address) of the burst has started. The delay will also be considered.

When the ready controller expects another channel valid signal; there is always a possibility that these signals will not be raised.

Note: The following ready controller has to be used with caution because it can cause a deadlock in certain scenarios:

`DENALI_CDN_AXI_READYCONTROL_STALL_UNTIL_OTHER_DIRECTION_CHANNEL_VALID`

`DENALI_CDN_AXI_READYCONTROL_STALL_UNTIL_BOTH_CHANNELS_VALID`

`DENALI_CDN_AXI_READYCONTROL_STALL_UNTIL_VALID_AND_OTHER_PHASE_STARTED`

`DENALI_CDN_AXI_READYCONTROL_STALL_UNTIL_VALID_AND_OTHER_PHASE_`

`STARTED_WITH_DELAY`

For example: If `wready` controller is `STALL_UNTIL_OTHER_DIRECTION_CHANNEL_VALID` but the burst sends data before the address, then the `wready` signal will stall until the address phase starts, but the address phase will not start until some data transfers are sent. This effectively causes a deadlock.

The two transaction fields, `EnforceReadyController` and `MaxDelayReadyController`, are added to overcome the possibility of a deadlock when using these controllers.

The following table describes the control and delay fields for each channel:

Table 6.2. Ready Signal Control Fields

Channel	Control Field	Delay Field
write address	AreadyControl	AddressDelay
read address	AreadyControl	AddressDelay
write data	WreadyControl	TransfersChannelDelay
read data	RreadyControl	TransfersChannelDelay
write response	BreadyControl	BreadyDelay

Note

The control and delay fields always affect the next item. So, constraining a burst will influence only the next burst. The same applies for transfers. This also means that only the second burst in a test and the second transfer in each burst can effectively be constrained.

6.3.4.2.1. Controlling AWREADY and ARREADY Slave Delay

The slave delay on the address channel is calculated as the number of cycles to keep `AWREADY` or `ARREADY` low after the completion of the burst address phase.

To control the slave delay on the address channel, constrain the `AddressDelay` and `AreadyControl` fields of `denaliCdn_axiTransaction`.

Example 6.7. Controlling AxREADY signal delay

```
slaveResp = new();
slaveResp.AddressDelay = 5;
slaveResp.AreadyControl = DENALI_CDN_AXI_READYCONTROL_DELAYED_ASSERTION;
status = activeSlave.transAdd(slaveResp, DENALI_CDN_AXI_QUEUE_Burst);
```

Control through callback: This delay can also be set using `transSet()` inside the callback `DENALI_CDN_AXI_CB_BeforeSendResponse` with transaction types `DENALI_CDN_AXI_TR_ReadData / DENALI_CDN_AXI_TR_WriteResponse.`

6.3.4.2.2. Controlling RREADY Master Delay

The delay for read transfers on the data channel is the number of cycles from the end of the transfer until the assertion of `RREADY`.

To control the master delay on the read data channels, constrain the `TransfersChannelDelay` and `RreadyControl` fields of `denaliCdn_axiTransaction.`

Example 6.8. Controlling RREADY signal delay:

```
masterBurst = new();
masterBurst.Direction = DENALI_CDN_AXI_DIRECTION_READ;
masterBurst.Length = 4;
masterBurst.RreadyControl = DENALI_CDN_AXI_READYCONTROL_DELAYED_ASSERTION;
masterBurst.TransfersChannelDelay = new [4];
for (int i=0; i< masterBurst.TransfersChannelDelay.size(); i++) begin
masterBurst.TransfersChannelDelay[i] = i;
end
status = activeMaster.transAdd(masterBurst, DENALI_CDN_AXI_QUEUE_Burst);
```

Control through callback: This delay can also be set using `transSet()` inside the callback `DENALI_CDN_AXI_CB_BeforeSend` with transaction type `DENALI_CDN_AXI_TR_Read`.

6.3.4.2.3. Controlling WREADY Slave Delay

The delay for write transfers on the data channel is the number of cycles from the end of the transfer until the assertion of `WREADY`.

To control the slave delay on the write data channel, constrain the `WreadyControl` and `TransfersChannelDelay` fields of `denaliCdn_axiTransaction.`

Example 6.9. Controlling WREADY Signal Delay

```
slaveResp = new();
slaveResp.Length = 4;
slaveResp.Direction = DENALI_CDN_AXI_DIRECTION_WRITE;
slaveResp.WreadyControl = DENALI_CDN_AXI_READYCONTROL_OSCILLATING;
slaveResp.TransfersChannelDelay = new [4];
for (int i=0; i< slaveResp.TransfersChannelDelay.size(); i++) begin
slaveResp.TransfersChannelDelay[i] = 2;
end
status = activeSlave.transAdd(slaveResp, DENALI_CDN_AXI_QUEUE_Burst);
```

Control through callback: This delay can also be set using `transSet()` inside the callback `DENALI_CDN_AXI_CB_BeforeSendResponse` with transaction type `DENALI_CDN_AXI_TR_WriteResponse.`

6.3.4.2.4. Controlling BREADY Master Delay

The master delay on the write response channel is calculated as the number of cycles from the end of the write burst (assertion of both `BREADY` and `BVALID`) until another assertion of `BREADY`.

To control the master delay on the write response channel, constrain the `BreadyDelay` and `BreadyControl` fields of `denaliCdn_axiTransaction`.

Example 6.10. Controlling BREADY Signal Delay

```
masterBurst = new();
masterBurst.Direction = DENALI_CDN_AXI_DIRECTION_WRITE;
masterBurst.BreadyDelay = 5;
masterBurst.BreadyControl = DENALI_CDN_AXI_READYCONTROL_DELAYED_ASSERTION;
status = activeMaster.transAdd(slaveResp, DENALI_CDN_AXI_QUEUE_Burst);
```

Control through callback: This delay can also be set using `transSet()` inside the callback `DENALI_CDN_AXI_CB_BeforeSend` with transaction type `DENALI_CDN_AXI_TR_Write`.

6.3.4.3. Setting all Master Delays to Zero

You can remove any delays for the *AXI ACTIVE MASTER* by setting the following fields:

- `TransfersChannelDelay` – Set all items to zero
- `TransmitDelay` – Set to zero
- `ChannelDelay` – Set to zero
- `BreadyDelay` – Set to zero
- `BreadyControl` – Set to `DENALI_CDN_AXI_READYCONTROL_DELAYED_ASSERTION`
- `RreadyControl` - Set to `DENALI_CDN_AXI_READYCONTROL_DELAYED_ASSERTION`

If you are using SystemVerilog randomization, you can add constraints on these fields. If not, you can set it procedurally for each transaction or set them in the `BeforeSend` callback. (Refer to Chapter 7, *CDN_AXI VIP Callbacks*.)

Note

Some SOMA configuration parameters can also cause delays on master transmissions.

These are:

- `stall_addr_after_max_write_depth`
- `stall_addr_after_max_read_depth`
- `stall_addr_after_max_write_acceptance_capability`
- `stall_addr_after_max_read_acceptance_capability`

All the above parameters should be set to *FALSE*.

6.3.5. Transaction Types and Fields

Table 6.3. Transaction Types (DENALI_CDN_AXI_TR_)

Transaction Type	Description
UNSET	Undefined type - this value should not be used
Write	High-level write transaction, is automatically converted to and from WriteAddress and WriteData
WriteAddress	Low-level write address transaction, transported via write address channel
WriteData	High-level transaction, is automatically converted to and from one or many WriteTransfer transactions
WriteTransfer	Low-level write transfer transaction, transported via write data channel
WriteResponse	Low-level write response transaction, transported via write response channel
WriteAcknowledge	Low-level write acknowledge transaction, transported via write response channel
Read	High-level read transaction, is automatically converted to and from ReadAddress and ReadData
ReadAddress	Low-level read address transaction, transported via read address channel
ReadData	High-level transaction, is automatically converted to and from one or many ReadTransfer transactions

Transaction Type	Description
ReadTransfer	Low-level read transfer transaction, transported via read data channel
ReadAcknowledge	Low-level read acknowledge transaction, transported via write response channel
Snoop	AMBA4 ACE only high-level snoop transaction, is automatically converted to and from SnoopAddress and SnoopData
SnoopAddress	AMBA4 ACE only low-level snoop address transaction, transported via snoop address channel
SnoopData	AMBA4 ACE only high-level transaction, is automatically converted to and from one or many SnoopTransfer transactions
SnoopTransfer	AMBA4 ACE only low-level snoop transfer transaction, transported via snoop data channel
SnoopResponse	AMBA4 ACE only low-level snoop response transaction, transported via snoop response channel
InternalMemory	Low-level internal transaction used to back access the memory, should not be used by default

Table 6.4. Fields Common to All Transaction Types

Name	R/W	Type	Size (Bits)	Description
Type	RW	denaliCdn_axiTrTypeT	32	Transaction type.
PortNum	RO	unsigned	32	Port/Queue number
Callback	RO	denaliCdn_axiCbPointT	32	ID of the callback currently associated with the transaction. Useful in the callback processing user code.
ErrorString	RO	string	*	Character string describing the transaction errors for the debug
ErrorId	RO	denaliCdn_axiErrorTypeT	32	This is the Error ID associated with error callbacks
ErrorInfo	RO	denaliCdn_axiErrorInfoT	32	Transaction error severity
UserData	RW	unsigned	32	Placeholder for the user-specific information
Delay	RW	unsigned	32	Transaction delay (in clock cycles). Please do not use this field
BreadyDelay	RW	unsigned	32	The number of cycles to keep BREADY low after a WRITE burst

Name	R/W	Type	Size (Bits)	Description
				has ended (i.e. BVALID and BREADY were high).
BreadyControl	RW	denaliCdn_axiReadyControlT	32	BREADY signal control.
Data	RW	array	*	Data list for write bursts only. Each entry is the data for the transfer in the same index in the transfers list. (Available also in AVIP mode)
TransmitDelay	RW	unsigned	32	The delay in cycles from the time the BFM first attempts to send the burst until asserting AVALID and putting the burst's control information on the address channel.
WriteAddressOffset	RW	unsigned	32	(Default: 0) Determines the transfer to whom the delay of the address will refer. The field is relevant only for write bursts In case of 0 - the adress delay will be calculate first. The delay of the first transfer would not be calculated before the beginning of the write address phase. In case greater than 0 - the address delay will be calculated from the beginning of the write transfer (as indicated by the field).
IdTag	RW	unsigned	20	Burst ID tag (Available also in AVIP mode)
Alen	RW	unsigned	8	(NC) The value driven to the ALEN signal
Size	RW	denaliCdn_axiTransferSizeT	32	Transfer size (Available also in AVIP mode)
Kind	RW	denaliCdn_axiBurstKindT	32	FIXED, INCR or WRAP (Available also in AVIP mode)
Access	RW	denaliCdn_axiAccessT	32	NORMAL, EXCLUSIVE or LOCKED (Available also in AVIP mode)
Direction	RW	denaliCdn_axiDirectionT	32	READ or WRITE (Available also in AVIP mode)
Bufferable	RW	denaliCdn_axiBufferModeT	32	NON_BUFFERABLE or BUFFERABLE (Available also in AVIP mode)

Name	R/W	Type	Size (Bits)	Description
Cacheable	RW	denaliCdn_axi CacheModeT	32	NON_CACHEABLE or CACHEABLE -- AMBA3 (Available also in AVIP mode)
ReadAllocate	RW	denaliCdn_axi ReadAllocateT	32	NO_READ_ALLOCATE or READ_ ALLOCATE (Available also in AVIP mode)
WriteAllocate	RW	denaliCdn_axi WriteAllocateT	32	NO_WRITE_ALLOCATE or WRITE_ ALLOCATE (Available also in AVIP mode)
Region	RW	unsigned	5	REGION -- AMBA4 (Available also in AVIP mode)
Qos	RW	unsigned	4	QOS -- AMBA4 (Available also in AVIP mode)
Auser	RW	array	*	User address signal
Buser	RW	array	*	User response signal
Length	RW	unsigned	32	The number of transfers in the burst (equals to ALEN + 1) (Available also in AVIP mode)
IsDiscarded	RW	boolean	32	Was the burst discarded
TransfersChannel Delay	RW	array	*	Array of all burst's transfers channel delay
TransfersResp	RW	array	*	Array of all burst's transfers responses (Available also in AVIP mode)
ModelGeneration	RW	boolean	32	if TRUE - use the core model's constraints for generation. if FALSE - use only SV constraints for generation.
DiType	RW	denaliCdn_axiDi TypeT	32	Symbolic Di type (like DENALI_ CDN_AXI_DI_MasterBurst).
UserDataArray	RW	array	*	Optional array of user spesific data in case the UserData field is insufficient.
StrobeArray	RW	array	*	Array of all WRITE burst's transfers strobe values (Available also in AVIP mode)
UserSignalArray	RW	array	*	Array of data channel user signals (RUSER / WUSER) visible and controlable in the burst level. In case the r/wuser signals width is up to 32 bits

Name	R/W	Type	Size (Bits)	Description
				- each entry in the array reflects one transfer's user signal. Multiple array entries reflects a transfer's user signal wider than 32 bits.
SpecVer	RW	denaliCdn_axiSpecVersionT	32	The version of the spec (Available also in AVIP mode)
SpecSubtype	RW	denaliCdn_axiSpecSubtypeT	32	The spec subtype (Available also in AVIP mode)
SpecInterface	RW	denaliCdn_axiSpecInterfaceT	32	The spec interface (FULL or LITE) (Available also in AVIP mode)
StartAddress	RW	unsigned	64	The first address in this burst (Available also in AVIP mode)
Privileged	RW	denaliCdn_axiPrivilegedModeT	32	NORMAL or PRIVILEGED (Available also in AVIP mode)
Secure	RW	denaliCdn_axiSecureModeT	32	SECURE or NONSECURE (Available also in AVIP mode)
DataInstr	RW	denaliCdn_axiFetchKindT	32	DATA or INSTRUCTION (Available also in AVIP mode)
HighestAddress	RW	unsigned	64	(NC) The highest address byte in the burst
LowestAddress	RW	unsigned	64	(NC) The lowest address byte in the burst
AddrPhaseStartTime	RW	unsigned	64	Address phase start time.
RespPhaseStartTime	RW	unsigned	64	Response phase start time.
DataPhaseStartTime	RW	unsigned	64	Data phase start time.
AddrPhaseEndTime	RW	unsigned	64	Address phase End Time
RespPhaseEndTime	RW	unsigned	64	Response phase End Time
DataPhaseEndTime	RW	unsigned	64	Data phase End Time
Duration	RW	unsigned	64	Burst duration
IsAddrPhaseStarted	RW	boolean	32	Is the address phase started.
IsRespPhaseStarted	RW	boolean	32	Is the Response phase started.
IsDataPhaseStarted	RW	boolean	32	Is the data phase started.
IsAddrPhaseFinished	RW	boolean	32	Is the Address phase ended.
IsRespPhaseFinished	RW	boolean	32	Is the Response phase ended.
IsDataPhaseFinished	RW	boolean	32	Is the Data phase ended.

Name	R/W	Type	Size (Bits)	Description
IsEnded	RW	boolean	32	Is the burst ended.
PsInternalId	RW	unsigned	32	this is a model internal field, this field shouldn't be accessed. this field will not be available in future versions.
AllowRespChange	RW	boolean	32	If TRUE allow the BFM to change the response for an EXCLUSIVE burst if it is necessary (the generated response is no longer valid).
Resp	RW	denaliCdn_axiResponseT	32	The write response. Used only for WRITE bursts. (Available also in AVIP mode)
RespPassDirty	RW	denaliCdn_axiSnoopRespPassDirtyT	32	Only for AMBA4 read response This value will tranculate to all the read transfer
RespIsShared	RW	denaliCdn_axiSnoopRespIsSharedT	32	Only for AMBA4 read response This value will tranculate to all the read transfer
ChannelDelay	RW	unsigned	32	The delay until we drive the response for a write burst. Used only for WRITE bursts. Note that if the early write response feature is enabled then this delay is counted from the end of the address phase
AddressDelay	RW	unsigned	32	Number of cycles to keep AREADY low after the completion of this burst address phase.
AreadyControl	RW	denaliCdn_axiReadyControlT	32	[AW/AR]READY signal control.
EnforceReadyController	RW	boolean	32	Indicates if the ready controller behavior will be enforced, even at the risk of a deadlock. The default is TRUE.
MaxDelayReadyController	RW	unsigned	32	If the field 'enforce_ready_controller' is FALSE, this field indicates how long from the initiation of the ready controller, it's ok to ignore the controller and raise the ready signal.
PhysicalData	RW	array	*	The data transmitted in this transfer as written to the bus.

Name	R/W	Type	Size (Bits)	Description
Strobe	RW	unsigned	128	The strobe for the data
User	RW	array	*	The user signal (Available also in AVIP mode)
Position	RW	denaliCdn_axi TransferPositionT	32	The position of the transfer in the burst : FIRST, MIDDLE or LAST.
Address	RW	unsigned	64	(NC) The address of this transfer. This field need to be generated since it is used in the generation of 'strobe'.
Last	RW	boolean	32	Whether this transfer is the last in the burst
IsStarted	RW	boolean	32	Has the transfer started.
PhysicalDataXMask	RW	array	*	Bit mask for the data monitored on the bus, where '1' indicates 'X' value
PhysicalDataZMask	RW	array	*	Bit mask for the data monitored on the bus, where '1' indicates 'Z' value
RreadyControl	RW	denaliCdn_axi ReadyControlT	32	RREADY signal control.
IgnoreConstraints	RW	boolean	32	The IgnoreConstraints field pertains to behavior of Slave agents for Read transactions. When set to TRUE, Read data, used in the Slave response, will be taken from Slave's sparse memory When set to FALSE, Read data will be determined by response transaction, that can be specified by the user, and placed into Slave's response. If no data was specified in the slave response, the Read Data will still be taken from Slave's sparse memory. Default: TRUE
WreadyControl	RW	denaliCdn_axi ReadyControlT	32	WREADY signal control.
TOTAL	RW	unsigned	32	

6.3.6. Controlling Transaction Fields

The following section specifies the controllability of the `denaliCdn_axiTransaction` fields:

- **Externally constrainable:** You can set the value of this field before sending a transaction, using `transAdd()`

- **Internally calculated:** The value of this field is calculated internally by the module and its value cannot be set by using `transAdd()`
- **Controllable through callback:** You can set the value of this field in the relevant callbacks using `transSet()`

The following sections describe the transaction fields and their related parameters.

6.3.6.1. Field Name:*Size*

Description	Determines transfer size
Controlled Signal(s)	ARSIZE/AWSIZE
Types / Pre-Defined Enumeration Types	DENALI_CDN_AXI_TRANSFERSIZE_UNSET Field unset DENALI_CDN_AXI_TRANSFERSIZE_BYTE 1 byte = 8 bits DENALI_CDN_AXI_TRANSFERSIZE_HALFWORD 2 bytes = 16 bits DENALI_CDN_AXI_TRANSFERSIZE_WORD 4 bytes = 32 bits DENALI_CDN_AXI_TRANSFERSIZE_TWO_WORDS 8 bytes = 64 bits DENALI_CDN_AXI_TRANSFERSIZE_FOUR_WORDS 16 bytes = 128 bits DENALI_CDN_AXI_TRANSFERSIZE_EIGHT_WORDS 32 bytes = 256 bits DENALI_CDN_AXI_TRANSFERSIZE_SIXTEEN_WORDS 64 bytes = 512 bits DENALI_CDN_AXI_TRANSFERSIZE_K_BITS 128 bytes = 1024 bits
Relevant Agent(s)	Master
Relevant Data Items	Write burst Read burst
Externally Constrainable	Yes
Internally Calculated	No

Controllable via Callbacks	**BeforeSend**
	• Relevant Transaction Types
	  • DENALI_CDN_AXI_TR_Write (awsize)
	  • DENALI_CDN_AXI_TR_Read (arsize)
	• Relevant Agent: Master

	BeforeSendAddress
	• Relevant Transaction Types
	  • DENALI_CDN_AXI_TR_WriteAddress (awsize)
	  • DENALI_CDN_AXI_TR_ReadAddress (arsize)
	• Relevant Agent: Master

6.3.6.2. Field Name:*Kind*

Externally constrainable

Controllable through callbacks:

- **BeforeSend**
  - Transaction types:
    - DENALI_CDN_AXI_TR_Write (awburst)
    - DENALI_CDN_AXI_TR_Read (arburst)
  - **Agent:** Master
- **BeforeSendAddress**
  - Transaction types:
    - DENALI_CDN_AXI_TR_WriteAddress (awburst)
    - DENALI_CDN_AXI_TR_ReadAddress (arburst)
  - **Agent:** Master

6.3.6.3. Field Name: *Direction*

Externally constrainable

Controllable through callbacks:

6.3.6.4. Field Name: *Type*

Externally constrainable

Controllable through callbacks:

6.3.6.5. Field Name: *Data*

Externally constrainable

Controllable through callbacks:

- **BeforeSend, BeforeSendAddress**
  - Transaction types:
    - DENALI_CDN_AXI_TR_Write
  - **Agent:** Master
- **BeforeSendResponse**
  - Transaction types:
    - DENALI_CDN_AXI_TR_ReadData
  - **Agent:** Slave
- **BeforeSendTransfer**
  - Transaction types:
    - DENALI_CDN_AXI_TR_ReadTransfer
  - **Agent:** Slave

6.3.6.6. Field Name: *StartAddress*

Externally constrainable

Controllable through callbacks:

- **BeforeSend**
  - Transaction types:
    - DENALI_CDN_AXI_TR_Write (awaddr)
    - DENALI_CDN_AXI_TR_Read (araddr)
  - **Agent:** Master
- **BeforeSendAddress**
  - Transaction types:

- DENALI_CDN_AXI_TR_WriteAddress (awaddr)
- DENALI_CDN_AXI_TR_ReadAddress (araddr)

- **Agent:** Master

6.3.6.7. Field Name: *Length*

Externally constrainable

Controllable through callbacks:

- **BeforeSend**
  - Transaction types:
    - DENALI_CDN_AXI_TR_Write (awlen)
    - DENALI_CDN_AXI_TR_Read (arlen)
  - **Agent:** Master
- **BeforeSendAddress**
  - Transaction types:
    - DENALI_CDN_AXI_TR_WriteAddress (awlen)
    - DENALI_CDN_AXI_TR_ReadAddress (arlen)
  - **Agent:** Master

6.3.6.8. Field Name: *Last*

Internally calculated

Controllable through callbacks:

- **BeforeSendTransfer**
  - Transaction types:
    - DENALI_CDN_AXI_TR_WriteTransfer (wlast)
  - **Agent:** Master
- **BeforeSendTransfer**

- Transaction types:
  - DENALI_CDN_AXI_TR_ReadTransfer (rlast)
- **Agent:** Slave

6.3.6.9. Field Name: *Strobe*

Internally calculated

Controllable through callbacks:

- **BeforeSendTransfer**
  - Transaction types:
    - DENALI_CDN_AXI_TR_WriteTransfer (wstrb)
  - **Agent:** Master

6.3.6.10. Field Name: *StrobeArray*

Internally calculated

Controllable through callbacks:

- **BeforeSend**
  - Transaction types:
    - DENALI_CDN_AXI_TR_Write
  - **Agent:** Master

6.3.6.11. Field Name: *ModelGeneration*

Internally constrainable

Not Controllable through callbacks:

6.3.6.12. Field Name: *Access*

Externally constrainable

Controllable through callbacks:

- **BeforeSend**
  - Transaction types:
    - DENALI_CDN_AXI_TR_Write (awlock)
    - DENALI_CDN_AXI_TR_Read (arlock)
  - **Agent:** Master
- **BeforeSendAddress**
  - Transaction types:
    - DENALI_CDN_AXI_TR_WriteAddress (awlock)
    - DENALI_CDN_AXI_TR_ReadAddress (arlock)
  - **Agent:** Master

6.3.6.13. Field Name: *Region*

Externally constrainable

Controllable through callbacks:

- **BeforeSend**
  - Transaction types:
    - DENALI_CDN_AXI_TR_Write (awregion)
    - DENALI_CDN_AXI_TR_Read (arregion)
  - **Agent:** Master
- **BeforeSendAddress**
  - Transaction types:
    - DENALI_CDN_AXI_TR_WriteAddress (awregion)
    - DENALI_CDN_AXI_TR_ReadAddress (arregion)
  - **Agent:** Master

6.3.6.14. Field Name: *IdTag*

Externally constrainable

Controllable through callbacks:

- **BeforeSend**
  - Transaction types:
    - DENALI_CDN_AXI_TR_Write (awid)
    - DENALI_CDN_AXI_TR_Read (arid)
  - **Agent:** Master
- **BeforeSendAddress**
  - Transaction types:
    - DENALI_CDN_AXI_TR_WriteAddress (awid)
    - DENALI_CDN_AXI_TR_ReadAddress (arid)
  - **Agent:** Master
- **BeforeSendTransfer (for write)**
  - Transaction types:
    - DENALI_CDN_AXI_TR_WriteTransfer (wid – only for AXI3 if supported)
  - **Agent:** Master
- **BeforeSendTransfer (for read)**
  - Transaction types:
    - DENALI_CDN_AXI_TR_ReadTransfer (rid)
  - **Agent:** Slave
- **BeforeSendResponse (for write)**
  - Transaction types:
    - DENALI_CDN_AXI_TR_WriteResponse (bid)
  - **Agent:**Slave

6.3.6.15. Field Name: *Bufferable*

Externally constrainable

Controllable through callbacks:

- **BeforeSend**
  - Transaction types:
    - DENALI_CDN_AXI_TR_Write (awcache[0])
    - DENALI_CDN_AXI_TR_Read (arcache[0])
  - **Agent:** Master
- **BeforeSendAddress**
  - Transaction types:
    - DENALI_CDN_AXI_TR_WriteAddress (awcache[0])
    - DENALI_CDN_AXI_TR_ReadAddress (arcache[0])
  - **Agent:** Master

6.3.6.16. Field Name: *Cacheable*

Externally constrainable

Controllable through callbacks:

- **BeforeSend**
  - Transaction types:
    - DENALI_CDN_AXI_TR_Write (awcache[1])
    - DENALI_CDN_AXI_TR_Read (arcache[1])
  - **Agent:** Master
- **BeforeSendAddress**
  - Transaction types:
    - DENALI_CDN_AXI_TR_WriteAddress (awcache[1])

    - DENALI_CDN_AXI_TR_ReadAddress (arcache[1])
  - **Agent:** Master

6.3.6.17. Field Name: *ReadAllocate*

Externally constrainable

Controllable through callbacks:

- **BeforeSend**
  - Transaction types:
    - DENALI_CDN_AXI_TR_Write (awcache[2])
    - DENALI_CDN_AXI_TR_Read (arcache[2])
  - **Agent:** Master
- **BeforeSendAddress**
  - Transaction types:
    - DENALI_CDN_AXI_TR_WriteAddress (awcache[2])
    - DENALI_CDN_AXI_TR_ReadAddress (arcache[2])
  - **Agent:** Master

6.3.6.18. Field Name: *WriteAllocate*

Externally constrainable

Controllable through callbacks:

- **BeforeSend**
  - Transaction types:
    - DENALI_CDN_AXI_TR_Write (awcache[3])
    - DENALI_CDN_AXI_TR_Read (arcache[3])
  - **Agent:** Master
- **BeforeSendAddress**

- Transaction types:
  - DENALI_CDN_AXI_TR_WriteAddress (awcache[3])
  - DENALI_CDN_AXI_TR_ReadAddress (arcache[3])
- **Agent:** Master

6.3.6.19. Field Name: *Qos*

Externally constrainable

Controllable through callbacks:

- **BeforeSend**
  - Transaction types:
    - DENALI_CDN_AXI_TR_Write (awqos)
    - DENALI_CDN_AXI_TR_Read (arqos)
  - **Agent:** Master
- **BeforeSendAddress**
  - Transaction types:
    - DENALI_CDN_AXI_TR_WriteAddress (awqos)
    - DENALI_CDN_AXI_TR_ReadAddress (arqos)
  - **Agent:** Master

6.3.6.20. Field Name: *Auser*

Externally constrainable

Controllable through callbacks:

- **BeforeSend**
  - Transaction types:
    - DENALI_CDN_AXI_TR_Write (awuser)
    - DENALI_CDN_AXI_TR_Read (aruser)

- **Agent:** Master

- **BeforeSendAddress**
  - Transaction types:
    - DENALI_CDN_AXI_TR_WriteAddress (awuser)
    - DENALI_CDN_AXI_TR_ReadAddress (aruser)
  - **Agent:** Master

6.3.6.21. Field Name: *Buser*

Externally constrainable

Controllable through callbacks:

- **BeforeSend**
  - Transaction types:
    - DENALI_CDN_AXI_TR_Write (buser)
  - **Agent:** Slave
- **BeforeSendAddress**
  - Transaction types:
    - DENALI_CDN_AXI_TR_WriteResponse (buser)
  - **Agent:** Slave

6.3.6.22. Field Name: *User*

Externally constrainable

Controllable through callbacks:

- **BeforeSendTransfer**
  - Transaction types:
    - DENALI_CDN_AXI_TR_WriteTransfer (wuser)
  - **Agent:** Master

- **BeforeSendTransfer**
  - Transaction types:
    - DENALI_CDN_AXI_TR_ReadTransfer (ruser)
  - **Agent:** Slave

6.3.6.23. Field Name: *Domain*

Externally constrainable

Controllable through callbacks:

- **BeforeSend**
  - Transaction types:
    - DENALI_CDN_AXI_TR_Write (awdomain)
    - DENALI_CDN_AXI_TR_Read (ardomain)
  - **Agent:** Master
- **BeforeSendAddress**
  - Transaction types:
    - DENALI_CDN_AXI_TR_WriteAddress (awdomain)
    - DENALI_CDN_AXI_TR_ReadAddress (ardomain)
  - **Agent:** Master

6.3.6.24. Field Name: *PhysicalData*

Externally constrainable

Controllable through callbacks:

- **BeforeSendTransfer**
  - Transaction types:
    - DENALI_CDN_AXI_TR_WriteTransfer (wdata)
  - **Agent:** Master

- **BeforeSendTransfer**
  - Transaction types:
    - DENALI_CDN_AXI_TR_ReadTransfer (rdata)
  - **Agent:** Slave

6.3.6.25. Field Name: *TransfersResp*

Externally constrainable

Controllable through callbacks:

- **BeforeSendResponse**
  - Transaction types:
    - DENALI_CDN_AXI_TR_ReadData (rresp)
  - **Agent:** Slave

6.3.6.26. Field Name: *Resp*

Externally constrainable

Controllable through callbacks:

- **BeforeSendResponse**
  - Transaction types:
    - DENALI_CDN_AXI_TR_WriteResponse (bresp)
  - **Agent:** Slave

6.3.6.27. Field Name: *Privileged*

Externally constrainable

Controllable through callbacks:

- **BeforeSend**
  - Transaction types:
    - DENALI_CDN_AXI_TR_Write (awprot[0])

    - DENALI_CDN_AXI_TR_Read (arprot[0])
  - **Agent:** Master
- **BeforeSendAddress**
  - Transaction types:
    - DENALI_CDN_AXI_TR_WriteAddress (awprot[0])
    - DENALI_CDN_AXI_TR_ReadAddress (arprot[0])
  - **Agent:** Master

6.3.6.28. Field Name: *Secure*

Externally constrainable

Controllable through callbacks:

- **BeforeSend**
  - Transaction types:
    - DENALI_CDN_AXI_TR_Write (awprot[1])
    - DENALI_CDN_AXI_TR_Read (arprot[1])
  - **Agent:** Master
- **BeforeSendAddress**
  - Transaction types:
    - DENALI_CDN_AXI_TR_WriteAddress (awprot[1])
    - DENALI_CDN_AXI_TR_ReadAddress (arprot[1])
  - **Agent:** Master

6.3.6.29. Field Name: *DataInstr*

Externally constrainable

Controllable through callbacks:

- **BeforeSend**
  - Transaction types:

    - DENALI_CDN_AXI_TR_Write (awprot[2])
    - DENALI_CDN_AXI_TR_Read (arprot[2])
  - **Agent:** Master
- **BeforeSendAddress**
  - Transaction types:
    - DENALI_CDN_AXI_TR_WriteAddress (awprot[2])
    - DENALI_CDN_AXI_TR_ReadAddress (arprot[2])
  - **Agent:** Master

6.3.6.30. Field Name: *EnforceReadyController*

Externally constrainable

Not Controllable through callbacks:

6.3.6.31. Field Name: *MaxDelayReadyController*

Externally constrainable

Not Controllable through callbacks:

6.3.6.32. Field Name: *BreadyDelay*

Externally constrainable

Controllable through callbacks:

- **BeforeSend**
  - Transaction types:
    - DENALI_CDN_AXI_TR_Write (bready)
  - **Agent:** Master

6.3.6.33. Field Name: *BreadyControl*

Externally constrainable

Controllable through callbacks:

- **BeforeSend**
  - Transaction types:
    - DENALI_CDN_AXI_TR_Write (bready)
  - **Agent:** Master

6.3.6.34. Field Name: *TransmitDelay*

Externally constrainable

Controllable through callbacks:

- **BeforeSend**
  - Transaction types:
    - DENALI_CDN_AXI_TR_Write (awvalid)
    - DENALI_CDN_AXI_TR_Read (arvalid)
  - **Agent:** Master

6.3.6.35. Field Name: *TransfersChannelDelay*

Externally constrainable

Controllable through callbacks:

- **BeforeSend**
  - Transaction types:
    - DENALI_CDN_AXI_TR_Read (rready)
    - DENALI_CDN_AXI_TR_Write (wvalid)
  - **Agent:** Master
- **BeforeSendResponse**
  - Transaction types:
    - DENALI_CDN_AXI_TR_WriteResponse (wready)
    - DENALI_CDN_AXI_TR_ReadData (rvalid)

- **Agent:** Slave

6.3.6.36. Field Name: *ChannelDelay*

Externally constrainable

Controllable through callbacks:

- **BeforeSendResponse**
  - Transaction types:
    - DENALI_CDN_AXI_TR_WriteResponse (bvalid)
  - **Agent:** Slave

6.3.6.37. Field Name: *AddressDelay*

Externally constrainable

Controllable through callbacks:

- **BeforeSendResponse**
  - Transaction types:
    - DENALI_CDN_AXI_TR_WriteResponse (awready)
  - **Agent:** Slave
- **BeforeSendResponse**
  - Transaction types:
    - DENALI_CDN_AXI_TR_WriteResponse (awready)
  - **Agent:** Slave

6.3.6.38. Field Name: *AreadyControl*

Externally constrainable

Controllable through callbacks:

- **BeforeSendResponse**
  - Transaction types:

        - DENALI_CDN_AXI_TR_WriteResponse (awready)
    - **Agent:** Slave
- **BeforeSendResponse**
    - Transaction types:
        - DENALI_CDN_AXI_TR_ReadData (aready)
    - **Agent:** Slave

6.3.6.39. Field Name: *RreadyControl*

Externally constrainable

Controllable through callbacks:

- **BeforeSend**
    - Transaction types:
        - DENALI_CDN_AXI_TR_Read (rready)
    - **Agent:** Master

6.3.6.40. Field Name: *WreadyControl*

Externally constrainable

Controllable through callbacks:

- **BeforeSendResponse**
    - Transaction types:
        - DENALI_CDN_AXI_TR_WriteResponse (wready)
    - **Agent:** Slave

6.3.6.41. Field Name: *IgnoreConstraints*

Internally calculated

Controllable through callbacks:

- **BeforeSendResponse**
  - Transaction types:
    - DENALI_CDN_AXI_TR_ReadData
  - **Agent:** Slave

6.3.6.42. Field Name: *WriteAddressOffset*

Externally constrainable

Controllable through callbacks:

- **BeforeSend**
  - Transaction types:
    - DENALI_CDN_AXI_TR_Write
  - **Agent:** Master

6.3.6.43. Field Name: *AllowRespChange*

Externally constrainable

Controllable through callbacks:

- **BeforeSendResponse**
  - Transaction types:
    - DENALI_CDN_AXI_TR_WriteResponse
  - **Agent:** Slave

6.3.6.44. Field Name: *Alen*

Internally calculated

NOT Controllable through callbacks

6.3.6.45. Field Name: *IsDiscarded*

Internally calculated

NOT Controllable through callbacks

6.3.6.46. Field Name: *DiType*

Internally calculated

NOT Controllable through callbacks

6.3.6.47. Field Name: *HighestAddress*

Internally calculated

NOT Controllable through callbacks

6.3.6.48. Field Name: *LowestAddress*

Internally calculated

NOT Controllable through callbacks

6.3.6.49. Field Name: *Duration*

Internally calculated

NOT Controllable through callbacks

6.3.6.50. Field Name: *IsEnded*

Internally calculated

NOT Controllable through callbacks

6.3.6.51. Field Name: *IsAddrPhaseStarted*

Internally calculated

NOT Controllable through callbacks

6.3.6.52. Field Name: *IsDataPhaseStarted*

Internally calculated

NOT Controllable through callbacks

6.3.6.53. Field Name: *IsRespPhaseStarted*

Internally calculated

NOT Controllable through callbacks

6.3.6.54. Field Name: *IsAddrPhaseFinished*

Internally calculated

Not Controllable through callbacks

6.3.6.55. Field Name: *IsRespPhaseFinished*

Internally calculated

NOT Controllable through callbacks

6.3.6.56. Field Name: *IsDataPhaseFinished*

Internally calculated

NOT Controllable through callbacks

6.3.6.57. Field Name: *AddrPhaseStartTime*

Internally calculated

NOT Controllable through callbacks

6.3.6.58. Field Name: *RespPhaseStartTime*

Internally calculated

NOT Controllable through callbacks

6.3.6.59. Field Name: *DataPhaseStartTime*

Internally calculated

NOT Controllable through callbacks

6.3.6.60. Field Name: *AddrPhaseEndTime*

Internally calculated

NOT Controllable through callbacks

6.3.6.61. Field Name: *RespPhaseEndTime*

Internally calculated

NOT Controllable through callbacks

6.3.6.62. Field Name: *DataPhaseEndTime*

Internally calculated

NOT Controllable through callbacks

6.3.6.63. Field Name: *TransferIndex*

Internally calculated

NOT Controllable through callbacks

6.3.6.64. Field Name: *Position*

Internally calculated

NOT Controllable through callbacks

6.3.6.65. Field Name: *Address*

Internally calculated

NOT Controllable through callbacks

6.3.6.66. Field Name: IsStarted

Internally calculated

NOT Controllable through callbacks

6.3.7. Auxiliary Transaction Fields

There are several other fields of `denaliCdn_axiTransaction` that are not set using `transAdd()` or `transSet()`. You must set these fields as part of the user constraints when extending `denaliCdn_axiTransaction`.

BurstMaxSize: The maximum size of a transfer in the burst; should be set to be the data bus width. The field is of type `denaliCdn_axiTransferSizeT`.

Here is an example that shows how Auxiliary transaction field is used:

```
class myTransaction extends denaliCdn_axiTransaction;

constraint user_dut_information {
BurstMaxSize == DENALI_CDN_AXI_TRANSFERSIZE_WORD;
}

endclass
```

The fields `SpecVer, SpecSubtype` and `SpecInterface` are also defined in the testbench:

```
class myTransaction extends denaliCdn_axiTransaction;

constraint user_dut_information {
BurstMaxSize == DENALI_CDN_AXI_TRANSFERSIZE_WORD;
}

endclass
```

For more examples, see Section 6.3, “*Transactions*”

Chapter 7. CDN_AXI VIP Callbacks

The callback mechanism enables you to monitor the incoming transactions. Every time some significant event happens, the CDN_AXI VIP model issues a callback so that you can see the incoming transactions and modify the transactions that are automatically generated by the model.

7.1. Callback Types

The following table shows a list of all callbacks provided by the CDN_AXI VIP pertaining to operations on transactions.

Table 7.1. Callback Types (DENALI_CDN_AXI_CB_)

Callback Type	Description
UNSET	Undefined type - this value should not be used
Error	A generic error has been detected. During this callback, Transaction field ErrorString stores character string that is a concatenation of all error messages associated with the transaction at the moment, field ErrorId holds the current error id, and field ErrorInfo holds the error severity.
AssertPass	(Not Implemented) Is activated when an assertion does not detect an error condition
ResetStarted	Callback ResetStarted is activated when the protocol ARESETn signal is de-asserted. (Available also in AVIP mode)
ResetEnded	Callback ResetEnded is activated when the protocol ARESETn signal is asserted. (Available also in AVIP mode)
RetrieveItem	Available when using ICM - Callback RetreiveItem is activated when user write to the RetriveItem register with the id of the wanted item
BeforeSend	Callback BeforeSend is activated when the PureSpec model generated a high-level transaction (read/write/snoop) but did not send it yet. During BeforeSend the user can modify the transaction.
BeforeSendAddress	Callback BeforeSendAddress is activated right before the address phase of the transaction is being send. (Before asserting AW/AR/ACVALID)
BeforeSendResponse	Callback BeforeSendResponse is activated right before the response phase of the transaction is being send. (Available also in AVIP mode)
BeforeSendTransfer	Callback BeforeSendTransfer is activated right before a data transfer is being send. (Before asserting CD/R/WVALID)

Callback Type	Description
Started	Callback Started is activated when the high-level transaction first appears on the wires. Same as StartedAddress unless when data before address. (Available also in AVIP mode)
StartedAddress	Callback StartedAddress is activated when the address phase of the transaction appears on the wires. (AW/AR/ACVALID high)
StartedResponse	Callback StartedResponse is activated when the response phase of the transaction appears on the wires. (CR/BVALID high or first RVALID high)
StartedTransfer	Callback StartedTransfer is activated when a data transfer appears on the wires. (Each CD/R/WVALID high)
Ended	Callback Ended is activated when the high-level transaction is done. For Read: after last read transfer, For Write: after write response, For Snoop: after snoop response / last snoop transfer. (Available also in AVIP mode)
EndedAddress	Callback EndedAddress is activated when the the address phase of the transaction is done. (Both VALID AND READY are high)
EndedResponse	Callback EndedResponse is activated when the the response phase of the transaction is done. (Both VALID AND READY are high)
EndedTransfer	Callback EndedTransfer is activated when a data transfer transmission is done. (Both VALID AND READY are high).
CoverageSample	An internal callback for coverage collection this callback is only for internal use.
TOTAL	This is not a real queue, but a symbolic constant useful to enumerate queue in a loop

7.2. Transaction Callbacks

You can set these callbacks at specific points in the flow of a transaction through the system. You can use the callbacks to intercept a transaction, change its characteristics, and re-insert it at the same point so that the transaction resumes its previous flow. You can also use callbacks as a means to know that certain events have occurred in the model, without the use of polling.

You can use data transaction flow callbacks to:

- Change a field in a transaction.
- Synchronize user transactions from the CDN_AXI VIP model with transaction received from the DUT.
- Implement scoreboards to enable the testbench to compare responses from the DUT with expected data.

- Inject errors into transactions after these have been generated by the CDN_AXI VIP, but before the transaction is transmitted on the physical link.

Within a test configuration, there may be many possible callback points. You must specifically enable a callback point in order to make use of it in your test cases.

7.2.1. Transaction Callback Interface

The CDN_AXI VIP transaction callback interface provides callback initialization and handling. The `denaliCdn_axiCallback` base module contains an integer variable that is changed by the model when a sensitized event occurs in the model. A change on this variable triggers a procedural block in the testbench to process the callback events.

Each pre-defined callback `DENALI_CDN_AXI_CB_<callback_name>` triggers a pre-defined function. For example:

```
virtual function int  <callback_name>CbF(ref denaliCdn_axiTransaction trans) ;
```

You can add procedural code to this function in the testbench to process the user-defined behavior. Each pre-defined callback triggers the default method as shown below:

```
virtual function int DefaultCbF(ref denaliCdn_axiTransaction trans);
```

You can write general code (for example, message print) that can be performed on all callback types.

7.2.2. Instantiating Transaction Callbacks

The pre-defined callbacks are specified in the `$DENALI/ddvapi/sv/denaliCdn_axiTypes.svh` file.

7.2.3. Enabling Transaction Callbacks

You can use the predefined callback type IDs in SystemVerilog by including the `denaliCdn_axiTypes.svh` file in your testbench module. The `denaliCdn_axiTypes.svh` file is supplied with the CDN_AXI VIP model. It is located at `$DENALI/ddvapi/sv`.

7.2.3.1. setCallback()

You can set up a model callback using the `setCallback()` function as shown below:

```
cdn_axiMaster.setCallback(DENALI_CDN_AXI_CB_Ended);
```

Here, `cdn_axiMaster` is one of the initialized models of type `denaliCdn_axiInstance` and `DENALI_CDN_AXI_CB_Ended` is one of the predefined callbacks of type `denaliCdn_axiCbPointT`.

7.2.3.2. clearCallback()

Callbacks that have been enabled with `setCallback()` can be disabled with the `clearCallback()` function as shown below:

```
cdn_axiMaster.clearCallback(DENALI_CDN_AXI_CB_Ended);
```

7.2.4. Processing Transaction Callbacks

The variable `Event` toggles when any enabled callback triggers. There is only a single event for each instance in the design hierarchy even if multiple callbacks are enabled. These functions are defined (for SystemVerilog) in `$DENALI/ddvapi/sv/denaliCdn_axi.sv`.

To process transaction callbacks, you can use the following functions:

7.2.4.1. reasonStr()

This function displays a text string for the reason for a callback.

7.2.4.2. Example

The following example shows how the above-mentioned functions are used:

```
class denaliCdn_axiInstance;
     ...
     virtual function int EndedCbf (ref denaliCdn_axiTransaction trans) ;
        if (trans == null) begin
            $display("==== TRANS is null");
        end
        else begin
            if (trans.Type == DENALI_CDN_AXI_TR_Write) begin
                $display("Detected DENALI_CDN_AXI_TR_Write");
                trans.printInfo();
                // Process write transaction callback
            end

            if (trans.Type == DENALI_CDN_AXI_TR_Read) begin
                $display("Detected DENALI_CDN_AXI_TR_Read");
                trans.printInfo();
                // Process read transaction callback
            end
        end
        return super.EndedCbf(trans);
     endfunction

     ...
endclass
```

Chapter 8. CDN_AXI VIP Simulation

8.1. Running the Model

This section describes the main steps to run the VIP model.

8.1.1. Setting Basic Logging and Simulation Parameters

Before running your simulation, ensure that you have appropriately directed the output to the desired log file locations. With the exception of the simulation time parameter, the following parameters are all in the `.denalirc` file. For details on the `.denalirc` parameters, refer to Section 5.1.1, *"General .denalirc Options"*.

- Set the history file location using the `HistoryFile` parameter.
- Set the history debugging level using the `HistoryDebug` parameter. Setting this to "On" copies history file information into your trace file for easy debugging.
- Set the trace file location using the `TraceFile` parameter.
- Set the simulation time parameter as described in Section 8.1.4, *"Ending Simulation"*.

8.1.2. Linking the CDN_AXI VIP Library and Running the Simulation

The CDN_AXI VIP product is supplied in the form of a binary object file. By linking in this library, the CDN_AXI VIP instance communicates with the simulator program through the API function calls. Notice these are in the same UNIX process.

Refer to Chapter 10, *CDN_AXI VIP Testbench Integration* for more details.

After the simulation completes, the history and trace files are created.

8.1.3. Viewing Results

After completing the simulation run, the CDN_AXI VIP creates the following files (subject to the corresponding settings in the `.denalirc` file):

History File — The history file is a very useful resource that is generated during simulation. With this file, you can view the details of the CDN_AXI VIP model at a given simulation time, which aids in debugging the design under test.

Trace File — The trace file contains all events the model receives, such as reads, writes, and so on. This is primarily used by *Cadence Customer Support* as a valuable diagnostic tool for understanding and recreating your simulation environment.

8.1.4. Ending Simulation

There are two primary ways to determine that the simulation has completed. The simulation can end after either one of these conditions:

- A specified duration of simulation time has elapsed.
- A specified number of data transaction have been processed.

  If you use this method, you can enable callbacks to look for the ID of the last transaction sent, and end the simulation when that transaction ID is detected.

You can also add a timeout to prevent your simulation from hanging indefinitely.

You can also set a time limit for your simulation in your testbench instantiation file, as shown in the example below:

```
initial begin
for(i=0;i< `SIMTIME; i=i+1)
    #1000 ;
    $finish;
end
```

The above code allows you to run the simulation for a set amount of time as defined by the variable `SIMTIME`. You can set this variable via the following command line when you run your simulation:

```
`define SIMTIME <x>
```

Here, <x> is a valid number such as 5000.

8.2. Controlling Model Behavior

You can control the model behavior using the run-time initialization file `.denalirc`. This file stores settings that control the CDN_AXI VIP model behavior during simulation.

The `.denalirc` file is a text file containing keyword-value pairs used to store your initialization settings. All lines beginning with # are comments and are ignored. The default `.denalirc` file in your installation also includes commented lines that provide a complete description of all the switches and their usage. The descriptions use mixed-case for clarity but the flags are NOT case-sensitive, even though the values might be.

Note

The name of the `.denalirc` file cannot be changed.

8.2.1. The .denalirc File Hierarchy

Before running the simulation for the first time, the CDN_AXI VIP needs the `.denalirc` to control the model behavior during simulation. You can use up to four `.denalirc` files to store these settings. These files are listed below in the order of precedence:

$DENALIRC	*Environment variable*
`.denalirc`	*Simulation specific defaults*
`.denalirc`	*User Defaults*
`$DENALI/.denalirc`	*System Defaults*

If the `$DENALIRC` environment variable is set, and a file exists in the specified path, that file is used, and the others are ignored. If `$DENALIRC` is not set (or if the file is not found at the specified location), then the simulator looks in the current directory for `.denalirc`.

For example, if you want to change only one setting for a particular simulation, create a `.denalirc` file in the working directory to store the specific simulation settings.

8.3. Output Files

You can enable the CDN_AXI VIP to generate different types of output log files by setting different options in the `.denalirc` file. These log files are helpful during the debug process. The following sections describe these output files and some of the aspects of each log file that can be used during the debug process.

8.3.1. History File

The history file allows you to view the details of the CDN_AXI VIP model at a given simulation time, which aids you in debugging your design. The history file includes the following information:

- Memory/register read and write along with address, values, and simulation time
- Model state transition along with the simulation time
- Model activities during different states of state machines
- Data flow at different callback points
- Data transmitted and received on different pins
- Order-of-model activities with respect to the simulation time

The history file contains all messages, so the size of this file can be significant depending on the test case intent.

Every line of the file represents an event. The syntax is as follows:

```
<instance_name> <time> <debug_flag> <action> <information>
```

where:

- `<instance_name>` is an instance, such as an endpoint device

- `<time>` is the time when the event occurred
- `<debug_flag>` is for lines that only appear if the `HistoryDebug` parameter is set to *On*
- `<action>` represents the type of action, such as **WrReg** or **SIM WRITE**. These are described below in more detail.
- `<information>` contains details related to the event

The history file contains detailed model simulation messages in order of simulation time. It maintains a consistent structure, and includes following information:

- Register Writes (**WrReg**)

  WrReg stands for Write to Register. Any line denoted by **WrReg** shows the details of the write. The message shows the Register Name and the value to be written.

- Memory Same (**WRREG_SAME**)

  Certain registers may not be written with the exact value specified by the write command due to access types of one or more bits. However, if the exact value is written, the **WREG_SAME** is issued.

- Memory Difference (**WRREG_DIFF**)

  Certain registers may not be written with the exact value specified by the write command due to access types of one or more bits. However, if the exact value is NOT written, the **WREG_DIFF** is issued.

- Memory Writes (**SIM WRITE**)

  SIM WRITE is the actual write to model memory resources. It shows both the address and the data value written to memory location. The address mapping can be found in the `denaliCdn_axiTypes.svh` file for SystemVerilog or `api_cdn_axi.h` for use with the DRAPE.

- Memory Reads (**SIM READ**)

  SIM READ is the actual read from model memory resources. It shows both the address and the data that is read.

- Event Information (**Info**)

  The line containing **Info** keyword shows general information about different events happening. This information is very useful for debugging purposes. Whenever any major event happens inside the model and could be useful for debugging, it is reported as Info.

- Protocol Layer Information

The Layer messages are general informational messages that describe events that occur with the model at different layers.

- State Machine Transition (**State**)

  The **State** history message shows a transition of state machine. This message gives information about state machine name and context along with previous state and new state.

- Cycle

  The **Cycle** message shows the value received or transmitted at a given simulation time on RX or TX bus respectively.

- Configuration Instance

  The configuration instance message shows whether a given port in a device is a downstream port or upstream port function.

- Memory Instance

  The memory instance message shows the memory resource defined by BAR and Expansion Rom BAR in the design.

- Bus Range

  The message shows the bus range defined by a bridge.

- Tag Management

  The tag messages are used for tag Management activities

- Reset

  The reset messages indicate activities related to the reset process.

- Data Flow Control (Callback)

  The history file also contains information about transaction flow when it passes different callback locations. This information can be helpful in debugging the contents of a transaction at different points of data flow.

8.3.2. Trace File

The trace file is a very useful to debug and reproduce issues offsite. A test case can be extracted from this trace file because it stores sufficient information to create a test case for a particular issue. The trace file includes the following information:

- Trace File Header

The top part of tracefile contains HDL simulator, operating system and the VIP version information used for simulation.

- `.denalirc` Settings

  The trace file stores the `.denalirc` parameters settings used during simulations.

- Device Instance

  The trace file contains information about instance name and corresponding instance ID. This instance ID is used in rest of the trace file to identify model. For example:

  2 i testbench.i0

  where,

  i0 – An instance in the testbench

  2 – The instance ID corresponding to *testbench.i0*. The trace file will use '2' (first character on the line) to represent any information related to *testbench.i0*.

  I/i – An instance in the trace file

- SOMA File

  The trace file contains information about the SOMA file used for each CDN_AXI VIP model. It contains the path of the SOMA file used for a model as well as all settings in the SOMA file.

  The CDN_AXI VIP model reads all these SOMA values for different parameters and writes to corresponding registers present inside model at 0 simulation time. All these writes can be seen in the trace file at the start of simulation.

- Simulation Time

  The trace file contains all the activities on a specific simulation time followed by simulation time stamp. The format for the time stamp is: `-<simulation time> <time unit>-`

- Event on Pin

  The Symbol 'E' represents an event on a particular pin in the trace file. The first character always represents the instance ID. For example:

  2 E RX 00000000

  2 – The instance ID

  E – The symbol for event followed by value (0000000 – Represents the data value)

RX – The pin name

- Scheduled Event on Pin

  The Symbol 'S' represents an event scheduled on a particular pin in the trace file. For example.:

  2 S TX 00001100 10 ns T

  2 – The Instance ID

  S – The symbol to represent event scheduling followed by value 10 ns – after 10 ns of current simulation time.

  TX – The TX pin name

- History File Info

  In the `.denalirc` file, if the setting `HistoryDebug` is *On* then all the history file information can be found in the trace file too. The symbol 'H' is used to represent the history information.

8.3.2.1. Creating a Trace File

To create a trace file:

- In your `.denalirc` file, turn on trace file generation by adding the following line:

  ```
  Tracefile denali.trc
  ```

  where `denali.trc` is the name of your trace file.

  Note

  A line containing `Tracefile denali.trc` might already be present but commented out in your `.denalirc` file. If so, uncomment it (by removing the "#" at the beginning of the line).

  The `.trc` suffix for the name of the trace file is not mandatory. It is recommended for ease of identification.

8.4. Verification Messages

8.4.1. Changing Message Severity

You can control the severity of a verification message by writing to the DENALI_CDN_AXI_REG_ErrCtrl register. Bits [3:0] of this register represent the severity field, which determines how the CDN_AXI VIP displays the error. Valid values are specified by the denaliCdn_axiErrSeverityCtrlT enum type. For example:

- DENALI_CDN_AXI_ERR_CONFIG_SEVERITY_Error = 2 /* Change the severity level to Error
- DENALI_CDN_AXI_ERR_CONFIG_SEVERITY_Warn = 3 /* Change the severity level to Warn
- DENALI_CDN_AXI_ERR_CONFIG_SEVERITY_Info = 4 /* Change the severity level to Info

The following code illustrates how to change the severity of an item in the testbench from Error to Info:

```
errId = <Some Protocol Error value>
severity = DENALI_CDN_AXI_ERR_CONFIG_SEVERITY_Info;
value = (errId << DENALI_CDN_AXI_Rpos__MODEL_ERROR_CTRL_errId) |
(severity << DENALI_CDN_AXI_Rpos__MODEL_ERROR_CTRL_severity);
regInst.writeReg(DENALI_CDN_AXI_REG_ErrCtrl,value);
```

To change the severity of several messages, you can write to this register multiple times. Every time you write to this register, the CDN_AXI VIP processes it immediately and stores the information.

Note
The method for changing message severity differs when using UVM. For information on changing message severity using UVM, see Section 10.3.5, "*Error Reporting and Control*".

8.4.2. Message List

Following is a list of the messages that can be reported by the CDN_AXI VIP. Each item consists of the enumeration, followed by the string (with all formatting parameters) that is printed during simulation. Below the string are listed various reporting criteria for that message.

1. **DENALI_CDN_AXI_UNSET**

   This field is not set

   Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

2. **DENALI_CDN_AXI_NONFATAL_WARNING_MASTER_DATA_ITEM_CREATION_INVALID_INPUT**

   Master could not create a legal VIP transaction from input and the item is discarded.

   Report on: Tx Rx *Severity: WARNING* *Callback: Yes* *Coverage: Yes*

3. **DENALI_CDN_AXI_NONFATAL_WARNING_SLAVE_DATA_ITEM_CREATION_INVALID_INPUT**

   Slave could not create a legal VIP transaction from input and the item is discarded.

   Report on: Tx Rx *Severity: WARNING* *Callback: Yes* *Coverage: Yes*

4. **DENALI_CDN_AXI_FATAL_expected_cache_states_after_read**

   .In agent ...:.The command/response does not fit the cache state for address (...).Cache state: ...Read command: ....rresp_pass_dirty: ....rresp_is_shared: ....

   Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

5. **DENALI_CDN_AXI_FATAL_expected_cache_states_after_write**

   .In agent ...:.The command/response does not fit the cache state for address (...).Cache state: ...Write command: ....

   Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

6. **DENALI_CDN_AXI_FATAL_expected_cache_states_after_snoop**

   .In agent ...:.The command/response does not fit the cache state for address (...).Cache state: ...Snoop command: ....Snoop response: ... , ... , ... , ...

   Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

7. **DENALI_CDN_AXI_NONFATAL_WARNING_VR_AXI1001_WRITE_BURST_ RESPONSE_TIMEOUT_AND_DISCARDED**

   The write response to a burst was not received after a configurable number of clock cycles from the beginning of the address phase.

   Report on: Tx Rx *Severity: WARNING* *Callback: Yes* *Coverage: Yes*

8. **DENALI_CDN_AXI_NONFATAL_WARNING_VR_AXI1002_READ_BURST_ RESPONSE_TIMEOUT_AND_DISCARDED**

   The last read transfer to a burst was not received after a configurable number of clock cycles from the beginning of the address phase.

   Report on: Tx Rx *Severity: WARNING* *Callback: Yes* *Coverage: Yes*

9. **DENALI_CDN_AXI_NONFATAL_ERR_VR_AXI200_SLAVE_BAD_SIGNAL_VALUE**

   At the rising edge of ACLK all signals must be in logical values (i.e. 0 or 1)

   Report on: Tx Rx *Severity: WARNING* *Callback: Yes* *Coverage: Yes*

10. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI239_SLAVE_BAD_DATA_SIGNAL_ VALUE_WHEN_RVALID_HIGH**

    At the rising edge of ACLK the data signal must be in logical values (i.e. 0 or 1)

    Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

11. **DENALI_CDN_AXI_NONFATAL_ERR_VR_AXI240_SLAVE_BAD_DATA_SIGNAL_ VALUE_WHEN_RVALID_LOW**

    At the rising edge of ACLK the data signal must be in logical values (i.e. 0 or 1)

    Report on: Tx Rx *Severity: WARNING* *Callback: Yes* *Coverage: Yes*

12. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI201_RVALID_NOT_STABLE**

    RVALID must remain asserted until RREADY is asserted

    Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

13. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI202_RLAST_NOT_STABLE**
    While RVALID is asserted, RLAST must remain stable
    Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

14. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI203_RDATA_NOT_STABLE**
    While RVALID is asserted, RDATA must remain stable
    Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

15. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI204_RRESP_NOT_STABLE**
    While RVALID is asserted, RRESP must remain stable
    Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

16. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI205_RID_NOT_STABLE**
    While RVALID is asserted, RID must remain stable
    Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

17. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI206_BVALID_NOT_STABLE**
    BVALID must remain asserted until BREADY is asserted
    Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

18. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI207_BRESP_NOT_STABLE**
    While BVALID is asserted, BRESP must remain stable
    Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

19. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI208_BID_NOT_STABLE**
    While BVALID is asserted, BID must remain stable
    Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

20. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI209_RLAST_NOT_ASSERTED**
    Slave must assert RLAST when sending the last transfer of a read burst.
    Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

21. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI210_INVALID_RLAST**
    Slave must not assert RLAST before sending the last read transfer of a read burst
    Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

22. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI211_RID_WITH_NO_MATCHING_ARID**
    Slave can drive a transfer on the read channel only if there was a burst with that id on the read address channel.

DENALI_CDN_AXI_FATAL_ERR_VR_AXI211_RID_WITH_NO_MATCHING_ARID

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

23. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI212_BID_WITH_NO_MATCHING_AWID**

Slave can assert BVALID only if it ended a write burst with the same ID before

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

24. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI213_EXOKAY_RESP_ON_EXCLUSIVE_WRITE_WITH_NO_EXCLUSIVE_READ**

Slave should not return EXOKAY to an exclusive write burst if there's no exclusive read monitored for its id.

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

25. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI214_EXOKAY_RESP_ON_INCOMPATIBLE_EXCLUSIVE_WRITE**

Slave should not return EXOKAY to an exclusive write burst if the exclusive read burst of its id isn't compatible with it.

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

26. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI215_OKAY_RESP_ON_VALID_EXCLUSIVE_WRITE**

Slave should not return OKAY to a valid exclusive write burst.

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

27. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI232_EXOKAY_RESP_ON_DIFFERENT_SEQ_EXCLUSIVE_WRITE**

Slave shouldn't return an EXOKAY for an exclusive write burst if the exclusive read of it's id belongs to another exclusive sequence. This can happen if an exclusive read starts while an exclusive write for a previous read is in progress. If the new exclusive read has the same id as the exclusive write then the sitaution occures.

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

28. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI245_EXOKAY_RESP_ON_EXCLUSIVE_WRITE_WITH_NO_FAILED_READ**

Exclusive write got EXOKAY even after the previous exclusive READ failed (didn't get EXOKAY response)

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

29. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI216_RVALID_HIGH_DURING_RESET**

Slave must drive RVALID low during reset

DENALI_CDN_AXI_FATAL_ERR_VR_AXI216_RVALID_HIGH_DURING_RESET

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

30. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI217_BVALID_HIGH_DURING_RESET**

Slave must drive BVALID low during reset

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

31. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI218_EXOKAY_READ_TRANSFER_RESPONSE_TO_NON_EXCLUSIVE_TRANSFER**

slave must not drive an EXOKAY response to a non-exclusive read transfer

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

32. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI219_EXOKAY_WRITE_BURST_RESPONSE_TO_NON_EXCLUSIVE_BURST**

slave must not drive an EXOKAY response to a non-exclusive write burst

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

33. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI227_TRANSFER_EXCEEDED_READ_DATA_REORDERING_DEPTH**

Check that read_data_reordering_depth is not exceeded

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

34. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI228_ARREADY_AFTER_MAX_READ_DEPTH**

Check that read_data_reordering_depth is not exceeded

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

35. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI243_ARREADY_AFTER_MAX_READ_ACCEPTANCE_CAPABILITY**

Check that read_data_reordering_depth is not exceeded

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

36. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI229_AWREADY_AFTER_MAX_WRITE_DEPTH**

Check that the address phase of an extra WRITE bursts was stalled.

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

37. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI244_AWREADY_AFTER_MAX_WRITE_ACCEPTANCE_CAPABILITY**

Check that AWREADY is low after reaching write_acceptance_capability

DENALI_CDN_AXI_FATAL_ERR_VR_AXI244_AWREADY_AFTER_MAX_WRITE_ACCEPTANCE_CAPABILITY

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

38. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI231_READ_TRANSFER_MAPPED_ADDRESS_AND_DECERR**

    Check that the slave responds with DECERR when there's a read transfer to an unmapped address, and doesn't respond with DECERR when there's a read transfer to a mapped address.

    Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

39. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI230_READ_TRANSFER_UNMAPPED_ADDRESS_AND_NO_DECERR**

    Check that the slave responds with DECERR when there's a read transfer to an unmapped address, and doesn't respond with DECERR when there's a read transfer to a mapped address.

    Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

40. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI238_READ_BURST_NON_ACCESSIBLE_ADDRESS_AND_NO_DECERR**

    Check that the slave responds with DECERR when there's a read transfer to an unmapped address, and doesn't respond with DECERR when there's a read transfer to a mapped address.

    Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

41. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI233_READ_TR_STARTED_BEFORE_ADDR_PHASE_FINISHED**

    Slave can drive a transfer on the read channel only if the transfer burst finished its address phase.

    Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

42. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI235_WRITE_BURST_MAPPED_ADDRESS_AND_DECERR**

    Check that the slave responds with DECERR when there's a write burst to an unmapped address, and doesn't respond with DECERR when there's a write burst to a mapped address. NOTE: We check the burst start address only.

    Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

43. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI234_WRITE_BURST_UNMAPPED_ADDRESS_AND_NO_DECERR**

    Check that the slave responds with DECERR when there's a write burst to an unmapped address, and doesn't respond with DECERR when there's a write burst to a mapped address. NOTE: We check the burst start address only.

    Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

44. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI237_WRITE_BURST_NON_ACCESSIBLE_ADDRESS_AND_NO_DECERR**

    Check that the slave responds with DECERR when there's a write burst to an unmapped address, and doesn't respond with DECERR when there's a write burst to a mapped address. NOTE: We check the burst start address only.

    Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

45. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI236_RLAST_ASSERTED_WHEN_RVALID_IS_LOW**

    Deprecated check! - When RVALID is low RLAST must be low.

    Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

46. **DENALI_CDN_AXI_NONFATAL_ERR_VR_AXI241_READ_CHANNEL_SIGNALS_NOT_CHANGED**

    RVALID and RREADY were asserted in previous cycle and this cycle the slave didn't change any of his read data signals (warning)

    Report on: Tx Rx *Severity: WARNING* *Callback: Yes* *Coverage: Yes*

47. **DENALI_CDN_AXI_NONFATAL_ERR_VR_AXI242_WRITE_RESP_CHANNEL_SIGNALS_NOT_CHANGED**

    BVALID and BREADY were asserted in previous cycle and this cycle the slave didn't change any of his write resp signals (warning)

    Report on: Tx Rx *Severity: WARNING* *Callback: Yes* *Coverage: Yes*

48. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI246_BUSER_NOT_STABLE**

    While BVALID is asserted, BUSER must remain stable

    Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

49. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI247_RUSER_NOT_STABLE**

    While RVALID is asserted, RUSER must remain stable

    Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

50. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI248_AWREADY_ARREADY_ARE_STUCK**

    AWREADY and ARREADY must go up once in a while. If not, its a deadlock

    Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

51. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI226_MEMORY_INCONSISTENCY**

    Check memory consistency of the data in the transfer with the data in the memory.

    Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

52. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI2201_NO_EXOKAY_WRITE_RESPONSE_IN_AMBA4_LITE**

    Slave sent write EXOKAY response in AMBA4-Lite

    Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

53. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI2202_NO_EXOKAY_READ_RESPONSE_IN_AMBA4_LITE**

    Slave sent read EXOKAY response in AMBA4-Lite

    Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

54. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3225_WRONG_BARRIER_READ_RESPONSE**

    Slave sent wrong barrier read response (expected RRESP is b'0)

    Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

55. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3226_WRONG_BARRIER_WRITE_RESPONSE**

    Slave sent wrong barrier write response (expected BRESP is b'0)

    Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

56. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3201_ACVALID_NOT_STABLE**

    ACVALID must remain asserted until ACREADY is asserted

    Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

57. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3202_ACADDR_NOT_STABLE**

    While ACVALID is asserted, ACADDR must remain stable

    Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

58. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3203_ACSNOOP_NOT_STABLE**

    While ACVALID is asserted, ACSNOOP must remain stable

    Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

59. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3205_ACPROT_NOT_STABLE**

    While ACVALID is asserted, ACPROT must remain stable

    Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

60. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3212_ACVALID_HIGH_DURING_RESET**

    During reset ACVALID must be low.

    Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

61. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3218_SNOOP_ADDRESS_CHANNEL_SIGNALS_NOT_CHANGED**

    ACVALID and ACREADY were asserted in previous cycle and this cycle the slave didn't change any of his write address signals (warning)

    Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

62. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3222_RRESP_2_NOT_STABLE**

    While RVALID is asserted, RRESP[2:2] must remain stable

    Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

63. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3223_RRESP_3_NOT_STABLE**

    While RVALID is asserted, RRESP[3:3] must remain stable

    Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

64. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3224_USED_RESERVED_ACSNOOP**

    The ACSNOOP must not be Reserved

    Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

65. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3227_RRESP_2_3_ARE_NOT_CONSTANT_WITHIN_THE_BURST**

    Check to see if the response[2:3] values of all READ transfers are the same

    Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

66. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3228_WRONG_RRESP_FOR_TRANSACTION**

    Check to see if the first transfer of the burst has a legal response[2:3] value according to the snoop type Because all the transfer responses[2:3] are the same, we only need to check the first transfer

    Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

67. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3229_WRONG_ADDRESS_SPACE_ON_SNOOP**

    Check if the address of a snoop burst is mapped as a shareable address

    Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

68. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3230_ACADDR_NOT_ALIGNED_TO_DATA_WIDTH**

    The ACADDR must be aligned to the snoop data channel width

    Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

69. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3231_SENDING_TO_ACTIVE_ADDR**

    Check that the interconnect stall the snoop burst until the previous one to the same cache line is responded

    Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

70. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3232_WRONG_RLAST_FOR_SPECIFIC_READ_SNOOPS**

    For Make and Clean read transactions, the response should consist of only one transfer

    Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

71. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3240_SENT_SNOOP_BURST_WHILE_WAITING_FOR_ACK**

    The interconnect can't send snoop burst to the same cache line that has an active READ/WRITE Coherent and Cache Maintenance transaction waiting for the ACK stage.

    Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

72. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3241_SENT_READ_TRANSACTION_RESPONSE_WHILE_SNOOP_IS_IN_PROGRESS**

    When a snoop transaction is sent to a master, the interconnect must ensure that no READ responses to the same cache line are sent until the snoop response has been received

    Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

73. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3242_SENT_WRITE_TRANSACTION_RESPONSE_WHILE_SNOOP_IS_IN_PROGRESS**

    When a snoop transaction is sent to a master, the interconnect must ensure that no WRITE responses to the same cache line are sent until the snoop response has been received (only relevant to WriteLineUnique and WriteUnique transactions)

    Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

74. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3243_SENT_SNOOP_BURST_WHILE_DATA_TRANSFERS_ARE_SENT**

    The interconnect can't send snoop burst to the same cache line that has an active READ transaction where read transfers have started

    Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

75. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3244_SENT_EXOKAY_RESP_TO_NON_WRITENOSNOOP_BURST**

    The EXOKAY response is only permitted for a WriteNoSnoop transaction.

    Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

76. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3245_SENT_EXOKAY_RESP_TO_NON_READNOSNOOP_BURST**

    Deprecated check! - The EXOKAY response is only permitted for a ReadNoSnoop transaction.

    Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

77. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3250_WRONG_ACADDR_IN_DVM_COMPLETE**

    Check that when the burst is DVM_Complete, all the ACADDR bits are 0

    Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

78. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3251_WRONG_AXADDR_IN_DVM_MESSAGE**

    Check that when the burst is DVM_Message, the ACADDR has a legal value

    Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

79. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3252_WRONG_RRESP_IN_DVM_COMPLETE**

    Check that when the READ burst is DVM_Complete, the response is 0b0000

    Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

80. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3253_WRONG_RRESP_IN_DVM_SYNC**

    Check that when the burst is DVM Sync, the response is 0b0000

    Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

81. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3254_WRONG_RRESP_IN_DVM**

    Check that when the READ burst is DVM, the RRESP is b'0010 or b'0000 (only b'0000 for Hint)

    Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

82. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3255_WRONG_SNOOP_TYPE_IN_ACE_LITE**

    Check that when in ACE Lite only, the restricted set of snoop bursts are sent. In normal ACE Lite, no snoop bursts are allowed. In ACE Lite with supported DVM, only DVM snoop bursts are allowed.

    Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

83. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3256_WRONG_RLAST_FOR_READ_BARRIER**

    Check that a READ barrier response is consist of only one transfer

    Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

84. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3257_WRONG_RLAST_FOR_DVM**

Check that when the READ burst is DVM, the response is consist of only one transfer

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

85. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3258_SLAVE_UNCOMPLETED_SYNC**

Check that each DVM sync transaction is answered by a DVM complete snoop in a timely manner

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

86. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3259_MULTI_DVM_ERROR**

Check that Multi part snoop DVM adheres to protocol: the two parts are transmitted consecutively the second part does not have additional parts (dvm_has_va FALSE)

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

87. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3260_SLAVE_UNEXPECTED_ COMPLETE_DVM**

Check that no DVM_Complete snoop is transmitted unless there was a previous DVM sync

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

88. **DENALI_CDN_AXI_NONFATAL_WARNING_VR_AXI3246_DVM_COMPLETE_ SNOOP_DURING_MULTIPART_DVM_SEQ_IS_DISCARDED**

WARNING..
DVM_COMPLETE_SNOOP_DURING_MULTIPART_DVM_SEQ_IS_DISCARDED. The slave tried to send a DVM Complete snoop transaction between the two sections of multi-part DVM sequence

Report on: Tx Rx *Severity: WARNING* *Callback: Yes* *Coverage: Yes*

89. **DENALI_CDN_AXI_NONFATAL_ERR_VR_AXI100_MASTER_BAD_SIGNAL_ VALUE**

At the rising edge of ACLK all signals must be in logical values (i.e. 0 or 1)

Report on: Tx Rx *Severity: WARNING* *Callback: Yes* *Coverage: Yes*

90. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI189_MASTER_BAD_DATA_SIGNAL_ VALUE_WHEN_WVALID_HIGH**

At the rising edge of ACLK the data signal must be in logical values (i.e. 0 or 1).

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

91. **DENALI_CDN_AXI_NONFATAL_ERR_VR_AXI190_MASTER_BAD_DATA_ SIGNAL_VALUE_WHEN_WVALID_LOW**

At the rising edge of ACLK the data signal must be in logical values (i.e. 0 or 1).

DENALI_CDN_AXI_NONFATAL_ERR_VR_AXI190_MASTER_BAD_DATA_SIGNAL_VALUE_WHEN_WVALID_LOW

Report on: Tx Rx *Severity: WARNING* *Callback: Yes* *Coverage: Yes*

92. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI101_AWVALID_NOT_STABLE**
AWVALID must remain asserted until AWREADY is asserted
Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

93. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI102_ARVALID_NOT_STABLE**
ARVALID must remain asserted until ARREADY is asserted
Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

94. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI103_AWADDR_NOT_STABLE**
While AWVALID is asserted, AWADDR must remain stable
Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

95. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI104_ARADDR_NOT_STABLE**
While ARVALID is asserted, ARADDR must remain stable
Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

96. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI105_AWLEN_NOT_STABLE**
While AWVALID is asserted, AWLEN must remain stable
Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

97. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI106_ARLEN_NOT_STABLE**
While ARVALID is asserted, ARLEN must remain stable
Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

98. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI107_AWSIZE_NOT_STABLE**
While AWVALID is asserted, AWSIZE must remain stable
Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

99. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI108_ARSIZE_NOT_STABLE**
While ARVALID is asserted, ARSIZE must remain stable
Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

100. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI109_AWBURST_NOT_STABLE**
While AWVALID is asserted, AWBURST must remain stable
Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

101. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI110_ARBURST_NOT_STABLE**
While ARVALID is asserted, ARBURST must remain stable
Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

102. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI111_AWLOCK_NOT_STABLE**
While AWVALID is asserted, AWLOCK must remain stable
Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

103. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI112_ARLOCK_NOT_STABLE**
While ARVALID is asserted, ARLOCK must remain stable
Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

104. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI113_AWCACHE_NOT_STABLE**
While AWVALID is asserted, AWCACHE must remain stable
Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

105. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI114_ARCACHE_NOT_STABLE**
While ARVALID is asserted, ARCACHE must remain stable
Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

106. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI115_AWPROT_NOT_STABLE**
While AWVALID is asserted, AWPROT must remain stable
Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

107. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI116_ARPROT_NOT_STABLE**
While ARVALID is asserted, ARPROT must remain stable
Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

108. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI117_AWID_NOT_STABLE**
While AWVALID is asserted, AWID must remain stable
Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

109. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI118_ARID_NOT_STABLE**
While ARVALID is asserted, ARID must remain stable
Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

110. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI119_WVALID_NOT_STABLE**
WVALID must remain asserted until WREADY is asserted

DENALI_CDN_AXI_FATAL_ERR_VR_AXI119_WVALID_NOT_STABLE

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

111. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI120_WLAST_NOT_STABLE**

While WVALID is asserted, WLAST must remain stable

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

112. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI121_WDATA_NOT_STABLE**

While WVALID is asserted, WDATA must remain stable

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

113. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI122_WSTRB_NOT_STABLE**

While WVALID is asserted, WSTRB must remain stable

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

114. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI124_WLAST_NOT_ASSERTED**

Master must assert WLAST when sending the last transfer of a write burst.

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

115. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI125_INVALID_WLAST**

Master must not assert WLAST before sending the last write transfer of a write burst.

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

116. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI126_WSTRB_WHEN_WVALID_LOW**

When WVALID is low, WSTRB should be low or stay in previous state (recommendation).

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

117. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI127_WRITE_CROSS_4K_BOUNDARY**

The master started a write burst that will cross a 4K boundary.

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

118. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI128_READ_CROSS_4K_BOUNDARY**

The master started a read burst that will cross a 4K boundary.

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

119. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI129_WRITE_WRONG_WRAP_LENGTH**

A WRAP burst can have only 2, 4, 8 or 16 transfers.

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

120. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI130_READ_WRONG_WRAP_LENGTH**

A WRAP burst can have only 2, 4, 8 or 16 transfers

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

121. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI1100_WRONG_LENGTH_IN_FIXED_WRITE**

A FIXED burst can have a maximum of 16 transfers.

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

122. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI1101_WRONG_LENGTH_IN_FIXED_READ**

A FIXED burst can have a maximum of 16 transfers.

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

123. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI1104_WRONG_LENGTH_IN_INCR_WRITE**

An INCR burst can have a maximum of 256 transfers in AMBA4 and 16 in AMBA3

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

124. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI1105_WRONG_LENGTH_IN_INCR_READ**

An INCR burst can have a maximum of 256 transfers in AMBA4 and 16 in AMBA3

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

125. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI131_WRITE_SIZE_TOO_BIG**

The size of a transfer must not exceed the bus width.

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

126. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI132_READ_SIZE_TOO_BIG**

The size of a transfer must not exceed the bus width.

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

127. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI133_USED_RESERVED_AWBURST**

The AWBURST must not be 3 - Reserved

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

128. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI134_USED_RESERVED_ARBURST**

The ARBURST must not be 3 - Reserved

DENALI_CDN_AXI_FATAL_ERR_VR_AXI134_USED_RESERVED_ARBURST

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

129. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI135_WRITE_BAD_START_ADDRESS_ON_WRAP**

The start address of a WRAP burst must be aligned with the size.

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

130. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI136_READ_BAD_START_ADDRESS_ON_WRAP**

The start address of a WRAP burst must be aligned with the size.

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

131. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI137_READ_ALLOCATE_HIGH_WHEN_CACHEABLE_LOW**

The read_allocate bit, must not be high if the cacheable is low.

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

132. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI194_READ_ALLOCATE_HIGH_WHEN_CACHEABLE_LOW_IN_WRITE_BURST**

The read_allocate bit, must not be high if the cacheable is low for write burst.

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

133. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI138_WRITE_ALLOCATE_HIGH_WHEN_CACHEABLE_LOW**

The write_allocate bit, must not be high if the cacheable is low.

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

134. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI195_WRITE_ALLOCATE_HIGH_WHEN_CACHEABLE_LOW_IN_READ_BURST**

The write_allocate bit, must not be high if the cacheable is low for read burst.

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

135. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI139_USED_RESERVED_AWLOCK**

The AWLOCK must not be 3 - Reserved.

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

136. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI140_USED_RESERVED_ARLOCK**

The ARLOCK must not be 3 - Reserved.

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

137. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI142_WRONG_AWLEN_ON_EXCLUSIVE_WRITE**

The length must not change between exclusive read and exclusive write.

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

138. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI143_WRONG_TOTAL_SIZE_OF_EXCLUSIVE_WRITE**

EXCLUSIVE burst total number of bytes should be in [1,2,4,8,16,32,64,128].

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

139. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI144_WRONG_TOTAL_SIZE_OF_EXCLUSIVE_READ**

EXCLUSIVE burst total number of bytes should be in [1,2,4,8,16,32,64,128].

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

140. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI1102_WRONG_LENGTH_IN_EXCLUSIVE_WRITE**

AMBA4 EXCLUSIVE burst maximum number of transfers should be 16.

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

141. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI1103_WRONG_LENGTH_IN_EXCLUSIVE_READ**

AMBA4 EXCLUSIVE burst maximum number of transfers should be 16.

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

142. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI145_EXCLUSIVE_WRITE_WITH_NO_EXCLUSIVE_READ**

The exclusive write should be called only after a call to the exclusive read.

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

143. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI186_EXCLUSIVE_WRITE_DATA_PHASE_WITH_NO_EXCLUSIVE_READ**

The exclusive write should be called only after a call to the exclusive read.

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

144. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI146_WRONG_AWADDR_ON_EXCLUSIVE_WRITE**

The address must not change between exclusive read and exclusive write.

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

145. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI147_WRONG_AWSIZE_ON_EXCLUSIVE_WRITE**

The transfer size must not change between exclusive read and exclusive write.

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

146. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI148_WRONG_AWBURST_ON_EXCLUSIVE_WRITE**

The burst type must not change between exclusive read and exclusive write

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

147. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI149_WRONG_AWCACHE_ON_EXCLUSIVE_WRITE**

The cache must not change between exclusive read and exclusive write.

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

148. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI150_WRONG_AWPROT_ON_EXCLUSIVE_WRITE**

The protection must not change between exclusive read and exclusive write.

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

149. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI151_CACHEABLE_WHEN_EXCLUSIVE_READ**

Deprecated check! - No cacheable access on exclusive read access.

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

150. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI152_CACHEABLE_WHEN_EXCLUSIVE_WRITE**

Deprecated check! - No cacheable access on exclusive write access.

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

151. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI158_WRONG_WSTRB**

A master must assert byte strobes that can have valid data according to the control fields.

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

152. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI159_AWVALID_HIGH_DURING_RESET**

During reset AWVALID must be low.

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

153. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI160_ARVALID_HIGH_DURING_RESET**

During reset ARVALID must be low.

DENALI_CDN_AXI_FATAL_ERR_VR_AXI160_ARVALID_HIGH_DURING_RESET

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

154. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI161_WVALID_HIGH_DURING_RESET**

During reset WVALID must be low.

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

155. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI162_CHANGED_ID_IN_LOCKED_SEQUENCE**

The master must ensure that all transactions within a locked sequence have the same id tag.

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

156. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI165_WRITE_TRANSFER_CHANGED_ID_IN_LOCKED_SEQUENCE**

The master must ensure that all transfers within a locked sequence have the same id tag.

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

157. **DENALI_CDN_AXI_NONFATAL_ERR_VR_AXI163_CHANGED_ACACHE_IN_LOCKED_SEQUENCE**

ARM recommends that the master will ensure that all transactions within a locked sequence have the same A(R/W)CACHE value

Report on: Tx Rx *Severity: WARNING* *Callback: Yes* *Coverage: Yes*

158. **DENALI_CDN_AXI_NONFATAL_ERR_VR_AXI164_CHANGED_APROT_IN_LOCKED_SEQUENCE**

ARM recommends that the master will ensure that all transactions within a locked sequence have the same A(R/W)PROT value

Report on: Tx Rx *Severity: WARNING* *Callback: Yes* *Coverage: Yes*

159. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI169_TRANSFER_EXCEEDED_WRITE_DATA_INTERLEAVING_DEPTH**

A master is not allowed to have more then the slave's write data interleaving depth WRITE bursts executing in parallel.

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

160. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI170_WRITE_BURST_EXCEEDED_WRITE_DATA_INTERLEAVING_DEPTH**

A master is not allowed to have more then the slave's write data interleaving depth WRITE bursts executing in parallel.

DENALI_CDN_AXI_FATAL_ERR_VR_AXI170_WRITE_BURST_EXCEEDED_WRITE_DATA_INTERLEAVING_DEPTH

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

161. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI171_BURST_EXCEEDED_WRITE_DATA_INTERLEAVING_DEPTH**

A master is not allowed to have more then the slave's write data interleaving depth WRITE bursts executing in parallel.

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

162. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI167_ARVALID_AFTER_MAX_READ_ISSUING_CAPABILITY**

Check that ARVALID is low after reaching read_issuing_capability

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

163. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI168_AWVALID_AFTER_MAX_WRITE_ISSUING_CAPABILITY**

Check that AWVALID is low after reaching read_issuing_capability

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

164. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI172_FIRST_TRANSFER_OUT_OF_ORDER**

The order in which a master sends the first transfer of each WRITE burst must be the same as the address phases order (spec 8.5,8-6).

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

165. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI174_EXCLUSIVE_WRITE_BEFORE_READ_ENDED**

The exclusive write should not start before the exclusive read ended.

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

166. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI187_EXCLUSIVE_WRITE_DATA_PHASE_BEFORE_READ_ENDED**

The exclusive write should not start before the exclusive read ended.

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

167. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI175_LOCK_STARTED_WHEN_BURSTS_ARE_IN_PROGRESS**

Locked bursts must not start until all other bursts finish.

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

168. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI188_LOCK_DATA_PHASE_STARTED_WHEN_BURSTS_ARE_IN_PROGRESS**

Locked bursts must not start until all other bursts finish.

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

169. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI177_ADDRESS_NOT_ALIGNED_TO_TOTAL_SIZE_ON_EXCLUSIVE_READ**

READ EXCLUSIVE burst address should be aligned to the burst total size.

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

170. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI178_ADDRESS_NOT_ALIGNED_TO_TOTAL_SIZE_ON_EXCLUSIVE_WRITE**

WRITE EXCLUSIVE address should be aligned to the burst total size.

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

171. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI180_WLAST_ASSERTED_WHEN_WVALID_IS_LOW**

Deprecated check! - When WVALID is low WLAST must be low.

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

172. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI181_RETROACTIVE_WLAST_NOT_ASSERTED**

Master must assert WLAST when sending the last transfer of a write burst. This check is performed when an address phase of a burst starts after its data phase.

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

173. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI182_RETROACTIVE_INVALID_WLAST**

Master must not assert WLAST before sending the last write transfer of a write burst. This check is performed when an address phase of a burst starts after its data phase.

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

174. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI183_RETROACTIVE_WRONG_WSTRB**

A master must assert byte strobes that can have valid data according to the control fields. This check is performed when an address phase of a burst starts after its data phase.

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

175. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI184_ADDRESS_PHASE_OUT_OF_ORDER**

The order in which a master sends the address phases and the data phases of write bursts must be identical.

DENALI_CDN_AXI_FATAL_ERR_VR_AXI184_ADDRESS_PHASE_OUT_OF_ ORDER

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

176. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI196_NEW_BURST_STARTED_BEFORE_ ADDRESS_PHASE_OF_PREVIOUS**

The order in which a master sends the address phases and the data phases of write bursts must be identical. This check will verify that if an address phase of a burst has started (address before data), there isn't any "data before address" burst before the address phase

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

177. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI185_EXCLUSIVE_READ_WHEN_ EXCLUSIVE_WRITES_ARE_IN_PROGRSS**

Exclusive read burst is not allowed to start while exclusive write bursts with the same id tag are in progress.

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

178. **DENALI_CDN_AXI_NONFATAL_ERR_VR_AXI191_READ_ADDRESS_CHANNEL_ SIGNALS_NOT_CHANGED**

ARVALID and ARREADY were asserted in previous cycle and this cycle the master didn't change any of his read address signals (warning)

Report on: Tx Rx *Severity: WARNING* *Callback: Yes* *Coverage: Yes*

179. **DENALI_CDN_AXI_NONFATAL_ERR_VR_AXI192_WRITE_ADDRESS_CHANNEL_ SIGNALS_NOT_CHANGED**

AWVALID and AWREADY were asserted in previous cycle and this cycle the master didn't change any of his write address signals (warning)

Report on: Tx Rx *Severity: WARNING* *Callback: Yes* *Coverage: Yes*

180. **DENALI_CDN_AXI_NONFATAL_ERR_VR_AXI193_WRITE_CHANNEL_SIGNALS_ NOT_CHANGED**

WVALID and WREADY were asserted in previous cycle and this cycle the master didn't change any of his write data signals (warning)

Report on: Tx Rx *Severity: WARNING* *Callback: Yes* *Coverage: Yes*

181. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI1106_AWUSER_NOT_STABLE**

While AWVALID is asserted, AWUSER must remain stable

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

182. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI1107_ARUSER_NOT_STABLE**

While ARVALID is asserted, ARUSER must remain stable

DENALI_CDN_AXI_FATAL_ERR_VR_AXI1107_ARUSER_NOT_STABLE

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

183. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI1108_WUSER_NOT_STABLE**

While WVALID is asserted, WUSER must remain stable

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

184. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI1109_AWVALID_ARVALID_ARE_STUCK**

AWVALID and ARVALID must go up once in a while. If not, its a deadlock

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

185. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI123_WID_NOT_STABLE**

While WVALID is asserted, WID must remain stable

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

186. **DENALI_CDN_AXI_NONFATAL_ERR_VR_AXI154_LOCKED_ACCESSES_USE_MORE_THAN_4K_ADDRESS_REGION**

Locked transaction sequence should not cross 4K boundary (recommendation).

Report on: Tx Rx *Severity: WARNING* *Callback: Yes* *Coverage: Yes*

187. **DENALI_CDN_AXI_NONFATAL_ERR_VR_AXI155_MORE_THAN_TWO_LOCKED_TRANSACTIONS**

Locked transaction sequence should not include more than two transactions (recommendation).

Report on: Tx Rx *Severity: WARNING* *Callback: Yes* *Coverage: Yes*

188. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI176_BURST_STARTED_BEFORE_LOCK_SEQUENCE_ENDED**

While waiting for a lock sequence to end other bursts must not start.

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

189. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI179_LAST_LOCKED_SEQ_BURST_STARTED_PREMATURELY**

The last, non-locked burst of a locked sequence is not allowed to start while other burst of the sequence are still in progress.

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

190. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI2101_USED_RESEVRED_SIGNAL_IN_AMBA4_IN_AWLOCKED**

Master started a write LOCKED burst started in AMBA4

DENALI_CDN_AXI_FATAL_ERR_VR_AXI2101_USED_RESEVRED_SIGNAL_IN_AMBA4_IN_AWLOCKED

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

191. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI2102_USED_RESEVRED_SIGNAL_IN_AMBA4_IN_ARLOCKED**

Master started a read LOCKED burst started in AMBA4

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

192. **DENALI_CDN_AXI_ERR_VR_AXI2103_STARTED_NEW_WRITE_BURST_WITH_NON_ZERO_AWREGION**

Deprecated check! - A new write burst started at the master with a AWREGION that is not zero. Only the interconnect can change the AWREGION

Report on: Tx Rx *Severity: IGNORE* *Callback: Yes* *Coverage: Yes*

193. **DENALI_CDN_AXI_ERR_VR_AXI2104_STARTED_NEW_READ_BURST_WITH_NON_ZERO_ARREGION**

Deprecated check! - A new read burst started at the master with a ARREGION that is not zero. Only the interconnect can change the ARREGION

Report on: Tx Rx *Severity: IGNORE* *Callback: Yes* *Coverage: Yes*

194. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI2105_AWREGION_NOT_STABLE**

While AWVALID is asserted, AWREGION must remain stable

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

195. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI2106_ARREGION_NOT_STABLE**

While ARVALID is asserted, ARREGION must remain stable

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

196. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI2107_AWQOS_NOT_STABLE**

While AWVALID is asserted, AWQOS must remain stable

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

197. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI2108_ARQOS_NOT_STABLE**

While ARVALID is asserted, ARQOS must remain stable

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

198. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3101_AWSNOOP_NOT_STABLE**

While AWVALID is asserted, AWSNOOP must remain stable

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

199. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3102_ARSNOOP_NOT_STABLE**
While ARVALID is asserted, ARSNOOP must remain stable
Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

200. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3105_AWDOMAIN_NOT_STABLE**
While AWVALID is asserted, AWDOMAIN must remain stable
Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

201. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3106_ARDOMAIN_NOT_STABLE**
While ARVALID is asserted, ARDOMAIN must remain stable
Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

202. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3107_AWBAR_NOT_STABLE**
While AWVALID is asserted, AWBAR must remain stable
Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

203. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3111_ARBAR_NOT_STABLE**
While ARVALID is asserted, ARBAR must remain stable
Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

204. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3103_CRVALID_NOT_STABLE**
once CRVALID is asserted, it must remain asserted until CRREADY is asserted
Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

205. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3128_CRRESP_NOT_STABLE**
While CRVALID is asserted, CRRESP must remain stable
Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

206. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3129_CDVALID_NOT_STABLE**
CDVALID must remain asserted until CDREADY is asserted
Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

207. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3115_CDDATA_NOT_STABLE**
While CDVALID is asserted, CDDATA must remain stable
Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

208. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3116_CDLAST_NOT_STABLE**
While CDVALID is asserted, CDLAST must remain stable

DENALI_CDN_AXI_FATAL_ERR_VR_AXI3116_CDLAST_NOT_STABLE

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

209. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3117_CRVALID_HIGH_DURING_RESET**

During reset CRVALID must be low.

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

210. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3118_CDVALID_HIGH_DURING_RESET**

During reset CDVALID must be low.

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

211. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3122_WRONG_RESP_TO_ TRANSACTION_TYPE**

Check that the snoop response is legal for the snoop type

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

212. **DENALI_CDN_AXI_NONFATAL_ERR_VR_AXI3121_UNEXPECTED_RESP_TO_ TRANSACTION_TYPE**

Check that the snoop response is excepted for the snoop type

Report on: Tx Rx *Severity: WARNING* *Callback: Yes* *Coverage: Yes*

213. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3123_CDLAST_NOT_ASSERTED**

Check that the last transfer has CDLAST TRUE This check is when the response already started. there is another check for transfers before the response phase.

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

214. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3124_RETROACTIVE_CDLAST_NOT_ ASSERTED**

Master must assert CDLAST when sending the last transfer of a write snoop burst. This check is performed when a response phase of a burst starts after its data phase.

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

215. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI4104_INVALID_CDLAST**

Master must not assert CDLAST before sending the last snoop transfer of a snoop burst.

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

216. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3125_STARTED_RESPONSE_ WITHOUT_MATCHING_ADDRESS**

Master must not assert CRVALID without any matching address phase on the snoop address channel.

DENALI_CDN_AXI_FATAL_ERR_VR_AXI3125_STARTED_RESPONSE_WITHOUT_MATCHING_ADDRESS

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

217. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3126_STARTED_DATA_WITHOUT_MATCHING_SNOOP**

Master must not assert CDVALID without a matching address phase on the snoop address channel and a correct response phase on the snoop response channel.

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

218. **DENALI_CDN_AXI_NONFATAL_ERR_VR_AXI3127_SNOOP_DATA_CHANNEL_SIGNALS_NOT_CHANGED**

CDVALID and CDREADY were asserted in previous cycle and this cycle the master didn't change any of his snoop data signals (warning)

Report on: Tx Rx *Severity: WARNING* *Callback: Yes* *Coverage: Yes*

219. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3153_WRONG_SIGNALS_IN_BARRIER_WRITE_PORTION**

Master sent wrong signals in a barrier write portion

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

220. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3154_WRONG_SIGNALS_IN_BARRIER_READ_PORTION**

Master sent wrong signals in a barrier read portion

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

221. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3155_WRONG_WRITE_PORTION_BARRIER_STARTED**

Master sent wrong write portion barrier started

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

222. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3156_WRONG_READ_PORTION_BARRIER_STARTED**

Master sent wrong read portion barrier started

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

223. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3151_WRONG_ID_FOR_WRITE_PORTION_OF_BARRIER**

Master sent wrong write id in the started barrier. Master sent a write barrier with an ID that already exists in a pending non-barrier transaction.

DENALI_CDN_AXI_FATAL_ERR_VR_AXI3151_WRONG_ID_FOR_WRITE_PORTION_OF_BARRIER

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

224. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3152_WRONG_ID_FOR_READ_PORTION_OF_BARRIER**

Master sent wrong read id in the started barrier Master sent a read barrier with an ID that already exists in a pending non-barrier transaction.

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

225. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3130_USED_RESERVED_AWSNOOP**

The AWSNOOP must not be Reserved

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

226. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3131_USED_RESERVED_ARSNOOP**

The ARSNOOP must not be Reserved

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

227. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3132_WRONG_AWDOMAIN_FOR_OUTER_ADDRESS_SPACE**

Check that if the write address is shareable, the domain is Inner/Outer

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

228. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3144_WRONG_AWDOMAIN_FOR_INNER_ADDRESS_SPACE**

Check that if the write address is shareable, the domain is Inner/Outer

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

229. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3133_WRONG_ARDOMAIN_FOR_OUTER_ADDRESS_SPACE**

Check that if the read address is shareable, the domain is Inner/Outer

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

230. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3145_WRONG_ARDOMAIN_FOR_INNER_ADDRESS_SPACE**

Check that if the read address is shareable, the domain is Inner/Outer

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

231. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3134_WRONG_AWDOMAIN_FOR_NON_SHAREABLE_ADDRESS_SPACE**

Check that if the write address is non shareable, the domain is NON_SHAREABLE or SYSTEM

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

232. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3135_WRONG_ARDOMAIN_FOR_NON_SHAREABLE_ADDRESS_SPACE**

Check that if the read address is non shareable, the domain is NON_SHAREABLE or SYSTEM

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

233. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3136_WRONG_AWSNOOP_FOR_NON_SHAREABLE_DOMAIN**

Check that if the domain is NON_SHAREABLE, the AWSNOOP is WriteUnique,WriteBack or WriteClean Check that if the domain is SYSTEM, the AWSNOOP is WriteUnique

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

234. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI4100_WRONG_AWSNOOP_FOR_SYSTEM_DOMAIN**

Check that if the domain is NON_SHAREABLE, the AWSNOOP is WriteUnique,WriteBack or WriteClean Check that if the domain is SYSTEM, the AWSNOOP is WriteUnique

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

235. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3137_WRONG_ARSNOOP_FOR_NON_SHAREABLE_DOMAIN**

Check that if the domain is NON_SHAREABLE, the ARSNOOP is ReadOnce, CleanShared, CleanInvalid, MakeInvalid Check that if the domain is SYSTEM, the ARSNOOP is ReadOnce

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

236. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI4101_WRONG_ARSNOOP_FOR_SYSTEM_DOMAIN**

Check that if the domain is NON_SHAREABLE, the ARSNOOP is ReadOnce, CleanShared, CleanInvalid, MakeInvalid Check that if the domain is SYSTEM, the ARSNOOP is ReadOnce

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

237. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3138_WRONG_AWSNOOP_FOR_CACHE_STATE**

Check that The shareable AWSNOOP matches the current cache line state

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

238. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3139_WRONG_ARSNOOP_FOR_CACHE_STATE**

Check that The shareable ARSNOOP matches the current cache line state

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

239. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3140_WRITE_BAD_START_ADDRESS_ON_INCR_SHAREABLE_TRANSACTION**

Check that for shareable INCR transactions on the WRITE channel, the address is aligned to the cache line size (only relevant to WriteLineUnique and Evict transactions)

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

240. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3141_READ_BAD_START_ADDRESS_ON_INCR_SHAREABLE_TRANSACTION**

Check that for shareable INCR transactions on the READ channel, the address is aligned to the cache line size (not relevant to ReadOnce transactions)

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

241. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3142_WRITE_WRONG_TOTAL_SIZE_OF_SHAREABLE_TRANSACTION**

Check that for shareable transactions on the WRITE channel, the total size of the burst is at line length (only relevant to WriteLineUnique and Evict transactions)

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

242. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3143_READ_WRONG_TOTAL_SIZE_OF_SHAREABLE_TRANSACTION**

Check that for shareable transactions on the READ channel, the total size of the burst is at line length (not relevant to ReadOnce, CleanInvalid and MakeInvalid transactions)

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

243. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3146_WACK_WITH_NO_WRITE_BURST_ATTACHED**

Check that if the WACK signal was raised, there is a Write burst which is attached. (The signal wasn't raised for nothing)

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

244. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3147_RACK_WITH_NO_READ_BURST_ATTACHED**

Check that if the RACK signal was raised, there is a Read burst which is attached. (The signal wasn't raised for nothing)

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

245. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3149_WRONG_STROBE_ON_WRITELINEUNIQUE**

Check that when a master issues a WriteLineUnique transaction, all the bits in the strobe must be asserted.

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

246. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3150_CACHE_INCONSISTENCY_ON_WRITE_CHANNEL**

The Data of a cache line in a master can't change if the line is in shared state. The transition from shared to unique must be through the correct transaction. (only relevant to WriteClean and WriteBack transactions)

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

247. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3166_RETROACTIVE_CACHE_INCONSISTENCY_ON_SNOOP_CHANNEL**

The Data of a cache line in a master can't change if the line is in shared state. The transition from shared to unique must be through the correct transaction. (relevant to snoop transactions in case of data before response on the snoop channel) we can only check the data when the snoop response phase started)

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

248. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3159_UNSUPPORTED_AWBURST_IN_SHAREABLE_TRANSACTION**

The Fixed burst type is not supported for WRITE shareable transactions

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

249. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3160_UNSUPPORTED_ARBURST_IN_SHAREABLE_TRANSACTION**

The Fixed burst type is not supported for READ shareable transactions

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

250. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3161_CACHE_INCONSISTENCY_ON_SNOOP_CHANNEL**

The Data of a cache line in a master can't change if the line is in shared state. The transition from shared to unique must be through the correct transaction. (relevant to snoop transactions)

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

251. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3162_WRONG_WRITE_TRANSACTION_TYPE_IN_ACE_LITE**

The ACE LITE Master only support a restricted set of shareable WRITE transaction types

DENALI_CDN_AXI_FATAL_ERR_VR_AXI3162_WRONG_WRITE_TRANSACTION_TYPE_IN_ACE_LITE

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

252. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3163_WRONG_READ_TRANSACTION_TYPE_IN_ACE_LITE**

The ACE LITE Master only support a restricted set of shareable READ transaction types

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

253. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3164_WRONG_TOTAL_SIZE_IN_SHAREABLE_WRITEUNIQUE**

Deprecated check! - In WriteUnique to a shareable memory, the size is less then a cache line size

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

254. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3168_WRONG_SIZE_FOR_SHAREABLE_READ**

When reading from a shareable memory the size should be the entire bus (not relevant to Read-Once transactions)

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

255. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3169_WRONG_SIZE_FOR_SHAREABLE_WRITE**

When writing to a shareable memory the size should be the entire bus (only relevant to Write-LineUnique and Evict transactions)

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

256. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3172_EXCLUSIVE_SHAREABLE_READ_BURST**

Deprecated check! - Check that shareable read transactions are not exclusive

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

257. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3173_EXCLUSIVE_SHAREABLE_WRITE_BURST**

Check that shareable write transactions are not exclusive

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

258. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3174_SENT_READ_DEVICE_TRANSACTION_TO_NON_SYSTEM_DOMAIN**

Check that shareable read transactions are cacheable All domains other than SYSTEM must have ARCACHE[1] bit asserted.

DENALI_CDN_AXI_FATAL_ERR_VR_AXI3174_SENT_READ_DEVICE_TRANSACTION_TO_NON_SYSTEM_DOMAIN

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

259. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3175_SENT_WRITE_DEVICE_TRANSACTION_TO_NON_SYSTEM_DOMAIN**

Check that shareable write transactions are cacheable All domains other than SYSTEM must have AWCACHE[1] bit asserted.

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

260. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3176_WRITE_INCR_BURST_CROSSES_CACHE_LINE_BOUNDARY**

Check that WriteClean and WriteBack transactions don't cross the cache line boundary

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

261. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI4103_WRITE_WRAP_BURST_CROSSES_CACHE_LINE_BOUNDARY**

Check that WriteClean and WriteBack transactions don't cross the cache line boundary

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

262. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3177_BARRIER_READ_PORTION_COUNT_EXCEEDS_LIMIT**

Check that the total read portion barrier count does not exceed 256

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

263. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3178_BARRIER_WRITE_PORTION_COUNT_EXCEEDS_LIMIT**

Check that the total write portion barrier count does not exceed 256

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

264. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3180_SENT_MAINTENANCE_TRANSACTION_WHILE_HAVING_OUTSTANDING_TRANSACTIONS**

The Master can't start a maintenance transaction while a transaction which can result in a dirty line (ReadNotSharedDirty / ReadShared / ReadUnique / MakeUnique) is active. starting from 0.27 this is also TRUE for ReadOnce

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

265. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3181_SENT_WRITE_TRANSACTION_WHILE_HAVING_OUTSTANDING_MAINTENANCE_TRANSACTION**

The Master can't start any shareable write transaction when it has outstanding maintenance transactions to the same cache line

DENALI_CDN_AXI_FATAL_ERR_VR_AXI3181_SENT_WRITE_TRANSACTION_WHILE_HAVING_OUTSTANDING_MAINTENANCE_TRANSACTION

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

266. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3182_SENT_READ_TRANSACTION_WHILE_HAVING_OUTSTANDING_MAINTENANCE_TRANSACTION**

The Master can't start any shareable read transaction when it has outstanding maintenance transactions to the same cache line

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

267. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3185_SENT_WRITE_TRANSACTION_WHILE_HAVING_OUTSTANDING_MAINTENANCE_TRANSACTION**

Deprecated check! - The Master can't start any write transaction when it has outstanding maintenance transactions to the same cache line

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

268. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3183_SENT_SNOOP_RESPONSE_WHILE_OUTSTANDING_WRITEBACK_WRITECLEAN_IN_PROGRESS**

The Master can't respond to a snoop transaction if there are outstanding WriteBack or WriteClean transactions to the same line

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

269. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3184_SENT_WRITEBACK_WRITECLEAN_WHILE_OUTSTANDING_SNOOP_DATA_IS_IN_PROGRESS**

The Master can't start a WriteBack/WriteClean at the same time it respond to a snoop on the same cache line

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

270. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3187_SENT_WRITEBACK_WRITECLEAN_WHILE_OUTSTANDING_WRITEUNIQUE_WRITELINEUNIQUE_IS_IN_PROGRESS**

The Master can't start a WriteBack/WriteClean at the same time a WriteUnique/WriteLineUnique is active (it doesn't have to be to the same cache line)

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

271. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3188_SENT_WRITEUNIQUE_WRITELINEUNIQUE_WHILE_OUTSTANDING_WRITEBACK_WRITECLEAN_IS_IN_PROGRESS**

The Master can't start a WriteUnique/WriteLineUnique at the same time a WriteBack/WriteClean is active (it doesn't have to be to the same cache line)

DENALI_CDN_AXI_FATAL_ERR_VR_AXI3188_SENT_WRITEUNIQUE_WRITELINEUNIQUE_WHILE_OUTSTANDING_WRITEBACK_WRITECLEAN_IS_IN_PROGRESS

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

272. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3192_SENT_CACHEABLE_READ_BURST_TO_SYSTEM_DOMAIN**

Cacheable transactions, as indicated by ARCACHE[3:2] ! b'00, must not use ARDOMAIN b'11, SYSTEM

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

273. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3193_SENT_CACHEABLE_WRITE_BURST_TO_SYSTEM_DOMAIN**

Cacheable transactions, as indicated by AWCACHE[3:2] ! b'00, must not use AWDOMAIN b'11, SYSTEM

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

274. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3189_MASTER_CACHE_STATE_INCOSISTENCE_WITH_MODEL**

The Master's cache must be consistent with the user-defined cache model, if one was defined. the checking of the cache model is performed using a user hook method in the cache `check_cache_consistency_with_model()` which by default returns TRUE.

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

275. **DENALI_CDN_AXI_NONFATAL_ERR_VR_AXI3194_SENT_NON_CACHEABLE_READ_TRANSACTION_TO_NONE_SYSTEM_DOMAIN**

When Read_allocate and write_allocate are LOW, it is recommended that domain will be SYSTEM

Report on: Tx Rx *Severity: WARNING* *Callback: Yes* *Coverage: Yes*

276. **DENALI_CDN_AXI_NONFATAL_ERR_VR_AXI3195_SENT_NON_CACHEABLE_WRITE_TRANSACTION_TO_NONE_SYSTEM_DOMAIN**

When Read_allocate and write_allocate are LOW, it is recommended that domain will be SYSTEM

Report on: Tx Rx *Severity: WARNING* *Callback: Yes* *Coverage: Yes*

277. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3196_SENT_SHAREABLE_READ_BURST_WITH_WRONG_LENGTH**

All read transaction other then ReadOnce the length is limited to 1,2,4,8,16

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

278. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3197_SENT_SHAREABLE_WRITE_BURST_WITH_WRONG_LENGTH**

On Evict and WriteLineUnique write transaction the length is limited to 1,2,4,8,16

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

279. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3198_WRITE_BAD_START_ADDRESS_ON_WRAP_SHAREABLE_TRANSACTION**

Check that for WriteBack, WriteClean, WriteLineUnique WRAP transactions, the address is aligned to the AWSIZE, which is equal to the data bus width

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

280. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3199_READ_BAD_START_ADDRESS_ON_WRAP_SHAREABLE_TRANSACTION**

Check that for shareable READ WRAP transaction (other then ReadOnce), the address is aligned to the AxSIZE, which is equal to the data bus width

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

281. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI4102_SENT_WRITEBACK_WRITECLEAN_BIGGER_THAN_SIXTEEN_TRANSFERS**

When WriteBack, WriteClean INCR transactions are sent, they must be less than 16 transfers

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

282. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI4110_WRONG_SIGNALS_IN_DVM**

Check that when the burst is DVM, the normal signals fit the spec

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

283. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI4111_WRONG_ARADDR_IN_DVM_COMPLETE**

Check that when the burst is DVM_Complete, all the ARADDR bits are 0

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

284. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI4112_WRONG_ARADDR_IN_DVM_MESSAGE**

Check that when the burst is DVM_Message, the ARADDR signal has a legal value

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

285. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI4113_WRONG_CRRESP_IN_DVM_COMPLETE**

Check that when the snoop burst is DVM_Complete, the response is 0b00000

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

286. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI4114_WRONG_CRRESP_IN_DVM_SYNC**

Check that when the snoop burst is DVM Sync, the response is 0b00000

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

287. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI4115_WRONG_CRRESP_IN_DVM**

Check that when the snoop burst is DVM, the CRRESP is 'b00000 or 'b00010. (only 'b00000 for Hint)

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

288. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI4116_WRONG_ID_FOR_DVM**

Check that when the burst is DVM, the ID is not taken by other kinds of transactions: (Barrier and normal)

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

289. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI4117_WRONG_ID_FOR_READ_NON_DVM_NON_BARRIER**

Check that when a READ burst is NON DVM and NON BARRIER, the ID is not taken by other kinds of transactions: (BARRIER and DVM)

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

290. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI4118_WRONG_ID_FOR_WRITE_NON_DVM_NON_BARRIER**

Check that when the WRITE burst is NON BARRIER, the ID is not taken by BARRIER transactions

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

291. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI4119_MASTER_UNCOMPLETED_SYNC**

Check that each DVM sync snoop is answered by a DVM complete in a timely manner

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

292. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI4120_MULTI_DVM_ERROR**

Check that Multi part DVM transaction adheres to protocol: 1) the two parts are transmitted consecutively 2) they both have the same id tag 3) the second part does not have additional parts (dvm_has_va FALSE)

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

293. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI4121_MASTER_UNEXPECTED_COMPLETE_DVM**

Check that no DVM_Complete is transmitted unless there was a previous DVM sync snoop

DENALI_CDN_AXI_FATAL_ERR_VR_AXI4121_MASTER_UNEXPECTED_COMPLETE_DVM

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

294. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI4122_WRONG_READ_SNOOP_ON_EXCLUSIVE_SHAREABLE_READ_BURST**

Check that shareable read transactions are only ReadShared, ReadClean, CleanUnique

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

295. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI4123_EXCLUSIVE_SHAREABLE_STORE_STARTED_WHILE_SHAREABLE_BURSTS_IN_PROGRESS**

Check that shareable read store is not sent while other shareable load/store are active

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

296. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI4124_EXCLUSIVE_SHAREABLE_STORE_STARTED_THOUGH_EXCLUSIVE_MONITOR_WAS_RESET**

Deprecated check! - Check that shareable read store is not sent to an address for which the exclusive monitor is reset

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

297. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI4125_EXCLUSIVE_SHAREABLE_STORE_TO_UNIQUE_STATE**

Deprecated check! - Check that shareable read store is not sent to an address in Unique state

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

298. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI4126_EXCLUSIVE_SHAREABLE_LOAD_TO_UNIQUE_STATE**

Check that shareable read load is not sent to an address in Unique state

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

299. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI4127_SENT_DVM_SYNC_WHILE_OUTSTANDING_DVM_SYNC_WAITING_FOR_COMPLETE**

Check that no DVM sync is transmitted by the master if it already has outstanding DVM sync which was not answered yet by a DVM Complete snoop transaction

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

300. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3600_SNOOP_FILTER_BAD_SIGNAL_VALUE**

At the rising edge of ACLK all signals must be in logical values (i.e. 0 or 1)

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

301. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3601_SNOOP_FILTER_INCOSISTENT_SIGNALS**

At the rising edge of ACLK all signals on the AR/AW/W/R/B channels must be equels on both interfaces (_ma and _sl)

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

302. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3602_SENT_SNOOP_BURST_WHILE_WAITING_FOR_ACK**

The snoop filter can't send snoop burst to the master with the same cache line that has an active transaction on the ACK stage

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

303. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3603_SENT_SNOOP_BURST_WHILE_DATA_TRASFERS_ARE_SENT**

The snoop filter can't send snoop burst to the same cache line that has an active READ transaction where read transfers have started

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

304. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3604_SENDING_SNOOP_BURST_NOT_RECIEVED_FROM_INTERCONNECT**

The snoop filter can't send a snoop burst to the master if it didn't receive the same snoop burst previously from the interconnect

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

305. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3605_WRONG_RESP_TO_INTERCONNECT**

The snoop filter cannot send a snoop response to the interconnect to a cached address (snoop that was forwarded to the master) if it didn't receive that response from the master.

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

306. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3606_WRONG_RESP_TO_INTERCONNECT**

The snoop filter cannot send a snoop response to the interconnect to a filtered snoop request where the response is other than the defualt response.

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

307. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3608_WRONG_FILTERING_OF_SNOOP_AFTER_EVICTION**

The snoop filter should not pass a snoop burst to the master if the snoop address is not in the snoop filter internal array (warning)

DENALI_CDN_AXI_FATAL_ERR_VR_AXI3608_WRONG_FILTERING_OF_SNOOP_AFTER_EVICTION

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

308. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3607_WRONG_FILTERING_OF_SNOOP**

The snoop filter should not pass a snoop burst to the master if the snoop address is not in the snoop filter internal array (warning)

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

309. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3609_WRONG_FILTERING_OF_SNOOP_FOR_ALLOCATE_ADDRESS**

The snoop filter MUST forwared a snoop requests to the master for addresses allocated in the master cache. This check is performed when the response is sent back to the interconnect.

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

310. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3610_WRONG_FORWARDING_ORDER_OF_SNOOP**

The snoop filter MUST forwared a snoop requests to the master for addresses allocated in the master cache. This check fires when the snoop filter forwared the wrong snoop to the interconnect which mean that one or more snoops for allocated addresses was worngly filtered. maybe this check is not required (covered in check 3609)

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

311. **DENALI_CDN_AXI_FATAL_ERR_VR_AXI3611_SENDING_SNOOP_TRANSFER_NOT_RECIEVED_FROM_MASTER**

The snoop filter forwared a snoop data transfer to the interconnect but this transfer was not previously recieved from the cached master.

Report on: Tx Rx *Severity: ERROR* *Callback: Yes* *Coverage: Yes*

312. **DENALI_CDN_AXI_NONFATAL_ERR_VR_AXI000_WRONG_DIRECTION_BURST_RECEIVED_FROM_DRIVER**

Check that the direction of the burst received from the driver is needed. If it is return TRUE, o/w FALSE.

Report on: Tx Rx *Severity: WARNING* *Callback: Yes* *Coverage: Yes*

313. **DENALI_CDN_AXI_NONFATAL_WARNING_VR_AXI3190_WRITE_BURST_DOESNT_MATCH_THE_CACHE_STATE_AND_DISCARDED**

WARNING.. WRITE BURST DOESNT MATCH THE CACHE STATE AND DISCARDED. The master tried to send a Write burst that is illegal or doesn't match the current cache state

Report on: Tx Rx *Severity: WARNING* *Callback: Yes* *Coverage: Yes*

314. **DENALI_CDN_AXI_NONFATAL_WARNING_VR_AXI3191_READ_BURST_DOESNT_MATCH_THE_CACHE_STATE_AND_DISCARDED**

WARNING.. READ BURST DOESNT MATCH THE CACHE STATE AND DISCARDED. The master tried to send a Read burst that is illegal or doesn't match the current cache state

Report on: Tx Rx *Severity: WARNING* *Callback: Yes* *Coverage: Yes*

315. **DENALI_CDN_AXI_NONFATAL_WARNING_VR_AXI4190_WRITE_BURST_TO_ACTIVE_SHAREABLE_ADDRESS_DISCARDED**

WARNING.. WRITE BURST TO ACTIVE SHAREABLE ADDRESS DISCARDED. The master tried to send a Write burst to an active shareable address

Report on: Tx Rx *Severity: WARNING* *Callback: Yes* *Coverage: Yes*

316. **DENALI_CDN_AXI_NONFATAL_WARNING_VR_AXI4191_READ_BURST_TO_ACTIVE_SHAREABLE_ADDRESS_DISCARDED**

WARNING.. READ BURST TO ACTIVE SHAREABLE ADDRESS DISCARDED. The master tried to send a Read burst to an active shareable address

Report on: Tx Rx *Severity: WARNING* *Callback: Yes* *Coverage: Yes*

317. **DENALI_CDN_AXI_NONFATAL_WARNING_VR_AXI4192_WRITE_BURST_WITH_ACTIVE_BARRIER_ID_DISCARDED**

WARNING.. WRITE BURST WITH ACTIVE BARRIER ID DISCARDED. The master tried to send a normal Write burst which share id with current barrier transaction

Report on: Tx Rx *Severity: WARNING* *Callback: Yes* *Coverage: Yes*

318. **DENALI_CDN_AXI_NONFATAL_WARNING_VR_AXI4193_READ_BURST_WITH_ACTIVE_BARRIER_ID_DISCARDED**

WARNING.. READ BURST WITH ACTIVE BARRIER ID DISCARDED. The master tried to send a normal or DVM Read burst which share id with current barrier transaction

Report on: Tx Rx *Severity: WARNING* *Callback: Yes* *Coverage: Yes*

319. **DENALI_CDN_AXI_NONFATAL_WARNING_VR_AXI4194_DVM_SYNC_WHILE_OUTSTANDING_DVM_SYNC_IS_DISCARDED**

WARNING.. DVM_SYNC_WHILE_OUTSTANDING_DVM_SYNC_IS_DISCARDED. The master tried to send more than one outstanding DVM Sync transaction

Report on: Tx Rx *Severity: WARNING* *Callback: Yes* *Coverage: Yes*

320. **DENALI_CDN_AXI_NONFATAL_WARNING_VR_AXI4195_DVM_COMPLETE_DURING_MULTIPART_DVM_SEQ_IS_DISCARDED**

WARNING.. DVM_COMPLETE_DURING_MULTIPART_DVM_SEQ_IS_DISCARDED. The master tried to send a DVM Complete transaction between the two sections of multi-part DVM sequence

DENALI_CDN_AXI_NONFATAL_WARNING_VR_AXI4195_DVM_COMPLETE_DURING_MULTIPART_DVM_SEQ_IS_DISCARDED

Report on: Tx Rx *Severity: WARNING* *Callback: Yes* *Coverage: Yes*

8.5. Debugging the Simulation

Running a test case with the CDN_AXI VIP often results in the display of a few errors or informational messages. The CDN_AXI VIP can recover from some errors and continue the execution of the test case as intended. However, serious errors on the part of the DUT might place the model in an unrecoverable state and halt the simulation. This section describes how you can effectively debug your model.

The first step is to carefully read through each error message completely. These messages contain a lot of information, such as transaction fields, time of error, state in which the error occurred, which may give a clue as to what is wrong with the DUT.

8.5.1. Check the Run Summary

At the end of the test, a transaction summary is printed to the screen as shown below:

Figure 8.1. Transaction Summary

```
Checking the test ...
Checking is complete - 0 DUT errors, 0 DUT warnings.
[97000] cci_test.aceMasterMonitor2 MON: End of Test Summary: Write bursts sent: 0       Read bursts sent: 0     Snoops sent: 0
[97000] cci_test.aceMasterMonitor1 MON: End of Test Summary: Write bursts sent: 0       Read bursts sent: 0     Snoops sent: 0
[97000] cci_test.aceMasterMonitor0 MON: End of Test Summary: Write bursts sent: 0       Read bursts sent: 3     Snoops sent: 0
[97000] cci_test.aceSlaveDevice2 MON: End of Test Summary: Write bursts sent: 0 Read bursts sent: 0     Snoops sent: 0
[97000] cci_test.aceSlaveDevice1 MON: End of Test Summary: Write bursts sent: 0 Read bursts sent: 0     Snoops sent: 0
[97000] cci_test.aceSlaveDevice0 MON: End of Test Summary: Write bursts sent: 0 Read bursts sent: 3     Snoops sent: 0
[97000] cci_test.aceSlaveMonitor4 MON: End of Test Summary: Write bursts sent: 0       Read bursts sent: 1     Snoops sent: 1
[97000] cci_test.aceSlaveMonitor3 MON: End of Test Summary: Write bursts sent: 0       Read bursts sent: 0     Snoops sent: 2
[97000] cci_test.aceSlaveMonitor2 MON: End of Test Summary: Write bursts sent: 0       Read bursts sent: 0     Snoops sent: 0
[97000] cci_test.aceSlaveMonitor1 MON: End of Test Summary: Write bursts sent: 0       Read bursts sent: 0     Snoops sent: 0
[97000] cci_test.aceSlaveMonitor0 MON: End of Test Summary: Write bursts sent: 5       Read bursts sent: 1     Snoops sent: 0
[97000] cci_test.aceMasterDevice4 BFM: End of Test Summary: Discarded read burst : 0    Discarded write burst : 0
[97000] cci_test.aceMasterDevice4 MON: End of Test Summary: Write bursts sent: 0       Read bursts sent: 1     Snoops sent: 1
[97000] cci_test.aceMasterDevice3 BFM: End of Test Summary: Discarded read burst : 0    Discarded write burst : 0
[97000] cci_test.aceMasterDevice3 MON: End of Test Summary: Write bursts sent: 0       Read bursts sent: 0     Snoops sent: 2
[97000] cci_test.aceMasterDevice2 BFM: End of Test Summary: Discarded read burst : 0    Discarded write burst : 0
[97000] cci_test.aceMasterDevice2 MON: End of Test Summary: Write bursts sent: 0       Read bursts sent: 0     Snoops sent: 0
[97000] cci_test.aceMasterDevice1 BFM: End of Test Summary: Discarded read burst : 0    Discarded write burst : 0
[97000] cci_test.aceMasterDevice1 MON: End of Test Summary: Write bursts sent: 0       Read bursts sent: 0     Snoops sent: 0
[97000] cci_test.aceMasterDevice0 BFM: End of Test Summary: Discarded read burst : 0    Discarded write burst : 0
[97000] cci_test.aceMasterDevice0 MON: End of Test Summary: Write bursts sent: 5       Read bursts sent: 1     Snoops sent: 0
Wrote 1 cover_struct to vr_axi_psif_api_top_1.ecov
End-of-test operations are completed
```

You can view a summary for each device in the environment, and monitor whether it matches your expectations.

Note

If you see any bursts getting discarded, search the log to find the reason. For more information, see Section 12.2, “*CDN_AXI Specific Troubleshooting*”.

8.5.2. Adding Transaction Recording to the Waveform (NCSIM only)

You can add information about the transaction to the waveform.

To activate the transaction recording, write `1` to the `DENALI_CDN_AXI_REG_HasTrRecording register`.

To add the transaction to the waveform, do the following:

1. In the Design Browser, navigate to the simulator using **sys->>>psif_cdn_axi_env_list** in the left side menu.

   The **psif_cdn_axi_env_list** contains the list of all the VIP agents.

2. Navigate to **<agent_num>** >>> **active_masters/active_slaves** to view the related signal on the right side window.

   Figure 8.2. Agent List

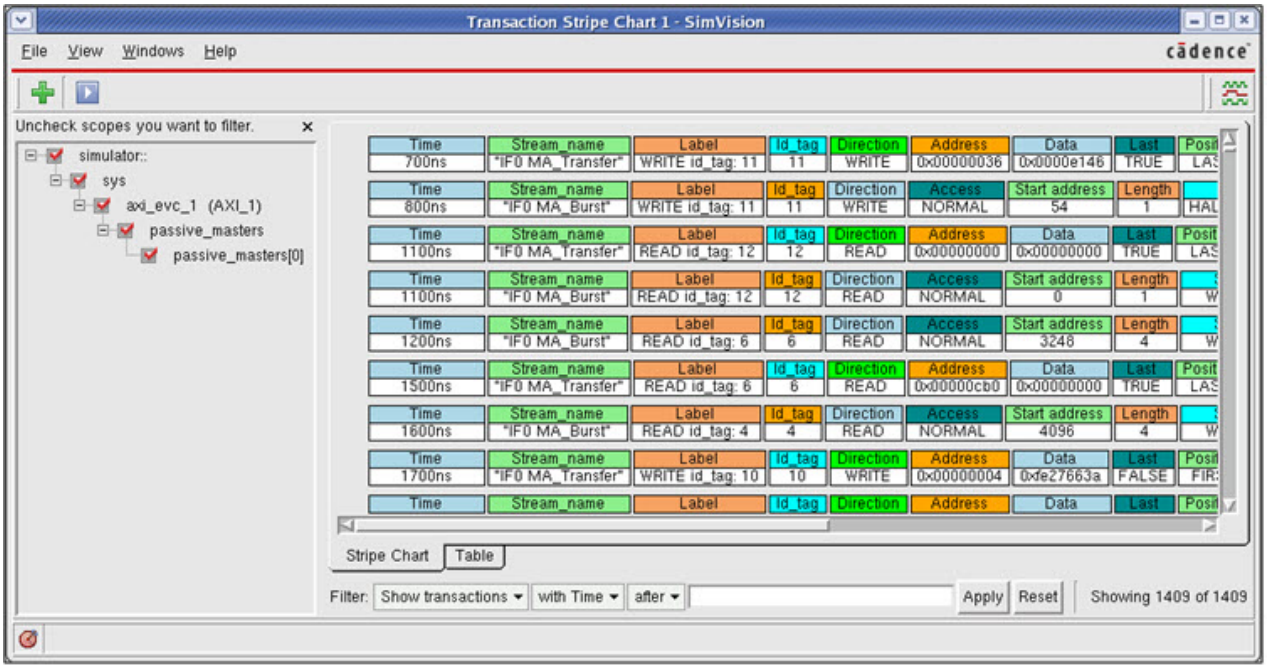
3. Right-click and select **Send to Waveform**.

8.5.3. Send Transaction Information to the Transaction Stripe Chart (NCSIM only)

1. Navigate to the simulator window using **Windows->>>New->>>Transaction Stripe Chart.**

2. Click on the green + icon at the top left corner, and select **All recorded transaction streams** to display the transaction table, as shown below.

   Figure 8.3. Transaction Stripe Chart



3. Select your preferred time in the table and click **OK** to view the selected transaction time.

Chapter 9. CDN_AXI Verification Test Scenarios

9.1. Test Flow

9.1.1. SystemVerilog UVM Test Flow

A typical SystemVerilog UVM test flow begins with writing a sequence. In the sequence, you have two options:

1. Use the SV constraints to randomize the transaction using `` `uvm_do() ``, `` `uvm_do_with() ``, or `` `uvm_rand_send() ``. When you use these macros, you can pass the transaction to the sequencer.

2. Set the transaction fields manually and pass the transaction to the sequencer using `` `uvm_send(). ``

The sequencer then passes the transaction to the CDN_AXI VIP. Before the CDN_AXI VIP passes the transaction to the wires, it activates the relevant `BeforeSend` callbacks (for more details, refer to Chapter 7, *CDN_AXI VIP Callbacks*). If you change the transaction in `BeforeSend`, you must use the `transSet()` function to pass the changes to the CDN_AXI VIP.

If you set `ModelGeneration` field, the CDN_AXI VIP will generate all the fields that you did not set.

Note

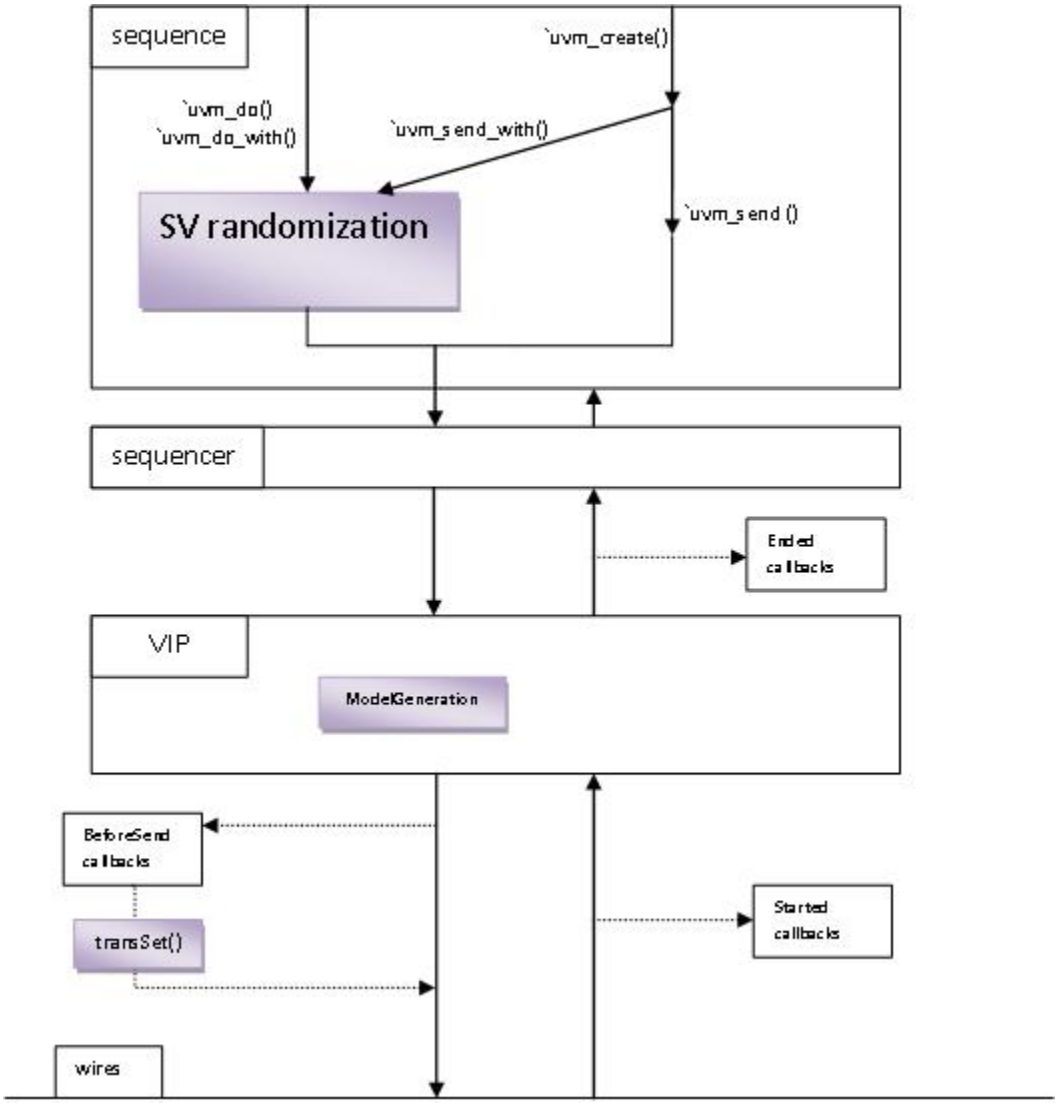
This is only relevant if you used `` `uvm_send() `` macro previously.

When the transaction gets to the wires, the model issues `Started` callbacks at the appropriate time.

When the transaction is ended, the model would issue the Ended callbacks.

Refer to the following figure that illustrates the overall SystemVerilog UVM test flow.

Figure 9.1. SystemVerilog UVM Test Flow



9.1.2. SystemVerilog Test Flow

A typical SystemVerilog test flow begins with writing a sequence. After you create a transaction using the `new()` command, you can manually edit the transaction or turn on or turn off SV constraints. You can then randomize the transaction using the `randomize()` function.

Once you are done with the transaction, pass it to the agent sequencer using the `transSet()` function.

The sequencer then passes the transaction to the CDN_AXI VIP. Before the CDN_AXI VIP passes the transaction to the wires, it activates the relevant `BeforeSend` callbacks (for more details, refer to Chapter 7, *CDN_AXI VIP Callbacks*). If you change the transaction in `BeforeSend`, you must use the `transSet()` function to pass the changes to the CDN_AXI VIP.

If you set the `ModelGeneration` field, the CDN_AXI VIP will generate all the fields that you did not set.

Note

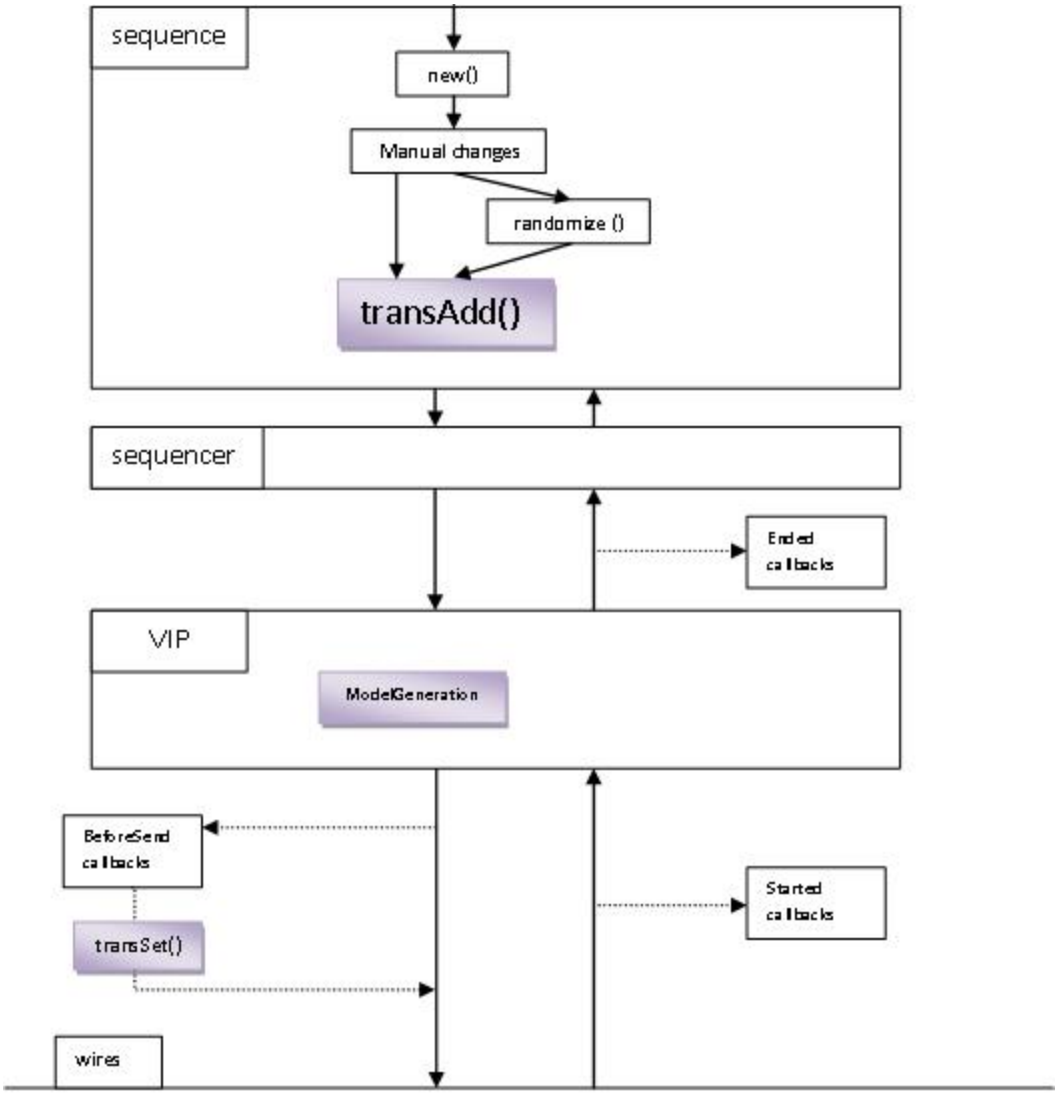
This is relevant only if you used the `randomize()` function previously.

When the transaction gets to the wires, the model issues `Started` callbacks at the appropriate time.

When the transaction is ended, the model issues the Ended callbacks.

The following figure illustrates the overall SystemVerilog test flow.

Figure 9.2. SystemVerilog Test Flow



9.2. Active Master Test Scenarios

The examples in the following sections work with the Slave DUT or Slave interface of an interconnect DUT.

9.2.1. Simple FIXED Write Burst

This example shows how to use `uvm_do_with to send a simple WRITE burst. The burst is FIXED, with Length 3 and size of half word. The rest of the values are randomized.

```
  `uvm_do_with(burst,
    {burst.Direction == DENALI_CDN_AXI_DIRECTION_WRITE;
    burst.Length == 3;
    burst.Size == DENALI_CDN_AXI_TRANSFERSIZE_HALFWORD;
    burst.Kind == DENALI_CDN_AXI_BURSTKIND_FIXED;
   });
```

Example of the output in the log:

```
UVM_INFO /myenv/cdnAxiUvmUserMasterDriver.sv(66) @ 1000:
uvm_test_top.axi_sve.myUvmEnv.activeMaster.driver [cdnAxiUvmUserMasterDriver]
Adding transaction to UVC Transmission Queue with Unique ID = 0
…
[1050] testbench.a_master BFM: Sending New WRITE burst: vr_axi_master_driven_burst-@293
kind:FIXED address:0xf2340a96012c2310 id:0x003f3 len:0x3 size:HALFWORD
```

9.2.2. Simple WRAP Read Burst

This example shows how to send a simple WRAP read burst. The burst start address is br ‘h3100 and it uses tag of 15 and size of byte.

```
    `uvm_do_with(burst,
    {burst.Direction == DENALI_CDN_AXI_DIRECTION_READ;
    burst.Kind == DENALI_CDN_AXI_BURSTKIND_WRAP;
    burst.StartAddress == 'h3100;
    burst.IdTag == 15;
    burst.Size == DENALI_CDN_AXI_TRANSFERSIZE_BYTE;
   });
```

Example of the output in the log:

```
UVM_INFO /myenv/axi3/cdnAxiUvmUserMasterDriver.sv(66) @ 4050:
uvm_test_top.axi_sve.myUvmEnv.activeMaster.driver [cdnAxiUvmUserMasterDriver]
Adding transaction to UVC Transmission Queue with Unique ID = 1
…
[4250] testbench.a_master BFM: Sending New READ burst: vr_axi_master_driven_burst-@317
kind:WRAP address:0x0000000000003100 id:0x0000f len:0x8
size:BYTE
```

9.2.3. Write Burst with Specific Data

This example shows how to send a random write burst with specific data.

Note that before assigning Data, the Data array must be properly sized

```
     `uvm_do_with(burst, {
               burst.Direction ==  DENALI_CDN_AXI_DIRECTION_WRITE;
```

```
                burst.Length == 4;
                burst.Kind == DENALI_CDN_AXI_BURSTKIND_INCR;
                burst.Size == DENALI_CDN_AXI_TRANSFERSIZE_HALFWORD;
                burst.IdTag == 4;
                burst.StartAddress == 'h2000;
                // Need to define the require amount of Data. The Data is an array of bytes, so
                // the size is number of byte in a transfer (2 for HALFWORD) * Length
                 burst.Data.size() == 8;
                burst.Data[0] == 'h0;
                burst.Data[1] == 'h11;
                burst.Data[2] == 'h22;
                burst.Data[3] == 'h33;
                burst.Data[4] == 'h44;
                burst.Data[5] == 'h55;
                burst.Data[6] == 'h66;
                burst.Data[7] == 'h77;
        })
```

Example of the output in the log:

```
UVM_INFO /myenv/axi3/cdnAxiUvmUserMasterDriver.sv(66) @ 6000:
uvm_test_top.axi_sve.myUvmEnv.activeMaster.driver [cdnAxiUvmUserMasterDriver]
Adding transaction to UVC Transmission Queue with Unique ID = 5
…
 [6050] testbench.a_master BFM: Sending New WRITE burst:
vr_axi_master_driven_burst-@349  kind:INCR address:0x0000000000002000 id:0x00004 len:0x4
size:HALFWORD
```

9.2.4. Write Non-Random Burst with Specific Data

This example shows how to send a non-random burst with specific fields set, including the data field .

```
  virtual task body();
      // allocate the burst using the common factory and initialize its properties
      `uvm_create(burst);
      // setting the required data items
      burst.Direction = DENALI_CDN_AXI_DIRECTION_WRITE;
      burst.Length = 3;
      burst.Size = DENALI_CDN_AXI_TRANSFERSIZE_BYTE;
      burst.Kind = DENALI_CDN_AXI_BURSTKIND_FIXED;
      // initializing the data according to the required Length
      burst.Data = new[3];
      burst.Data[0] = 'h12;
      burst.Data[1] = 'h34;
      burst.Data[2] = 'h56;
      burst.StartAddress = 'h1000;
      // sending the burst to the sequencer without allocation and randomization
      `uvm_send(burst);
  endtask : body
```

Example of the output in the log:

```
UVM_INFO /myenv/axi3/cdnAxiUvmUserMasterDriver.sv(66) @ 2000:
uvm_test_top.axi_sve.myUvmEnv.activeMaster.driver [cdnAxiUvmUserMasterDriver]
Adding transaction to UVC Transmission Queue with Unique ID = 1
…
[2050] testbench.a_master BFM: Sending New WRITE burst: vr_axi_master_driven_burst-@302
kind:FIXED address:0x0000000000001000 id:0x00009 len:0x3 size:BYTE
```

9.2.5. Read after Write Burst

This example shows how to send a Write burst with 4 transfers to address ‘h2000. After a short wait, it sends a Read burst to the same address with the same Length.

```
    `uvm_do_with(burst, {
              burst.Direction ==  DENALI_CDN_AXI_DIRECTION_WRITE;
              burst.Length == 4;
              burst.Kind == DENALI_CDN_AXI_BURSTKIND_INCR;
              burst.Size == DENALI_CDN_AXI_TRANSFERSIZE_HALFWORD;
              burst.StartAddress == 'h2000;
              // Need to define the require amount of Data. The Data is an array of bytes, so
              // the size is number of byte in a transfer (2 for HALFWORD) * Length
              burst.Data.size() == 8;
              burst.Data[0] == 'h0;
              burst.Data[1] == 'h11;
              burst.Data[2] == 'h22;
              burst.Data[3] == 'h33;
              burst.Data[4] == 'h44;
              burst.Data[5] == 'h55;
              burst.Data[6] == 'h66;
              burst.Data[7] == 'h77;
       })
       #5000;
       `uvm_do_with(burst, {
              burst.Direction ==  DENALI_CDN_AXI_DIRECTION_READ;
              burst.Length == 4;
              burst.Kind == DENALI_CDN_AXI_BURSTKIND_INCR;
              burst.Size == DENALI_CDN_AXI_TRANSFERSIZE_HALFWORD;
              burst.StartAddress == 'h2000;
       })
```

Example of the output in the log:

```
[6050] testbench.a_master BFM: Sending New WRITE burst: vr_axi_master_driven_burst-@341
kind:INCR address:0x0000000000002000 id:0x0099c len:0x4 size:HALFWORD
…
[11050] testbench.a_master BFM: Sending New READ burst: vr_axi_master_driven_burst-@370
kind:INCR address:0x0000000000002000 id:0x02c8d len:0x4 size:HALFWORD
```

9.2.6. Burst with Delays

In the CDN_AXI VIP, you can control the delays between bursts and transfers. The following example shows how to send a write burst with 0 delay on the AWVALID signal, and 0 delay on the WVALID signal before the first transfer. If there is no outstanding write burst, sending such a burst means that AWVALID and WVALID will be raised at the same cycle. If there are write transfers waiting to be sent, this will not happen.

For more information, refer to Section 6.3.4, *"Controlling Delays on AXI Channels"*.

```
       `uvm_do_with(burst, {
              burst.Direction ==  DENALI_CDN_AXI_DIRECTION_WRITE;
              burst.Length == 4;
              burst.Kind == DENALI_CDN_AXI_BURSTKIND_INCR;
              burst.Size == DENALI_CDN_AXI_TRANSFERSIZE_HALFWORD;
              burst.StartAddress == 'h9000;
              burst.TransmitDelay == 0;
              burst.TransfersChannelDelay[0] == 0;
       })
```

Example of the output in the log (sending the burst and the transfer at the same cycle):

```
[1050] testbench.a_master BFM: Sending New WRITE burst: vr_axi_master_driven_burst-@293
kind:INCR address:0x0000000000009000 id:0x05857 len:0x4size:HALFWORD
[1050] testbench.a_master BFM: Sending New WRITE transfer:
```

```
vr_axi_master_driven_transfer-@294  address:0x0000000000009000 id:0x05857 last:FALSE
```

9.2.7. Exclusive Burst

This example shows how to send an Exclusive read burst and then an Exclusive write burst.

Note

Sending an Exclusive write without a matching Exclusive read will result in an error.

```
        `uvm_create(burst);
         // This predefined constraint is constraining the Access field to be Normal
         burst.normal_access_const.constraint_mode(0);

         `uvm_rand_send_with(burst, {
                burst.Direction == DENALI_CDN_AXI_DIRECTION_READ;
                burst.Access == DENALI_CDN_AXI_ACCESS_EXCLUSIVE;
                burst.Kind == DENALI_CDN_AXI_BURSTKIND_INCR;
                burst.StartAddress == 'h7000;
                burst.IdTag == 10;
                burst.Size == DENALI_CDN_AXI_TRANSFERSIZE_WORD;
                burst.Cacheable == DENALI_CDN_AXI_CACHEMODE_CACHEABLE;
                burst.ReadAllocate == DENALI_CDN_AXI_READALLOCATE_NO_READ_ALLOCATE;
                burst.WriteAllocate == DENALI_CDN_AXI_WRITEALLOCATE_NO_WRITE_ALLOCATE;
                burst.Length == 2;
        });
        #10000;
        `uvm_create(burst);
         // This predefined constraint is constraining the Access field to be Normal
         burst.normal_access_const.constraint_mode(0);

        `uvm_rand_send_with(burst, {
                burst.Direction == DENALI_CDN_AXI_DIRECTION_WRITE;
                burst.Access == DENALI_CDN_AXI_ACCESS_EXCLUSIVE;
                burst.Kind == DENALI_CDN_AXI_BURSTKIND_INCR;
                burst.StartAddress == 'h7000;
                burst.IdTag == 10;
                burst.Size == DENALI_CDN_AXI_TRANSFERSIZE_WORD;
                burst.Cacheable == DENALI_CDN_AXI_CACHEMODE_CACHEABLE;
                burst.ReadAllocate == DENALI_CDN_AXI_READALLOCATE_NO_READ_ALLOCATE;
                burst.WriteAllocate == DENALI_CDN_AXI_WRITEALLOCATE_NO_WRITE_ALLOCATE;
                burst.Length == 2;
        });
```

Example of the output in the log:

```
20050] testbench.a_master BFM: Sending New READ burst: vr_axi_master_driven_burst-@403
kind:INCR address: 0x0000000000007000 id:10 len:2 size:WORD
…
[30050] testbench.a_master BFM: Sending New WRITE burst:
vr_axi_master_driven_burst-@426  kind:INCR address: 0x0000000000007000  id:10 len:2 size:WORD
```

Example of a related error message that might appear in the log file:

```
ERROR: Trying to send exclusive write when there was no exclusive read with the same id
```

9.2.8. Lock Burst

This example shows how to send a locked burst, and then send a normal burst with the same parameters to unlock the address.

To send a locked burst, you need to turn off the `normal_access_const` constraint. This constraint is setting the `Access` field to be not Exclusive and not Locked.

```
 // The following variables are defined outside the body. We will use them
 // to ensure that the second burst is using the same parameters as the first burst.
 rand reg [19:0] idTag;
 denaliCdn_axiPrivilegedModeT privileged;
 denaliCdn_axiSecureModeT secure;
 denaliCdn_axiFetchKindT dataInstr;
 denaliCdn_axiBufferModeT bufferable;
 denaliCdn_axiCacheModeT cacheable;
 denaliCdn_axiReadAllocateT readAllocate;
 denaliCdn_axiWriteAllocateT writeAllocate;

  …

      `uvm_create(burst);
  // This predefined constraint is constraining the Access field to be Normal
  burst.normal_access_const.constraint_mode(0);

  `uvm_rand_send_with(burst,
          {burst.Access == DENALI_CDN_AXI_ACCESS_LOCKED;
          burst.StartAddress == 'h3000;
  });

  bufferable = burst.Bufferable;
  cacheable = burst.Cacheable;
  readAllocate = burst.ReadAllocate;
  writeAllocate = burst.WriteAllocate;
  privileged = burst.Privileged;
  secure = burst.Secure;
  dataInstr = burst.DataInstr;
  idTag = burst.IdTag;

  #5000;

  `uvm_rand_send_with(burst,
          {burst.Access == DENALI_CDN_AXI_ACCESS_NORMAL;
          burst.StartAddress == 'h3000;
          burst.Bufferable == bufferable;
          burst.Cacheable == cacheable;
          burst.ReadAllocate == readAllocate;
          burst.WriteAllocate == writeAllocate;
          burst.Privileged == privileged;
          burst.Secure == secure;
          burst.DataInstr == dataInstr;
          burst.IdTag == idTag;
  });

  burst.normal_access_const.constraint_mode(1);
```

Example of the output in the log:

```
[18450] testbench.a_master BFM: Sending New READ burst: vr_axi_master_driven_burst-@404
kind:INCR address:0x0000000000003000 id:0x026e9 len:0x9 size:FOUR_WORDS
…
[19650] testbench.a_slave BFM: Sent READ transfer vr_axi_slave_driven_transfer_response-@407
address:0x0000000000003000 id:0x026e9 last:FALSE resp:OKAY
```

Examples of related error message that might appear in the log file:

```
ERROR: Trying to send exclusive burst while in a locked transaction

ERROR: burst id tag: 0x06a59 is different from current locked transaction id: 0x026e9
```

```
WARNING: burst start address: 0x0000000000007000 is not in the same4K region as the current
locked transaction(start of block): 0x0000000000003000
```

9.2.9. Unaligned Transfers

The CDN_AXI VIP supports unaligned transfers. For any burst that is made of data transfers wider than one byte, the first bytes accessed might be unaligned with the natural address boundary. For example, a 32-bit data packet that starts at a byte address of 0x1002 is not aligned to the natural 32-bit address boundary.

By default, the CDN_AXI VIP will not send unaligned transfers based on a built-in SV constraint. To enable the VIP to generate such transfers, you must turn-off the `constraintAligned` constraint:

```
masterBurst.constraintAligned.constraint_mode(0);
```

The following two examples show how to send unaligned transfers using SV UVM.

Example 1: Sending non-random unaligned write burst.

Note

All unspecified fields will be set by the VIP core.

```
`uvm_create(masterBurst);
masterBurst.constraintAligned.constraint_mode(0);
masterBurst.Direction = DENALI_CDN_AXI_DIRECTION_WRITE;
masterBurst.StartAddress = 'h2;
masterBurst.ModelGeneration = 1;
`uvm_send(masterBurst);
```

Example 2: Sending unaligned write burst using `uvm_do_with`

```
`uvm_create(masterBurst);
masterBurst.constraintAligned.constraint_mode(0);
`uvm_rand_send_with(masterBurst, {
   masterBurst.StartAddress == 'h7;
   masterBurst.Direction == DENALI_CDN_AXI_DIRECTION_WRITE;
});
```

The following two examples show how to send unaligned transfers using SV.

Example 1: Sending non-random unaligned write burst.

Note

All unspecified fields will be set by the VIP core.

```
  masterBurst = new();
  masterBurst.constraintAligned.constraint_mode(0);
  masterBurst.Direction = DENALI_CDN_AXI_DIRECTION_WRITE;
  masterBurst.StartAddress = 'h2;
  masterBurst.ModelGeneration = 1;
  status = axiActiveMaster.transAdd(masterBurst, DENALI_CDN_AXI_QUEUE_Burst);
```

Example 2: Sending unaligned write burst using randomize.

```
masterBurst = new();
```

```
masterBurst.constraintAligned.constraint_mode(0);
  assert(masterBurst.randomize() with {
    StartAddress == 'h7;
    Direction == DENALI_CDN_AXI_DIRECTION_WRITE;
  });
status = axiActiveMaster.transAdd(masterBurst, DENALI_CDN_AXI_QUEUE_Burst);
```

Note

As per the *AMBA AXI Specification*, the unaligned transactions are relevant only for transactions of type INCR and FIXED.

9.2.10. Unaligned Data

The CDN_AXI VIP enables you to send transactions with unaligned data (that is, the data in the last transfer of the transaction can be partial). In that case, the strobe of the last transfer will be partial.

For example:

```
Data width 64bit, strobe 8bit
Address 0x100, Length 2, Size TWO_WORDS (8 bytes), kind INCR
Data: 9d d0 59 0f 91 76 41 ec 17 d5 72 63 (12 bytes)
```

The strobes will be:

1st transfer: 0xff

2nd transfer: 0x0f

By default the CDN_AXI VIP will not generate or allow such transactions, based on a built-in SV constraint. To enable this VIP behavior, you must turn off the `constraintAligned` constraint:

```
masterBurst.constraintAligned.constraint_mode(0);
```

When this constraint is turned off, the CDN_AXI VIP will generate and allow randomizing such transactions for INCR type only.

You can enable this behavior for WRAP and FIXED transactions by using `` `uvm_create() `` or `new()`.

Note the following:

- If you choose to create your transactions using `` `uvm_create `` or `new()` and you are setting the data in the `Data` field, you must provide the necessary fields:
  - StartAddress
  - Size
  - Kind

  If these fields are not set, the transaction will be dropped.

- You must provide the minimal number of Data bytes based on the fields you set. That is, the last transfer should have at least one valid byte in it.

  For example:

  For INCR burst of length 3, address 0x0, Size WORD, there should be at least 9 bytes of Data. For an INCR burst of length 3, address 0x2, Size WORD, there should be at least 7 bytes of data. If the `Data` field is set but not enough data is provided the transaction will be dropped.

The following three examples show legal transactions (data width is 256 bits) using SV UVM.

Example 1: INCR burst with unaligned address and unaligned data.

```
`uvm_create(masterBurst);
masterBurst.constraintAligned.constraint_mode(0);
uvm_rand_send_with(masterBurst, {
   masterBurst.Direction == DENALI_CDN_AXI_DIRECTION_WRITE;
   masterBurst.StartAddress == 2;
   masterBurst.Length == 3;
   masterBurst.Size == DENALI_CDN_AXI_TRANSFERSIZE_WORD;
   masterBurst.Kind == DENALI_CDN_AXI_BURSTKIND_INCR;
   masterBurst.ModelGeneration == 1;
   masterBurst.Data.size() == 8;
   masterBurst.Data[0] == 'hd;
   masterBurst.Data[1] == 'he;
   masterBurst.Data[2] == 'hf;
   masterBurst.Data[4] == 'h11;
   masterBurst.Data[5] == 'h12;
   masterBurst.Data[6] == 'h13;
   masterBurst.Data[7] == 'h14;
   });
```

Strobes will be: 0000000c 000000f0 00000300

Example 2: WRAP burst with unaligned data.

```
`uvm_create(masterBurst);
masterBurst.constraintAligned.constraint_mode(0);
masterBurst.Direction = DENALI_CDN_AXI_DIRECTION_WRITE;
masterBurst.StartAddress = 'h10;
masterBurst.Length = 2;
masterBurst.Size = DENALI_CDN_AXI_TRANSFERSIZE_FOUR_WORDS;
masterBurst.Kind = DENALI_CDN_AXI_BURSTKIND_WRAP;
masterBurst.Data = new[20];
for (int ii=0;ii<masterBurst.Data.size();ii++) begin
   masterBurst.Data[ii] = ('hd)+ii;
end
masterBurst.ModelGeneration = 1;
`uvm_send(masterBurst);
```

Strobes will be: ffff0000 0000000f

Example 3: FIXED burst with unaligned address and unaligned data.

```
`uvm_create(masterBurst);
masterBurst.constraintAligned.constraint_mode(0);
masterBurst.Direction = DENALI_CDN_AXI_DIRECTION_WRITE;
masterBurst.StartAddress = 2;
masterBurst.Length = 3;
masterBurst.Size = DENALI_CDN_AXI_TRANSFERSIZE_WORD;
```

```
masterBurst.Kind = DENALI_CDN_AXI_BURSTKIND_FIXED;
masterBurst.Data = new[5];
for (int ii=0;ii<masterBurst.Data.size();ii++) begin
   masterBurst.Data[ii] = ('hd)+ii;
end
//all fields that were not set by the user will be randomize in the
//VIP core. If you use `uvm_rand_send, then all the fields will be
//randomize according to the SV constraints
masterBurst.ModelGeneration = 1;

`uvm_send(masterBurst);
```

Strobes will be: 0000000c 0000000c 00000004

The following three examples show legal transactions (data width is 256 bits) using SV.

Example 1: INCR burst with unaligned address and unaligned data.

```
masterBurst = new();
masterBurst.constraintAligned.constraint_mode(0);
assert(masterBurst.randomize() with {
   Direction == DENALI_CDN_AXI_DIRECTION_WRITE;
   StartAddress == 2;
   Length == 3;
   Size == DENALI_CDN_AXI_TRANSFERSIZE_WORD;
   Kind == DENALI_CDN_AXI_BURSTKIND_INCR;
   ModelGeneration == 1;
   Data.size() == 8;
   Data[0] == 'hd;
   Data[1] == 'he;
   Data[2] == 'hf;
   Data[3] == 'h10;
   Data[4] == 'h11;
   Data[5] == 'h12;
   Data[6] == 'h13;
   Data[7] == 'h14;
 });
status = axiActiveMaster.transAdd(masterBurst, DENALI_CDN_AXI_QUEUE_Burst);
```

Strobes will be: 0000000c 000000f0 00000300

Example 2: WRAP burst with unaligned data.

```
masterBurst = new();
masterBurst.constraintAligned.constraint_mode(0);
masterBurst.Direction = DENALI_CDN_AXI_DIRECTION_WRITE;
masterBurst.StartAddress = 'h10;
masterBurst.Length = 2;
masterBurst.Size = DENALI_CDN_AXI_TRANSFERSIZE_FOUR_WORDS;
masterBurst.Kind = DENALI_CDN_AXI_BURSTKIND_WRAP;
masterBurst.Data = new[20];
for (int ii=0;ii<masterBurst.Data.size();ii++) begin
   masterBurst.Data[ii] = ('hd)+ii;
end
masterBurst.ModelGeneration = 1;
status = axiActiveMaster.transAdd(masterBurst, DENALI_CDN_AXI_QUEUE_Burst);
```

Strobes will be: ffff0000 0000000f

Example 3: FIXED burst with unaligned address and unaligned data.

```
masterBurst = new();
masterBurst.constraintAligned.constraint_mode(0);
```

```
masterBurst.Direction = DENALI_CDN_AXI_DIRECTION_WRITE;
masterBurst.StartAddress = 2;
masterBurst.Length = 3;
masterBurst.Size = DENALI_CDN_AXI_TRANSFERSIZE_WORD;
masterBurst.Kind = DENALI_CDN_AXI_BURSTKIND_FIXED;
masterBurst.Data = new[5];
for (int ii=0;ii<masterBurst.Data.size();ii++) begin
   masterBurst.Data[ii] = ('hd)+ii;
end
//all the fields that were not set by the user will be randomize in the
//VIP core. If you use `uvm_rand_send, then all the fields will be
//randomize according to the SV constraints
masterBurst.ModelGeneration = 1;

status = axiActiveMaster.transAdd(masterBurst, DENALI_CDN_AXI_QUEUE_Burst);
```

Strobes will be: 0000000c 0000000c 00000004

9.3. Active Slave Test Scenarios

The examples in the following sections work with Master DUT or Master interface of an interconnect DUT.

9.3.1. Response with BVALID Slave Delay

This example shows how to send a response to a write transaction with delay on the BVALID signal.

For more details, refer to Section 6.3.4, *"Controlling Delays on AXI Channels"*.

```
   `uvm_do_with(burst, {
        burst.Direction == DENALI_CDN_AXI_DIRECTION_WRITE;
        burst.ChannelDelay == 5;
    });
```

Example of the output in the log (note that the transaction was added to the slave response queue at time 1000):

```
[6150] testbench.a_slave BFM: Sending Response for Address phase to WRITE burst:
vr_axi_slave_driven_burst_response-@298  kind:INCR address:36864
id:22615 len:4 size:HALFWORD
```

9.3.2. How to Control Data in Slave Read Responses

There are three (3) ways to control data in slave read responses:

- Pre-populate the slave's sparse memory with a specific data pattern using backdoor writes as explained in Section 6.1.2.2, *"Backdoor Reads and Writes to Slave Memory"*.

  When the slave receives the read transaction, it returns a response with the data taken from the sparse memory model.

- Intercept the slave response in the `BeforeSendResponse` callback, modify the Data field, and put the modified response back on the stack for transmission using `transSet()`.

- Define the slave response with specific data, and place it in the slave's transmission queue.

Make sure to set the `IgnoreConstraints` field to zero, as shown in the code snippet below. When the slave receives the read transaction, it will get the response from the transmission queue, then transmit it.

```
  `uvm_info(`gtn, "Pre-loading Slave Response", UVM_LOW);
     `uvm_create_on(slaveResp, p_sequencer.slave_seqr);

     `uvm_rand_send_with(slaveResp, {
               slaveResp.Direction == DENALI_CDN_AXI_DIRECTION_READ;
               slaveResp.Data.size() == 4;
               slaveResp.Data[0] == 'h12;
               slaveResp.Data[1] == 'h34;
               slaveResp.Data[2] == 'h56;
               slaveResp.Data[3] == 'h78;
               slaveResp.IgnoreConstraints == 0;
      })
```

9.4. Built-in SV Constraints

The CDN_AXI VIP has some built-in SV constraints to control the basic generation in the VIP. The SV constraints are located in `$DENALI/ddvapi/sv/denaliCdn_axi.sv`

You can turn *OFF* any of the built-in constraints by using `constraint_mode(0)` or turn *ON* a constraint by using `constraint_mode(1)`.

List of AXI Built-in SV Constraints

```
Arraysizes_const_TransfersChannelDelay :
Keep TransfersChannelDelay.size == Length
Internal constraints, it is not recommended to disable this constraint

Arraysizes_const_TransfersResp :
Keep TransfersResp.size == Length
Internal constraints, it is not recommended to disable this constraint

BurstSize_const :
Keep auxiliary var BurstSize to correct Size
Internal constraints, it is not recommended to disable this constraint

ConstraintType :
Keep basic burst fields to supported values
Internal constraints, it is not recommended to disable this constraint

DisableWriteAddressOffset_const :
Keep WriteAddressOffset == 0
User should disable this constraint to send write data before address bursts

Drop_delay :
This constraint shouldn't be disabled

TransfersChannelDelay_defaultTiming_const :
Keep the delay values in TransfersChannelDelay < 10
User should disable this constraint to use larger values for TransfersChannelDelay

TransmitDelay_const :
Keep TransmitDelay == 0
User should disable this constraint for using different TransmitDelay values

WriteAddressOffset_const :
Keep the number of data transfers sent before address phase (WriteAddressOffset) < Length
Internal constraints, it is not recommended to disable this constraint
```

```
alen_const :
Keep Alen singal to be (burst Length -1)
Internal constraints, it is not recommended to disable this constraint

allowresp_change_const :
Keep AllowRespChange == 0 for Non-Exclusive bursts

aready_delay_const :
Keep AreadyControl and AddressDelay in supported values
Internal constraints, it is not recommended to disable this constraint

axi4_lite_const :
Keep correct burst fields for AXI4 LITE
Internal constraints, it is not recommended to disable this constraint

bready_control_const :
Keep BreadyControl and BreadyDelay in supported values
Internal constraints, it is not recommended to disable this constraint

burst_max_size_const :
Keep burst Size <= BurstMaxSize
Internal constraints, it is not recommended to disable this constraint

constraint4k_boundry :
Keep bursts not crossing 4K boundry
Internal constraints, it is not recommended to disable this constraint

constraintAligned :
Keep bursts with StartAddress aligned
User should disable this constraint for sending unaligned bursts

constraintAttributes :
Keep correct Cacheable, ReadAllocate and WriteAllocate values
Internal constraints, it is not recommended to disable this constraint

constraintBurstBurstSize :
Keep burst Size in supported values
Internal constraints, it is not recommended to disable this constraint

constraintBurstLength :
Keep correct burst Length values based on:
AXI3 --> length <=16
WRAP bursts --> Length in [2,4,8,16], FIXED bursts --> Length <=16
EXCLUSIVE bursts --> Length <=16
Default --> 0 < Length <= 256

constraintCombo :
Keep StartAddress aligned for Exclusive bursts
Internal constraints, it is not recommended to disable this constraint

constraintData :
Keep Data.size == DataByteLength
Internal constraints, it is not recommended to disable this constraint

constraintDataByteLength :
Keep auxiliary var DataByteLength in correct values
Internal constraints, it is not recommended to disable this constraint

constraintUnaligned :
Keep correct StartAddress, Size and Length when sending unaligned bursts

delay_timing_const_AddressDelay :
Keep AddressDelay distribution default mostly in [0:5] values

delay_timing_const_BreadyDelay :
Keep BreadyDelay distribution default mostly in [0:5] values
```

```
delay_timing_const_ChannelDelay :
Keep ChannelDelay distribution default mostly in [0:5] values

delay_timing_const_TransmitDelay :
Keep TransmitDelay distribution default mostly in [0:10] values

fast_di_const :
Keep ModelGeneration == 0 by default when using SV randomization

no_ace_const :
Disable ACE related transaction types when in AXI3 or AXI4
Internal constraints, it is not recommended to disable this constraint

no_exokay_response :
Do not send EXOKAY responses to Non-Exclusive bursts
Internal constraints, it is not recommended to disable this constraint

normal_access_const :
Keep burst Access to NORMAL
User should disable this constraint to send EXCLUSIVE and LOCKED bursts

region_4_bit :
Keep burst Region <= 15

region_const :
Keep burst Region == 0

rready_control_const :
Keep RreadyControl and TransfersChannelDelay in supported values
Internal constraints, it is not recommended to disable this constraint

wready_control_const :
Keep WreadyControl and TransfersChannelDelay in supported values
Internal constraints, it is not recommended to disable this constraint
```

Example 9.1. AXI Locked and Exclusive built-in constraint.

By default the CDN_AXI VIP will not send exclusive and locked bursts based on the following constraint:

```
constraint normal_access_const {
      Access != DENALI_CDN_AXI_ACCESS_EXCLUSIVE;
      Access != DENALI_CDN_AXI_ACCESS_LOCKED;
}
```

To turn off this constraint and have the VIP generate all types of Access:

```
masterBurst.normal_access_const.constraint_mode(0);
```

Example 9.2. AXI Write Data before address built-in constraint

By default, the CDN_AXI VIP will not generate write data before address transactions, based on the following constraint:

```
constraint DisableWriteAddressOffset_const {
    WriteAddressOffset == 0;
}
```

To turn off this constraint and have the VIP generate write data before address transactions:

```
masterBurst.DisableWriteAddressOffset_const.constraint_mode(0);
```

9.5. Error Injection

By default, transactions will be generated with no errors. The CDN_AXI VIP lets you inject errors through `transSet()` inside the different callbacks. When using `transSet()` in callbacks to modify field values in a transaction, the values will not be checked for validity before being sent.

Therefore, you can use this mechanism to purposely inject errors or set the values in `transSet()` correctly if you are not interested in error injection.

9.5.1. Injecting Wrong Strobe on a WRITE Transfer Using BeforeSendTransfer Callback

The following example shows how to inject wrong strobe on a WRITE transfer using the `BeforeSendTransfer` callback:

```
// Set the BeforeSendTransfer callback for the ACTIVE Master
void'(axiActiveMaster.setCallback(DENALI_CDN_AXI_CB_BeforeSendTransfer));

virtual function int BeforeSendTransferCbF(ref denaliCdn_axiTransaction trans);
if (trans.Type == DENALI_CDN_AXI_TR_WriteTransfer) begin
  trans.Strobe = 'h0f00ffff;
  void'(trans.transSet());
 end
return super.BeforeSendTransferCbF(trans);
endfunction

//send one transaction
masterBurst.Size = DENALI_CDN_AXI_TRANSFERSIZE_FOUR_WORDS;
masterBurst.Type = DENALI_CDN_AXI_TR_Write;
masterBurst.Direction = DENALI_CDN_AXI_DIRECTION_WRITE;
status = axiActiveMaster.transAdd(masterBurst,DENALI_CDN_AXI_QUEUE_Burst);
```

Error result: `ERR_VR_AXI158_WRONG_WSTRB` since the `wstrb` enabled bytes are outside the valid range (5 bytes instead of 4 bytes).

9.5.2. Injecting Wrong Length on a WRITE Burst Using BeforeSend Callback

The following example shows how to inject wrong length on a WRITE burst using the `BeforeSend` callback:

```
// Set the BeforeSend callback for the ACTIVE Master
void'(axiActiveMaster.setCallback(DENALI_CDN_AXI_CB_BeforeSend));

virtual function int BeforeSendCbF(ref denaliCdn_axiTransaction trans);
if (trans.Type == DENALI_CDN_AXI_TR_Write) begin
  trans.Length = 10;
  void'(trans.transSet());
 end
return super.BeforeSendCbF(trans);
endfunction

//send one transaction
masterBurst.Type = DENALI_CDN_AXI_TR_Write;
masterBurst.Direction = DENALI_CDN_AXI_DIRECTION_WRITE;
masterBurst.Length = 4;
status = axiActiveMaster.transAdd(masterBurst, DENALI_CDN_AXI_QUEUE_Burst);
```

Error result: `ERR_VR_AXI125_INVALID_WLAST` since the master drove `WLAST` high in the 4th transfer while the length is 10.

9.5.3. Injecting Wrong Transfer Response on a READ Burst Using BeforeSendResponse Callback

The following example shows how to inject a wrong transfer response on a READ burst using the BeforeSendResponse callback:

```
// Set the BeforeSendResponse callback for the Active Slave
void'(axiActiveSlave.setCallback(DENALI_CDN_AXI_CB_BeforeSendResponse));

virtual function int BeforeSendResponseCbF(ref denaliCdn_axiTransaction trans);
 if (trans.Type == DENALI_CDN_AXI_TR_ReadData) begin
  trans.TransfersResp = new[3];
  trans.TransfersResp[0] = DENALI_CDN_AXI_RESPONSE_OKAY;
  trans.TransfersResp[1] = DENALI_CDN_AXI_RESPONSE_DECERR;
  trans.TransfersResp[2] = DENALI_CDN_AXI_RESPONSE_OKAY;
  void'(trans.transSet());
 end
 return super.BeforeSendResponseCbF(trans);
endfunction

//send one transaction
masterBurst.StartAddress = 0;
masterBurst.Type = DENALI_CDN_AXI_TR_Read;
masterBurst.Direction = DENALI_CDN_AXI_DIRECTION_READ;
masterBurst.Length = 3;
status = axiActiveMaster.transAdd(masterBurst, DENALI_CDN_AXI_QUEUE_Burst);
```

Error result: ERR_VR_AXI231_READ_TRANSFER_MAPPED_ADDRESS_AND_DECERR since the slave received a read transfer to a mapped address - 0 - but responded with DECERR.

Chapter 10. CDN_AXI VIP Testbench Integration

This chapter describes the CDN_AXI VIP integration with supported testbench interfaces and simulators. Currently, only SystemVerilog is supported.

The CDN_AXI VIP provides a native class-based object-oriented interface to support SystemVerilog testbench methodologies, such as UVM and OVM.

- UVM - Cadence provides the UVM layer for CDN_AXI. The UVM layer is located at `$CDN_VIP_ROOT/tools/denali/ddvapi/sv/uvm/cdn_axi`. For details on getting started with the UVM Layer, refer to Section 10.3, *"SystemVerilog Interface for UVM"*. This section also provides an example test case.
- OVM - Cadence provides the OVM layer for CDN_AXI. The OVM layer is located at `$CDN_VIP_ROOT/tools/denali/ddvapi/sv/ovm/cdn_axi`. For a description of each component in the OVM Layer, refer to the UVM Layer description at Section 10.3, *"SystemVerilog Interface for UVM"*. You may use the UVM example as a reference for OVM provided you change the UVM specific naming conventions and syntax to OVM specific naming conventions and syntax.

10.1. Simulator Integration

The VIPCAT installation provides a set of utility scripts in `$CDN_VIP_ROOT/bin`. These scripts set up your environment for VIP simulation for various simulators and provide key simulator-specific options to integrate the VIP into your overall simulation setup, using your scripts or Makefiles.

If you are using Incisive Enterprise Simulator (IES) for your simulation environment, then you can also use *irun* to easily run a test case with the VIP models. Starting from IES product release 12.1, *irun* recognizes the VIP models from their instantiation interface and handles most of the invocation details for you.

10.1.1. VIP Scripts

The VIP scripts need to know the location of the VIPCAT installation. Throughout these instructions, the environment variable `CDN_VIP_ROOT` is used to refer to the path of the VIP installation. Although you are not required to set this environment variable, you are encouraged to do so. Otherwise, you need to specify the `-cdn_vip_root` option for all scripts.

Before running these scripts, you must ensure that you have set up all required third-party tools available in the environment (for example, the simulator, GCC, and so on).

The VIP scripts provide the following functions:

- Environment Setup (cdn_vip_setup_env)

Generates shell commands that set up the simulation environment. This script needs to know the VIPCAT installation path, the specific simulator, and the testbench methodology - either SystemVerilog with UVM layer, or basic SystemVerilog.

- Example Setup (cdn_vip_setup_example)

  Generates simulator-specific commands to run a VIP example from the release. This script needs to know the specific example being set up, along with the other key environment components, such as VIPCAT installation, specific simulator, and so forth.

  Note

  All scripts have an `-h` option that provides information on arguments to the script. For example, `cdn_vip_check_env -h` lists detailed information on each argument to the script.

10.1.1.1. cdn_vip_setup_env

The `cdn_vip_setup_env` script generates a shell file with necessary commands to set up your environment. This script requires the information described below:

- The VIPCAT installation directory is expected to be specified with the CDN_VIP_ROOT environment variable. Or, it must be explicitly specified with the `-cdn_vip_root` optional argument to the script.
- The simulator must be explicitly specified with the `-sim` argument to the script. The script is capable of setting up the environment for the following simulators: NCSIM (multi-step mode), NCSIM (single-step mode), VCS, and MTI. The corresponding simulator executables (ncsim, vcs, vlog) must be available in the path environment variable. Or, the simulator installation directory can be explicitly specified with the `-sim_root` optional argument to the script.
- The testbench methodology must be explicitly specified with the `-method` argument to the script. The script is capable of setting up the environment for SystemVerilog with UVM Layer or basic SystemVerilog.
- NCSIM users only: The user directory containing VIP-compiled libraries must be explicitly specified with the `-cdn_vip_lib` option to the script.
- NCSIM users only: The `-install` option must be specified in the following three scenarios:
  - First VIP download
  - Download of updated VIP (with additional functionality, etc.)
  - Upgrade of Cadence simulator installation

  This option installs the VIP Compiled libraries necessary for NCSIM in the specified user directory (specified with `-cdn_vip_lib <VIP_Compiled_Library_Directory>`).

10.1.1.1.1. Completing Environment Setup

`cdn_vip_setup_env` generates Bourne shell scripts `cdn_vip_env_<...>.sh` by default. If you use this shell (/bin/sh) or its derivatives, such as Bash (/bin/bash) or Z (/bin/zsh), you need to execute the following command to complete your environment setup (on your command-line prompt, or, in your script):

```
. cdn_vip_env_<...>.sh
```

If you use C shell (/bin/csh) or its derivatives such as Tenex C shell (/bin/tcsh), you need to specify an additional `-csh` option to generate shell scripts `cdn_vip_env_<...>.csh`. Then, you need to execute the following command to complete your environment set up, either on the command line or in your script:

```
source cdn_vip_env_<...>.csh
```

10.1.1.2. cdn_vip_setup_example

The cdn_vip_setup_example script generates a shell file with simulator-specific commands to compile and simulate a VIP example in the release. This script requires the key information detailed below. Information on other/optional arguments to the script is available by executing the following command:

```
cdn_vip_setup_example -h
```

- The VIPCAT installation directory is expected to be specified with CDN_VIP_ROOT environment variable. Or, it must be explicitly specified with the `-cdn_vip_root` optional argument to the script.
- The specific release example to be setup must be explicitly specified with the `-exampledir` argument to the script.
- The simulator must be explicitly specified with the `-sim` argument to the script. The script is capable of setting up examples for the following simulators: NCSIM (multi-step mode), NCSIM (single-step mode), VCS and MTI.
- The testbench methodology must be explicitly specified with the "-method"argument to the script. The script is capable of setting up examples for SystemVerilog with UVM Layer, basic SystemVerilog.
- NCSIM users only: The user directory containing VIP Compiled libraries must be explicitly specified with the `-cdn_vip_lib` option to the script.

`cdn_vip_setup_example` generates a Bourne shell script `cdn_vip_run_<...>.sh`. The optional `-run` argument to `cdn_vip_setup_example` also executes the generated script.

Important: for the generated script to compile/simulate successfully, the environment must be set up correctly with `cdn_vip_setup_env`.

10.1.1.3. cdn_vip_check_env

The cdn_vip_check_env script checks that the environment (path, LD_LIBRARY_PATH, etc.) are setup correctly for VIP simulation. This script requires the key information detailed below. Information on other/optional arguments to the script is available by executing the following command:

```
cdn_vip_check_env -h
```

- The VIPCAT installation directory is expected to be specified with CDN_VIP_ROOT environment variable. Or, it must be explicitly specified with the `-cdn_vip_root` optional argument to the script.
- The simulator must be explicitly specified with the `-sim` argument to the script. The script is capable of checking the environment for the following simulators: NCSIM (multi-step mode), NCSIM (single-step mode), VCS and MTI.
- The testbench methodology must be explicitly specified with the `-method` argument to the script. The script is capable of checking the environment for SystemVerilog with UVM Layer or basic SystemVerilog.
- NCSIM users only: The user directory containing VIP-compiled libraries must be explicitly specified with the `-cdn_vip_lib` option to the script. The script confirms that the VIP-compiled libraries have been correctly set up in the user directory.
- The optional argument `-licensing` lists all Cadence VIP licenses available in the environment in the file `cdn_vip_licenses.log`.

10.1.2. NCSIM

The `cdn_vip_setup_env` and `cdn_vip_setup_example` scripts generate shell commands to set up your simulation environment and run an example in the VIPCAT release with NCSIM (in either three-step mode or single-step mode).

10.1.2.1. Three-Step Mode

The key arguments (libraries and command-line options) for VIP simulations with NCSIM in three-step mode (compile, elaborate, simulate) are clearly specified in the generated script `cdn_vip_run_ncsim_sv.sh` (for NCSIM three-step simulations in a 32-bit environment). After confirming that the provided example compiles and simulates successfully, you are strongly encouraged to modify this generated script, and compile and simulate your own SystemVerilog testbench before proceeding to integrate the VIP with other components in your overall NCSIM three-step simulation framework (your Makefiles, scripts).

Examples of script invocations for NCSIM in three-step mode:

- Setting up C shell environment for the first VIP download (VIP compiled libraries need to be installed with -i option)

```
cdn_vip_setup_env -s ncsim_3s -cdn_vip_root <VIPCAT_Installation_Directory> -m sv -i axi -
cdn_vip_lib <VIP_Compiled_Library_Directory> -csh
```

```
source cdn_vip_env_ncsim_sv.csh
```

- Setting up (and running) an example provided in the VIPCAT release:

```
cdn_vip_setup_example -s ncsim_3s -cdn_vip_root <VIPCAT_Installation_Directory> -e
 <VIPCAT_Example_Directory> -m sv -cdn_vip_lib <VIP_Compiled_Library_Directory>
cdn_vip_run_ncsim_sv.sh
```

Exact script invocations to set up one specific example in the VIPCAT release for NC-SIM in three-step mode are shown below. The complete script with all steps is available at `$CDN_VIP_ROOT/tools/denali/example/cdn_axi/svExamples/simpleExample/example_setup_ncsim.csh`.

```
 #!/bin/csh -f
    #Step 1. Create cdn_vip_env_ncsim_sv.csh with environment setup commands
    #Note: NCSIM Compiled libraries path to be provided + libraries need to be installed
${CDN_VIP_ROOT}/bin/cdn_vip_setup_env -s ncsim_3s -cdn_vip_root ${CDN_VIP_ROOT} -m sv -csh -
cdn_vip_lib vip_lib -i axi
    #Step 2. Complete environment setup by executing generated shell commands
source cdn_vip_env_ncsim_sv.csh

    #Step 3. Create cdn_vip_run_ncsim_sv.sh with simulator comands
    #Note: CDN_VIP_ROOT environment variable guaranteed to be available after Step 2
${CDN_VIP_ROOT}/bin/cdn_vip_setup_example -s ncsim_3s -e ${CDN_VIP_ROOT}/tools/denali/example/cdn_axi/
svExamples/simpleExample -m sv -cdn_vip_lib vip_lib
    #Step 4. Execute generated script with compile/simulate commands (also doable by -r option to
 cdn_vip_setup_example)
./cdn_vip_run_ncsim_sv.sh
```

Please confirm the following two requirements are met before executing the above scripts:

- The CDN_VIP_ROOT environment variable is correctly specified.
- The 'ncsim' executable is available in your path (and you have an ncsim license).

10.1.2.2. Single-Step Mode

The key arguments (libraries/command-line options) for VIP simulations with NCSIM in single-step mode (irun) are clearly specified in the generated script cdn_vip_run_irun.sh (for NCSIM single-step simulations in a 32-bit environment). After confirming that the provided example compiles/simulates successfully, you are strongly encouraged to modify this generated script and compile/simulate your own SystemVerilog testbench before proceeding to integrate the VIP with other components in your overall NCSIM single-step simulation framework (your Makefiles, scripts).

Examples of script invocations for NCSIM in single-step mode:

- Setting up C shell environment for the first VIP time (VIP compiled libraries need to be installed with -i option)

```
cdn_vip_setup_env -s ncsim_irun -cdn_vip_root <VIPCAT_Installation_Directory> -m sv -i axi -
cdn_vip_lib <VIP_Compiled_Library_Directory> -csh
source cdn_vip_env_irun_sv.csh
```

- Setting up (and running) an example provided in the VIPCAT release:

```
cdn_vip_setup_example -s ncsim_irun -cdn_vip_root <VIPCAT_Installation_Directory> -e
 <VIPCAT_Example_Directory> -m sv -cdn_vip_lib <VIP_Compiled_Library_Directory>
cdn_vip_run_irun_sv.sh
```

Exact script invocations to set up one specific example in the VIPCAT release for NC-SIM in single-step mode are shown below. The complete script with all steps is available at `$CDN_VIP_ROOT/tools/denali/example/cdn_axi/svExamples/simpleExample/example_setup_irun.csh`.

```
 #!/bin/csh -f
    #Step 1. Create cdn_vip_env_irun_sv.csh with environment setup commands
    #Note: NCSIM Compiled libraries path to be provided + libraries need to be installed
${CDN_VIP_ROOT}/bin/cdn_vip_setup_env -s ncsim_irun -cdn_vip_root ${CDN_VIP_ROOT} -m sv -csh -
cdn_vip_lib vip_lib -i axi
    #Step 2. Complete environment setup by executing generated shell commands
source cdn_vip_env_irun_sv.csh

    #Step 3. Create cdn_vip_run_irun_sv.sh with simulator comands
    #Note: CDN_VIP_ROOT environment variable guaranteed to be available after Step 2
${CDN_VIP_ROOT}/bin/cdn_vip_setup_example -s ncsim_irun -e ${CDN_VIP_ROOT}/tools/denali/example/
cdn_axi/svExamples/simpleExample -m sv -cdn_vip_lib vip_lib
    #Step 4. Execute generated script with compile/simulate commands (also doable by -r option to
 cdn_vip_setup_example)
./cdn_vip_run_irun_sv.sh
```

Please confirm the following two requirements are met before executing the above scripts:

- The CDN_VIP_ROOT environment variable is correctly specified.
- The 'ncsim' executable is available in your path (and you have an ncsim license).

10.1.2.3. Using *irun*

If your simulator is NCSIM, you can use the *irun* invocation tool to handle the details of incorporating VIP models. When you add an option to specify the location of the VIPCAT installation, *irun* uses that information to treat the VIP models as native simulation components. *irun* automatically compiles the model and loads the necessary VIP shared objects, so all you need to do is specify the model instantiation interface and the VIP HDL packages that you are using.

To get native treatment for the VIP models, you need the following:

- Incisive Enterprise Simulator release 12.1 or later
- The VIP model instantiation interface that originates from the VIP release 11.30.012-s or later. If you are using older versions, you should regenerate them with Pureview or get new copies from Cadence customer support.

The `-CDN_VIP_ROOT` option tells irun where the VIPCAT installation is located. Throughout these instructions, the environment variable `CDN_VIP_ROOT` is used to refer to the path to the VIP installation. You are not required to set this environment variable; it is simply a shorthand for this location.

Here is an example that uses *irun* to simulate with the CDN_AXI VIP:

```
irun \
   -cdn_vip_root ${CDN_VIP_ROOT} \
   ${CDN_VIP_ROOT}/tools/denali/ddvapi/sv/denaliMem.sv \
   ${CDN_VIP_ROOT}/tools/denali/ddvapi/sv/denaliCdn_axi.sv \
   ${CDN_VIP_ROOT}/tools/denali/example/cdn_axi/svExamples/simpleExample/activeMaster.v \
   ${CDN_VIP_ROOT}/tools/denali/example/cdn_axi/svExamples/simpleExample/activeSlave.v \
   ${CDN_VIP_ROOT}/tools/denali/example/cdn_axi/svExamples/simpleExample/tb.sv \
   -incdir ${CDN_VIP_ROOT}/tools/denali/example/cdn_axi/svExamples/simpleExample \
   -timescale 1ps/1ps -top pstest
```

Note

The files used in this example are located in the VIPCAT installation. If you want to run your own copy of the example, copy the example directory into a local directory and update the paths to the files in the `cdn_axi_irun.csh` script.

Note

If you use an old HDL instantiation interface by mistake, you will get an elaborator error that begins like this:

```
ncelab: *W,MISSYST
```

You can verify that you have new code by looking for the following line at the top of the file:

```
// pragma cdn_vip_model -class cdn_axi
```

10.1.3. VCS

The cdn_vip_setup_env and cdn_vip_setup_example scripts generate shell commands to set up your simulation environment and run an example in the VIPCAT release with VCS.

The key arguments (libraries and command-line options) for VIP simulations with VCS are clearly specified in the generated script `cdn_vip_run_vcs_sv.sh` (for VCS simulations in a 32-bit environment). After confirming that the provided example compiles and simulates successfully, you are strongly encouraged to modify this generated script, and compile and simulate your own SystemVerilog testbench before proceeding to integrate the VIP with other components in your overall VCS simulation framework (your Makefiles, scripts).

Examples of script invocations for VCS:

- Setting up your Bourne shell environment:

```
cdn_vip_setup_env -s vcs -cdn_vip_root <VIPCAT_Installation_Directory> -m sv
. cdn_vip_env_vcs_sv.sh
```

- Setting up (and running) an example provided in the VIPCAT release:

```
cdn_vip_setup_example -s vcs -cdn_vip_root <VIPCAT_Installation_Directory> -e
 <VIPCAT_Example_Directory> -m sv
cdn_vip_run_vcs_sv.sh
```

Exact script invocations to set up one specific example in the VIPCAT release for VCS are shown below. The complete script with all steps is available at `$CDN_VIP_ROOT/tools/denali/example/cdn_axi/svExamples/simpleExample/example_setup_vcs.csh`.

```
 #!/bin/csh -f
    #Step 1. Create cdn_vip_env_vcs_sv.csh with environment setup commands
${CDN_VIP_ROOT}/bin/cdn_vip_setup_env -s vcs -cdn_vip_root ${CDN_VIP_ROOT} -m sv -csh
    #Step 2. Complete environment setup by executing generated shell commands
source cdn_vip_env_vcs_sv.csh

    #Step 3. Create cdn_vip_run_vcs_sv.sh with simulator comands
    #Note: CDN_VIP_ROOT environment variable guaranteed to be available after Step 2
${CDN_VIP_ROOT}/bin/cdn_vip_setup_example -s vcs -e ${CDN_VIP_ROOT}/tools/denali/example/cdn_axi/
svExamples/simpleExample -m sv
    #Step 4. Execute generated script with compile/simulate commands (also doable by -r option to
 cdn_vip_setup_example)
./cdn_vip_run_vcs_sv.sh
```

Please confirm the following two requirements are met before executing the above scripts:

- The CDN_VIP_ROOT environment variable is correctly specified.
- The 'vcs' executable is available in your path (and you have a vcs license).

10.1.4. MTI

The `cdn_vip_setup_env` and `cdn_vip_setup_example` scripts generate shell commands to set up your simulation environment and run an example in the VIPCAT release with MTI.

The key arguments (libraries and command-line options) for VIP simulations with MTI are clearly specified in the generated script `cdn_vip_run_mti_sv.sh` (for MTI simulations in a 32-bit environment). After confirming that the provided example compiles and simulates successfully, you are strongly encouraged to modify this generated script, and compile and simulate your own SystemVerilog testbench before proceeding to integrate the VIP with other components in your overall MTI simulation framework (your Makefiles, scripts).

Examples of script invocations for MTI:

- Setting up your C shell environment:

```
cdn_vip_setup_env -s mti -cdn_vip_root <VIPCAT_Installation_Directory> -m sv
source cdn_vip_env_mti_sv.sh
```

- Setting up (and running) an example provided in the VIPCAT release:

```
cdn_vip_setup_example -s mti -cdn_vip_root <VIPCAT_Installation_Directory> -e
 <VIPCAT_Example_Directory> -m sv
cdn_vip_run_mti_sv.sh
```

Exact script invocations to set up one specific example in the VIPCAT release for MTI are shown below. The complete script with all steps is available at `$CDN_VIP_ROOT/tools/denali/example/cdn_axi/svExamples/simpleExample/example_setup_mti.csh`.

```
 #!/bin/csh -f
    #Step 1. Create cdn_vip_env_mti_sv.csh with environment setup commands
${CDN_VIP_ROOT}/bin/cdn_vip_setup_env -s mti -cdn_vip_root ${CDN_VIP_ROOT} -m sv -csh
    #Step 2. Complete environment setup by executing generated shell commands
source cdn_vip_env_mti_sv.csh

    #Step 3. Create cdn_vip_run_mti_sv.sh with simulator comands
    #Note: CDN_VIP_ROOT environment variable guaranteed to be available after Step 2
${CDN_VIP_ROOT}/bin/cdn_vip_setup_example -s mti -e ${CDN_VIP_ROOT}/tools/denali/example/cdn_axi/
svExamples/simpleExample -m sv
    #Step 4. Execute generated script with compile/simulate commands (also doable by -r option to
 cdn_vip_setup_example)
./cdn_vip_run_mti_sv.sh
```

Please confirm the following two requirements are met before executing the above scripts:

- The CDN_VIP_ROOT environment variable is correctly specified.
- The 'vsim' executable is available in your path (and you have a vsim license).

10.2. SystemVerilog Interface

This interface is used when you instantiate transaction objects in your SystemVerilog testbench and would like to employ tasks that can be used to perform the actions, such as transaction creation, call-back processing, and so on.

10.2.1. Transaction Interface

10.2.1.1. Instance Class

10.2.1.1.1. Class denaliCdn_axiInstance

The `denaliCdn_axiInstance` corresponds to the CDN_AXI VIP instance instantiated in the testbench. Each such instance must have a corresponding `denaliCdn_axiInstance` object associated with it.

10.2.1.1.1.1. Constructor

```
function new(string instName, string cbFuncName = "");
```

This creates a new transaction to be initiated from the specified instance and returns a transaction handle pointing to the newly created empty transaction.

The `instName` must be a full path and the `cbFuncName` can be null for the constructor. However, if you want to define an explicit DPI callback function, it must be set before any callback point is added for monitoring.

The format of `cbFuncName` is `"<hdlScope>::<funcName>"`.

```
denaliCdn_axiInstance inst;
inst = new(top.i0);
```

10.2.1.1.1.2. Methods

10.2.1.1.1.2.1. getInstName()

Name

getInstName — Retrieves the testbench instance name that corresponds to this `denaliCdn_axiInstance`.

Synopsis

```
function string getInstName() ;
```

Description

Retrieves the testbench instance name that corresponds to this `denaliCdn_axiInstance`.

Example

```
denaliCdn_axiInstance inst;
inst = new("top.i0");
$display("InstName = %s", inst.getInstName());
```

10.2.1.1.1.2.2. setCbFuncName()

Name

setCbFuncName — Sets the DPI callback function name.

Synopsis

```
function void setCbFuncName(string cbFuncName) ;
```

Description

Sets the DPI callback function name.

The format of `cbFuncName` is `"<hdlScope>::<funcName>"`.

Example

```
denaliCdn_axiInstance inst;
inst = new("top.i0");
inst.setCbFuncName("top.my_if::myCbFunc");
```

10.2.1.1.1.2.3. getCbFuncName()

Name

getCbFuncName() — Gets the DPI callback function name.

Synopsis

```
function string getCbFuncName() ;
```

Description

Gets the DPI callback function name.

Example

```
denaliCdn_axiInstance inst;
inst = new("top.i0");
inst.setCbFuncName("top.my_if::myCbFunc");
$display("FuncName = %s", inst.getCbFuncName());
```

10.2.1.1.1.2.4. transAdd()

Name

transAdd — Adds a newly created transaction to the model’s user queue.

Synopsis

```
function integer transAdd(denaliCdn_axi obj, integer portNumOrQueueNum, denaliArgTypeT insertType =
DENALI_ARG_trans_append) ;
```

Description

Adds a newly created transaction to the model’s user queue. Any transaction fields that have been assigned are propagated to the internal model transaction.

> **Note**
> In CDN_AXI, only one queue exists called DENALI_CDN_AXI_QUEUE_Burst

10.2.1.1.1.2.5. setCallback()

Name

setCallback — Adds a given callback reason to the list of callback points being monitored.

Synopsis

```
virtual function integer setCallback(denaliCdn_axiCbPointT cbRsn) ;
```

Description

Adds a given callback reason to the list of callback points being monitored.

10.2.1.1.1.2.6. clearCallback()

Name

clearCallback — Removes a given callback reason from the list of callback points being monitored.

Synopsis

```
virtual function integer clearCallback(denaliCdn_axiCbPointT cbRsn) ;
```

Description

Removes a given callback reason to the list of callback points being monitored.

10.2.1.1.1.2.7. getQueuePending()

Name

getQueuePending — Returns the number of pending transactions in the instance's transaction queue.

Synopsis

```
function int getQueuePending(integer portNum) ;
```

Description

Retrieves the number of pending transactions currently in the instance's transaction queue.

Example

```
denaliCdn_axiInstance inst;
inst = new("top.i0");
$display("Queue pending items=", inst.getQueuePending(0));
```

10.2.1.2. Packet Classes

10.2.1.2.1. Class denaliCdn_axiTransaction

10.2.1.2.1.1. Fields

Table 10.1. Class denaliCdn_axiTransaction Fields

Field Name	Field Type	Description
Type	rand denaliCdn_axiTrTypeT	Transaction type.
PortNum	rand reg [31:0]	Port/Queue number
Callback	rand denaliCdn_axiCbPointT	ID of the callback currently associated with the transaction. Useful in the callback processing user code.
ErrorString	rand string	Character string describing the transaction errors for the debug
ErrorId	rand denaliCdn_axiErrorTypeT	This is the Error ID associated with error callbacks
ErrorInfo	rand reg [31:0]	Transaction error severity
UserData	rand reg [31:0]	Placeholder for the user-specific information
Delay	rand reg [31:0]	Transaction delay (in clock cycles). Please do not use this field
BreadyDelay	rand reg [31:0]	The number of cycles to keep BREADY low after a WRITE burst has ended (i.e. BVALID and BREADY were high).
BreadyControl	rand denaliCdn_axiReady ControlT	BREADY signal control.
Data [*]	rand reg [7:0]	Data list for write bursts only. Each entry is the data for the transfer in the same index in the

Field Name	Field Type	Description
		transfers list. (Available also in AVIP mode)
TransmitDelay	rand reg [31:0]	The delay in cycles from the time the BFM first attempts to send the burst until asserting AVALID and putting the burst's control information on the address channel.
WriteAddressOffset	rand reg [31:0]	(Default: 0) Determines the transfer to whom the delay of the address will refer. The field is relevant only for write bursts In case of 0 - the adress delay will be calculate first. The delay of the first transfer would not be calculated before the beginning of the write address phase. In case greater than 0 - the address delay will be calculated from the beginning of the write transfer (as indicated by the field).
IdTag	rand reg [19:0]	Burst ID tag (Available also in AVIP mode)
Alen	rand reg [7:0]	(NC) The value driven to the ALEN signal
Size	rand denaliCdn_axiTransferSizeT	Transfer size (Available also in AVIP mode)
Kind	rand denaliCdn_axiBurstKindT	FIXED, INCR or WRAP (Available also in AVIP mode)
Access	rand denaliCdn_axiAccessT	NORMAL, EXCLUSIVE or LOCKED (Available also in AVIP mode)
Direction	rand denaliCdn_axiDirectionT	READ or WRITE (Available also in AVIP mode)
Bufferable	rand denaliCdn_axiBufferModeT	NON_BUFFERABLE or BUFFERABLE (Available also in AVIP mode)
Cacheable	rand denaliCdn_axiCacheModeT	NON_CACHEABLE or CACHEABLE -- AMBA3 (Available also in AVIP mode)

Field Name	Field Type	Description
ReadAllocate	rand denaliCdn_axiRead AllocateT	NO_READ_ALLOCATE or READ_ALLOCATE (Available also in AVIP mode)
WriteAllocate	rand denaliCdn_axiWrite AllocateT	NO_WRITE_ALLOCATE or WRITE_ALLOCATE (Available also in AVIP mode)
Region	rand reg [4:0]	REGION -- AMBA4 (Available also in AVIP mode)
Qos	rand reg [3:0]	QOS -- AMBA4 (Available also in AVIP mode)
Auser [*]	rand reg [31:0]	User address signal
Buser [*]	rand reg [31:0]	User response signal
Length	rand reg [31:0]	The number of transfers in the burst (equals to ALEN + 1) (Available also in AVIP mode)
IsDiscarded	rand reg	Was the burst discarded
TransfersChannelDelay [*]	rand reg [31:0]	Array of all burst's transfers channel delay
TransfersResp [*]	rand denaliCdn_axiResponseT	Array of all burst's transfers responses (Available also in AVIP mode)
ModelGeneration	rand reg	if TRUE - use the core model's constraints for generation. if FALSE - use only SV constraints for generation.
DiType	rand denaliCdn_axiDiTypeT	Symbolic Di type (like DENALI_CDN_AXI_DI_ MasterBurst).
UserDataArray [*]	rand reg [31:0]	Optional array of user spesific data in case the UserData field is insufficient.
StrobeArray [*]	rand reg [31:0]	Array of all WRITE burst's transfers strobe values (Available also in AVIP mode)
UserSignalArray [*]	rand reg [31:0]	Array of data channel user signals (RUSER / WUSER) visible and controlable in the burst level. In case the r/wuser signals width is up to 32 bits - each

Field Name	Field Type	Description
		entry in the array reflects one transfer's user signal. Multiple array entries reflects a transfer's user signal wider than 32 bits.
SpecVer	rand denaliCdn_axiSpecVersionT	The version of the spec (Available also in AVIP mode)
SpecSubtype	rand denaliCdn_axiSpecSubtypeT	The spec subtype (Available also in AVIP mode)
SpecInterface	rand denaliCdn_axiSpecInterfaceT	The spec interface (FULL or LITE) (Available also in AVIP mode)
StartAddress	rand reg [63:0]	The first address in this burst (Available also in AVIP mode)
Privileged	rand denaliCdn_axiPrivilegedModeT	NORMAL or PRIVILEGED (Available also in AVIP mode)
Secure	rand denaliCdn_axiSecureModeT	SECURE or NONSECURE (Available also in AVIP mode)
DataInstr	rand denaliCdn_axiFetchKindT	DATA or INSTRUCTION (Available also in AVIP mode)
HighestAddress	rand reg [63:0]	(NC) The highest address byte in the burst
LowestAddress	rand reg [63:0]	(NC) The lowest address byte in the burst
AddrPhaseStartTime	rand reg [63:0]	Address phase start time.
RespPhaseStartTime	rand reg [63:0]	Response phase start time.
DataPhaseStartTime	rand reg [63:0]	Data phase start time.
AddrPhaseEndTime	rand reg [63:0]	Address phase End Time
RespPhaseEndTime	rand reg [63:0]	Response phase End Time
DataPhaseEndTime	rand reg [63:0]	Data phase End Time
Duration	rand reg [63:0]	Burst duration
IsAddrPhaseStarted	rand reg	Is the address phase started.
IsRespPhaseStarted	rand reg	Is the Response phase started.
IsDataPhaseStarted	rand reg	Is the data phase started.
IsAddrPhaseFinished	rand reg	Is the Address phase ended.
IsRespPhaseFinished	rand reg	Is the Response phase ended.
IsDataPhaseFinished	rand reg	Is the Data phase ended.

Field Name	Field Type	Description
IsEnded	rand reg	Is the burst ended.
PsInternalId	rand reg [31:0]	this is a model internal field, this field shouldn't be accessed. this field will not be available in future versions.
AllowRespChange	rand reg	If TRUE allow the BFM to change the response for an EXCLUSIVE burst if it is necessary (the generated response is no longer valid).
Resp	rand denaliCdn_axiResponseT	The write response. Used only for WRITE bursts. (Available also in AVIP mode)
RespPassDirty	rand denaliCdn_axiSnoopRespPassDirtyT	Only for AMBA4 read response This value will tranculate to all the read transfer
RespIsShared	rand denaliCdn_axiSnoopRespIsSharedT	Only for AMBA4 read response This value will tranculate to all the read transfer
ChannelDelay	rand reg [31:0]	The delay until we drive the response for a write burst. Used only for WRITE bursts. Note that if the early write response feature is enabled then this delay is counted from the end of the address phase
AddressDelay	rand reg [31:0]	Number of cycles to keep AREADY low after the completion of this burst address phase.
AreadyControl	rand denaliCdn_axiReadyControlT	[AW/AR]READY signal control.
EnforceReadyController	rand reg	Indicates if the ready controller behavior will be enforced, even at the risk of a deadlock. The default is TRUE.
MaxDelayReadyController	rand reg [31:0]	If the field 'enforce_ready_controller' is FALSE, this field indicates how long from the initiation of the ready controller,

Field Name	Field Type	Description
		it's ok to ignore the controller and raise the ready signal.
PhysicalData [*]	rand reg [31:0]	The data transmitted in this transfer as written to the bus.
Strobe	rand reg [127:0]	The strobe for the data
User [*]	rand reg [31:0]	The user signal (Available also in AVIP mode)
Position	rand denaliCdn_axiTransferPositionT	The position of the transfer in the burst : FIRST, MIDDLE or LAST.
Address	rand reg [63:0]	(NC) The address of this transfer. This field need to be generated since it is used in the generation of 'strobe'.
Last	rand reg	Whether this transfer is the last in the burst
IsStarted	rand reg	Has the transfer started.
PhysicalDataXMask [*]	rand reg [31:0]	Bit mask for the data monitored on the bus, where '1' indicates 'X' value
PhysicalDataZMask [*]	rand reg [31:0]	Bit mask for the data monitored on the bus, where '1' indicates 'Z' value
RreadyControl	rand denaliCdn_axiReadyControlT	RREADY signal control.
IgnoreConstraints	rand reg	The IgnoreConstraints field pertains to behavior of Slave agents for Read transactions. When set to TRUE, Read data, used in the Slave response, will be taken from Slave's sparse memory When set to FALSE, Read data will be determined by response transaction, that can be specified by the user, and placed into Slave's response. If no data was specified in the slave response, the Read Data will still be taken from Slave's sparse memory. Default: TRUE

Field Name	Field Type	Description
WreadyControl	rand denaliCdn_axiReady ControlT	WREADY signal control.

10.2.1.2.1.2. Constructor

10.2.1.2.1.2.1. new()

Name

new — Constructor for the class denaliCdn_axiTransaction.

Synopsis

```
function new();
```

Description

This constructs an object of the class denaliCdn_axiTransaction.

10.2.1.2.1.3. Methods

10.2.1.2.1.3.1. transSet()

Name

transSet — Propagates all modified transaction fields to the internal model transaction.

Synopsis

```
virtual function transSet();
```

Description

Propagates all modified transaction fields to the internal model transaction.

10.2.1.2.1.3.2. printInfo()

Name

printInfo — Prints the fields/contents of the transaction.

Synopsis

```
virtual function integer printInfo(integer arraySize = 32);
```

Description

Prints the fields/contents of the transaction. The 'arraySize' parameter specifies the maximum number of elements to print for arrays.

10.2.2. Coverage Interface / CDN_AXI VIP Coverage

Coverage is based on monitor events/callbacks and data structures. It lets you see both how well you are proceeding with the verification task and provides statistics about your DUT in the AXI environment.

The VIP has predefined protocol coverage defined for all major events, data items and their fields. You can add your own coverage groups and items, and you can disable coverage collection from parts of your environment or specific coverage groups. Cadence recommends adjusting the coverage specifically for your DUT.

The Coverage can be used in specifics instance in your SystemVerilog or SystemVerilog UVM test-bench.

All the coverage files are under `$DENALI/ddvapi/sv/coverage/cdn_axi`.

10.2.2.1. Coverage Class

The denaliCdn_axiCoverageInstance class is instantiated under the CDN_AXI VIP instance once your enable coverage for that instance.

10.2.2.1.1. cover_sample()

Name

cover_sample — Triggers covergroup sampling.

Description

This method is called by an internal pre-defined CDN_AXI VIP callback every time a covergroup needs to be sampled. The method gets a covergroup value and a relevant transaction, and based on the covergroup value it got, it triggers the relevant covergroup sampling.

10.2.2.1.2. Coverage Definitions

The coverage class includes the definition of CDN_AXI predefined protocol coverage. The coverage is divided to covergroups sampled on a relevant callback, when each group contains several cover-points.

10.2.2.2. Coverage File Structure

The following table describes coverage files, which can be found in `$DENALI/ddvapi/sv/coverage/cdn_axi`.

Table 10.2. Coverage File Structure

Filename	Description
denaliCdn_axiCoverage.svh	This file contains the VIP coverage definitions and the trigger for the covergroups sampling.
denaliCdn_axiCoverageInstance.sv	This file contains definition of the coverage class.
denaliCdn_axiCoverGroupNew.svh	This file contains new actions for the cover-groups.
denaliCdn_axiCoverGroupEnable.svh	This file contains *_enable fields for each cover group that can be used to disable specific cover

Filename	Description
	groups from being collected. By default all are set to 1 (enabled).

10.2.2.3. Enabling Coverage Collection

To enable coverage collection in your simulation, please use the following steps:

1. Enable the coverage option in your simulator.
2. Set up coverage collection for the relevant instances, as described in Section 10.2.2.3.2, *"SystemVerilog Coverage Setup"*.
3. Enable the coverage:

   For the SystemVerilog interface, see Section 10.2.2.3.3, *"Enabling Coverage using SystemVerilog"*.
   For the SystemVerilog for UVM, see Section 10.2.2.3.4, *"Enabling Coverage using UVM SystemVerilog"*.

10.2.2.3.1. Enable the Coverage Option in your Simulator

In addition to setting up the VIP for coverage, you need to ensure your run command includes any flags required by the simulator to collect coverage.

For example, for ncsim, you need to provide the following flags:

```
-coverage functional -covoverwrite -write_metrics
```

10.2.2.3.2. SystemVerilog Coverage Setup

For coverage collection, you must first enable the collection of coverage for the relevant instance from the `.denalirc` file, using the following field:

```
EnableCoverage <instance pattern>
```

The following are examples of <instance pattern>.

- `*` is used to specify all instances.
- `*[Pp]assive*` is used to specify all the passive instances.
- `top.<instance_name>` is used to specify an instance called instance_name under top, where top is your top module. (The patterns come from the HDL instance names.)

10.2.2.3.3. Enabling Coverage using SystemVerilog

- Start with doing the SystemVerilog coverage setup using the `.denalirc` file. For more details, refer to Section 10.2.2.3.2, *"SystemVerilog Coverage Setup"*.

- After you instantiate the relevant denaliCdn_axiInstance using its `new` function, call the instance `create_cover()` function. See the code example in Section 10.2.4, "*Example Testcase*".

10.2.2.3.4. Enabling Coverage using UVM SystemVerilog

- Start with doing the SystemVerilog coverage setup using the `.denalirc` file. For more details, refer to Section 10.2.2.3.2, "*SystemVerilog Coverage Setup*".
- To enable your UVM monitor to collect coverage, set the **coverageEnable** bit to 0. For details, refer to the UVM user monitor code example in Section 10.3.9, "*Example Testcase*".

10.2.2.4. Creating DUT-Specific Coverage Definitions

This section discusses how to filter predefined coverage definitions and add customized coverage definitions.

10.2.2.4.1. Filtering Predefined or Irrelevant Coverage Definitions

Some coverage definitions might be irrelevant in your verification environment. In that case, you can filter the predefined coverage definitions to disable some of the coverage.

To filter covergroups, we use configuration fields which are defined in the coverage class.

For each covergroup we have an enable field called <*covergroup name*>_enable field. These fields are used to disable creation of the covergroup in the new function (covergroups can be created only in the class constructor, per the SystemVerilog LRM), and to check whether the covergroup needs to be sampled.

By default, all the relevant fields for the used VIP configuration are enabled (set to 1).

10.2.2.4.1.1. Filtering Irrelevant Covergroups using the UVM SV intreface

Use the set_config_field method in your testbench to disable the *_enable field you want, by setting its value to 0.

10.2.2.4.1.2. Filtering Irrelevant Covergroups using the SV interface without UVM

The coverage class contains an empty hook method called userDisableCoverGroups ().

To disable the covergroup you don't need:

1. Create your own coverage class which inherits from the given coverage class (denaliCdn_axiCoverageInstance).
2. In this class, implement the userDisableCoverGroups () method to set the relevant cover group *_enable field to 0, and by that disable the equvivalent covergroups.
3. Create your own VIP instance which inherits from the given VIP instance (denaliCdn_axiInstance).

4. In this VIP instance, re-implement the new_cover_class() method to new the coverInst field with the coverage class you created instead if the given coverage class.

```
   // Implement the new_cover_class to create the coverInst of the new coverage class you have
created

   virtual function void new_cover_class();

    // Create a new instance of your coverage class type
    // which extends the provided coverage class and assign it for coverInst
    userCoverageClass myCoverInst;
    myCoverInst = new(instName, this);
    coverInst = myCoverInst;
   endfunction
```

10.2.2.4.2. Adding Customized Coverage Definitions

Cadence recommends adding coverage groups specific to your DUT implementation, as required.

10.2.2.4.2.1. Adding Covergroup Definitions using UVM SV

To add coverage definitions:

1. Create your own UVM coverage class which inherits from the base VIP UVM coverage class (cdnAxiUvmCoverage)
2. In this class:
   a. Define the required cover groups using the denaliCdn_axiTransaction field (defined in the base coverage class), and the cdnAxiInstance register space.
   b. Be sure to new the new coverage groups in your class new function.
3. Ensure you call the new covergroups sample method on the right events in your environment code.
4. In you monitor, which inherits from cdnAxiUvmMonitor, use the factory to ensure that the coverage class instance is created using your new coverage class.

10.2.2.4.2.2. Adding Covergroup Definitions using the SV interface without UVM

To add coverage definitions:

1. Create your own coverage class which inherits from the given coverage class (denaliCdn_axiCoverageInstance)
2. In this class:
   a. Define the required cover groups using the denaliCdn_axiCoverageInstance class denaliCdn_axiTransaction field, and the denaliCdn_axiInstance register space.
   b. Be sure to new the new coverage groups in your class new function.

3. Create your own VIP instance which inherits from the given VIP instance (denaliCdn_axiInstance).

4. In this VIP instance class:

   a. Re-implement the new_cover_class() method to new the coverInst field with the coverage class you created instead if the given coverage class.

```
     // Implement the new_cover_class to create the coverInst of the new coverage class you
have created

     virtual function void new_cover_class();

      // Create a new instance of your coverage class type
      // which extends the provided coverage class and assign it for coverInst
      userCoverageClass myCoverInst;
      myCoverInst = new(instName, this);
      coverInst = myCoverInst;
     endfunction
```

   b. Implement the relevant denaliCdn_axiInstance callbacks to activate your covergroups sample().

10.2.3. SystemVerilog File Structure

The file structure that supports the CDN_AXI VIP SystemVerilog interface includes the following:

- `$DENALI/ddvapi/sv/denaliCdn_axi.sv`

  This file defines all the CDN_AXI VIP transaction classes for SystemVerilog.

  You must import everything that is defined within `DenaliSvCdn_axi` package within your testbench to make use of these transaction classes.

- `$DENALI/ddvapi/sv/denaliCdn_axiTypes.svh`

  This file defines all the enumeration types for each of the registers supported within the CDN_AXI configuration space, the common header and all support capability structures.

- `$DENALI/ddvapi/sv/denaliCdn_axiImports.sv`

  This file defines all the SystemVerilog-DPI function calls mapped to equivalent C functions as included and used within the `denaliCdn_axi.sv` and `denaliCdn_axiSvIf.c` files.

- `$DENALI/ddvapi/sv/denaliCdn_axiSvIf.c`

  This file defines all the integration code between SystemVerilog and C-API needed to pass to the model interface.

- `$DENALI/ddvapi/sv/denaliMem.sv`

This file defines an optional memory support package that can be used for memory models as well as internal memories and registers. Package `DenaliSvMem` contains global routines to support a traditional procedural programming style, and class `denaliMemInstance` to support an object oriented programming style. The package includes utilities for reads/writes, callbacks, memory transactions, SOMA parameter value access, TCL command evaluation, and so on.

- `$DENALI/ddvapi/sv/denaliMemSvIf.c`

  This file defines the integration interface between `denaliMem.sv` and the model C code, and is required to be compiled as part of creating your testbench if you are using the `DenaliSvMem` package.

10.2.4. Example Testcase

To create a testcase:

1. Use PureView to create SOMA files for requisite model instances. For details, refer to Chapter 4, *PureView Graphical Tool*.
2. Use PureView and the same configuration to create an HDL instantiation interface for those instances. For details, refer to Chapter 4, *PureView Graphical Tool*.
3. Create the top level and instantiate the CDN_AXI VIP and DUT models in it.
4. Set the required `.denalirc` variables.
5. Create the testbench and add transactions to the user queue of the host model.

The following example shows the `tb.sv` file.

```
// Import the DDVAPI ACE SV interface and the generic Mem inetrface
import DenaliSvCdn_axi::*;
import DenaliSvMem::*;

class myTransaction extends denaliCdn_axiTransaction;

 function new();
  super.new();
  this.SpecVer = DENALI_CDN_AXI_SPECVERSION_AMBA3;
  this.SpecSubtype = DENALI_CDN_AXI_SPECSUBTYPE_BASE;
  this.SpecInterface = DENALI_CDN_AXI_SPECINTERFACE_FULL;
   endfunction

    constraint user_dut_information {

        (StartAddress >= 'h0) && (StartAddress <= 'h1FFF);
        BurstMaxSize == DENALI_CDN_AXI_TRANSFERSIZE_EIGHT_WORDS;
        IdTag < (1 << 14);
      Cacheable == DENALI_CDN_AXI_CACHEMODE_NON_CACHEABLE;

      ModelGeneration == 0;
    }

endclass
```

```
// A denaliCdn_axiInstance should be instantiated per agent
// You should extend it and create your own, to implement callbacks
class axiInstance extends denaliCdn_axiInstance;

    denaliMemInstance regInst ; // Handle to the register-space

    function new(string instName);
        super.new(instName);
        regInst = new( { instName, "(registers)" } );
    endfunction

    function void mapMemorySegment(reg [63:0] fromAddress, reg [63:0] toAddress);
        regWrite( DENALI_CDN_AXI_REG_RegionAddressFromHigh,fromAddress[63:32]);
        regWrite( DENALI_CDN_AXI_REG_RegionAddressFromLow, fromAddress[31:0]);
        regWrite( DENALI_CDN_AXI_REG_RegionAddressToHigh ,toAddress[63:32]);
        regWrite( DENALI_CDN_AXI_REG_RegionAddressToLow ,toAddress[31:0]);
        regWrite( DENALI_CDN_AXI_REG_RegionType ,DENALI_CDN_AXI_DOMAIN_NON_SHAREABLE);
    endfunction

    virtual function int DefaultCbF(ref denaliCdn_axiTransaction trans);

     if (trans != null && trans.Callback == DENALI_CDN_AXI_CB_CoverageSample) return
 super.DefaultCbF(trans);
        if (trans != null)
        $display("*******************************");
        $display("DefaultCbF :  Type:  ",trans.Type.name());
        $display("DefaultCbF :  Callback:   ",trans.Callback.name());
        $display("DefaultCbF :  Agent name:   ", instName);
        $display("*******************************");

        begin
            if (trans.Callback == DENALI_CDN_AXI_CB_Error) begin
                $display("DefaultCbF :  ErrorInfo:  ",trans.ErrorInfo);
                $display("DefaultCbF :  ErrorId:  ",trans.ErrorId);
            end
        end

        return super.DefaultCbF(trans);
    endfunction

    //
    // Read from a 32-bit denali register
    //
    virtual function reg [31:0] regRead( denaliCdn_axiRegNumT addr );
        denaliMemTransaction trans = new();
        trans.Address = addr ;
        void'( regInst.read( trans ) );
        regRead[31:24] = trans.Data[0];
        regRead[23:16] = trans.Data[1];
        regRead[15:08] = trans.Data[2];
        regRead[07:00] = trans.Data[3];
    endfunction

    //
    // Write to a 32-bit denali register
    //
    virtual function void regWrite( denaliCdn_axiRegNumT addr, reg [31:0] data );
        denaliMemTransaction trans = new();
        trans.Address = addr ;
        trans.Data = new[4];
        trans.Data[0] = data[31:24];
        trans.Data[1] = data[23:16];
        trans.Data[2] = data[15:08];
        trans.Data[3] = data[07:00];
        void'( regInst.write( trans ) );
    endfunction
```

```
endclass

module pstest;

        reg aclk;
        reg aresetn;
        integer status;

  wire awvalid;
  wire [63:0] awaddr;
  wire [3:0] awlen;
  wire [2:0] awsize;
  wire [1:0] awburst;
  wire [1:0] awlock;
  wire [3:0] awcache;
  wire [2:0] awprot;
  wire [14:0] awid;
  wire awready;
  wire [255:0] awuser;
  wire wvalid;
  wire wlast;
  wire [255:0] wdata;
  wire [31:0] wstrb;
  wire [14:0] wid;
  wire wready;
  wire [255:0] wuser;
  wire bvalid;
  wire [1:0] bresp;
  wire [14:0] bid;
  wire bready;
  wire [255:0] buser;
  wire arvalid;
  wire [63:0] araddr;
  wire [3:0] arlen;
  wire [2:0] arsize;
  wire [1:0] arburst;
  wire [1:0] arlock;
  wire [3:0] arcache;
  wire [2:0] arprot;
  wire [14:0] arid;
  wire arready;
  wire [255:0] aruser;
  wire rvalid;
  wire rlast;
  wire [255:0] rdata;
  wire [1:0] rresp;
  wire [14:0] rid;
  wire rready;
  wire [255:0] ruser;

        always #50 aclk = ~aclk;

        initial
        begin
            aclk = 1'b0;
            aresetn = 1'b1;
            #100
            aresetn = 1'b0;
            #500
            aresetn = 1'b1;
        end

        activeMaster axiMasterDevice(aclk,
    aresetn,
    awvalid,
    awaddr,
    awlen,
```

```
    awsize,
    awburst,
    awlock,
    awcache,
    awprot,
    awid,
    awready,
    awuser,
    wvalid,
    wlast,
    wdata,
    wstrb,
    wid,
    wready,
    wuser,
    bvalid,
    bresp,
    bid,
    bready,
    buser,
    arvalid,
    araddr,
    arlen,
    arsize,
    arburst,
    arlock,
    arcache,
    arprot,
    arid,
    arready,
    aruser,
    rvalid,
    rlast,
    rdata,
    rresp,
    rid,
    rready,
    ruser);

       activeSlave axiSlaveDevice(aclk,
          aresetn,
          awvalid,
          awaddr,
          awlen,
          awsize,
          awburst,
          awlock,
          awcache,
          awprot,
          awid,
          awready,
          awuser,
          wvalid,
          wlast,
          wdata,
          wstrb,
          wid,
          wready,
          wuser,
          bvalid,
          bresp,
          bid,
          bready,
          buser,
          arvalid,
          araddr,
          arlen,
```

```
          arsize,
          arburst,
          arlock,
          arcache,
          arprot,
          arid,
          arready,
          aruser,
          rvalid,
          rlast,
          rdata,
          rresp,
          rid,
          rready,
          ruser);

       axiInstance axiActiveMaster;
       axiInstance axiActiveSlave;

       //myTransaction extends denaliCdn_axiTransaction with DUT specfic constraints
       myTransaction masterBurst;
       myTransaction slaveResp;

   initial
   begin
       $display("Going to create new instances ");
       axiActiveMaster = new ("pstest.axiMasterDevice");
       axiActiveSlave = new ("pstest.axiSlaveDevice");
       $display("Created new instances");

       void'(axiActiveMaster.setCallback(DENALI_CDN_AXI_CB_Error));
       void'(axiActiveMaster.setCallback(DENALI_CDN_AXI_CB_Ended));

       void'(axiActiveSlave.setCallback(DENALI_CDN_AXI_CB_Error));

       $display("Did the set call backs for the active instances");

       // Enable coverage collection
       void'(axiActiveMaster.create_cover());

       //we can encaplsuate the logic with USER code to avoid writing too much code
       axiActiveMaster.mapMemorySegment(64'h0,64'h1FFF);
       axiActiveSlave.mapMemorySegment(64'h0,64'h1FFF);

       //set test verbosity to HIGH
       //we can use the enum value as set in the denaliCdn_axiTypes.svh file
       axiActiveMaster.regWrite( DENALI_CDN_AXI_REG_Verbosity,
DENALI_CDN_AXI_MESSAGEVERBOSITY_HIGH );
       axiActiveSlave.regWrite( DENALI_CDN_AXI_REG_Verbosity, DENALI_CDN_AXI_MESSAGEVERBOSITY_HIGH );

       //setting this check to IGNORE
       axiActiveMaster.regWrite(DENALI_CDN_AXI_REG_ErrorID,
CDN_AXI_NONFATAL_ERR_VR_AXI000_WRONG_DIRECTION_BURST_RECEIVED_FROM_DRIVER);
       axiActiveMaster.regWrite(DENALI_CDN_AXI_REG_ErrorSeverity, DENALI_CDN_AXI_CHECKEFFECT_IGNORE);

    axiActiveMaster.regWrite(DENALI_CDN_AXI_REG_ResetSignalsSimStart, 1);
       axiActiveSlave.regWrite(DENALI_CDN_AXI_REG_ResetSignalsSimStart, 1);

       axiActiveMaster.regWrite(DENALI_CDN_AXI_REG_WriteDataInterleavingDepth, 6);
       axiActiveSlave.regWrite(DENALI_CDN_AXI_REG_WriteDataInterleavingDepth, 6);

       masterBurst = new();
       slaveResp = new();

       #1000;

       //send one transaction
```

```
        masterBurst.StartAddress = 0;
        status = axiActiveMaster.transAdd(masterBurst, DENALI_CDN_AXI_QUEUE_Burst);
        #1000;

        //you can send several transactions in a loop
        for (int i=0; i<50; i++) begin
            assert(masterBurst.randomize() with {
                StartAddress >= 'h0;
                StartAddress <= 'h1FFF;
            });
            status = axiActiveMaster.transAdd(masterBurst, DENALI_CDN_AXI_QUEUE_Burst);

            #100;

        end

        #500000;

        $finish;

    end

endmodule
```

10.3. SystemVerilog Interface for UVM

This section provides details on how to integrate CDN_AXI VIP into a UVM compliant test environments.

The CDN_AXI VIP UVM layer is located under `$CDN_VIP_ROOT/tools/denali/ddvapi/sv/uvm/cdn_axi`. All UVM layer classes are defined under the `cdnAxiUvm` package.

10.3.1. Prerequisites

The prerequisites to integrate CDN_AXI VIP to the UVM environment are as follows:

- Understanding of the SystemVerilog UVM. For details on UVM, refer to the website http://www.uvmworld.org [http://www.uvmworld.org].
- Understanding of the CDN_AXI VIP, including the SystemVerilog Interface (Section 10.2, *"SystemVerilog Interface"*).

You can also refer to the *Cadence UVM SystemVerilog User Guide* installed in the Incisive release.

Each release also includes a UVM API reference document that is automatically generated with information about structs, fields, methods, and events. This document can be very helpful for debugging. To read this reference, open the following file with your Web browser:

- `$CDN_VIP_ROOT/doc/cdn_axi/axi_uvm_sv_ref/index.html`

The Cadence UVM Layer supports UVM version 1.1 and supports simulators NCSIM 10.2, VCS 2011.03-SP1-2, and MTI 10.0c.

10.3.2. Using UVM with Different HDL Simulators

Refer to Section 10.1, "*Simulator Integration*".

Here are some examples that show how you can simulate the CDN_AXI VIP and SystemVerilog UVM with different HDL simulators.

10.3.2.1. NCSIM

This section provides examples of VIP script invocations for setting up the environment and running UVM examples in the release with NCSIM.

Three-Step Mode

Examples of script invocations for NCSIM in three-step mode:

- Setting up C shell environment for the first time (VIP compiled libraries need to be installed with the -i option):

```
cdn_vip_setup_env -s ncsim_3s -cdn_vip_root <VIPCAT_Installation_Directory> -m sv_uvm -i axi -
cdn_vip_lib  <VIP_Compiled_Library_Directory> -csh
source cdn_vip_env_ncsim_sv_uvm.csh
```

- Setting up (and running) an example provided in the VIPCAT release:

```
cdn_vip_setup_example -s ncsim_3s -cdn_vip_root <VIPCAT_Installation_Directory> -e
 <VIPCAT_Example_Directory> -m sv_uvm -cdn_vip_lib  <VIP_Compiled_Library_Directory>
cdn_vip_run_ncsim_sv_uvm.sh
```

Exact script invocations to set up one specific UVM example in the VIPCAT release for NCSIM in three-step mode are shown below. The complete script with all steps is available at `$CDN_VIP_ROOT/tools/denali/ddvapi/sv/uvm/cdn_axi/examples/axiSimpleExample/example_setup_ncsim.csh`.

```
 #!/bin/csh -f
    #Step 1. Create cdn_vip_env_ncsim_sv_uvm.csh with environment setup commands
    #Note: NCSIM Compiled libraries path to be provided + libraries need to be installed
${CDN_VIP_ROOT}/bin/cdn_vip_setup_env -s ncsim_3s -cdn_vip_root ${CDN_VIP_ROOT} -m sv_uvm -csh -
cdn_vip_lib vip_lib -i axi
    #Step 2. Complete environment setup by executing generated shell commands
source cdn_vip_env_ncsim_sv_uvm.csh

    #Step 3. Create cdn_vip_run_ncsim_sv_uvm.sh with simulator comands
    #Note: CDN_VIP_ROOT environment variable guaranteed to be available after Step 2
${CDN_VIP_ROOT}/bin/cdn_vip_setup_example -s ncsim_3s -e ${CDN_VIP_ROOT}/tools/denali/ddvapi/sv/uvm/
cdn_axi/examples/axiSimpleExample -m sv_uvm -cdn_vip_lib vip_lib
    #Step 4. Execute generated script with compile/simulate commands (also doable by -r option to
 cdn_vip_setup_example)
./cdn_vip_run_ncsim_sv_uvm.sh
```

Please confirm the following two requirements are met before executing the above scripts:

- CDN_VIP_ROOT environment variable is correctly specified.

- The 'ncsim' executable is available in your path (and you have a ncsim license).

Single-Step Mode

Examples of script invocations for NCSIM in single-step mode:

- Setting up C shell environment for the first time (VIP compiled libraries need to be installed with the -i option):

```
cdn_vip_setup_env -s ncsim_irun -cdn_vip_root <VIPCAT_Installation_Directory> -m sv_uvm -i axi -
cdn_vip_lib  <VIP_Compiled_Library_Directory> -csh
source cdn_vip_env_irun_sv_uvm.csh
```

- Setting up (and running) an example provided in the VIPCAT release:

```
cdn_vip_setup_example -s ncsim_irun -cdn_vip_root <VIPCAT_Installation_Directory> -e
 <VIPCAT_Example_Directory> -m sv_uvm -cdn_vip_lib <VIP_Compiled_Library_Directory>
cdn_vip_run_irun_sv_uvm.sh
```

Exact script invocations to set up one specific UVM example in the VIPCAT release for NC-SIM in single-step mode are shown below. The complete script with all steps is available at `$CDN_VIP_ROOT/tools/denali/ddvapi/sv/uvm/cdn_axi/examples/axiSimpleExample/example_setup_irun.csh`.

```
 #!/bin/csh -f
    #Step 1. Create cdn_vip_env_irun_sv_uvm.csh with environment setup commands
    #Note: NCSIM Compiled libraries path to be provided + libraries need to be installed
${CDN_VIP_ROOT}/bin/cdn_vip_setup_env -s ncsim_irun -cdn_vip_root ${CDN_VIP_ROOT} -m sv_uvm -csh -
cdn_vip_lib vip_lib -i axi
    #Step 2. Complete environment setup by executing generated shell commands
source cdn_vip_env_irun_sv_uvm.csh

    #Step 3. Create cdn_vip_run_irun_sv_uvm.sh with simulator comands
    #Note: CDN_VIP_ROOT environment variable guaranteed to be available after Step 2
${CDN_VIP_ROOT}/bin/cdn_vip_setup_example -s ncsim_irun -e ${CDN_VIP_ROOT}/tools/denali/ddvapi/sv/uvm/
cdn_axi/examples/axiSimpleExample -m sv_uvm -cdn_vip_lib vip_lib
    #Step 4. Execute generated script with compile/simulate commands (also doable by -r option to
 cdn_vip_setup_example)
./cdn_vip_run_irun_sv_uvm.sh
```

Please confirm the following two requirements are met before executing the above scripts:

- CDN_VIP_ROOT environment variable is correctly specified.
- The 'ncsim' executable is available in your path (and you have a ncsim license).

10.3.2.1.1. Single-Step Mode with Incremental Elaboration

Exact script invocations demonstrating incremental elaboration in single-step mode are shown below. The complete script with all steps is available at `$CDN_VIP_ROOT/tools/denali/ddvapi/sv/uvm/cdn_axi/examples/axiIncrIrunExample/example_setup_irun_incr.csh`.

```
 #!/bin/csh -f
    #Step 1. Create cdn_vip_env_irun_sv_uvm.csh with environment setup commands
    #Note: NCSIM Compiled libraries path to be provided + libraries need to be installed
${CDN_VIP_ROOT}/bin/cdn_vip_setup_env -s ncsim_irun -cdn_vip_root ${CDN_VIP_ROOT} -m sv_uvm -csh -
cdn_vip_lib vip_lib -i axi
    #Step 2. Complete environment setup by executing generated shell commands
source cdn_vip_env_irun_sv_uvm.csh

    #Step 3. Create cdn_vip_run_irun_sv_uvm.sh with simulator comands
    #Note: CDN_VIP_ROOT environment variable guaranteed to be available after Step 2
${CDN_VIP_ROOT}/bin/cdn_vip_setup_example -s ncsim_irun_incr -e ${CDN_VIP_ROOT}/tools/denali/ddvapi/
sv/uvm/cdn_axi/examples/axiIncrIrunExample -m sv_uvm -cdn_vip_lib vip_lib
    #Step 4. Execute generated script with compile/simulate commands (also doable by -r option to
 cdn_vip_setup_example)
    #Important: This script contains 2 irun commands; The second irun command
    #can be executed repeatedly (incremental elab) for multiple tests
./cdn_vip_run_irun_incr_sv_uvm.sh
```

Please confirm the following two requirements are met before executing the above scripts:

- CDN_VIP_ROOT environment variable is correctly specified.
- The 'ncsim' executable is available in your path (and you have a ncsim license).

10.3.2.2. VCS

This section provides examples of VIP script invocations for setting up the environment and running UVM examples in the release with VCS.

Examples of script invocations for VCS:

- Setting up C shell environment:
- The environment variable `CDN_VIP_ROOT` is used to refer to the path to the VIP installation.

```
cdn_vip_setup_env -s vcs -cdn_vip_root <VIPCAT_Installation_Directory> -m sv_uvm
source cdn_vip_env_vcs_sv_uvm.csh
```

- Setting up (and running) an example provided in the VIPCAT release:

```
cdn_vip_setup_example -s vcs -cdn_vip_root <VIPCAT_Installation_Directory> -e
 <VIPCAT_Example_Directory> -m sv_uvm
source cdn_vip_env_vcs_sv_uvm.csh
```

Exact script invocations to set up one specific UVM example in the VIPCAT release with VCS are shown below. The complete script with all steps is available at `$CDN_VIP_ROOT/tools/denali/ddvapi/sv/uvm/cdn_axi/examples/axiSimpleExample/example_setup_vcs.csh`.

```
 #!/bin/csh -f
    #Step 1. Create cdn_vip_env_vcs_sv_uvm.csh with environment setup commands
${CDN_VIP_ROOT}/bin/cdn_vip_setup_env -s vcs -cdn_vip_root ${CDN_VIP_ROOT} -m sv_uvm -csh
    #Step 2. Complete environment setup by executing generated shell commands
source cdn_vip_env_vcs_sv_uvm.csh
```

```
    #Step 3. Create cdn_vip_run_vcs_sv_uvm.sh with simulator comands
    #Note: CDN_VIP_ROOT environment variable guaranteed to be available after Step 2
${CDN_VIP_ROOT}/bin/cdn_vip_setup_example -s vcs -e ${CDN_VIP_ROOT}/tools/denali/ddvapi/sv/uvm/
cdn_axi/examples/axiSimpleExample -m sv_uvm
    #Step 4. Execute generated script with compile/simulate commands (also doable by -r option to
 cdn_vip_setup_example)
./cdn_vip_run_vcs_sv_uvm.sh
```

Please confirm the following two requirements are met before executing the above scripts:

- CDN_VIP_ROOT environment variable is correctly specified.
- The 'vcs' executable is available in your path (and you have a vcs license).

10.3.2.3. MTI

This section provides examples of VIP script invocations for setting up the environment and running UVM examples in the release with MTI.

Examples of script invocations for MTI:

- Setting up C shell environment:

```
cdn_vip_setup_env -s mti -cdn_vip_root <VIPCAT_Installation_Directory> -m sv_uvm
source cdn_vip_env_mti_sv_uvm.csh
```

- Setting up (and running) an example provided in the VIPCAT release:

```
cdn_vip_setup_example -s mti -cdn_vip_root <VIPCAT_Installation_Directory> -e
 <VIPCAT_Example_Directory> -m sv_uvm
cdn_vip_run_mti_sv_uvm.sh
```

Exact script invocations to set up one specific UVM example in the VIPCAT release with MTI are shown below. The complete script with all steps is available at `$CDN_VIP_ROOT/tools/denali/ddvapi/sv/uvm/cdn_axi/examples/axiSimpleExample/example_setup_mti.csh`.

```
 #!/bin/csh -f
    #Step 1. Create cdn_vip_env_mti_sv_uvm.csh with environment setup commands
${CDN_VIP_ROOT}/bin/cdn_vip_setup_env -s mti -cdn_vip_root ${CDN_VIP_ROOT} -m sv_uvm -csh
    #Step 2. Complete environment setup by executing generated shell commands
source cdn_vip_env_mti_sv_uvm.csh

    #Step 3. Create cdn_vip_run_mti_sv_uvm.sh with simulator comands
    #Note: CDN_VIP_ROOT environment variable guaranteed to be available after Step 2
${CDN_VIP_ROOT}/bin/cdn_vip_setup_example -s mti -e ${CDN_VIP_ROOT}/tools/denali/ddvapi/sv/uvm/
cdn_axi/examples/axiSimpleExample -m sv_uvm
    #Step 4. Execute generated script with compile/simulate commands (also doable by -r option to
 cdn_vip_setup_example)
./cdn_vip_run_mti_sv_uvm.sh
```

Please confirm the following two requirements are met before executing the above scripts:

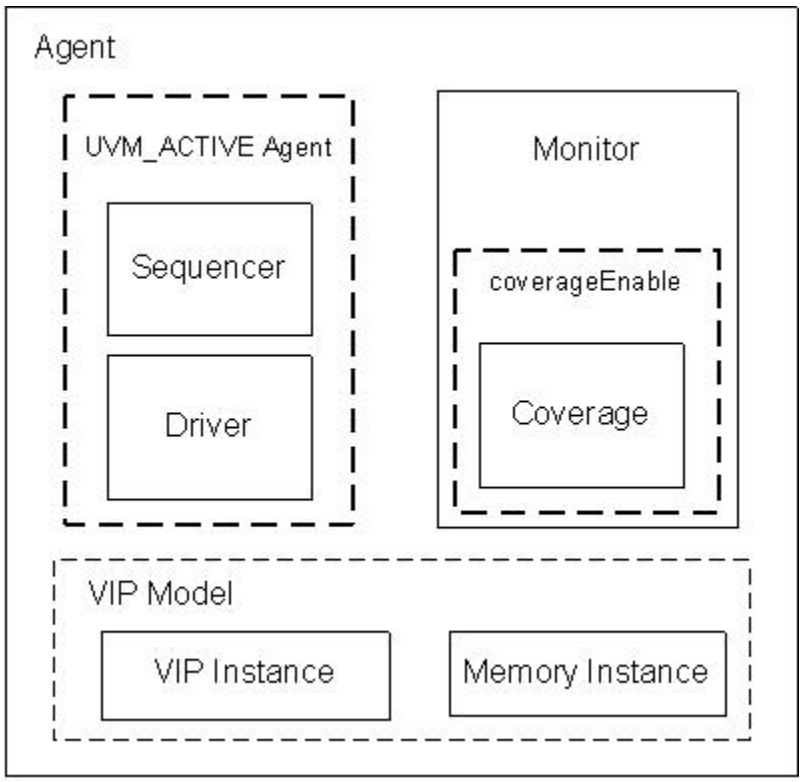
- CDN_VIP_ROOT environment variable is correctly specified.

- The 'vsim' executable is available in your path (and you have a vsim license).

10.3.3. Architecture

The following diagram shows the CDN_AXI VIP UVM agent architecture.

Figure 10.1. The CDN_AXI VIP UVM Agent Architecture



The CDN_AXI VIP SystemVerilog interface contains the following UVM components.

10.3.3.1. Agent

The Agent is an independent UVM device that contains the standard VIP UVM components. The CDN_AXI VIP UVM agent is of type cdnAxiUvmAgent and inherits from uvm_agent. The agent instantiates monitor (of type cdnAxiUvmMonitor with a reference name **monitor**), driver (of type cdnAxiUvmDriver with a reference name **driver**), Seqeuncer (of type cdnAxiUvmSequencer with a reference name **sequencer**), instance (of type cdnAxiUvmInstance with a reference name **inst**) and memory instance (of type cdnAxiUvmMemInstance with a reference name **regInst**).The driver and sequencer get instantiated only when `is_active` is set to UVM_ACTIVE.

You must configure the agent hdlPath to have the full path of the testbench module that correspond to the specific agent. Refer to example shown in the `cdnAxiUvmUserSve.sv` file.

Refer to the CDN_AXI VIP UVM agent example at `$CDN_VIP_ROOT/tools/denali/ddvapi/sv/uvm/cdn_axi/examples/axi3/cdnAxiUvmUserAgent.sv`, the CDN_AXI VIP UVM Master agent example at `$CDN_VIP_ROOT/tools/denali/ddvapi/sv/uvm/`

`cdn_axi/examples/axi3/cdnAxiUvmUserMasterAgent.sv` and the CDN_AXI VIP UVM Slave agent example at `$CDN_VIP_ROOT/tools/denali/ddvapi/sv/uvm/cdn_axi/examples/axi3/cdnAxiUvmUserSlaveAgent.sv`.

10.3.3.2. Monitor

The monitor is a passive entity that samples DUT signals but does not drive them. The monitor collects transactions, extracts events, performs checking and coverage and in general provides information about the device activity.

The monitor instantiates coverage (of type cdnAxiUvmCoverage with reference name coverModel), the coverage is instantiated only when coverageEnable is set to 1.

The CDN_AXI VIP monitor is of type cdnAxiUvmMonitor and inherits from uvm_monitor. It contains the following analysis ports and events.

Note that the relevant analysis port/event is triggered when the equivalent callback is called by the model and only if the callback is enabled. To enable a callback use the Instance's `setCallback()` method. For details on the CDN_AXI VIP Callbacks, refer to Chapter 7, *CDN_AXI VIP Callbacks*.

10.3.3.2.1. CDN_AXI Events and Ports

The following table lists the CDN_AXI events and ports used in the UVM layer. For more details on the CDN_AXI callbacks, refer to Section 7.1, "*Callback Types*".

Table 10.3. CDN_AXI Events

Event Name	Description
DefaultCbEvent	This event is triggered when the DefaultCbF function in the instance class is called.
ErrorCbEvent	This event is triggered when the ErrorCbF function in the instance class is called if the Error callback is enabled. For more information, refer to the Error callback.
AssertPassCbEvent	This event is triggered when the AssertPassCbF function in the instance class is called if the AssertPass callback is enabled. For more information, refer to the AssertPass callback.
ResetStartedCbEvent	This event is triggered when the ResetStartedCbF function in the instance class is called if the ResetStarted callback is enabled. For more information, refer to the ResetStarted callback.
ResetEndedCbEvent	This event is triggered when the ResetEndedCbF function in the instance class is called if the ResetEnded callback is enabled. For more information, refer to the ResetEnded callback.
RetrieveItemCbEvent	This event is triggered when the RetrieveItemCbF function in the instance class is called if the RetrieveItem callback is enabled. For more information, refer to the RetrieveItem callback.

BeforeSendCbEvent	This event is triggered when the BeforeSendCbF function in the instance class is called if the BeforeSend callback is enabled. For more information, refer to the BeforeSend callback.
BeforeSendAddressCbEvent	This event is triggered when the BeforeSendAddressCbF function in the instance class is called if the BeforeSendAddress callback is enabled. For more information, refer to the BeforeSendAddress callback.
BeforeSendResponseCbEvent	This event is triggered when the BeforeSendResponseCbF function in the instance class is called if the BeforeSendResponse callback is enabled. For more information, refer to the BeforeSendResponse callback.
BeforeSendTransferCbEvent	This event is triggered when the BeforeSendTransferCbF function in the instance class is called if the BeforeSendTransfer callback is enabled. For more information, refer to the BeforeSendTransfer callback.
StartedCbEvent	This event is triggered when the StartedCbF function in the instance class is called if the Started callback is enabled. For more information, refer to the Started callback.
StartedAddressCbEvent	This event is triggered when the StartedAddressCbF function in the instance class is called if the StartedAddress callback is enabled. For more information, refer to the StartedAddress callback.
StartedResponseCbEvent	This event is triggered when the StartedResponseCbF function in the instance class is called if the StartedResponse callback is enabled. For more information, refer to the StartedResponse callback.
StartedTransferCbEvent	This event is triggered when the StartedTransferCbF function in the instance class is called if the StartedTransfer callback is enabled. For more information, refer to the StartedTransfer callback.
EndedCbEvent	This event is triggered when the EndedCbF function in the instance class is called if the Ended callback is enabled. For more information, refer to the Ended callback.
EndedAddressCbEvent	This event is triggered when the EndedAddressCbF function in the instance class is called if the EndedAddress callback is enabled. For more information, refer to the EndedAddress callback.
EndedResponseCbEvent	This event is triggered when the EndedResponseCbF function in the instance class is called if the EndedResponse callback is enabled. For more information, refer to the EndedResponse callback.
EndedTransferCbEvent	This event is triggered when the EndedTransferCbF function in the instance class is called if the EndedTransfer callback is enabled. For more information, refer to the EndedTransfer callback.

Table 10.4. CDN_AXI Ports

Port Name	Description

DefaultCbPort	This port is written when the DefaultCbF function in the instance class is called.
ErrorCbPort	This port is written when the ErrorCbF function in the instance class is called if the Error callback is enabled. For more information refer to the Error callback.
AssertPassCbPort	This port is written when the AssertPassCbF function in the instance class is called if the AssertPass callback is enabled. For more information refer to the AssertPass callback.
ResetStartedCbPort	This port is written when the ResetStartedCbF function in the instance class is called if the ResetStarted callback is enabled. For more information refer to the ResetStarted callback.
ResetEndedCbPort	This port is written when the ResetEndedCbF function in the instance class is called if the ResetEnded callback is enabled. For more information refer to the ResetEnded callback.
RetrieveItemCbPort	This port is written when the RetrieveItemCbF function in the instance class is called if the RetrieveItem callback is enabled. For more information refer to the RetrieveItem callback.
BeforeSendCbPort	This port is written when the BeforeSendCbF function in the instance class is called if the BeforeSend callback is enabled. For more information refer to the BeforeSend callback.
BeforeSendAddressCbPort	This port is written when the BeforeSendAddressCbF function in the instance class is called if the BeforeSendAddress callback is enabled. For more information refer to the BeforeSendAddress callback.
BeforeSendResponseCbPort	This port is written when the BeforeSendResponseCbF function in the instance class is called if the BeforeSendResponse callback is enabled. For more information refer to the BeforeSendResponse callback.
BeforeSendTransferCbPort	This port is written when the BeforeSendTransferCbF function in the instance class is called if the BeforeSendTransfer callback is enabled. For more information refer to the BeforeSendTransfer callback.
StartedCbPort	This port is written when the StartedCbF function in the instance class is called if the Started callback is enabled. For more information refer to the Started callback.
StartedAddressCbPort	This port is written when the StartedAddressCbF function in the instance class is called if the StartedAddress callback is enabled. For more information refer to the StartedAddress callback.
StartedResponseCbPort	This port is written when the StartedResponseCbF function in the instance class is called if the StartedResponse callback is enabled. For more information refer to the StartedResponse callback.

StartedTransferCbPort	This port is written when the StartedTransferCbF function in the instance class is called if the StartedTransfer callback is enabled. For more information refer to the StartedTransfer callback.
EndedCbPort	This port is written when the EndedCbF function in the instance class is called if the Ended callback is enabled. For more information refer to the Ended callback.
EndedAddressCbPort	This port is written when the EndedAddressCbF function in the instance class is called if the EndedAddress callback is enabled. For more information refer to the EndedAddress callback.
EndedResponseCbPort	This port is written when the EndedResponseCbF function in the instance class is called if the EndedResponse callback is enabled. For more information refer to the EndedResponse callback.
EndedTransferCbPort	This port is written when the EndedTransferCbF function in the instance class is called if the EndedTransfer callback is enabled. For more information refer to the EndedTransfer callback.

For more information on how to use the CDN_AXI monitor, refer to the CDN_AXI VIP UVM example at `$CDN_VIP_ROOT/tools/denali/ddvapi/sv/uvm/cdn_axi/examples`.

10.3.3.2.2. Implementing CDN_AXI Callbacks with Ports in UVM

This example shows the implementation of the `Ended` callback to the UVM monitor. You can also refer to this example located at `$DENALI/ddvapi/sv/uvm/cdn_axi/examples/axi3`.

Here are the steps:

1. Add a declaration in `CdnAxiUvmUserTypes.sv`:

```
`uvm_analysis_imp_decl(_cdn_axi_burst_ended)
```

2. The implementation gets added in `CdnAxiUvmUserMonitor` (extending `CdnAxiUvmMonitor`):

```
class cdnAxiUvmUserMonitor extends cdnAxiUvmMonitor;

uvm_analysis_imp_cdn_axi_burst_ended #(denaliCdn_axiTransaction, cdnAxiUvmUserMonitor) endedImp;
...
endclass:cdnAxiUvmUserMonitor
```

3. Add the method of the callback implementation. This example simply prints information about the ended burst:

```
  virtual function void write_cdn_axi_burst_ended (denaliCdn_axiTransaction trans);
    if (trans != null) begin
      printEndedBurst(trans);
    end
  endfunction : write_cdn_axi_burst_ended
```

4. Add the port creation in the class constructor:

```
  function new(string name = "cdnAxiUvmUserMonitor", uvm_component parent = null);
    super.new(name, parent);
    endedImp = new("endedImp", this);
  endfunction : new
```

5. Connect the analysis ports to imp ports:

```
  virtual function void connect();
      super.connect();
      this.EndedCbPort.connect(this.endedImp);
  endfunction
```

Note

All the analysis ports for the callback are available at `$DENALI/ddvapi/sv/uvm/cdn_axi/cdnAxiUvmMonitor.sv`, the format of the callback port names is <*callback_name*>`CbPort`.

10.3.3.2.3. Implementing CDN_AXI Callbacks with Events in UVM

This example shows how to add the event `writeEnded` that is triggered when a `Read` transaction ended. You can also refer to this example located at `$DENALI/ddvapi/sv/uvm/cdn_axi/examples/axi3`.

Add the event inside the `cdnAxiUvmUserMonitor` (extending `cdnAxiUvmMonitor`).

Here are the steps:

1. Add the event declaration to the Monitor:

```
class cdnAxiUvmUserMonitor extends cdnAxiUvmMonitor;
uvm_event readEnded;
...
endclass : cdnAxiUvmUserMonitor
```

2. Add the event to the registration macro:

```
`uvm_component_utils_begin(cdnAxiUvmUserMonitor)
...
 `uvm_field_event(readEnded, UVM_ALL_ON)
`uvm_component_utils_end
```

3. Add the event creation in the class constructor:

```
function new(string name = "cdnAxiUvmUserMonitor", uvm_component parent = null);
   super.new(name, parent);
   ...
   readEnded = new("readEnded");
endfunction : new
```

4. Add the event triggering in the Ended callback:

```
virtual function void write_cdn_axi_burst_ended (denaliCdn_axiTransaction trans);
   if (trans.Type == DENALI-CDN_AXI_TR_READ) begin
      this.readEnded.trigger(trans);
```

```
   end
endfunction : write_cdn_axi_burst_ended
```

5. Catch the event in the test:

```
class simple_seq extends cdnAxiUvmVirtualSequence;
  cdnAxiUvmUserMonitor event_monitor;

virtual task body();
 $cast(event_monitor, p_sequencer.pEnv.activeMaster.monitor);

 `uvm_info(get_type_name(), "Virtual sequence sending master read burst", UVM_LOW);
     `uvm_do_on_with(masterBurst, p_sequencer.masterSeqr, {
        masterBurst.Direction == DENALI_CDN_AXI_DIRECTION_READ;
        masterBurst.StartAddress < 'h100;
      })
      event_monitor.readEnded.wait_trigger();
  endtask

endclass : simple_seq
```

10.3.3.3. Driver

The Driver is a verification component that takes items from the Sequencer and drive them to the DUT.

The CDN_AXI VIP UVM driver is of type cdnAxiUvmDriver and inherits from uvm_driver. It includes the predefined implementation of the `run_phase()` method. For details about the Cadence UVM driver implementation and predefined methods refer to `$CDN_VIP_ROOT/tools/denali/ddvapi/sv/uvm/cdn_axi/cdnAxiUvmDriver.sv`

For details on how to use the different CDN_AXI VIP UVM drivers, refer to the examples at `$CDN_VIP_ROOT/tools/denali/ddvapi/sv/uvm/cdn_axi/examples/axi3/cdnAxiUvmUserMasterDriver.sv` and`$CDN_VIP_ROOT/tools/denali/ddvapi/sv/uvm/cdn_axi/examples/axi3/cdnAxiUvmUserSlaveDriver.sv.`

The CDN_AXI VIP UVM driver is instantiated only under UVM_ACTIVE agent.

10.3.3.4. Sequencer

The Sequencer is an advanced stimulus generator that controls the items provided to the driver for execution.

The CDN_AXI VIP UVM sequencer is of type cdnAxiUvmSequencer and inherits from uvm_sequencer. The CDN_AXI VIP UVM sequencer's sequence item is of type: denaliCdn_axiTransaction.

Refer to section on Transaction Interface in Section 10.2.1, *"Transaction Interface"* for more information on the CDN_AXI VIP Transaction Interface.

10.3.3.5. Sequence

A sequence is a stream of data items generated and sent to the DUT. The sequence represents a scenario.

The CDN_AXI VIP UVM Sequence is of type cdnAxiUvmSequence and inherits from uvm_sequence. The CDN_AXI VIP UVM sequence's data item is of type `denaliAxiTransaction`.

Refer to Section 10.2.1, "*Transaction Interface*" for more information.

For details on how to use CDN_AXI VIP UVM Sequence, refer to `$CDN_VIP_ROOT/tools/denali/ddvapi/sv/uvm/cdn_axi/examples/axi3/cdnAxiUvmUvmUserSeqLib.sv` and`$CDN_VIP_ROOT/tools/denali/ddvapi/sv/uvm/cdn_axi/examples/axi3/cdnAxiUvmUvmUserVirtualSeqLib.sv`.

10.3.3.6. Transaction

The CDN_AXI VIP contains a UVM compliant data item `denaliCdn_axiTransaction` class, which models the data items required to generate the protocol-specific traffic and test scenarios. This class is extended from the `uvm_sequence_item` base class to facilitate the use of UVM sequencers to generate protocol traffic.

For details on how to use CDN_AXI VIP UVM Sequence, refer to `$CDN_VIP_ROOT/tools/denali/ddvapi/sv/uvm/cdn_axi/examples/axi3/cdnAxUvmUserSeqLib.sv` and`$CDN_VIP_ROOT/tools/denali/ddvapi/sv/uvm/cdn_axi/examples/axi3/cdnAxiUvmUserVirtualSeqLib.sv`.

10.3.3.7. CDN_AXI Instance

The CDN_AXI VIP UVM agent contains a reference to cdnAxiUvmInstance which inherits from denaliCdn_axiInstance class that provides an interface to access the CDN_AXI VIP model. For example you can activate the model callbacks using the instance's `setCallback()` method.

For more information about denaliCdn_axiInstance, refer to Section 10.2.1.1, "*Instance Class*".

For details on how to trigger callbacks using the denaliCdn_axiInstance instance, refer to `$CDN_VIP_ROOT/tools/denali/ddvapi/sv/uvm/cdn_axi/examples/axi3/cdnAxUvmUserEnv.sv`.

Note that setting the model callbacks, using `setCallback()` method (Section 10.2.1.1.1.2.5, "*setCallback()*"), enables the monitor equivalent analysis ports and events. For details on the monitor analysis ports and events, refer to Section 10.3.3.2, "*Monitor*".

For details on CDN_AXI VIP Callbacks, refer to Chapter 7, *CDN_AXI VIP Callbacks*.

10.3.3.8. Memory Instance

The CDN_AXI VIP UVM agent contains a reference to cdnAxiUvmMemInstance which inherits from denaliMemInstance class that provides an interface to access the CDN_AXI Verification IP

model configuration and status registers. For more information about the control and status registers of the CDN_AXI VIP, refer to Chapter 5, *CDN_AXI VIP Model Configuration*.

10.3.3.9. Coverage

The CDN_AXI VIP UVM monitor contains a reference to cdnAxiUvmCoverage which inherits form denaliCdn_axiCoverageInstance class that contains CDN_AXI VIP coverage definitions.

For more information about denaliCdn_axiCoverageInstance, refer to Section 10.2.2, *"Coverage Interface / CDN_AXI VIP Coverage"*.

10.3.4. Sequence-Related Features

10.3.4.1. Sending Processed Data Back to Sequence

In some sequences, a generated value depends on the response to previously generated data. By default, the driver is non-blocking, which means that the `item_done` is called at the same cycle the transaction is send to driver using one of the `uvm_do` macros. This causes the `uvm_do` operation to finish.

Sequences can wait for the transmission to be completed using the UVM sequence task `get_response` method. Refer to the example below:

```
    `uvm_do_with(tr,
  {…
  })
  //Wait till transaction is transmitted to DUT
  get_response(response,tr.get_transaction_id());
```

10.3.4.1.1. CDN_AXI Get Response Callbacks

Transaction transmission is completed when the following instance function callbacks occur:

- Ended

10.3.4.2. Modify Transaction Sequence

This sequence enables you to modify transaction that is already in process. It is very useful specially for error injection and responses modification.

The `cdnAxiUvmModifyTransactionSequence` encapsulates the logic of the scenario and the logic of the modification in one class. The scenario logic is available in the sequence `body()` task.

The modification logic is available in the sequence `modifyTransaction()` function.

To use the `cdnAxiUvmModifyTransactionSequence` perform the following steps:

1. Create your own sequence class which extends from the `cdnAxiUvmModifyTransaction-Sequence`.

2. Describe the scenario you want to perform in the sequence `body()` task.

3. Describe the errors/transaction modification you want to perform in the sequence predefined `modifyTransaction` function.

From the time when the `cdnAxiModifyTransactionSequence` sequence is started till the sequence ends, the `modifyTransaction` function will be called to any transaction being transmitted by the model.

By default, only transactions with the same sequence id will trigger the `modifyTransaction` functions.

10.3.4.2.1. CDN_AXI Modify Sequence Triggering Callbacks

The `modifyTransaction` function is called at the following instance callbacks:

- BeforeSend
- BeforeSendResponse

10.3.4.2.2. Example

Here is an example:

```
//This example shows how to use the cdnAxiModifyTransactionSequence to
//perform error injection:
//1. sequence is sending a legal transaction
//2. transaction fields then get modified before transaction is send by model, by the code in
//   the modifyTransaction function

class cdnAxiUvmUserErrInjectionSequence extends cdnAxiUvmModifyTransactionSequence;

  // ************************************************************
  // Use the UVM registration macro for this class.
  // ************************************************************
  `uvm_object_utils(cdnAxiUvmUserErrInjectionSequence)


  // ************************************************************
  // Method : new
  // Desc.  : Call the constructor of the parent class.
  // ************************************************************
  function new(string name = "cdnAxiUvmUserErrInjectionSequence");
    super.new(name);
  endfunction : new

  denaliCdn_axiTransaction burst;

  virtual task pre_body();
    if (starting_phase != null) begin
      starting_phase.raise_objection(this);
    end
  endtask

  // ************************************************************
  // Method : post_body
  // Desc.  : Drop the objection raised earlier
  // ************************************************************
```

```
  virtual task post_body();
    if (starting_phase != null) begin
      starting_phase.drop_objection(this);
    end
  endtask


  virtual task body();
    denaliCdn_axiTransaction response;

    `uvm_do_with(burst,
    {burst.Direction == DENALI_CDN_AXI_DIRECTION_READ;
      burst.IdTag < (1 << 4);
      burst.Size == DENALI_CDN_AXI_TRANSFERSIZE_BYTE;
      burst.Length == 8;
      burst.Kind == DENALI_CDN_AXI_BURSTKIND_WRAP;
    })

    //Wait till transaction is transmitted to DUT
    //   we want the sequence to end only after the transaction transmission is completed.
    //   when sequence ends no error injections will occur any more
    get_response(response,burst.get_transaction_id());


    `uvm_info(get_type_name(), "Finished Error injection Sequence", UVM_LOW)
  endtask : body

  //Error injection behavior
  //   This is where we decide what error is injected.  which items should it effect.
  virtual function void modifyTransaction(denaliCdn_axiTransaction tr);
    bit status;
    //in this case we choose to inject an error only to a specific burst this sequence has generated.
    //if there was no condition then the error injection would have occurred to any burst being sent.
    //by default only bursts created in this sequence will be affected

    assert(burst == tr);
    if (burst != null && tr.UserData ==  burst.UserData)
    begin
      `uvm_info(get_type_name(), "Starting Error injection", UVM_LOW)
      tr.Size = DENALI_CDN_AXI_TRANSFERSIZE_FOUR_WORDS;
      tr.StartAddress = tr.StartAddress + 1;
      //Update the model transaction with the new values
      //   transSet() is being used to update that fields were changed - see section X.X.X for more
 information.
      status = tr.transSet();


      `uvm_info(get_type_name(), "Finished Error injection", UVM_LOW)
    end
  endfunction

endclass
```

10.3.5. Error Reporting and Control

10.3.5.1. Changing Message Severity using UVM

All errors from the VIP are reported by the monitor. You can control the severity of an error message using UVM functions such as `set_report_severity_id_override`. For example, if an item should not be reported as an *Error*, you can convert its severity from *Error* to *Info* in the testbench using a function such as the following:

```
agent.monitor.set_report_severity_id_override(UVM_ERROR," <Some Protocol Error value>",UVM_WARNING);
```

10.3.6. The UVM Layer File Structure

You can find the CDN_AXI VIP UVM related files under `$DENALI/ddvapi/sv/uvm/cdn_axi`. The following table lists and describes UVM files.

Table 10.5. UVM Files

Filename	Description
cdnAxiUvmTop.sv	This file defines the cdnAxiUvm package and includes all the CDN_AXI UVM Layer files. To have CDN_AXI UVM Layer you must compile this file and import the cdnAxiUvm package
cdnAxiUvmAgent.sv	This file defines the cdnAxiUvmAgent class.
cdnAxiUvmMonitor.sv	This file defines the cdnAxiUvmMonitor class, that includes all available events and analysis ports.
cdnAxiUvmDriver.sv	This file defines the cdnAxiUvmDriver class, that includes the default implementation of the driver run_phase method.
cdnAxiUvmSequencer.sv	This file defines the cdnAxiUvmSequencer.
cdnAxiUvmSequence.sv	This file defines the cdnAxiUvmSequence.
cdnAxiUvmInstance.sv	This file defines the cdnAxiUvmInstance, that includes triggering and writing the monitor events and analysis ports. For details on Instance Class, refer to Section 10.2.1.1, “*Instance Class*”.
cdnAxiUvmMemInstance.sv	This file defines the cdnAxiUvmMemInstance, that includes predefined methods for reading from registers, writing to registers and implementing memory callbacks. For details on registers, refer to the chapter on Registers.
cdnAxiUvmCoverage.sv	This file defines the cdnAxiUvmCoverage, that includes coverage definitions and collection. For details on Coverage class refer to Section 10.2.2, “*Coverage Interface / CDN_AXI VIP Coverage*”.

10.3.7. UVM Flow

This section provides step-by-step details on the recommended flow that you can use to work with the UVM layer.

10.3.7.1. Creating the UVM Environment

To create the UVM environment:

1. Update the testbench file.

   a. Include and import the relevant packages in the testbench module.

```
`include "uvm_macros.svh"

// Test module
module testbench;

import uvm_pkg::*;

   // Import the DDVAPI CDN_AXI SV interface and the generic Mem interface
   import DenaliSvCdn_axi::*;
   import DenaliSvMem::*;

   // Include the VIP UVM base classes
   import cdnAxiUvm::*;

   // Include the User UVM classes and sequences
   `include "cdnAxiUvmUserTop.sv"

   // tests are here
   `include "cdnAxiUvmUserTest.sv"
```

   b. Instantiate the VIP HDL instantiation interface in the testbench module.

```
// activeMaster is defined in the HDL instantiation interface generated using PureView

activeMaster a_master(aclk,aresetn,awvalid, awaddr,awlen,awsize,awburst,awlock,
awcache, awprot,awid, awready,awuser,wvalid,wlast,wdata,wstrb,wid,wready,
wuser,bvalid,bresp,bid,bready,buser,arvalid,araddr,arlen,arsize,arburst,arlock,
arcache,arprot,arid,arready,aruser,rvalid,rlast,rdata,rresp,rid,rready,ruser);
```

   c. Call `run_test()`.

```
module testbench;
…
 initial
   begin
    run_test();
   end
…
```

2. Instantiate the UVM agent in your environment.

   This example shows the instantiation of the user-specific agent class that is inherited from the UVM agent.

   a. Inherit from the UVM agent (optional).

```
class cdnAxiUvmUserAgent extends cdnAxiUvmAgent;
// your specific additions to the UVM agent
…

class cdnAxiUvmUserMasterAgent extends cdnAxiUvmUserAgent;
// Specific implementation for your master agent
…
```

   b. Instantiate the agent(s) in the environment.

```
class cdnAxiUvmUserEnv extends uvm_env;
```

```
  `uvm_component_utils(cdnAxiUvmUserEnv)
  // ************************************************************
  // The environment instantiates Master and Slave components
  // ************************************************************
  cdnAxiUvmUserMasterAgent activeMaster;
  cdnAxiUvmUserMasterAgent passiveMaster;
  cdnAxiUvmUserSlaveAgent activeSlave;
…
```

c. Set `is_active` field and *create* the agent in the build phase.

```
virtual function void build_phase(uvm_phase phase);
    …
    set_config_int("activeMaster","is_active",UVM_ACTIVE);
    set_config_int("passiveMaster","is_active",UVM_PASSIVE);
    set_config_int("activeSlave","is_active",UVM_ACTIVE);

    // Active Master
    activeMaster = cdnAxiUvmUserMasterAgent::type_id::create("activeMaster", this);
    // Passive Master
    passiveMaster = cdnAxiUvmUserMasterAgent::type_id::create("passiveMaster", this);
    // Active Slave
    activeSlave = cdnAxiUvmUserSlaveAgent::type_id::create("activeSlave", this);class
…
```

d. Enable the relevant model callbacks, per agent instance, in the `end_of_elaboration` phase.

When you enable the model callbacks, it updates the UVM Monitor through its Analysis Ports and Events about internal activities.

```
function void end_of_elaboration_phase(uvm_phase phase);

    super.end_of_elaboration_phase(phase);
    …
    void'(activeMaster.inst.setCallback( DENALI_CDN_AXI_CB_BeforeSend));
    void'(activeMaster.inst.setCallback( DENALI_CDN_AXI_CB_BeforeSendTransfer));
    void'(activeMaster.inst.setCallback( DENALI_CDN_AXI_CB_Ended));
    …
```

3. Instantiate the Environment class in the System Verification Environment (SVE). Unlike the `env` class, the SVE is not considered reusable and may change between different tests.

a. Instantiate the environment, and potentially your virtual sequencer in the SVE.

```
class cdnAxiUvmUserSve extends uvm_env;

  cdnAxiUvmUserEnv myUvmEnv;
  cdnAxiUvmUserVirtualSequencer vs;
…
```

b. Update the factory about your new classes in the constructor.

```
function new(string name = "cdnAxiUvmUserSve", uvm_component parent);
    super.new(name,parent);
    factory.set_type_override_by_type
       (cdnAxiUvmSequencer::get_type(), cdnAxiUvmUserSequencer::get_type());
    factory.set_type_override_by_type
       (cdnAxiUvmInstance::get_type(),cdnAxiUvmUserInstance::get_type());
    factory.set_type_override_by_type
```

```
      (cdnAxiUvmMonitor::get_type(),cdnAxiUvmUserMonitor::get_type());
   factory.set_type_override_by_type
      (cdnAxiUvmMemInstance::get_type(),cdnAxiUvmUserMemInstance::get_type());
   …
endfunction // new
```

c. Set the agent's `hdlPath` field and create the environment class.

```
 virtual function void build_phase(uvm_phase phase);
    …
   //set Passive Master and Passive Slave monitors to collect coverage
   set_config_int("myUvmEnv.passiveMaster.monitor", "coverageEnable",1);

   //set the full HDL path of the agent - this setting is mandatory.
   set_config_string("myUvmEnv.activeMaster","hdlPath", "testbench.a_master");
   set_config_string("myUvmEnv.passiveMaster","hdlPath", "testbench.p_master");
   set_config_string("myUvmEnv.activeSlave","hdlPath", "testbench.a_slave");
   …
   myUvmEnv = cdnAxiUvmUserEnv::type_id::create("myUvmEnv", this);
   vs = cdnAxiUvmUserVirtualSequencer::type_id::create("vs", this);
 endfunction
```

4. Create your tests.

a. Instantiate the SVE in the test and *create* it in the build phase.

```
 class cdnAxiUvmUserTest extends uvm_test;

 cdnAxiUvmUserSve axi_sve;
   …
 virtual function void build_phase(uvm_phase phase);
   super.build_phase(phase);
   axi_sve = cdnAxiUvmUserSve::type_id::create("axi_sve", this);
 endfunction : build_phase
   …
```

b. Set the default sequence to the (virtual) sequencer or sequencers.

```
 virtual function void build_phase(uvm_phase phase);
   super.build_phase(phase);

   //set the starting sequence to system
   uvm_config_db#(uvm_object_wrapper)::set(this, "axi_sve.vs.run_phase",
"default_sequence",simple_seq::type_id::get());
   …
 endfunction : build_phase
```

10.3.7.2. Creating Sequence Library

Cadence recommends that you create a rich and a modular sequence library that can be used as building blocks in other sequences and virtual sequences and directly in your tests.

To create a sequence library:

1. Extend the `Transaction` base class to create your own transactions with your own constraints and settings.

```
class myAxiTransaction extends denaliCdn_axiTransaction;

  `uvm_object_utils(myAxiTransaction)
```

```
  function new(string name = "myAxiTransaction");
    super.new(name);
    this.SpecVer = DENALI_CDN_AXI_SPECVERSION_AMBA3;
    …
  endfunction : new

  constraint user_dut_information {
   BurstMaxSize == DENALI_CDN_AXI_TRANSFERSIZE_EIGHT_WORDS;
    IdTag < (1 << 15);
  }

endclass
```

2. Create a sequence using your own new transaction.

```
// Read seqeunce
class cdnAxiUvmReadSeq extends cdnAxiUvmSequence;

  …
  rand myAxiTransaction burst;

  rand reg [63:0] address;
  rand reg[3:0] length;
  rand denaliCdn_axiTransferSizeT size;
  rand denaliCdn_axiBurstKindT kind;

  …

  virtual task body();
    `uvm_do_with(burst,
      {burst.Direction == DENALI_CDN_AXI_DIRECTION_READ;
      burst.StartAddress == address;
      burst.Length == length;
      burst.Size == size;
      burst.Kind == kind;
    });
  endtask : body

endclass
```

10.3.7.3. Creating Virtual Sequence

The virtual sequence is typically used to create system scenarios or scenarios that involve controlling more than one agent.

To create virtual sequencer and sequences:

1. Create a virtual sequencer with the pointers to the relevant sequencers that will be used by the virtual sequences.

```
class cdnAxiUvmUserVirtualSequencer extends uvm_sequencer;

    cdnAxiUvmUserSequencer masterSeqr;
    cdnAxiUvmUserSequencer slaveSeqr;
     …
  endclass
```

2. Connect the sequencers to the pointers in the `connect_phase` function of the SVE class.

```
class cdnAxiUvmUserSve extends uvm_env;
  …
```

```
  virtual function void connect_phase(uvm_phase phase);
    super.connect_phase(phase);
    $cast(vs.slaveSeqr, myUvmEnv.activeSlave.sequencer);
    $cast(vs.masterSeqr, myUvmEnv.activeMaster.sequencer);
  endfunction
  …
endclass
```

3. Create the virtual sequences.

   a. Create the virtual sequence base class.

      Cadence recommends that you create a base class for all your Virtual Sequences to enable you to write functionality that applies for all sequences only one time.

      In the following example, raising and dropping objections is in the pre_body and post_body functions.

```
class cdnAxiUvmVirtualSequence extends uvm_sequence;

  `uvm_declare_p_sequencer(cdnAxiUvmUserVirtualSequencer)
   …
  virtual task pre_body();
    if (starting_phase != null) begin
      starting_phase.raise_objection(this);
    end
  endtask

  virtual task post_body();
    if (starting_phase != null) begin
      starting_phase.drop_objection(this);
    end
  endtask
 …
endclass : cdnAxiUvmVirtualSequence
```

   b. Use your virtual sequence base class and your sequence library to create virtual sequences.

```
// Read after write example sequence
class readAfterWriteSeq extends cdnAxiUvmVirtualSequence;

  cdnAxiUvmWriteSeq writeSeq;
  cdnAxiUvmReadSeq readSeq;

  rand reg [3:0] same_length;

  reg [63:0] same_address = 64'h0;
  denaliCdn_axiTransferSizeT same_size = DENALI_CDN_AXI_TRANSFERSIZE_WORD;
  denaliCdn_axiBurstKindT same_kind = DENALI_CDN_AXI_BURSTKIND_INCR;

  reg [7:0] w_data[];

  …

  virtual task body();

 …

    // Send a Write following by a Read Transfer to the same address:
    `uvm_do_on_with(writeSeq, p_sequencer.masterSeqr, {
        writeSeq.address == same_address;
```

```
        writeSeq.length  == same_length;
        writeSeq.size    == same_size;
        writeSeq.kind    == same_kind;
    })

    // wait for write burst to end
    wait_for_EndedCbEvent();
…
    `uvm_do_on_with(readSeq, p_sequencer.masterSeqr, {
        readSeq.address == same_address;
        readSeq.length  == same_length;
        readSeq.size    == same_size;
        readSeq.kind    == same_kind;
    })
…
    // wait for read burst to end
    wait_for_EndedCbEvent();
…
  endtask // body

  //This task waits for Ended Callback to be triggered
  virtual task wait_for_EndedCbEvent();
     …
    p_sequencer.pEnv.activeMaster.monitor.EndedCbEvent.wait_trigger_data(obj);
…
  endtask
endclass
```

10.3.8. Generating a Test Case

To create a test case:

1. Refer to Chapter 4, *PureView Graphical Tool*.
2. Create the top level and instantiate the CDN_AXI VIP and DUT models in it.
3. Set the required `.denalirc` variables.
4. Create the testbench and add transactions to the user queue of the host model.

The CDN_AXI VIP UVM example illustrates how to build UVM compliant test environment CDN_AXI models and classes that are part of the VIPCAT installation.

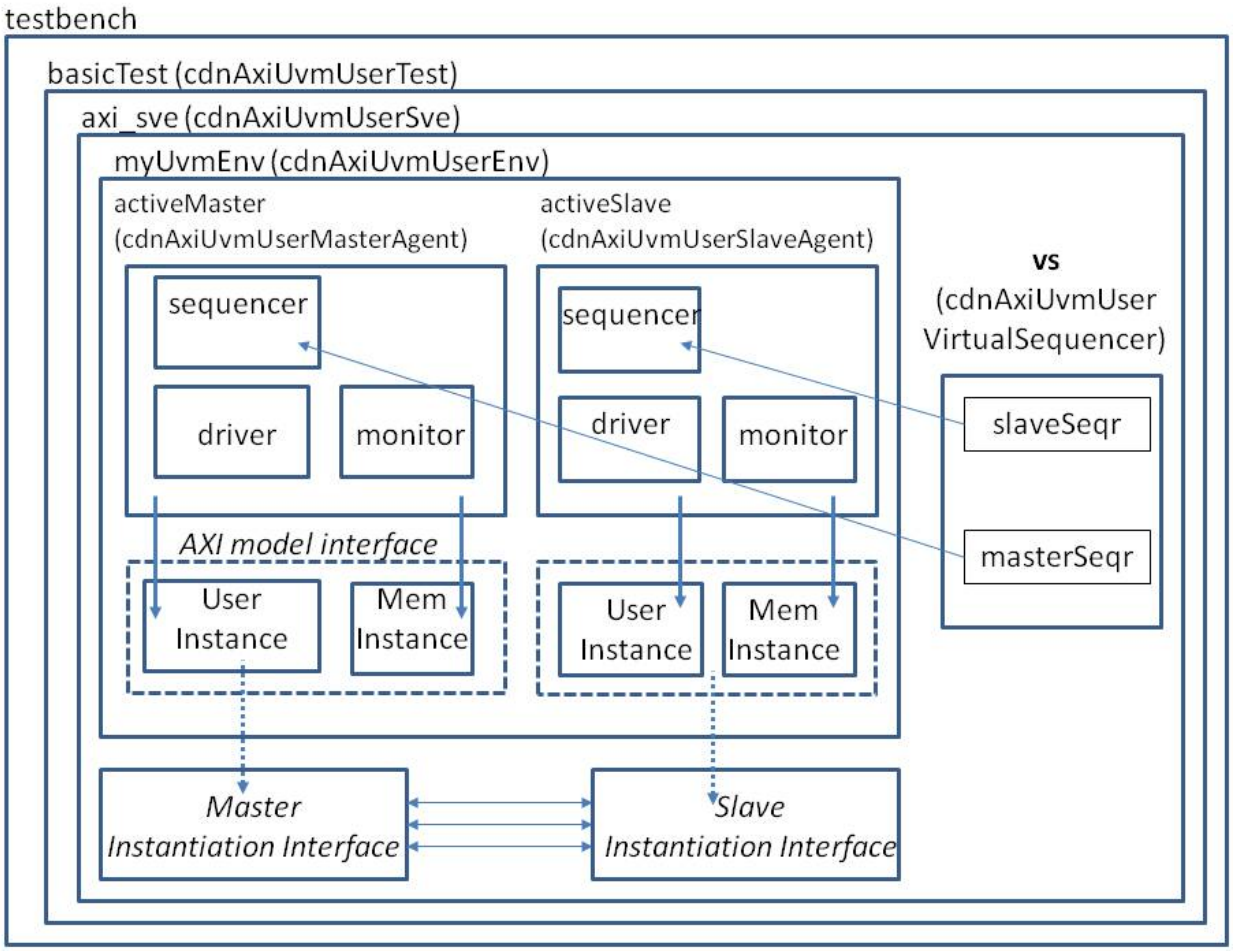
10.3.9. Example Testcase

You can access the testbench examples located at `<VIPCAT>/tools/denali/ddvapi/sv/uvm/cdn_axi/examples`.

Each example contains the verification environment with instantiations of three agents - active Master (mimicking DUT); passive Master (to shadow DUT); and an active Slave, SVE (System Verification Environment) that instantiates the environment and a virtual sequencer, and a test that instantiates the SVE.

The following diagram illustrates the CDN_AXI VIP UVM example architecture.

Figure 10.2. Example Testcase Architecture



The following table describes the testcase example files.

Table 10.6. Testcase Example Files

Filename	Description
cdnAxiUvmUserTb.sv	This file creates a testbench, including model and instantiation interface.
cdnAxiUvmUserTest.sv	This file includes the definition of a basic test class. This test: - Creates the CDN_AXI VIP SVE. - Sets the default sequence for the virtual sequencer. - Sets the log's verbosity levels.

Filename	Description
	- Sends several random transactions.
cdnAxiUvmUserSve.sv	This file includes the definition of the CDN_AXI VIP UVM SVE class. This code does the following: - Instantiates the CDN_AXI VIP UVM SVE. - Instantiates the virtual sequencer. - Overrides the types of all the UVM layer base components to the user-specific classes - Sets the agents hdlPath to the testbench module instantiations. - Connects the virtual sequencer pointers to the agent's sequencers. - Instantiates a coverage model by setting the relevant agent monitor coverageEnable bit to TRUE using `set_config_int()`.
cdnAxiUvmUserEnv.sv	This file includes the definition of the user-specific CDN_AXI environment. This code does the following: - Instantiates agents. In this example, activeMaster is considered as DUT. - Activates each agent's callbacks.
cdnAxiUvmUserMasterCfg.sv	This file includes the definition of the Master configuration object. It enables you to set some of the Master's configuration setting at the beginning of the run. It also enables the mapping of the Master agent's address segments.
cdnAxiUvmUserSlaveCfg.sv	This file includes the definition of the Slave configuration object. It enables you to set some of the Slave's configuration setting at the beginning of the run. It also enables the mapping of the Slave agent's address segments.
cdnAxiUvmUserAgent.sv	This file includes the definition of the user-specific CDN_AXI agent. In examples, this file imple-

Filename	Description
	ments the additional memory spaces to access the main memory space.
cdnAxiUvmUserMasterAgent.sv	This file includes the definition of the Master agent component. This code does the following: - Instantiates the master's configuration object (Defined in cdnAxiUvmUserMasterCfg.sv file) - Randomizes the configuration values according the user-defined constraints. - Sets the master agent's configuration by a set of register writes. - Sets the master address segments mapping.
cdnAxiUvmUserSlaveAgent.sv	This file includes the definition of the Slave agent component. This code does the following: - Instantiates the slave's configuration object (Defined in cdnAxiUvmUserSlaveCfg.sv file) - Randomizes the configuration values according the user-defined constraints. - Sets the slave agent's configuration by a set of register writes. - Sets the slave address segments mapping.
cdnAxiUvmUserMonitor.sv	This file includes the definition of the user-specific CDN_AXI monitor. This example: - Instantiates a coverage model. - Connects the coverage model analysis important to the monitor relevant analysis ports
cdnAxiUvmUserMasterDriver.sv	This file includes the definition of the user-specific CDN_AXI Master's driver.
cdnAxiUvmUserSlaveDriver.sv	This file includes the definition of the user-specific CDN_AXI Slave's driver.
cdnAxiUvmUserSequencer.sv	This file includes the definition of the user-specific CDN_AXI sequencer. The user sequencer in the example does not have any specific functionality.
cdnAxiUvmUserSeqLib.sv	This file includes an example of the user-specific sequence library.
cdnAxiUvmUserVirtualSeuquecer.sv	This file includes an example of user implementation of the virtual sequencer. The virtual sequencer in this example has pointers to the agent's sequencers.
cdnAxiUvmUserVirtualSeqLib.sv	This file includes an example of a user virtual sequence library.

Chapter 11. Frequently Asked Questions

11.1. Generic FAQs

11.1.1. How programmable is CDN_AXI VIP? Is there a list of the programmable fields available?

The models are extremely programmable. The CDN_AXI VIP provides a configuration file called SOMA that can be used to specify the device in terms of configuration and feature set. Additionally, the CDN_AXI VIP models allow dynamic changes or updates to the complete protocol-specific configuration space (if any) and CDN_AXI VIP specific configuration registers.

11.1.2. How can I change the pin names in CDN_AXI VIP model?

You can change pin names by using the PureView user interface. Regenerate the model instantiation interface. Do not manually edit the instantition code or the SOMA file. Both should be edited and saved using PureView.

11.1.3. What is the difference between .spc and .soma files?

The `.spc` and `.soma` files contain the same information, in a different format. The `.soma` file is in compressed XML format and the `.spc` file is in ASCII text format. Cadence recommends that you save the files from PureView in the XML syntax.

11.1.4. How can I disable the `timescale directive in SystemVerilog?

You can use the following macros to disable `timescale directive in the VIP interface:

Table 11.1. Macros to Disable `timescale Directive

Filename	Macro Name
denaliCdn_axi.sv	DENALI_SV_CDN_AXI_NO_TIMESCALE
denaliMem.sv file	DENALI_SV_MEM_NO_TIMESCALE
Interface files	DENALI_SV_NO_TIMESCALE

You can use the `DENALI_SV_NO_PKG` macro to disable interface files declaration as package.

11.1.5. How can I print a message in hexadecimal format?

You can print a message in a hexadecimal format by adding `set_config(print, radix, hex)` command to `SPECMAN_PRE_COMMANDS` as shown below:

```
setenv SPECMAN_PRE_COMMANDS "set_config(print, radix, hex);"
```

11.1.6. Is it ok to use double slashes (//) in paths for executing Specman commands?

When using the scripts: `cdn_vip_env.[c]sh`, `run_vcs_sv.sh`, `run_mti_sv.sh`, and `run_nc_sv.sh`, as well as when executing Specman commands directly, do not pass a path that contains double slashes because Specman will treat everything followed by the double slashes as a comment.

For example, the following command will fail:

```
sn_compile.sh -32 -shlib -exe -o uvc /path/to/some/top/e//file/ -enable_DAC
```

Error signatures may appear as:

```
1. *** Error: No match for file '/path/to/some/top/e.e' in command
(SPECMAN_PRE_COMMANDS)

2. *** Error: In preprocessing: Input is empty. Maybe no opening <'
and closing '> were found at line 1 in @...
```

11.1.7. What should the LD_LIBRARY_PATH be set to?

You must set the `LD_LIBRARY_PATH` to the following:

```
setenv LD_LIBRARY_PATH ${DENALI}/verilog:${DENALI}/lib:${DE-
NALI}/../lib:${DENALI }/../psui/lib:$LD_LIBRARY_PATH
```

> **Note**
>
> When you use Specman along with the CDN_AXI, Symbol resolutions happen through the `$INSTALL_ROOT/specman/libsn.so`. This path should be part of the `LD_LIBRARY_PATH` or should be passed through the `–pli` option.

11.1.8. Is there anything I need to do before compiling the C-libraries and system Verilog files in the run script?

Compile the `denaliCdn_axiSvIf.c` file into the DPI libraries before the compilation of corresponding SV files, which call functions like `denaliCdn_axiInit`.

11.1.9. Where does the DENALI variable point to?

`$DENALI` points to the `<package_location>/tools/denali` directory.

11.1.10. How do I change SPECMAN_HOME?

When you want to run a simulation with a different Specman version, SPECMAN_HOME can be changed in the following manner:

```
setenv SPECMAN_HOME <Package Installation Directory>/tools/specman
setenv PATH <Package Installation Directory>/tools/specman:$PATH
```

11.1.11. What should I do when the message "Couldn't create a legal Pkt transaction from input. Item is dropped." comes during simulation ?

This message comes when a randomization failure happens while generating a packet. To aid debugging the randomization failure in such cases, the following commands should be set in succession on the Specman command prompt before running the simulation:

- break on error
- break on gen error

When you run the simulation with these commands set, the debugger will show the exact contradicting constraints, and the constraints can be modified accordingly.

11.2. CDN_AXI Specific FAQs

11.2.1. How can I change address, ID and data signals width?

You can change them at time 0 of simulation by writing the desired value to the `DENALI_CDN_AXI_REG_ReadDataWidth`, `DENALI_CDN_AXI_REG_WriteDataWidth`, `DENALI_CDN_AXI_REG_AddrWidth, DENALI_CDN_AXI_REG_IdWidth`, register. For more more information see Section 5.3, “*Registers*”.

Chapter 12. Troubleshooting

12.1. Generic Troubleshooting

12.1.1. Specman license attempted a checkout

When using Specman installed under VIPCAT, you must set the environment variable `UVC_MSI_MODE` in your environment/run scripts/setup scripts. Not setting this variable will cause Specman to invoke a standard Specman license.

For example:

```
setenv UVC_MSI_MODE 1
```

You can find more details in the *VIP Catalog User Guide*, section *Using the UVC Virtual Machine*.

12.1.2. CDN_PSIF_ASSERT_0031

Error signature:

```
Generating the test with IntelliGen using seed 1...
                *** Error: CDN_PSIF_ASSERT_0031
Internal error or configuration problem at line 503 in encrypted module cdn_psif_e_utils_ext

No PS memories alocated. Check your setup and configuration
Failed condition: (FALSE)
```

Problem:

The stub file `<CDN_VIP_ROOT>/tools/psui/lib/[64bit/][vcs|mti]_psui.sv` should be the last SystemVerilog file passed to the vcs/vlog command. If the order has been modified, this error is raised.

Solution:

If the order of SV files cannot be changed, do the following to resolve the issue:

1. Remove "test;" from the SPECMAN_PRE_COMMANDS environment variable.
2. Do **one** of the following
   - Invoke "test" from the simulator by typing "sn test".
   - Add the following code into an initial block:

     ```
     initial
        begin
           $sn("test");
     end
     ```

     This code has to happen during time 0, before any register write (before the instantiation).

Note

The "test;" command must either be specified in SPECMAN_PRE_COMMANDS or invoked by the simulator. When modifying the default behavior, please make sure the test command is executed only once.

A missing test command will result the following error:

```
*** Error: Specman cannot start running: 'test' command not issued yet
```

12.1.3. Library Error During Simulation

When using VCS or MTI with VIPCAT, you might run into the following error

```
*** Error: Cannot open library
Failed to open `$CDN_VIP_ROOT/tools.lnx86/ucd/lib/libucdb.so' shared library
Operating System error:
<path to gcc>/gcc/*/lib/libstdc++.so.6: version `GLIBCXX_3.4.*' not found (required by $CDN_VIP_ROOT/
tools.lnx86/ucd/lib/libucdb.so)
```

Problem

Both VIPCAT and the simulator use libstdc++ which is loaded just once. If the simulator loads the libstdc++ before VIPCAT does, there will be no other load of the same library with the same major version. If the libucdb.so used for the simulation was compiled with an earlier gcc version than the one used to compile the VIPCAT libraries, this error is issued.

Workaround

As a first step, make sure your LD_LIBRARY_PATH includes the following path in the beginning: `$CDN_VIP_ROOT/tools/lib[64bit]`.

When executing `cdn_vip_env.[c]sh`, `run_vcs_sv.sh`, `run_mti_sv.sh` or `run_nc_sv.sh` with the `-setup` flag enabled, the `cdn_vip_setup_runme.sh/runme.sh` file created will include this step.

The above workaround is expected to resolve the issue, unless advanced environment settings were define to change/override LD_LIBRARY_PATH settings. In that case:

1. Check whether you have a wrapper script for VCS or MTI that changes the LD_LIBRARY_PATH.
2. Check `/etc/ld.so.conf` file/imports.
3. Check whether the `rpath` flag is set in your gcc configuration. `rpath` is a linker flag for searching dynamic libraries that have precedence over the `LD_LIBRARY_FLAG` variable.
4. Make sure the option `-noautoldlibpath` is not passed to MTI (since the default of the vsim command WILL change `LD_LIBRARY_FLAG`).

Which gcc version should you use if no standard location contains an advanced enough version? The safest thing to do is to:

1. Pick up gcc from VIPCAT (located under `$CDN_VIP_ROOT/tools.lnx86/systemc/gcc/bin[/64bit]`).

2. Set the environment variable NCROOT_OVERRIDE in your env to point to the vipcat root.

12.2. CDN_AXI Specific Troubleshooting

Table 12.1. Troubleshooting

Symptom	Reason	Solution

End of test summary says that the items were dropped in the test.
 	
OR
 	
You get the one of following messages in the log:
 	
`DENALI_CDN_AXI_`
 	
`WARNING_VR_AXI3190_`
 	
`WRITE_BURST_DOESNT_`
 	
`MATCH_THE_CACHE_`
 	
`STATE_AND_DISCARDED.`
 	
`DENALI_CDN_AXI_`
 	
`WARNING_VR_AXI3191_`
 	
`READ_BURST_DOESNT_`
 	
`MATCH_THE_CACHE_`
 	
`STATE_AND_DISCARDED.`
	The master tried to send a burst that is illegal or it did not match the current cache state.
	
	Note that for bursts sent to a shareable domain that are longer than one cache line, all relevant cache lines must be legal in order to send this burst.
		1. Search in the log file for the following string "Couldn't create a legal Burst transaction from input Item is dropped".
		
		2. See the reason, and fix constraints according to it. For example: `ERROR: size is set to TWO_WORDS but data width is 3`.