SYNOPSYS®

VC Verification IP
CHI
VMM User Guide

Version O-2018.12, December 2018

Copyright Notice and Proprietary Information

Right to Copy Documentation

The license agreement with Synopsys permits licensee to make copies of the documentation for its internal use only. Each copy shall include all copyrights, trademarks, service marks, and proprietary rights notices, if any. Licensee must

assign sequential numbers to all copies. These copies shall contain the following legend on the cover page:

> This document is duplicated with the permission of Synopsys, Inc., for the exclusive use of
>
> _____ and its employees. This is copy number_____."

Destination Control Statement

All technical data contained in this publication is subject to the export control laws of the United States of America. Disclosure to nationals of other countries contrary to United States law is prohibited. It is the reader's responsibility to determine the applicable regulations and to comply with them.

Disclaimer

SYNOPSYS, INC., AND ITS LICENSORS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Registered Trademarks (®)

Synopsys, AMPS, Cadabra, CATS, CRITIC, CSim, Design Compiler, DesignPower, DesignWare, EPIC, Formality, HSIM, HSPICE, iN-Phase, in-Sync, Leda, MAST, ModelTools, NanoSim, OpenVera, PathMill, Photolynx, Physical Compiler, PrimeTime, SiVL, SNUG, SolvNet, System Compiler, TetraMAX, VCS, Vera, and YIELDirector are registered trademarks of Synopsys, Inc.

Trademarks (™)

Synopsys and certain Synopsys product names are trademarks of Synopsys, as set forth at http://www.synopsys.com/Company/Pages/Trademarks.aspx.

All other product or company names may be trademarks of their respective owners.

Synopsys, Inc.

690 E. Middlefield Road

Mountain View, CA 94043

www.synopsys.com

Preface

About This Manual

This manual contains installation, setup, and usage material for SystemVerilog VMM users of the Synopsys Discovery CHI VIP, and is for design or verification engineers who want to verify CHI operation using an VMM testbench written in SystemVerilog. Readers are assumed to be familiar with CHI, Object Oriented Programming (OOP), SystemVerilog, and Verification Methodology Manual (VMM) techniques.

Manual Organization

The chapters of this databook are organized as follows:

- Chapter 1, "Introduction", introduces the Synopsys Discovery CHI VIP and its features.
- Chapter 2, "Installation and Setup", describes system requirements and provides instructions on how to install, configure, and begin using the Synopsys Discovery CHI VIP.
- Chapter 3, "General Concepts", introduces the CHI VIP within the VMM environment and describes the data objects and components that comprise the VIP.
- Chapter 4, "Verification Features", describes the verification features supported by CHI VIP such as, Verification Planner and Protocol Analyzer.
- Chapter 5, "Verification Topologies", describes the topologies to verify RN, SN and interconnect DUT.
- Chapter 6, "Using CHI Verification IP", shows how to install and run a getting started example.
- Chapter 7, "Troubleshooting", describes the debug port.
- Appendix A, "Reporting Problems", outlines the process for working through and reporting Synopsys Discovery CHI VIP issues.

Web Resources

- Documentation through SolvNet: https://solvnet.synopsys.com (Synopsys password required)
- Synopsys Common Licensing (SCL): http://www.synopsys.com/keys

Customer Support

To obtain support for your product, choose one of the following:

- Open a Case through SolvNet.
  - Go to https://onlinecase.synopsys.com/Support/OpenCase.aspx and provide the requested information, including:
    - Product: **Verification IP**
    - Sub Product: **amba_svt**
    - Tool Version: **O-2018.12**
    - Fill in the remaining fields according to your environment and your issue.
  - If applicable, provide the information noted in Appendix A, "Reporting Problems" on page 93.
- Send an e-mail message to support_center@synopsys.com.
  - Include the Product name, Sub Product name, and Tool Version (as noted above) in your e-mail so it can be routed correctly.
  - If applicable, provide the information noted in Appendix A, "Reporting Problems" on page 93.
- Telephone your local support center.
  - North America:

    Call 1-800-245-8005 from 7 AM to 5:30 PM Pacific time, Monday through Friday.
  - All other countries:

    http://www.synopsys.com/Support/GlobalSupportCenters

1 Introduction

This chapter gives a basic introduction, overview and features of the CHI VMM Verification IP. This chapter discusses the following topics:

1.1 Introduction

The Synopsys CHI Verification IP supports verification of SoC designs that include interfaces implementing the CHI Specification. This document describes the use of this VIP in testbenches that comply with the SystemVerilog Verification Methodology Manual (VMM). This approach leverages advanced verification technologies and tools that provide,

- Protocol functionality and abstraction
- Constrained random verification
- Functional coverage
- Rapid creation of complex tests
- Modular testbench architecture that provides maxmum reuse, scalability and modularity
- Proven verification approach and methodology
- Transaction-level models
- Self-checking tests
- Object oriented interface that allows OOP techniques

1.2 Prerequisites

- Familiarize with CHI, object oriented programming, SystemVerilog, and the current version of VMM

1.3 References

For more information on CHI Verification IP, refer to the following documents:

- Class Reference for Synopsys Discovery Verification IP for AMBA® CHI is available at:

  $DESIGNWARE_HOME/vip/svt/amba_svt/latest/doc/chi_svt_vmm_class_reference/html/index.html

1.4 Product Overview

The CHI VMM VIP is a suite of VMM-based verification components that are compatible for use with SystemVerilog-Compliant testbenches. The Synopsys CHI VIP suite simulates CHI transactions through active groups, as defined by the CHI specification.

The VIP provides a CHI System Group that contains the RN (Request Node) groups, SN (Slave Node) groups and CHI System Monitor. The RN and SN groups support all the functionality normally associated with active and passive VMM components.

The CHI System Env itself is contained within top level System VIP component AMBA System Env. User can either use AMBA System Env, CHI System Env, or standalone RN & SN groups in the testbench.

1.5 Language and Simulator Support

The current version of Synopsys CHI VIP suite is qualified with the following languages and simulators:

- Languages
  - SystemVerilog
- Simulators
  - VCS
- Methodology
  - VMM 1.2

1.6 Features Supported

- Protocol features
  - Support for RN-F, RN-I and SN-F node types
  - Support for CHI interconnect
  - Support for all CHI transaction types
  - CHI Memory within SN group
  - CHI protocol features as per CHI Version 5.0 spec. ARM-EPM-020383 5.0
- Verification Features
  - Analysis Port
  - Transaction Coverage
  - Scenario collection
  - Native FSDB dumping

- Methodology features
    - Active and passive mode of RN and SN groups.
    - Support for CHI System Group .
    - Port and system level checks.
    - Callbacks

1.7 Features Not Supported

Refer to section “Known Issues and Limitations” in the AMBA SVT VIP Release Notes:
$DESIGNWARE_HOME/vip/svt/amba_svt/latest/doc/amba_svt_release_notes.pdf

2
Installation and Setup

This section leads you through installing and setting up the Synopsys Discovery AMBA CHI VIP. When you complete this checklist, the provided example testbench will be operational and the Synopsys Discovery CHI VIP will be ready to use.

The checklist consists of the following major steps:

1. "Verifying the Hardware Requirements"
2. "Verifying Software Requirements"
3. "Preparing for Installation"
4. "Downloading and Installing"
5. "Setting Up a Testbench Design Directory"
6. "What's Next?"

If you encounter any problems with installing the Synopsys Discovery CHI VIP, see "Customer Support" on page 8.

2.1 Verifying the Hardware Requirements

400 MB available disk space for installation

- 1 GB available swap space

1 GB RAM (physical memory) recommended

- FTP anonymous access to ftp.synopsys.com (optional)

2.2 Verifying Software Requirements

The Synopsys Discovery CHI VIP is qualified for use with certain versions of platforms and simulators. This section lists software that the Synopsys Discovery CHI VIP requires.

2.2.1 Platform/OS and Simulator Software

- **Platform/OS and VCS**: You need versions of your platform/OS and simulator that have been qualified for use. To see which platform/OS and simulator versions are qualified for use with the CHI VIP, check the support matrix for "SVT-based" VIP in the following document:

Support Matrix for SVT-Based Synopsys CHI VIP is in:

Synopsys AMBA VIP Release Notes

2.2.2 Synopsys Common Licensing (SCL) Software

The SCL software provides the licensing function for the Synopsys Discovery CHI VIP. Acquiring

the SCL software is covered here in the installation instructions in "Licensing Information" on page 16.

2.2.3 Other Third Party Software

- **Adobe Acrobat**: Synopsys Discovery CHI VIP documents are available in Acrobat PDF files. You can get Adobe Acrobat Reader for free from http://www.adobe.com.
- **HTML browser:** Synopsys Discovery CHI VIP includes class reference documentation in HTML. The following browser/platform combinations are supported:
  - Microsoft Internet Explorer 6.0 or later (Windows)
  - Firefox 1.0 or later (Windows and Linux)
  - Netscape 7.x (Windows and Linux)

2.3 Preparing for Installation

1. Set DESIGNWARE_HOME to the absolute path where Synopsys CHI VIP is to be installed:

```
setenv DESIGNWARE_HOME absolute_path_to_designware_home
```

2.
   - DESIGNWARE_HOME/bin – The absolute path as described in the previous step.
   - LM_LICENSE_FILE – The absolute path to a file that contains the license keys for your third- party tools. Also, include the absolute path to the third party executable in your PATH variable.

     % setenv LM_LICENSE_FILE <my_license_file | port@host>
   - SNPSLMD_LICENSE_FILE – The absolute path to a file that contains the license keys for Vera and Synopsys Common Licensing software or the *port@host* reference to this file.

     % setenv SNPSLMD_LICENSE_FILE $LM_LICENSE_FILE

2.4 Downloading and Installing

Attention The Electronic Software Transfer (EST) system only displays products your site is entitled to download. If the product you are looking for is not available, contact est-ext@synopsys.com.

Follow the instructions below for downloading the software from Synopsys. You can download from the Download Center using either HTTPS or FTP, or with a command-line FTP session. If your Synopsys SolvNet password is unknown or forgotten, go to http://solvnet.synopsys.com.

Passive mode FTP is required. The passive command toggles between passive and active mode. If your FTP

utility does not support passive mode, use http. For additional information, refer to the following web page:

https://www.synopsys.com/apps/protected/support/EST-FTP_Accelerator_Help_Page.html

2.4.1 Downloading From the Electronic Software Transfer (EST) System (Download Center)

a. Point your web browser to "https://solvnet.synopsys.com/DownloadCenter". b.
Enter your Synopsys SolvNet Username and Password.

c. Click "Sign In" button.
d. Choose "Discovery Verification IP" from the list of available products under "My Product Releases"
e. Select the Product Version from the list of available versions. f.
Click the "Download Here" button for HTTPS download.

g. After reading the legal page, click on "Yes, I agree to the above terms".
h. Click the download button(s) next to the file name(s) of the file(s) you wish to download. i.
Follow browser prompts to select a destination location.

j. You may download multiple files simultaneously.

2.4.2 Downloading Using FTP with a Web Browser

a. Follow the above instructions through the product version selection step.
b. Click the "Download via FTP" link instead of the "Download Here" button. c.
Click the "Click Here To Download" button.

d. Select the file(s) that you want to download.
e. Follow browser prompts to select a destination location.

Note

If you are unable to download the Verification IP using above instructions, refer to "Customer Support" section to obtain support for download and installation.

2.5 Setting Up a Testbench Design Directory

A *design directory* is where the CHI VIP is set up for use in a testbench. A design directory is required for using VIP and, for this, the dw_vip_setup utility is provided.

The dw_vip_setup utility allows you to:

- Create the design directory (design_dir), which contains the transactors, support files (include files), and examples (if any)
- Add a specific version of CHI VIP from DESIGNWARE_HOME to a design directory

For a full description of dw_vip_setup, refer to "The dw_vip_setup Utility" on page 20.

To create a design directory and add a model so it can be used in a testbench, use the following command:

```
% $DESIGNWARE_HOME/bin/dw_vip_setup -path design_dir -a chi_system_group_svt -svtb
```

The models provided with CHI VIP include:

- chi_system_group_svt
- chi_rn_group_svt
- chi_sn_group_svt

When you install chi_system_group_svt, models chi_rn_group_svt and chi_sn_group_svt.

Refer to AXI VMM User Guide for information on AXI models.

2.6 What's Next?

The remainder of this chapter describes the details of the different steps you performed during installation and setup, and consists of the following sections:

- "Licensing Information" on page 16
- "Environment Variable and Path Settings" on page 17
- "Determining Your Model Version" on page 18
- "Integrating a Synopsys Discovery VIP into Your Testbench" on page 18

2.7 Licensing Information

The CHI VMM VIP uses the Synopsys Common Licensing (SCL) software to control its usage. You can find general SCL information at: http://www.synopsys.com/keys

The Synopsys VIP for AMBA uses a licensing mechanism that is enabled by the following license features:

- VIP-AMBA-CHI-SVT
- VIP-LIBRARY-SVT + DesignWare-Regression

The order in which licenses are checked out is as describes below:

- When CHI components are instantiated, VIP first looks for VIP-AMBA-CHI-SVT license. If this license is found, it is checked out.
- If VIP-AMBA-CHI-SVT license is not found, VIP looks for VIP-LIBRARY-SVT + DesignWare-Regression.

Note that VIP-AMBA-CHI-SVT license does not cover AXI3/AXI4/ACE/AHB/APB/ATB protocols. Refer to the respective user guides for details of licenses needed for these protocols.

Only one set of licenses as described above is consumed per simulation, no matter how many VIP instances are instantiated in the test bench.

The licensing key must reside in files that are indicated by specific environment variables. For information about setting these licensing environment variables, refer to "Environment Variable and Path Settings" on page 17.

2.7.1 Controlling License Usage

Using the DW_LICENSE_OVERRIDE environment variable, you can control which license is used as follows.

To use only DesignWare-Regression and VIP-LIBRARY-SVT licenses, set DW_LICENSE_OVERRIDE to:
DesignWare-Regression VIP-LIBRARY-SVT

To use only a VIP-AMBA-CHI-SVT license, set DW_LICENSE_OVERRIDE to:
VIP-AMBA-CHI-SVT

If DW_LICENSE_OVERRIDE is set to any value and the corresponding feature is not available, a license error message is issued.

2.7.1.1 License Polling

If you request a license and none are available, license polling allows your request to exist until a license becomes available instead of exiting immediately. To control license polling, you use the DW_WAIT_LICENSE environment variable as follows:

- To enable license polling, set the DW_WAIT_LICENSE environment variable to 1.
- To disable license polling, unset the DW_WAIT_LICENSE environment variable. By default, license polling is disabled.

2.7.1.2 Simulation License Suspension

All Synopsys Verification IP products support license suspension. Simulators that support license suspension allow a model to check in its license token while the simulator is suspended, then check the license token back out when the simulation is resumed.

Note This capability is simulator-specific; not all simulators support license check-in during suspension.

Environment Variable and Path Settings

llowing are environment variables and path settings required by the CHI VIP verification models:

e fo

DESIGNWARE_HOME – The absolute path to where the VIP is installed.

- SNPSLMD_LICENSE_FILE – The absolute path to a file that contains the license keys for Synopsys Common Licensing software or the *port@host* reference to this file.
- LM_LICENSE_FILE – The absolute path to a file that contains the license keys for your third-party tools. Also, include the absolute path to the third party executable in your PATH variable.

2.8.1 Simulator-Specific Settings

Your simulation environment and PATH variables must be set as required to support your simulator.

2.9 Determining Your Model Version

The following steps tell you how to check the version of the models you are using.

Verification IP products are released and versioned by the suite and not by individual model. The version number of a model indicates the suite version.

- To determine the versions of VIP models installed in your $DESIGNWARE_HOME tree, use the setup utility as follows:

```
% $DESIGNWARE_HOME/bin/dw_vip_setup -i home
```

- To determine the versions of VIP models in your design directory, use the setup utility as follows:

```
% $DESIGNWARE_HOME/bin/dw_vip_setup -p design_dir_path -i design
```

2.10 Integrating a Synopsys Discovery VIP into Your Testbench

After installing a Synopsys Discovery VIP, follow these procedures to set up VIP for use in testbenches:

- "Creating a Testbench Design Directory"
- "The dw_vip_setup Utility"

2.10.1 Creating a Testbench Design Directory

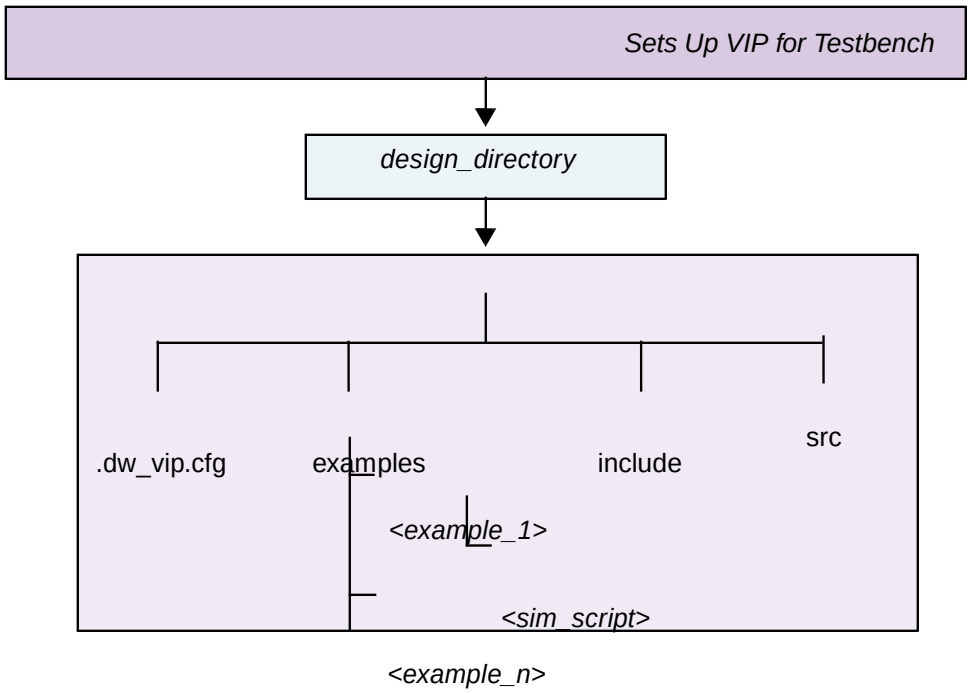
A *design directory* contains a version of VIP that is set up and ready for use in a testbench. You use the dw_vip_setup utility to create design directories. For the full description of dw_vip_setup, refer to "The dw_vip_setup Utility" on page 20.

If you move a design directory, the references in your testbenches to the include files will need to be revised to point to the new location. Also, any simulation scripts in the examples directory will need to be recreated.

A design directory gives you control over the version of Synopsys Discovery VIP in your testbench because it is isolated from the DESIGNWARE_HOME installation. When you want, you can use dw_vip_setup to update the VIP in your design directory. Figure 2-1 shows this process and the contents of a design directory.

Figure 2-1 Design Directory Created by dw_vip_setup

$DESIGNWARE_HOME/bin/ dw_vip_setup



A design directory contains:

examples Each VIP includes example testbenches. The dw_vip_setup utility adds them in this directory, along with a script for simulation. If an example testbench is specified on the command line, this directory contains all files required for model, suite, and system testbenches.

include Language-specific include files that contain critical information for VIP models. This directory is specified in simulator command lines.

src VIP-specific include files (not used by all VIPs). This directory may be specified in simulator command lines.

.dw_vip.cfg A database of all VIP models being used in the testbench. The dw_vip_setup program reads this file to rebuild or recreate a design setup.

Do not modify this file because dw_vip_setup depends on the original contents.

This section contains three examples that show common usage scenarios.

- "Adding or Updating Synopsys Discovery VIP Models In a Design Directory"
- "Removing Synopsys VIP Models from a Design Directory"
- "Reporting Information About DESIGNWARE_HOME or a Design Directory"

2.10.1.1 Adding or Updating Synopsys Discovery VIP Models In a Design Directory

CHI VIP models include:

- chi_system_group_svt
- chi_rn_group _svt
- chi_sn_group _svt

The following example adds a CHI VIP model to a design directory:

```
% $DESIGNWARE_HOME/bin/dw_vip_setup -a chi_system_group_svt -svtb
```

The following example updates a CHI VIP model in a design directory:

```
% $DESIGNWARE_HOME/bin/dw_vip_setup -p design_dir -u chi_system_group_svt -svtb
```

In these examples, the dw_vip_setup utility does the following:

1. Creates an include directory under the current directory and copies:
   - All files in the chi_system_group_svt model include directory
   - All include files in the CHI VIP suite
   - The latest library include files into the include directory
2. Creates the Synopsys Discovery CHI VIP suite libraries and VMM libraries.

2.10.1.2 Removing Synopsys VIP Models from a Design Directory

This example shows how to remove all listed models in the design directory at "/d/test2/daily" using the model list in the file "del_list" in the scratch directory under your home directory. The dw_vip_setup program command line is:

```
% $DESIGNWARE_HOME/bin/dw_vip_setup -p /d/test2/daily -r -m ~/scratch/del_list
```

The models in the *del_list* file are removed, but object files and include files are not.

2.10.1.3 Reporting Information About DESIGNWARE_HOME or a Design Directory

In these examples, the setup program sends output to STDOUT.

The following example lists the Synopsys Discovery VIP libraries, models, example testbenches, and license version in a DESIGNWARE_HOME installation:

```
% $DESIGNWARE_HOME/bin/dw_vip_setup -i home
```

The following example lists the VIP libraries, models, and license version in a testbench design directory:

```
% $DESIGNWARE_HOME/bin/dw_vip_setup -p design_dir -i design
```

2.10.2 The dw_vip_setup Utility

The dw_vip_setup utility

- Adds, removes, or updates CHI VIP models in a design directory
- Adds example testbenches to a design directory, the CHI VIP models they use (if necessary), and creates a script for simulating the testbench using any of the supported simulators
- Restores (cleans) example testbench files to their original state
- Reports information about your installation or design directory, including version information

2.10.2.1 Setting Environment Variables

Before running dw_vip_setup, the following environment variables must be set:

- DESIGNWARE_HOME – Points to where the Synopsys Discovery VIP is installed

2.10.2.2 The dw_vip_setup Command

You invoke dw_vip_setup from the command prompt. The dw_vip_setup program checks command line argument syntax and makes sure that the requested input files exist. The general form of the command is:

```
% dw_vip_setup [-p[ath] directory] switch (model [-v[ersion] latest | version_no] ) …
```

or

```
% dw_vip_setup [-p[ath] directory] switch -m[odel_list] filename
```

where

[**-p**[ath] *directory*] — The optional -path argument specifies the path to your design directory.

When omitted, dw_vip_setup uses the current working directory.

switch — The *switch* argument defines dw_vip_setup operation. Table 2-1 lists the switches and their applicable sub-switches.

Table 2-1 Setup Program Switch Descriptions

Switch	Description
-a[dd] (*model* [**-v**[ersion] *version*]) …	Adds the specified model or models to the specified design directory or current working directory. If you do not specify a version, the latest version is assumed. The model names are: • chi_system_group_svt • chi_rn_group_svt • chi_sn_group _svt The -add switch causes dw_vip_setup to build suite libraries from the same suite as the specified models, and to copy the other necessary files from $DESIGNWARE_HOME.
-r[emove] *model*	Removes ***all versions*** of the specified model or models from the design. The dw_vip_setup program does not attempt to remove any include files used solely by the specified model or models. The model names are: • chi_system_group _svt • chi_rn_group_svt • chi_sn_group_svt

Table 2-1 Setup Program Switch Descriptions (Continued)

Switch	Description
-u[pdate] (*model* [**-v**[ersion] *version*]) …	Updates to the specified model version for the specified model or models. The dw_vip_setup script updates to the latest models when you do not specify a version. The model names are: • chi_system_group_svt • chi_rn_group_svt • chi_sn_group_svt The -update switch causes dw_vip_setup to build suite libraries from the same suite as the specified models, and to copy the other necessary files from $DESIGNWARE_HOME.
-e[xample] {*scenario* | *model*/*scenario*} [**-v**[ersion] *version*]	The dw_vip_setup script configures a testbench example for a single model or a system testbench for a group of models. The program creates a simulator run program for all supported simulators. If you specify a *scenario* (or system) example testbench, the models needed for the testbench are included automatically and do not need to be specified in the command. **Note:** Use the -info switch to list all available system examples.
-ntb	Not supported.
-svtb	Use this switch to set up models and example testbenches for SystemVerilog VMM. The resulting design directory is streamlined and can only be used in SystemVerilog simulations.
-c[lean] {*scenario* | *model*/*scenario*}	Cleans the specified scenario/testbench in either the design directory (as specified by the *-path* switch) or the current working directory. This switch deletes *all files in the specified directory*, then restores all Synopsys created files to their original contents.
-i[nfo] *design* | *home*	When you specify the -info *design* switch, dw_vip_setup prints a list of all models and libraries installed in the specified design directory or current working directory, and their respective versions. Output from -info design can be used to create a model_list file. When you specify the -info *home* switch, dw_vip_setup prints a list of all models, libraries, and examples available in the currently-defined $DESIGNWARE_HOME installation, and their respective versions. The reports are printed to STDOUT.

Table 2-1 Setup Program Switch Descriptions (Continued)

Switch	Description
-h[elp]	Returns a list of valid dw_vip_setup switches and the correct syntax for each.
model	Synopsys Discovery CHI VIP models are: • chi_system_group_svt • chi_rn_group_svt • chi_sn_group_svt The *model* argument defines the model or models that dw_vip_setup acts upon. This argument is not needed with the -info or -help switches. All switches that require the *model* argument may also use a model list. You may specify a version for each listed *model*, using the -version option. If omitted, dw_vip_setup uses the latest version. The -update switch ignores *model* version information.
-m[odel_list] *filename*	The -model_list argument causes dw_vip_setup to use a user-specified file to define the list of models that the program acts on. The model_list, like the *model* argument, can contain model version information. Each line in the file contains: *model_name* [**-v** *version*] –or– # **Comments**
-b/ridge	Updates the specified design directory to reference the current DESIGNWARE_HOME installation. All product versions contained in the design directory must also exist in the current DESIGNWARE_HOME installation.

Note The dw_vip_setup program treats all lines beginning with “#” as comments.

For more information on installing and running SystemVerilog VMM example testbenches, refer to “SystemVerilog VMM Example Testbenches” and “Installing and Running the Examples” sections.

Change in use model from AMBA SVT N-2017.12 release onwards

AXI VIP COMPONENTS WITHIN CHI SYSTEM ENV

Note that the AXI VIP components within CHI System Env are not supported. However, the API svt_chi_system_configuration::create_sub_cfg() has the arguments to pass the number of AXI VIP components.

The VIP will issue a warning message when this API is supplied with non-zero values corresponding to number of AXI VIP components.

It is recommended to define the following compile time macro to remove the arguments corresponding to number of AXI VIP components: `SVT_AMBA_EXCLUDE_AXI_IN_CHI_SYS_ENV

Existing proto type of the API:

```
function void svt_chi_system_configuration::create_sub_cfgs(int num_chi_rn = 1, int num_chi_sn = 1, int num_chi_ic_rn = 0, int num_chi_ic_sn = 0, int num_axi_masters = 1, int num_axi_slaves = 1, int num_axi_ic_master_ports = 0, int num_axi_ic_slave_ports = 0);
```

With the compile macro `SVT_AMBA_EXCLUDE_AXI_IN_CHI_SYS_ENV defined, the proto type of the API:

```
function void svt_chi_system_configuration::create_sub_cfgs(int num_chi_rn = 1, int num_chi_sn = 1, int num_chi_ic_rn = 0, int num_chi_ic_sn = 0);
```

In a future release AMBA SVT N-2017.12-1, the arguments to specify number of AXI VIP components will be removed and the proto type of this API will always be the following:

```
function void svt_chi_system_configuration::create_sub_cfgs(int num_chi_rn = 1, int num_chi_sn = 1, int num_chi_ic_rn = 0, int num_chi_ic_sn = 0);
```

3
General Concepts

This chapter describes the usage of CHI VIP in an VMM environment, and it's user interface.

3.1 Introduction to VMM

VMM is an object-oriented approach. It provides a blueprint for building testbenches using constrained random verification. The resulting structure also supports directed testing.

This chapter describes the usage of CHI VIP in VMM environment, and its user interface. Refer to the Class

Reference HTML for a description of attributes and properties of the objects mentioned in this chapter.

This chapter assumes that you are familiar with SystemVerilog and VMM. For more information:

- For the IEEE SystemVerilog standard, see:
  - IEEE Standard for SystemVerilog—Unified Hardware Design, Specification, and Verification

    Language

3.2 CHI VIP in an VMM Environment

The following sections describe these components.

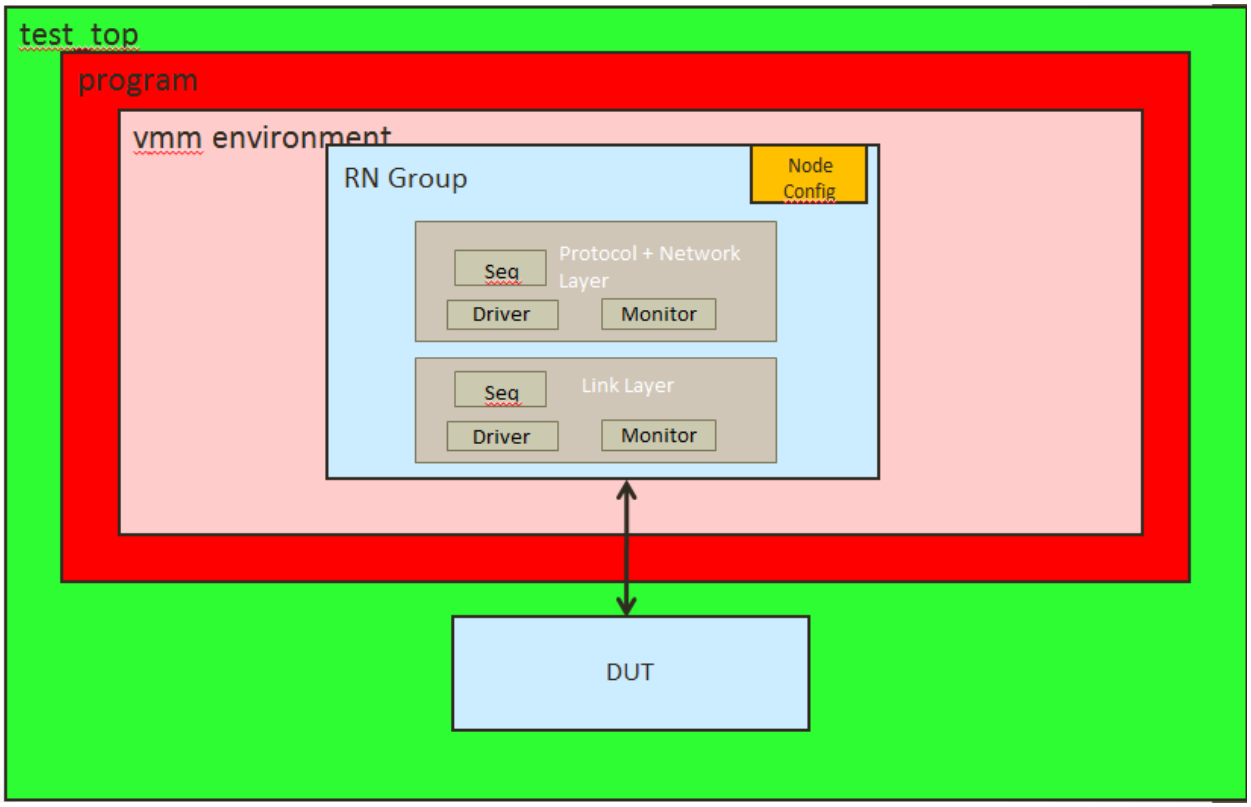
3.2.1 RN Group

The RN Group encapsulates RN Transaction Sequencer, RN Protocol Driver, RN Protocol Monitor, TX Flit Sequencers, RN Link Driver and RN Link Monitor The RN Group can be configured to operate in active mode and passive mode. You can provide CHI scenarios to the RN Generator.

The RN Group is configured using a node configuration, which is available in the system configuration. The node configuration should be provided to the RN Group in the build phase of the test.

The Protocol layer driver is connected to Link layer driver through flit sequencers. A flit sequencer exists for each Virtual Channel corresponding to a transmit path. Within the RN Group , the RN Protocol driver gets scenario from the RN Sequencer. The RN Protocol Driver then drives Flit objects to the RN Link driver through these sequencers. The RN Link driver drives the CHI transactions on the corresponding CHI Virtual Channel signals. After the CHI transaction on the bus is complete, the completed sequence item is provided to the analysis port of Protocol Monitor and Link Monitor for use by the testbench.

Figure 3-1 Usage with Standalone RN Group



3.2.2 SN Group

The SN Group encapsulates SN Transaction Sequencer, SN Protocol Driver, SN Protocol Monitor, TX Flit Sequencers, SN Link Driver and SN Link Monitor. The SN Group can be configured to operate in active mode and passive mode. You can provide CHI response scenarios to the SN Generator.

The SN Group is configured using node configuration, which is available in the system configuration. The node configuration should be provided to the SN Group in the build phase of the test or the testbench environment.

In the SN Group, the Link Monitor samples the CHI port signals. When a new transaction is detected, the Link Monitor provides a response request to Protocol Monitor. The Protocol Monitor in turn provides response request sequence item to the SN Transaction . The SN response scenario within the SN Transaction sequencer programs the appropriate SN response. The updated response sequence item is then provided by the SN Transaction Sequencer to the SN Protocol Driver. The SN Protocol driver is connected to Link layer driver through flit sequencers. A flit sequencer exists for each Virtual Channel corresponding to a transmit path. The SN Protocol Driver then drives Flit objects to the SN Link driver through these sequencers. The SN Link driver drives the CHI responses on the corresponding CHI Virtual Channel signals.

Note

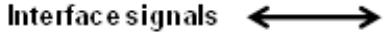
The SN Protocol driver expects the SN response sequence to,

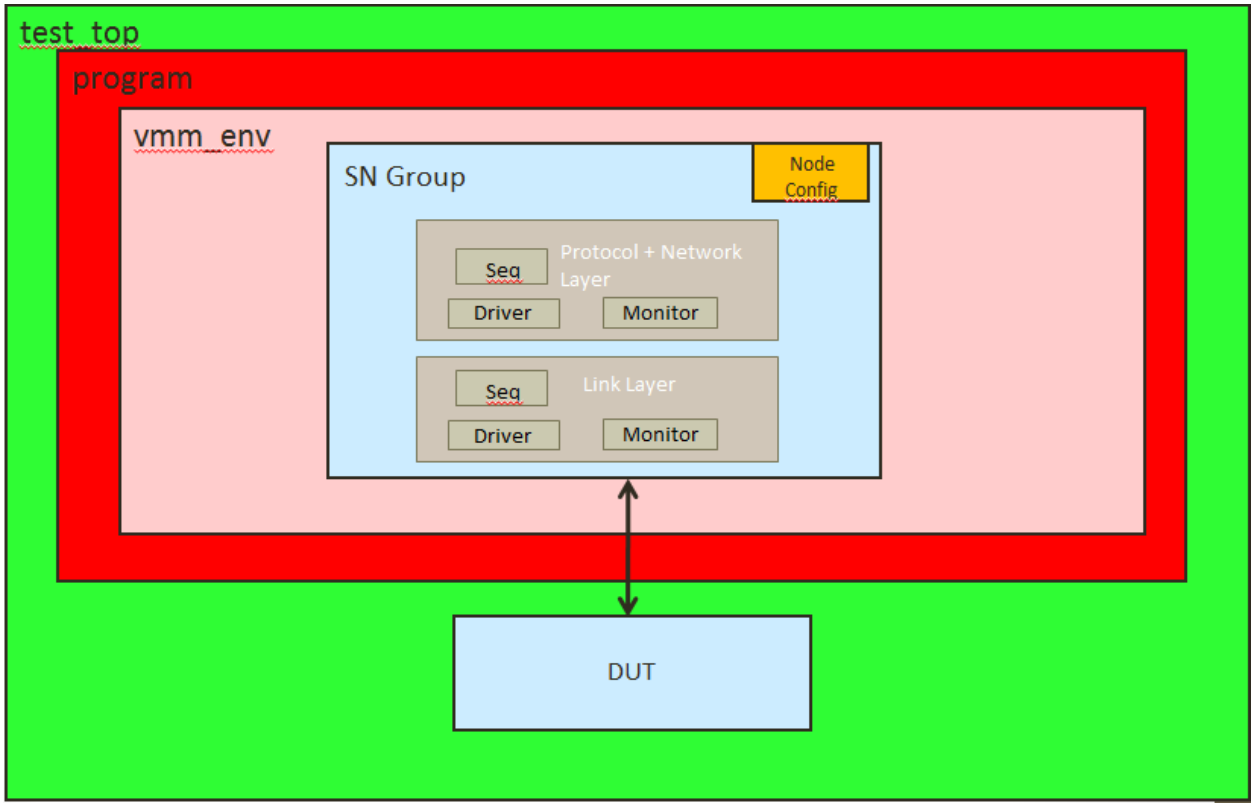
- Return same handle of the SN response object as provided to the sequencer by the SN Protocol monitor
- Return the SN response object in zero time, that is, without any delay after sequencer receives object from the SN Protocol monitor

If any of the above conditions is violated, the SN group issues a FATAL message.

After the CHI transaction on the bus is complete, the completed sequence item is provided to the analysis port of Protocol Monitor and Link Monitor for use by the testbench.

Figure 3-2 Usage with Standalone SN Group

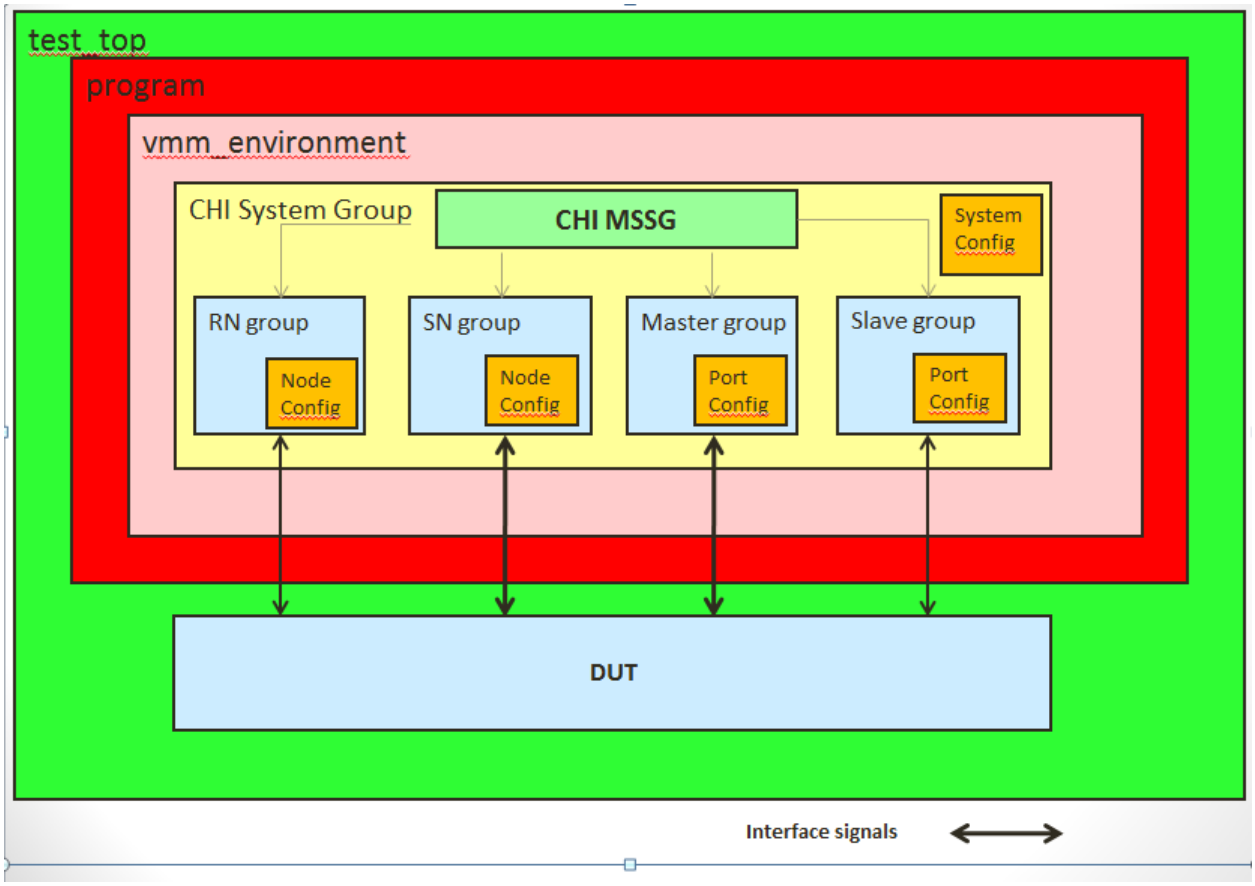




3.2.3 System Group

The CHI System group encapsulates the RN groups, SN Groups, System Monitor and the CHI system configuration. CHI System Group also contains a virtual sequencer that contains references to all the sequencers within the contained groups. The number of configured RN, SN, AXI Master, AXI Slave Groups is based on the system configuration provided by you. In the build phase, the System Group builds the RN, SN, AXI Master, AXI Slave groups. After the Groups are built, they are configured by System Group by using the CHI Node configuration and AXI Port configuration information available in the system configuration.

Figure 3-3 Usage with System Group


test_top
program
vmm_environment
CHI System Group
CHI MSSG
System Config
RN group
Node Config
SN group
Node Config
Master group
Port Config
Slave group
Port Config
DUT
Interface signals

3.2.4 Active and Passive Mode

Table 3-1 lists the behavior of RN and SN Groups in active and passive modes.

Table 3-1 Groups in Active and Passive Mode

Component behavior in active mode	Component behavior in passive mode
In active mode, RN and SN components generate transactions on the signal interface.	In passive mode, RN and SN components do not generate transactions on the signal interface. These components only sample the signal interface.
RN and SN components continue to perform passive functionality of coverage and protocol checking. You can enable/disable this functionality through configuration.	RN and SN components monitor the input and output signals, and perform passive functionality such as coverage and protocol checking. You can enable/disable this functionality through configuration options.
The Monitor components within the group perform protocol checks only on sampled signals. That is, it does not perform checks on the signals that are driven by the group. This is because when the group is driving an exception (exceptions are not supported in this release) the Monitor should not flag an error, since it knows that it is driving an exception. Exception means error injection.	The Monitor components within the group perform protocol checks on all signals. In passive mode, signals are considered as input signals.
The delay values reported in the CHI transaction provided by the RN and SN groups are the transaction values received from sequences, and not the sampled delay values.	The delay values reported in the CHI transaction provided by the RN and SN groups are the sampled delay values on the bus.

3.3 CHI VMM User Interface

The following sections give an overview of the user interface into the CHI VIP.

3.3.1 Configuration Objects

Configuration data objects convey the system level and node level testbench configuration. The configuration of groups is done in the build() phase of environment or the testcase. If the configuration needs to be changed later, it can be done through reconfigure() method of the RN Group, SN Group , AXI Master Group, AXI Slave Group or System Group.

The configuration can be of the following two types:

- Static configuration properties

  Static configuration parameters specify a configuration value which cannot be changed when the system is running. Examples of static configuration parameters are number of RNs and SNs, data bus width, and address width.

- Dynamic configuration properties

  Dynamic configuration parameters specify configuration value which can be changed at any time, regardless of whether the system is running or not. An example of a dynamic configuration parameter is a timeout value.

The configuration data objects contain built-in constraints, which come into effect when the configuration objects are randomized.

The CHI VIP defines following configuration classes:

- System configuration (svt_chi_system_configuration)

  The System configuration class contains configuration information which is applicable across the entire system. You can specify the system level configuration parameters through this class. You need to provide the system configuration to the system group from the environment or the testcase. The system configuration mainly specifies:

  - Number of RN and SN groups, and the number of AXI Master and Slave groups in the system group
  - Node configurations for RN and SN groups
  - Port configurations for AXI Master and Slave groups
  - Virtual top level CHI interface
  - Address map
  - Timeout values

- Node configuration (svt_chi_node_configuration)

  The Node configuration class contains configuration information which is applicable to individual CHI RN or SN groups in the system group. Some of the important information provided by node configuration class is:

  - Active/Passive mode of the RN/SN port group
  - Enable/disable protocol checks
  - Enable/disable port-level coverage
  - Interface type (RN_F, RN_I, RN_D, SN_F, SN_I)
  - Virtual interface for RN and SN nodes

The node configuration objects within the system configuration object are created in the constructor of the system configuration.

- Port configuration (svt_axi_port_configuration)

  The Port configuration class contains configuration information which is applicable to individual AXI master or slave groups in the system group. Some of the important information provided by port configuration class is:

  - Active/Passive mode of the master/slave port group
  - Enable/disable protocol checks
  - Enable/disable port-level coverage
  - Interface type (AXI3/AXI4/AXI4-Lite)
  - Port configuration contains the virtual interface for the port

For details on individual members of configuration classes, refer to the CHI VIP Class reference HTML documentation.

3.3.2 Transaction Objects

Transaction objects, which are extended from the vmm_sequence_item base class, define a unit of CHI protocol information that is passed across the bus. The attributes of transaction objects are public and are accessed directly for setting and getting values. Most transaction attributes can be randomized. The transaction object can represent the desired activity to be simulated on the bus, or the actual bus activity that was monitored.

CHI transaction data objects store data content and protocol execution information for CHI transactions in terms of timing details of the transactions.

These data objects extend from the vmm_sequence_item base class and implement all methods specified by

VMM for that class.

CHI transaction data objects are used to:

- Generate random stimulus
- Report observed transactions
- Generate random responses to transaction requests
- Collect functional coverage statistics

Class properties are public and accessed directly to set and read values. Transaction data objects support randomization and provide built-in constraints. Two set of constraints are provided: valid_ranges and reasonable constraints.

- valid_ranges constraints limit generated values to those acceptable to the drivers. These constraints ensure basic VIP operation and should never be disabled.
- reasonable_* constraints, which can be disabled individually or as a block, limit the simulation by:
  - Enforcing the protocol. These constraints are typically enabled unless errors are being injected into the simulation.
  - Setting simulation boundaries. Disabling these constraints may slow the simulation and introduce system memory issues.

The VIP supports extending transaction data classes for customizing randomization constraints. This allows you to disable some reasonable_* constraints and replace them with constraints appropriate to your system.

Individual reasonable_* constraints map to independent fields, each of which can be disabled. The class provides the reasonable_constraint_mode() method to enable or disable blocks of reasonable_* constraints.

CHI VIP defines following transaction classes:

- CHI RN transaction (svt_chi_rn_transaction)

  The RN transaction class extends from the CHI transaction class svt_chi_transaction. The RN transaction class contains the constraints for RN specific members in the base transaction class. At the end of each transaction, the RN group provides object of type svt_chi_rn_transaction from its analysis ports, in active and passive mode.

- CHI SN transaction (svt_chi_sn_transaction)

  The SN transaction class extends from the CHI transaction base class svt_chi_transaction. The SN transaction class contains the constraints for SN specific members in the base transaction class. At the end of each transaction, the SN group provides object of type svt_chi_sn_transaction from its analysis ports, in active and passive mode.

  The RN and SN transactions inherit a handle to configuration object of type svt_chi_node_configuration, which provides the configuration of the node on which this transaction would be applied. The node configuration is used during randomizing the transaction. The node configuration is available in the sequencer of the RN/SN group.

  You should initialize the node configuration handle in the transaction using the node configuration available in the sequencer of the RN/SN group. If the node configuration handle in the transaction is null at the time of randomization, the transaction will issue a fatal message.

- CHI Snoop transaction (svt_chi_snoop_transaction)

  This is the class for snoop transaction type. The svt_chi_snoop_transaction also inherits a handle to configuration object of type svt_chi_node_configuration, which provides the configuration of the port on which this transaction would be applied. The node configuration is used during randomizing the transaction.

- CHI Flit transaction (svt_chi_flit)

  This is the class for flit transaction type. This class contains attributes for flit like flit type and flit opcode. The svt_chi_flit also inherits a handle to configuration object of type svt_chi_node_configuration, which provides the configuration of the port on which this transaction would be applied. The node configuration is used during randomizing the transaction.

For details on individual members of transaction classes, refer to the CHI VIP Class reference HTML documentation.

3.3.3 Analysis Ports

The Protocol and Link monitors within the RN and SN Group provide an analysis port . At the end of the transaction, the RN and SN Groups respectively write the completed svt_chi_rn_transaction and svt_chi_sn_transaction object to the analysis port of Protocol monitor. On observation of a flit at the link layer, the RN and SN Groups write svt_chi_flit object to the analysis port of Link monitor. This holds true in active as well as passive mode of operation of the RN/SN group. You can use this analysis port for connecting to scoreboard, or any other purpose, where a transaction object for the completed transaction is required.

3.3.7 Overriding System Constants

The VIP uses include files to define system constants that, in some cases, you may override so the VIP matches your expectations. For example, you can override the maximum delay values. You can also adjust the default simulation footprint, like maximum address width.

The system constants for the VIP are specified (or referenced) in the following files (the first three files reside at $DESIGNWARE_HOME/vip/svt/amba_svt/latest/sverilog/include):

- svt_chi_defines.svi

  Top-level include file. It allows for the inclusion of the common define symbols and the port define symbols in a single file. Also, it contains a `include to read user overrides if the

  `SVT_CHI_INCLUDE_USER_DEFINES symbol is defined.

- svt_chi_common_defines.svi

  This file defines common constants used by the CHI VIP components. You can override only the

  "User Definable" constants, which are declared in "ifndef" statements, such as the following:

  ```
  `ifndef SVT_CHI_MAX_NUM_OUSTANDING_XACT
    `define SVT_CHI_MAX_NUM_OUSTANDING_XACT 256
  `endif
  ```

- svt_chi_port_defines.svi

  This file contains the constants that set the default maximum footprint of the environment. These values determine the wire bit widths in the 'wire frame'-- they do not (necessarily) define the actual bit widths used by the components, which is determined by the configuration classes. For example:

  ```
  `define SVT_CHI_MAX_REQ_FLIT_WIDTH 100
  `define SVT_CHI_MAX_RSP_FLIT_WIDTH 45
  `define SVT_CHI_MAX_SNP_FLIT_WIDTH 65
  ```

- svt_chi_user_defines.svi

  This file contains override values that you define. This file can reside anywhere-- specify its location on the simulator command line.

For example, to override the SVT_CHI_DAT_FLIT_MAX_DATA_WIDTH constant from the svt_chi_port_defines.svi file:

- Redefine the corresponding symbol in the svt_chi_user_defines.svi file. For example:

  ```
  `define SVT_CHI_DAT_FLIT_MAX_DATA_WIDTH 256
  ```

- In the simulator compile command,
  - Ensure that the directory containing svt_chi_user_defines.svi is provided to the simulator

- ✦ Provide SVT_CHI_INCLUDE_USER_DEFINES on the simulator command line as follows:
  +define+SVT_CHI_INCLUDE_USER_DEFINES

Note the following restrictions when overriding the default maximum footprint:

- ❖ Do not use a value of 0 for a MAX_*_WDTH value. The value must be >= 1
- ❖ The maximum footprint set at compile time must work for the full design. If you are using multiple instances of CHI VIP, only one maximum footprint can be set and must therefore satisfy the largest requirement.

3.3.8 Format Specification for Data, Byte Enable fields

3.3.8.1 Specifying Data, Byte enable in WYSIWYG format

- The svt_chi_node_configuration::wysiwyg_enable is set to 1 (default value)
- Data:
  - Number of bytes should be equal to full cache line size of 64 byte irrespective of data size of the transaction
- Byte enable (applicable only for Write transactions):
  - Number of bits should correspond to full cache line size of 64 byte irrespective of data size of the transaction. That is, should be always 64 bits

3.3.8.2 Specifying Data, Byte enable in Right aligned format

- The svt_chi_node_configuration::wysiwyg_enable is set to 0
- Data:
  - number of bytes should be equal to data size of the transaction
  - in case of unaligned address, irrespective of memory type, the data provided should be from: Previous data size boundary, up to: Next data size boundary
- Byte enable (applicable only for Write transactions):
  - number of bits should be equal to data size of the transaction
  - in case of unaligned address, irrespective of memory type, the data provided should be from: Previous data size boundary, up to: Next data size boundary

3.3.8.3 Examples

Consider the following for the below examples:

- The data width of the DATFLIT is 16 bytes.
- The data size of the transaction is 32 bytes.
- For unaligned address example, that is, address that is not aligned to data size, consider that the address is ‘h3A.
- For aligned address example, that is, address that is aligned to data size, consider that the address is ‘h20.
- Also assume that the 32 byte data from address ‘h20 to ‘h3F is: {128’bdeadbeef_feedbeef_deadbeef_feedbeef, 128’bdeadbeef_feedbeef_deadbeef_feedbeef}

3.3.8.3.1 Normal memory read transactions

Address aligned to data size

Data provided should correspond to 32 bytes (256 bits) starting from ‘h20 to ‘h3F.

So: svt_chi_transaction::data = {data corresponding to ‘h3F, …. , data corresponding to ‘h20}

The data in this example will be: data = {128’bdeadbeef_feedbeef_deadbeef_feedbeef,
128’bdeadbeef_feedbeef_deadbeef_feedbeef}

Same applies to svt_chi_snoop_transaction::data

Address unaligned to data size

Data provided should correspond to 32 bytes (256 bits) starting from ‘h20 to ‘h3F.

So: svt_chi_transaction::data = {data corresponding to ‘h3F, …. , data corresponding to ‘h20}

The data in this example will be: data = {128’hdeadbeef_feedbeef_deadbeef_feedbeef,
128’hdeadbeef_feedbeef_deadbeef_feedbeef}

Same applies to svt_chi_snoop_transaction::data

3.3.8.3.2 Normal memory write transactions

Address aligned to data size

Data, byte enable provided should correspond to 32 bytes starting from ‘h20 to ‘h3F.

So: svt_chi_transaction::data = {data corresponding to ‘h3F, …. , data corresponding to ‘h20}
svt_chi_transaction::byte_enable = {byte_enable corresponding to ‘h3F, …, byte_enable corresponding to ‘h20}

The data in this example will be: data = {128’bdeadbeef_feedbeef_deadbeef_feedbeef,
128’bdeadbeef_feedbeef_deadbeef_feedbeef}

The byte enable in the example specifically should be: byte_enable = {16’hFC00,16’h00FF}

Same applies to svt_chi_snoop_transaction::data, svt_chi_snoop_transaction::byte_enable

Address unaligned to data size

Data, byte enable provided should correspond to 32 bytes starting from ‘h20 to ‘h3F.

So: svt_chi_transaction::data = {data corresponding to ‘h3F, …. , data corresponding to ‘h20}
svt_chi_transaction::byte_enable = {byte_enable corresponding to ‘h3F, …, byte_enable corresponding to ‘h20}

The data in this example will be: data = {128’bdeadbeef_feedbeef_deadbeef_feedbeef,
128’bdeadbeef_feedbeef_deadbeef_feedbeef}

The byte enable in the example specifically should be: byte_enable = {16’hFC00,16’h00FF}

3.3.8.3.3 Device memory read transactions

Address aligned to data size

In this case, the valid data bytes would be similar to the normal memory case as described in Address aligned to data size.

Address unaligned to data size

The svt_chi_transaction::data = {data corresponding to ‘h3F, …. , data corresponding to ‘h3A, data observed/provided on invalid bytes corresponding from ‘h20 to ‘h39}

Note that still it is required to provide the data for all the 32 bytes in this case, including the invalid bytes from ‘h20 to ‘h39.

The data in this example will be: data = {128’bdeadbeef_feedxxxx_xxxxxxxx_xxxxxxxx, 128’bxxxxxxxx_xxxxxxxx xxxxxxxx_xxxxxxxx}

Note that the ‘x’ above indicates the data that correspond to invalid bytes.

The same will be the case even with svt_chi_snoop_transaction::data

3.3.8.3.4 Device memory write transactions

Address aligned to data size

In this case, the valid data bytes, byte enables would be similar to the normal memory case as described in Address aligned to data size.

Address unaligned to data size

The svt_chi_transaction::data = {data corresponding to ‘h3F, …. , data corresponding to ‘h3A, data observed/provided on invalid bytes corresponding from ‘h20 to ‘h39}

The svt_chi_transaction::byte_enable = {byte_enable corresponding to ‘h3F, …. , byte_enable corresponding to ‘h3A, byte_enable observed/provided on invalid bytes corresponding from ‘h20 to ‘h39}

Note that still it is required to provide the data, byte enable for all the 32 bytes in this case, including the invalid bytes from ‘h20 to ‘h39.

The data in this example will be: data = {128’bdeadbeef_feedxxxx_xxxxxxxx_xxxxxxxx, 128’bxxxxxxxx_xxxxxxxx xxxxxxxx_xxxxxxxx}

Note that the ‘x’ above indicates the data that correspond to invalid bytes.

The byte enable in the example specifically should be: byte_enable = {16’hFC00,16’h0000}

The same will be the case even with
svt_chi_snoop_transaction::data,
svt_chi_snoop_transaction::byte_enable

3.3.11 Delays

To know the various delay attributes supported by CHI SVT VIP, refer to the attributes under following groups in HTML class reference documentation.

- svt_chi_node_configuration – chi_config_delays
- svt_chi_flit – chi_flit_delays

3.4 Reset Functionality

Currently, reset functionality is only supported at power on. Dynamic reset during simulation is not yet supported.

PROGRAMMING CHI SYSTEM ADDRESS RANGES AND RELATED SETTINGS

PROGRAMMING THE HN RELATED INFORMATION

Following information is essential for the CHI VIP components to understand the CHI Interconnect configuration and HN address ranges. These details are applicable irrespective of whether interconnect VIP is enabled or not (svt_chi_system_configuration::use_interconenct).

Refer to CHI VIP HTML class reference for the details on the attributes and APIs mentioned below.

- Number of HNs:
  - Program using the attribute svt_chi_system_configuration::num_hn
- The term 'HN index' corresponds to the value in the range [0:svt_chi_system_configuration::num_hn-1]
- HN index to node ID mapping for each of the HNs:
  - Program using the API svt_chi_system_configuration::set_hn_node_id(int node_id[])
- HN index to HN type mapping for each of the HNs:
  - Program using the API svt_chi_system_configuration::set_hn_interface_type(svt_chi_address_configuration::hn_interface_type_enum interface_type[])
- SN index to HN index mapping:
  - Program using the API svt_chi_system_configuration:: set_sn_to_hn_map(int sn_idx, int hn_idx[]);
  - When 'Three SN-F striping' feature is enabled, this API is not needed to program SN-F index to HN-F index programming; however, still this API is needed to program SN index to HN-I index programming. Refer to the section on 'Three SN-F striping' for more details.
- MN node ID programming:
  - Program using the attribute svt_chi_system_configuration::misc_node_id

NOTE: The CHI VIP expects one to one correspondence between number of HNs(through svt_chi_system_configuration::num_hn) and HN interface types programmed(through svt_chi_system_configuration::set_hn_interface_type()). It is required to program the HN interface types using svt_chi_system_configuration::set_hn_interface_type() for all the HNs present in the system(svt_chi_system_configuration::num_hn) appropriately. **Otherwise the CHI system configuration will be considered as invalid and the simulation fails.** Note that this is needed irrespective of whether VIP interconnect is enabled or not (svt_chi_system_configuration::use_interconnect).

NOTE: The order of programming HN nodes (for node ID, HN interface type) should be such that:

- First, all HN-F nodes are programmed
- Then, followed by all HN-I nodes are programmed

Refer to programming examples in the user guide and CHI UVM intermediate example for the same.
Note that from next release, not following this programming order will result in simulation to fail with ERRORs indicating invalid CHI system configuration (svt_chi_system_configuration::do_is_valid())

PROGRAMMING MN ADDRESS RANGES

Following API needs to be used to program MN address ranges. Refer to HMTL class reference for more details.

svt_chi_system_configuration::set_mn_addr_range(int mn_idx, bit [`SVT_CHI_MAX_ADDR_WIDTH-1:0] start_addr, bit [`SVT_CHI_MAX_ADDR_WIDTH-1:0] end_addr)

Typically, MN shares the same node ID as that of HN-I;. So, the HN index that corresponds to HN-I is the applicable for MN as well. In such case, the argument 'mn_idx' corresponds to HN index of HN-I. It's possible to program multiple ranges for a given 'mn_idx' by passing different set of start_addr and end_addr.

Typically, the MN address range corresponds to Interconnect's register address space.

EXAMPLE

Consider that hn_index 8 corresponds to MN, and it's start address and end address are defined through `TB_START_ADDR_MN, `TB_END_ADDR_MN respectively.

The usage is as follows: Here <chi_sys_cfg> refers to the object of type svt_chi_system_configuration.

```
  /** Program the address ranges for MN index. Typically,
   *  this is subset of HN-I address ranges
   */
<chi_sys_cfg>.set_mn_addr_range(8, `TB_START_ADDR_MN, `TB_END_ADDR_MN);
```

PROGRAMMING THE HN ADDRESS RANGES

Following approaches are permitted for programming HN address ranges.

- Using APIs
- Using factory override for svt_chi_system_configuration

PROGRAMMING USING APIS

Following API needs to be used to program HN address ranges. Refer to HTML class reference for more details.

svt_chi_system_configuration:: set_hn_addr_range(int hn_idx, bit [`SVT_CHI_MAX_ADDR_WIDTH-1:0] start_addr, bit [`SVT_CHI_MAX_ADDR_WIDTH-1:0] end_addr)

PROGRAMMING HN-I ADDRESS RANGES

Use the svt_chi_system_configuration:: set_hn_addr_range() with 'hn_idx' that corresponds to the HN-I. It's possible to program multiple ranges for a given 'hn_idx' by passing different set of start_addr and end_addr.

EXAMPLE

Consider that hn_index 8 corresponds to HN-I, and it's start address and end address are defined through `TB_START_ADDR_HN_I, `TB_END_ADDR_HN_I respectively.

The usage is as follows: Here <chi_sys_cfg> refers to the object of type svt_chi_system_configuration.

```
/** Program the address ranges for HN index that corresponds to HN-I */
<chi_sys_cfg>.set_hn_addr_range(8, `TB_START_ADDR_HN_I, `TB_END_ADDR_HN_I);
```

PROGRAMMING HN-F ADDRESS RANGES

DEFAULT BEHAVIOR

If user doesn't program the HN-F address ranges using svt_chi_system_configuration::set_hn_addr_range(), the below details are applicable.

By default, the VIP doesn't require the user to program HN-F address ranges. In this case, VIP relies on the HN interface type information programmed using svt_chi_system_configuration::set_hn_interface_type(). So, it is important to program the HN interface type details appropriately.

Using this, VIP determines the number of HN-Fs present in the system. The HN index 'hn_f_num' corresponding to a given address 'addr' is determined based on number of HN-Fs, using the following XOR function:

If the number of HN-Fs are 2:

hn_f_num[0] = ^addr[43] … ^addr[7] ^ addr[6]

If the number of HN-Fs are 4:

hn_f_num[1:0] = addr[43:42] ^ .... addr[9:8] ^ addr[7:6]

If the number of HN-Fs are 8:

hn_f_num[2:0] = {1'b0,addr[43:42]} ^ addr[41:39] ^ addr[38:36] ^ .... ^ addr[11:9] ^ addr[8:6]

Note:

The CHI VIP expects one to one correspondence between number of HNs(through svt_chi_system_configuration::num_hn) and HN interface types programmed(through svt_chi_system_configuration::set_hn_interface_type()). It is required to program the HN interface types using svt_chi_system_configuration::set_hn_interface_type() for all the HNs present in the system(svt_chi_system_configuration::num_hn) appropriately. **Otherwise the CHI system configuration will be considered as invalid and simulation fails.** Note that this is needed irrespective of whether VIP interconnect is enabled or not (svt_chi_system_configuration::use_interconnect setting).

EXPLICITLY PROGRAMMING HN-F ADDRESS RANGES

The API svt_chi_system_configuration:: set_hn_addr_range() can be used to program the address ranges corresponding to different 'HN-Fs'. The 'hn_idx' that corresponds to a given HN-F along with the start_addr and end_addr needs to be passed to this API. It's possible to program multiple ranges for a given 'hn_idx' by passing different set of start_addr and end_addr.

If HN-F address ranges are programmed explicitly using this mechanism, the default behavior described for address to HN-F mapping will not be enabled.

EXAMPLE

Consider that hn_index 3 corresponds to HN-F, and it's start address and end address are defined through `TB_START_ADDR_HN_F_IDX_3, `TB_END_ADDR_HN_F_IDX_3 respectively.

The usage is as follows: Here <chi_sys_cfg> refers to the object of type svt_chi_system_configuration.

```
/** Program the address ranges for HN index that corresponds to HN-F index 3 */
<chi_sys_cfg>.set_hn_addr_range(3, `TB_START_ADDR_HN_F_IDX_3, `TB_END_ADDR_HN_F_IDX_3);
```

ADDRESS TO HN MAPPING MECHANISM USED BY THE VIP

Following is the sequence the VIP uses to determine the address to HN mapping:

- If HN-I address ranges are programmed using APIs:
  - Return the HN-I that is mapped to the address.
- If address is not mapped to any HN-I, and HN-F ranges are explicitly programmed using APIs:
  - Return the HN-F that is mapped to the address
- If address is not mapped to any HN-I, and HN-F ranges are not explicitly programmed using the APIs:
  - Return the HN-F that is mapped to the address based on number of HN-Fs and the associated XOR function

USING FACTORY OVERRIDE FOR SVT_CHI_SYSTEM_CONFIGURATION

In this approach, any user programming of the HN-I/HN-F address ranges using svt_chi_system_configuration::set_hn_addr_range() are not needed. If they are programmed, those settings will be ignored.

User can extend svt_chi_system_configuration and override the implementation of the API get_hn_node_id_for_addr(bit [`SVT_CHI_MAX_ADDR_WIDTH-1:0] addr) in the extended class.

EXAMPLE TO PROGRAM HN-I ADDRESS RANGES USING FACTORY OVERRIDE

The base test needs to have the factory override in place using the methodology specific mechanism.

The svt_chi_system_configuration needs to be overridden with cust_svt_chi_system_configuration.

```
/**

 * Abstract:

 * Class cust_svt_chi_test_suite_system_configuration is basically used to
encapsulate all

 * the configuration information specific to the test suite. It extends chi
test suite system

 * configuration.

 */



`ifndef GUARD_CUST_SVT_CHI_TEST_SUITE_SYSTEM_CONFIGURATION_SV

`define GUARD_CUST_SVT_CHI_TEST_SUITE_SYSTEM_CONFIGURATION_SV

`include "svt_chi_user_defines.svi"



class cust_svt_chi_test_suite_system_configuration extends
svt_chi_test_suite_system_configuration;



----
```

```
 function int get_hn_node_id_for_addr(bit [`SVT_CHI_MAX_ADDR_WIDTH-1:0]
addr);

    int hnf_node_id;

    bit[2:0] new_hnf_id;

    bit [`SVT_CHI_MAX_ADDR_WIDTH-1:0] addr_hn_id_mask;

    // TBD:

    // Edit this line for addresses that are routed to HN-I index 8.

    if ((addr >= 44'h000_0000_0000) && (addr <= 44'h000_3fff_ffff)) begin

      get_hn_node_id_for_addr = 16; //HN-I node ID

    end

    // Default implementation retrieves HN-F based on address

    else begin

      hnf_node_id = chi_addr_cfg.get_hn_node_id_for_addr(addr);

      get_hn_node_id_for_addr = hnf_node_id;

    end

      endfunction

----

endclass

`endif //GUARD_CUST_SVT_CHI_TEST_SUITE_SYSTEM_CONFIGURATION_SV
```

EXAMPLE TO PROGRAM HN-I ADDRESS RANGES AND 2 HN-F BASED XOR FUNCTION USING FACTORY MECHANISM

The base test needs to have the factory override in place using the methodology specific mechanism.

The svt_chi_system_configuration needs to be overridden with cust_svt_chi_system_configuration.

```
/**

 * Abstract:

 * Class cust_svt_chi_system_configuration is basically used to encapsulate
all

 * the configuration information specific to the CHI system. It extends chi
system

 * configuration.

 * The method get_hn_node_id_for_addr() is overridden to implement the

 * address map with 4 HN-Fs using XOR function.
```

```
 */


`ifndef GUARD_CUST_SVT_CHI_SYSTEM_CONFIGURATION_SV
  `define GUARD_CUST_SVT_CHI_SYSTEM_CONFIGURATION_SV
  `include "svt_chi_user_defines.svi"


class cust_svt_chi_system_configuration extends svt_chi_system_configuration;


----


  /**
   * Returns the node_id of the HN configured for this address.
   * @param addr Address to be used in the lookup
   */
  function int get_hn_node_id_for_addr(bit [`SVT_CHI_MAX_ADDR_WIDTH-1:0]
addr);


    // TBD: Edit this line for addresses that are mapped to HN-I.
    if ((addr >= 44'h000_0000_0000) && (addr <= 44'h000_3fff_ffff)) begin
      get_hn_node_id_for_addr = 0; //Node ID of HN-I
    end
    // Address to HN-F node ID mapping is esablished as follows
    // with 4 HN-Fs.
    else begin
      // Stores to which HN-F the address belongs to, in the range [0:3]
      bit [1:0] hn_f_num;
      // Variable to aid in computing the hash function
      bit [1:0] _partial_addr;


      _partial_addr = addr[7:6];


      // The HN-F address map when number of HN-Fs are 4:
      // hn_f_num[1:0] = addr[43:42] ^ .... addr[9:8] ^ addr[7:6]
```

```
      for (int lsb = 8; lsb <= 42; lsb+=2) begin
        _partial_addr = (_partial_addr[1:0] ^ addr[lsb+:2]);
      end

      // The result of above hash function is the HN-F to which
      // the address belongs to.
      hn_f_num = _partial_addr;

      // Retrieve Node ID corresponding to the hn_f_num
      get_hn_node_id_for_addr = get_hn_node_id(hn_f_num);
    end
  endfunction
----
endclass
`endif //GUARD_CUST_SVT_CHI_SYSTEM_CONFIGURATION_SV
```

SN INDEX TO HN INDEX MAPPING

The relation between SN index to HN index needs to be programmed using the API svt_chi_system_configuration:: set_sn_to_hn_map(int sn_idx, int hn_idx[])

'sn_idx' corresponds to the SN index within the range [0:svt_chi_system_configuration::num_sn-1].

Values of the array 'hn_idx[]' correspond to HN indices within the range [0:svt_chi_system_configuration::num_hn-1].

Note that when 'Three SN-F striping' feature is enabled, this API needs to be used only to program SN to HN-I index mapping. SN-F to HN-F index mapping is not required to be programmed. Refer to the section on 'Three SN-F striping' for more details.

EXAMPLE

Consider the example that there are two SNs(SN indices in range[0:1]) and 9 HNs (HN indices in range [0:8]).

SN index 0 maps to HN indices {0,1,2,3,4,5,6,7}.

SN index 1 maps to HN index 8.

The usage is as follows: Here <chi_sys_cfg> refers to the object of type svt_chi_system_configuration.

```
/** Map SN to HNs: SN with index 0 maps to HN indices {0,1,2,3,4,5,6,7}).
 * SN with index 1 maps to HN index 8
*/

   <chi_sys_cfg>.set_sn_to_hn_map(0,{0,1,2,3,4,5,6,7});
```

```
<chi_sys_cfg>.set_sn_to_hn_map(1,{8});
```

RECOMMENDED PROGRAMMING EXAMPLES

This below code snippets provide the example to:

- Program HN related information
- Program MN, HN-I address ranges
- Rely on VIP default implementation for address to HN-F map based on number of HN-Fs and associated XOR function

Here <chi_sys_cfg> refers to the object of type svt_chi_system_configuration.

```
/**  Local variable used to program node IDs corresponding to HN indices */
int hn_node_id[];
/**  Local variable used to program HN interface type corresponding to HN
indices*/
svt_chi_address_configuration::hn_interface_type_enum hn_interface_type[];


/** Program total number of HNs present in the CHI system*/
<chi_sys_cfg>.num_hn = 9;


   /** Program the node IDs for each of the HN indices */
   hn_node_id = new[chi_sys_cfg.num_hn];
   hn_node_id[0] = 8;
   hn_node_id[1] = 9
   hn_node_id[2] = 10;
   hn_node_id[3] = 11;
   hn_node_id[4] = 12;
   hn_node_id[5] = 13;
   hn_node_id[6] = 14;
   hn_node_id[7] = 15;
   hn_node_id[8] = 16;
   <chi_sys_cfg>.set_hn_node_id(hn_node_id);


   /** Program the MN node ID. In this case, this is same as HN with index 8
*/
```

```
   <chi_sys_cfg>.misc_node_id = hn_node_id[8];


   /** Program the HN interface types for each of the HN indices */
   hn_interface_type = new[<chi_sys_cfg>.num_hn];
   hn_interface_type[0] = svt_chi_address_configuration::HN_F;
   hn_interface_type[1] = svt_chi_address_configuration::HN_F;
   hn_interface_type[2] = svt_chi_address_configuration::HN_F;
   hn_interface_type[3] = svt_chi_address_configuration::HN_F;
   hn_interface_type[4] = svt_chi_address_configuration::HN_F;
   hn_interface_type[5] = svt_chi_address_configuration::HN_F;
   hn_interface_type[6] = svt_chi_address_configuration::HN_F;
   hn_interface_type[7] = svt_chi_address_configuration::HN_F;
   hn_interface_type[8] = svt_chi_address_configuration::HN_I;
   <chi_sys_cfg>.set_hn_interface_type(hn_interface_type);


   /** Program the address ranges for HN index that corresponds to HN-I */
   <chi_sys_cfg>.set_hn_addr_range(8, `TB_START_ADDR_HN_I,
`TB_END_ADDR_HN_I);


   /** Program the address ranges for MN index. Typically, this is subset of
HN-I address ranges */
   <chi_sys_cfg>.set_mn_addr_range(8, `TB_START_ADDR_MN, `TB_END_ADDR_MN);


   /** Map SN to HNs: SN with index 0 maps to HN indices {0,1,2,3,4,5,6,7}.
SN with index 1 maps to HN index 8 */
   <chi_sys_cfg>.set_sn_to_hn_map(0,{0,1,2,3,4,5,6,7});
   <chi_sys_cfg>.set_sn_to_hn_map(1,{8});
```

THREE SN-F STRIPING

OVERVIEW

In case of three SN-F striping, HN-Fs route the transactions to SN-Fs, the addresses are striped among three SN-Fs. In this case, each of the HN-F can route the transactions to all the three SN-Fs present in the system. The addresses are striped at 256 B granularities across the three SN-Fs, with 512 B granularities at 2 KB address boundary.

The total addressable space must be equally distributed across the three SN-Fs, with the addressable space per each SN-F be such that it can be resented as (2 ^ integer value). The minimum addressable space expected is 256 MB per SN-F, that is, 768 MB across the three SN-Fs. The maximum addressable space expected is 4 TB per SN-F, that is, 12 TB across the three SN-Fs.

Based on the total addressable space at the SN-Fs, a given SN-F presents only the relevant lower order bits to the DRM that is connected downstream to that SN-F. The total addressable space is decided based on the top two address bit indices, from which and beyond these up to 43, the SN-Fs will not present to the DRAM that is connected downstream. That is, only the bit indices from 0 up to the top address bit indices will be present to the DRAM. Note that address aliasing can take place when the RN-Fs generate different addresses, but that map to same physical locations in the DRAM that is connected to the downstream of the SN-Fs.

Refer to the class reference of 'svt_chi_system_configuration' under the group 'chi_sys_config_three_sn_f_striping' for more details.

USER INTERFACE

ATTRIBUTES

Following are the attributes that **must be** programmed in svt_chi_system_configuration to use this feature:

- three_sn_f_striping_enable must be set to 1
- Top address bit indices should be programmed together appropriately based on total addressable space across the three SN-Fs
    - three_sn_f_striping_top_address_bit_0
        - Should be in the range [28:43]
    - three_sn_f_striping_top_address_bit_1
        - should be in the range [28:43]
    - Also note that the value of three_sn_f_striping_top_address_bit_1 should be greater than or equal to three_sn_f_striping_top_address_bit_0. Typically,
        - three_sn_f_striping_top_address_bit_1 = three_sn_f_striping_top_address_bit_0+1

APIS

Following APIs are present in the svt_chi_system_configuration. Refer to the class reference for more details.

- get_three_sn_f_striping_addressable_space()
    - To know the total addressable space based on the values of top address fields programmed through svt_chi_system_configuration::three_sn_f_striping_top_address_bit_1, svt_chi_system_configuration::three_sn_f_striping_top_address_bit_1.
- get_three_sn_f_striping_based_sn_f_idx()
    - Returns the SN-F index that corresponds to the address input argument passed, based on three SN-F striping
- is_three_sn_f_striping_enable_valid()
    - Indicates whether enabling this feature is valid or not, based on the related CHI system configuration values programmed

Note: When 'three SN-F striping is enabled', the API svt_chi_system_configuration::hn_sn_map() needs to be used only to program the 'SN to HN-I map'; any mapping programmed for 'SN-F to HN-F map' will be ignored.

PROGRAMMING EXAMPLE

```
class cust_svt_chi_system_configuration extends svt_chi_system_configuration;

------
```

```
  /** Program number of HNs(5) = number of HN-Fs(4) + number of HN-Is(1) */
  num_hn = 5;

   /** Program the HN interface types for each of the HN indices: 4 HN-Fs, 1
HN-I */
   hn_interface_type = new[num_hn];
   hn_interface_type[0] = svt_chi_address_configuration::HN_F;
   hn_interface_type[1] = svt_chi_address_configuration::HN_F;
   hn_interface_type[2] = svt_chi_address_configuration::HN_F;
   hn_interface_type[3] = svt_chi_address_configuration::HN_F;
   hn_interface_type[4] = svt_chi_address_configuration::HN_I;
   set_hn_interface_type(hn_interface_type);

   /** Program the node IDs for each of the HN indices: 4 HN-Fs, 1 HN-I */
   hn_node_id = new[num_hn];
   hn_node_id[0] = 3;
   hn_node_id[1] = 5;
   hn_node_id[2] = 11;
   hn_node_id[3] = 13;
   hn_node_id[4] = 0;
   set_hn_node_id(hn_node_id);

   /** Number of SN-Fs that are not mapped to HN-I should be 3 */
    num_sn = 3;
    foreach(sn_cfg[i]) sn_cfg[i].chi_interface_type =
svt_chi_node_configuration::SN-F;

  /** Three SN striping related settings */
   three_sn_f_striping_enable = 1;
   three_sn_f_striping_top_address_bit_0 = 42;
   three_sn_f_striping_top_address_bit_1 = 43;
------
endclass
```

SN-F SELECTION

For a given address ‘addr’ at HN-F, following are the parameters that influence the targeted SN-F.

- addr[16:8]
- top tow address bits
    - bit[1:0] high_addr_bits = {addr[top_addr_bit_1], addr[top_addr_bit_0]}

Following is the algorithm that indices, out of the three SN-Fs, which one is the targeted SN-F for a given ‘addr’:

- int sn_f_num = (high_addr_bits[1:0] + addr[16:14] + addr[13:11] + addr[10:8])%3

The order of selection of SN-Fs depends on the value of these fields, and primarily influenced by the top address bits.

TOTAL ADDRESSABLE SPACE

The total addressable space depends on the top address bit field values.

- Addressable space per SN-F = (1/4) * (2^(top_addr_bit_0+2))
- Total addressable space across three SN-Fs – 3 *(addressable space per SN-F)

EXAMPLE:

Consider top_addr_bit_0 = 30, top_addr_bit_1 = 31.

- Addressable space per SN-F = 1 GB
- Total addressable space across the three SN-Fs = 3 GB

DIFFERENT TOP ADDRESS BIT FIELD VALUES AND ADDRESSABLE SPACE VALUES

top_bit1	top_bit0	size_per_SN-F	size_across_3_SN-Fs
29	28	256MB	768MB
30	29	512MB	1536MB
31	30	1GB	3GB
32	31	2GB	6GB
33	32	4GB	12GB
34	33	8GB	24GB
35	34	16GB	48GB
36	35	32GB	96GB
37	36	64GB	192GB
38	37	128GB	384GB
39	38	256GB	768GB
40	39	512GB	1536GB
41	40	1TB	3TB
42	41	2TB	6TB
43	42	4TB	12TB

ADDRESS ALIASING

As described in earlier section, the SN-F selection depends on addr[top_addr_bit_0], addr[top_addr_bit_1], addr[16:8]. As the CHI addresses are 44 bit wide, it’s possible that multiple addresses at RN-F can get mapped to same physical address in the DRAM.

EXAMPLE:

Consider top_addr_bit_0=30, top_addr_bit_1=31 where total addressable space is 3 GB. However, if the addresses are generated such that addr[31:30] is 2'b11, that is beyond 3 GB, it gets mapped to same SN-F with addr[31:30] is 2'b00. This implies the same physical address of DRAM will be accessed.

Note that address aliasing is not a violation, but it's a scenario where multiple addresses can get mapped to same physical address due to the total address space available at each SN-F.

CHI MEMORY WITHIN SN GROUP

The CHI SN group contains a memory of type svt_chi_memory instantiated. The CHI system monitor's slave_data_integrity_check relies on the memory within the SN group.

- In the active mode, the memory can be updated when the SN's generator type is selected as Memory response generator (svt_chi_node_configuration::prot_generator_type = svt_chi_node_configuration::MEMORY_RESPONSE_GEN). Refer to the HTML class reference and tb_chi_svt_vmm_basic_sys for the usage.
- In the passive mode, the SN group updates the memory for write transactions that are completed successfully. To update the memory for the reads transactions in passive mode, the attribute svt_chi_node_configuration:: memory_update_for_read_xact_enable needs to be programmed to 1. Refer to HTML class reference for details.
- When the read is performed to a location that was not written to earlier, the corresponding transaction's field svt_chi_transaction::is_read_data_unknown is set to 1

CHI INTERFACE CLOCK MODE

COMMON CLOCK MODE

- This is the default use model.
- Common clock 'clk' needs to be passed as an argument to top level CHI interface (svt_chi_if) instance.
- The same 'clk' is passed by the top level interface to all the RN, SN sub interfaces and corresponding Interconnect port sub interfaces.

MULTI CLOCK MODE

- This is not the default use model. This needs to be enabled when different clocks are needed for each of the RNs and SNs.
- This mode is enabled when the compile time macro `SVT_CHI_ENABLE_MULTI_CLOCK is defined.
- Instead of passing 'clk' as argument to the top level CHI interface (svt_chi_if) instance, arrays of RN clocks (rn_clk[`SVT_CHI_MAX_NUM_RNS-1:0]) and SN clocks (sn_clk[`SVT_CHI_MAX_NUM_SNS-1:0]) are passed in this case.
    - Note that the array of clocks should be declared as shown above, that is from [*_MAX_*-1:0].
- The rn_clk[rn_idx] then gets connected to corresponding CHI RN sub interface svt_chi_if::chi_rn_if[rn_idx], and corresponding port on Interconnect svt_chi_if::chi_ic_rn_if[rn_idx].
- Similarly, The sn_clk[sn_idx] then gets connected to corresponding CHI SN sub interface svt_chi_if::chi_sn_if[sn_idx], and corresponding port on Interconnect svt_chi_if::chi_ic_sn_if[sn_idx].
- Refer to the documentation of the CHI top level interface file svt_chi_if.svi for more details.
- Also refer to tb_chi_svt_uvm_basic_sys/top.sv for the use model example and corresponding README file tb_chi_svt_vmm_basic_sys/top.sv

USAGE EXAMPLE

```
module test_top;

  // System clock, reset
  bit                     SystemClock;
  bit                     SystemReset;
  // Individual clocks for RNs, SNs
`ifdef SVT_CHI_ENABLE_MULTI_CLOCK
  bit                     rn_clk[`SVT_CHI_MAX_NUM_RNS-1:0];
  bit                     sn_clk[`SVT_CHI_MAX_NUM_SNS-1:0];
`endif

  /** VIP Interface instance representing the CHI system */
`ifdef SVT_CHI_ENABLE_MULTI_CLOCK
  svt_chi_if chi_if_1(rn_clk, sn_clk, SystemReset);
`else
  svt_chi_if chi_if_1(SystemClock,SystemReset);
`endif

  /** Generate System clock, RN and SN clocks.
   *  Below is for demonstration purpose. */
  initial begin
    #(simulation_cycle/2); // No clock edge at T=0
    SystemClock = 0;
    forever begin
`ifdef SVT_CHI_ENABLE_MULTI_CLOCK
      foreach (rn_clk[i]) begin
         rn_clk[i] = SystemClock;
      end

      foreach (sn_clk[i]) begin
          sn_clk[i] = SystemClock;
      end
`endif
      #(simulation_cycle/2) SystemClock = ~SystemClock;
    end
  end // initial begin

-----
endmodule
```

CHI SYSTEM VIRTUAL SCENARIO LIBRARY

Following categories of virtual scenarios are available:

- Single node scenarios
- Flow control scenarios
- Outstanding scenarios

SINGLE NODE SCENARIOS

System level single node RN transaction virtual scenarios exercises a given transaction type from one of the RNs. These scenarios provide controls for configurability through vmm_opts.

Refer to public attributes of the class svt_chi_system_single_node_rn_coherent_transaction_base_virtual_scenario of CHI VMM class reference:
$DESIGNWARE_HOME/vip/svt/amba_svt/latest/doc/chi_svt_vmm_public/html/class_svt_chi_system_single_node_rn_coherent_transaction_base_virtual_scenario.html

FLOW CONTROL SCENARIOS

System level Multi node RN transaction virtual scenarios exercise CHI protocol flow control features. These scenarios provide the control for the user to select RN/HN node indices from the test through vmm_opts.

Following are the list of scenarios available under this category:

- class svt_chi_system_protocol_flow_ctrl_read_resp_same_txnid_diff_rn_diff_hn_virtual_scenario
- class svt_chi_system_protocol_flow_ctrl_read_resp_same_txnid_diff_rn_same_hn_virtual_scenario
- class svt_chi_system_protocol_flow_ctrl_cmo_resp_same_txnid_diff_rn_diff_hn_virtual_scenario
- class svt_chi_system_protocol_flow_ctrl_cmo_resp_same_txnid_diff_rn_same_hn_virtual_scenario
- class svt_chi_system_protocol_flow_ctrl_write_resp_same_txnid_diff_rn_diff_hn_virtual_scenario
- class svt_chi_system_protocol_flow_ctrl_write_resp_same_txnid_diff_rn_same_hn_virtual_scenario
- class svt_chi_system_protocol_flow_ctrl_copyback_resp_same_txnid_diff_rn_diff_hn_virtual_scenario
- class svt_chi_system_protocol_flow_ctrl_copyback_resp_same_txnid_diff_rn_same_hn_virtual_scenario

Refer to public attributes of the class svt_chi_system_protocol_flow_ctrl_cmo_resp_same_txnid_diff_rn_diff_hn_virtual_scenario of CHI VMM class reference:
$DESIGNWARE_HOME/vip/svt/amba_svt/latest/doc/chi_svt_vmm_public/html/svt_chi_system_protocol_flow_ctrl_cmo_resp_same_txnid_diff_rn_diff_hn_virtual_scenario.html

OUTSTANDING SCENARIOS

System level Multi node RN outstanding transaction virtual scenarios exercise CHI protocol outstanding transactions features. These scenarios provide the control for the user to select RN/HN node indices from the test through vmm_opts.

Following are the list of scenarios available under this category.

- class svt_chi_system_protocol_flow_ctrl_read_outstanding_diff_rn_diff_hn_virtual_scenario
- class svt_chi_system_protocol_flow_ctrl_read_outstanding_diff_rn_same_hn_virtual_scenario
- class svt_chi_system_protocol_flow_ctrl_read_outstanding_same_rn_same_hn_virtual_scenario
- class svt_chi_system_protocol_flow_ctrl_cmo_outstanding_diff_rn_diff_hn_virtual_scenario
- class svt_chi_system_protocol_flow_ctrl_cmo_outstanding_diff_rn_same_hn_virtual_scenario
- class svt_chi_system_protocol_flow_ctrl_cmo_outstanding_same_rn_same_hn_virtual_scenario
- class svt_chi_system_protocol_flow_ctrl_write_outstanding_diff_rn_diff_hn_virtual_scenario
- class svt_chi_system_protocol_flow_ctrl_write_outstanding_diff_rn_same_hn_virtual_scenario
- class svt_chi_system_protocol_flow_ctrl_write_outstanding_same_rn_same_hn_virtual_scenario
- class svt_chi_system_protocol_flow_ctrl_copyback_outstanding_diff_rn_diff_hn_virtual_scenario
- class svt_chi_system_protocol_flow_ctrl_copyback_outstanding_diff_rn_same_hn_virtual_scenario
- class svt_chi_system_protocol_flow_ctrl_copyback_outstanding_same_rn_same_hn_virtual_scenario

Refer to public attributes of the class svt_chi_system_protocol_flow_ctrl_read_outstanding_diff_rn_diff_hn_virtual_scenario of CHI VMM class reference:
$DESIGNWARE_HOME/vip/svt/amba_svt/latest/doc/chi_svt_vmm_public/html/svt_chi_system_protocol_flow_ctrl_read_outstanding_diff_rn_diff_hn_virtual_scenario.html

CONTROLLING RN SNOOP RESPONSE GENERATION

The snoop responses from the RN group in active mode with the interface type RN-F can be controlled using the following mechanism. The snoop response generator callback needs to be extended and registered with the snoop response generator within the RN-F.

Below is the example. The chi_svt_vmm_basic_sys example is updated to demonstrate the same.

```
//------------------------------------------------------------------------------
/**
 * Snoop response generator callback to control the generated snoop response
 * before it is fed to RN driver transactor.
 * In this case, the responses for SnpShared, SnpClean, SnpPonce snoop requests
```

```
 * is updated to have pass_dirty as 0 when the final state at the snooped RN
 * is going to be Shared and the response includes the data.
 */
//------------------------------------------------------------------------------
class cust_svt_chi_rn_snoop_response_gen_callback extends svt_chi_rn_snoop_response_gen_callback;
  vmm_log log;
  function new();
    super.new();
  endfunction // new

  virtual function void generate_snoop_response(svt_chi_rn_snoop_response_gen gen, svt_chi_rn_snoop_transaction
xact);
    log = gen.log;
    case (xact.snp_req_msg_type)
      svt_chi_snoop_transaction::SNPSHARED,
      svt_chi_snoop_transaction::SNPCLEAN,
      svt_chi_snoop_transaction::SNPONCE: begin
        if ((xact.snp_rsp_isshared == 1) &&
            (xact.snp_rsp_datatransfer == 1) &&
            (xact.resp_pass_dirty == 1)) begin
          `svt_debug("generate_snoop_response_callback",
                     {`SVT_CHI_SNP_PRINT_PREFIX(xact),
                      "isshared = 1. datatransfer = 1. Changing pass_dirty from 1 to 0"});
          xact.resp_pass_dirty = 0;
          `svt_verbose("generate_snoop_response_callback",
                  $sformatf("Updated pass_dirty to 1 \n%0s", xact.`SVT_DATA_PSDISPLAY("updated_snp_xact:: ")));
        end
        else begin
          `svt_note("generate_snoop_response_callback",
               {`SVT_CHI_SNP_PRINT_PREFIX(xact), "not updated"});
        end // else: !if((xact.snp_rsp_isshared == 1) &&...
      end // case: svt_chi_snoop_transaction::SNPSHARED,...
      default: begin
        `svt_note("generate_snoop_response_callback",
             {`SVT_CHI_SNP_PRINT_PREFIX(xact), "not updated"});
      end
    endcase // case (xact.snp_req_msg_type)
  endfunction // generate_snoop_response

endclass // cust_svt_chi_rn_snoop_response_gen_callback
```

Following are the environment updates needed to integrate the above callback:

```
class chi_basic_group extends vmm_group;
------
  /** RN-F Snoop response generator callback */
  cust_svt_chi_rn_snoop_response_gen_callback rn_snp_rsp_cb[];
------

// ----------------------------------------------------------------------------
function void chi_basic_group::connect_ph();
  -----
    // Instantiate snp rsp gen callback and register the same with each of the RN-Fs
    // within chi system group 'chi_active_group'.
    for (int idx=0; idx < active_cfg.num_rn; idx++) begin
    if (active_cfg.rn_cfg[idx] != null) begin
      if (active_cfg.rn_cfg[idx].chi_interface_type == svt_chi_node_configuration::RN_F) begin
        rn_snp_rsp_cb = new[rn_snp_rsp_cb.size()+1](rn_snp_rsp_cb);
        rn_snp_rsp_cb[rn_snp_rsp_cb.size()-1] = new();
        chi_active_group.rn[idx].snoop_response_gen.append_callback(rn_snp_rsp_cb[rn_snp_rsp_cb.size()-1]);
      end
    end
  end
  -----
endfunction

endclass
```

4
Verification Features

This chapter describes the various verification features available along with Synopsys Discovery AXI Verification IP. These features are not yet supported.

4.1 Verification Planner

CHI VIP provides verification plans which can be used for tracking verification progress. A set of top-level plans and su $DESIGNWARE_HOME/vip/svt/amba_svt/latest/doc/VerificationPlans.

For more information, refer to the README file, which is available at:
$DESIGNWARE_HOME/vip/svt/amba_svt/latest/doc/VerificationPlans/README

4.2 Protocol Analyzer Support

CHI VIP supports Synopsys® Protocol Analyzer. Protocol Analyzer is an interactive graphical application which provides protocol-oriented analysis and debugging capabilities.

For the CHI SVT VIP, protocol file generation is enabled or disabled through the variable svt_chi_node_configuration::enable_xact_xml_gen, and svt_chi_node_configuration::enable_fsm_xml_gen.

The default value of these variables is zero, which means that protocol file generation is disabled by default. To enable protocol file generation, set the value of these variables to one in the node configuration of each RN or SN component for which protocol file generation is desired.

For CHI system monitor, the protocol file generation is enabled or disabled through the variable svt_chi_system_configuration::enable_xml_gen. The default value of this variable is zero, which means that protocol generation file is disabled by default. When set to one, this enables protocol file generation fro CHI system monitor, that associates coherent transaction with associated snoop transactions.

The next time the environment is simulated, protocol files will be generated according to the node configurations. The protocol files will be in.xml format. Import these files into the Protocol Analyzer to view the protocol transactions.

For Verdi documentation, refer to $NOVAS_HOME/doc/LCAFeaturesGuide.pdf

NOTE: Protocol Analyzer has been enhanced to read FSDB transactions and Verdi can load the FSDB transactions into the browser.

4.2.1Support for Native Dumping of FSDB

Native FSDB supported in CHI VIP.

❖ FSBD Generation: Protocol Analyzer uses transaction-level dump database. You can use the following settings to dump the transaction database:

✦ Compile Time Options:

✧ -lca -kdb // dumps the work.lib++ data for source coding view

✧ +define+SVT_FSDB_ENABLE // enables FSDB dumping

✧ Use any of the following options to enable Verdi PLI libraries: -P ${NOVAS_HOME}/share/PLI/VCS/${novas_platform} /novas.tab ${NOVAS_HOME}/share/PLI/VCS/${novas_platform}/pli.a

✦ New configuration parameter added svt_chi_system_configuration::pa_format_type that controls the format of PA output file, is for XML/FSDB generation in svt_chi_system_configuartion.sv. It is a system configuration variable. Add the following setting in system configuration to enable the generation of XML/FSDB:

/** Enable protocol file generation for Protocol Analyzer */

this.rn_cfg[0].enable_xact_xml_gen = 1;

this.sn_cfg[0].enable_xact_xml_gen = 1;

this.rn_cfg[0].enable_fsm_xml_gen = 1;

this.sn_cfg[0].enable_fsm_xml_gen = 1;

this.enable_xml_gen = 1;

this.pa_format_type = 1; // 0 is XML, 1 FSDB and 2 both XML and FSDB, default it will be zero See the HTML class reference for the documentation of these attributes for more details.

❖ Invoking Protocol Analyzer: Perform the following steps to invoke Protocol Analyzer in interactive or post-processing mode:

✦ Post-processing Mode

✧ Load the transaction dump data and issue the following command to invoke the GUI: verdi - ssf <dump.fsdb> -lib work.lib++

✧ In Verdi, navigate to **Tools > Transaction Debug** and select the Protocol Analyzer option in the main window to invoke Protocol Analyzer.

✦ Interactive Mode

✧ Issue the following command to invoke Protocol Analyzer in an interactive mode: <simv> -gui=Verdi

You can invoke the Protocol Analyzer as described above using Verdi. The Protocol Analyzer transaction view gets updated during the simulation.

4.3 DEBUG FEATURES:

CHI SYSTEM MONITOR:

When CHI system monitor is enabled by setting svt_chi_system_configuration::system_monitor_enable to 1, following features are applicable

- svt_chi_system_configuration::display_summary_report Enables displaying coherent and snoop transaction summary, SN transaction summary towards end of the simulation with UVM_LOW verbosity
- svt_chi_system_configuration::enable_summary_reporting Enables interactive display of coherent and snoop transaction summary, SN transaction summary as and when its available with VMM_VERBOSE verbosity
- svt_chi_system_configuration::enable_summary_tracing Enables interactive writing of coherent and snoop transaction summary, SN transaction summary into respective seperate files as and when its available

CHI RN GROUP, CHI SN GROUP:

- svt_chi_node_configuration::enable_pl_reporting Enables interactive display of coherent and snoop transaction(only for RN group) tracing summary when these transactions are complete with VMM_VERBOSE verbosity
- svt_chi_node_configuration::enable_pl_tracing Enables interactive writing of coherent and snoop transaction(only for RN group) tracing summary with corresponding flit info to separate files when these transactions are complete
- svt_chi_node_configuration::enable_ll_reporting Enables interactive display of flit tracing summary when these flits are complete with VMM_VERBOSE verbosity
- svt_chi_node_configuration::enable_pl_tracing Enables interactive writing of flit tracing summary to separate files when these flits are complete

5
Verification Topologies

This chapter shows you from a high-level, how the CHI VIP can be used to test Interconnect DUT.

5.1 Interconnect DUT and RN/SN VIP

In this scenario, DUT is a CHI Interconnect tested by a RN and SN VIP: Assuming that the CHI Interconnect has *m* RN ports and *s* SN ports, configure the CHI System Group to have *s* RN groups and *m* SN groups, in active mode. The active RN groups will generate CHI transactions towards the interconnect SN ports, and active SN groups connected would respond to the transactions generated by interconnect RN ports. The RN and SN groups would also perform passive functions such as protocol checking, coverage generation and transaction logging.

Implementation of this topology requires the setting of the following properties:

- Assuming instance name of system configuration is "chi_sys_cfg"
- Assuming number of RN ports on interconnect = 2
- Assuming number of SN ports on interconnect = 2

System configuration settings:

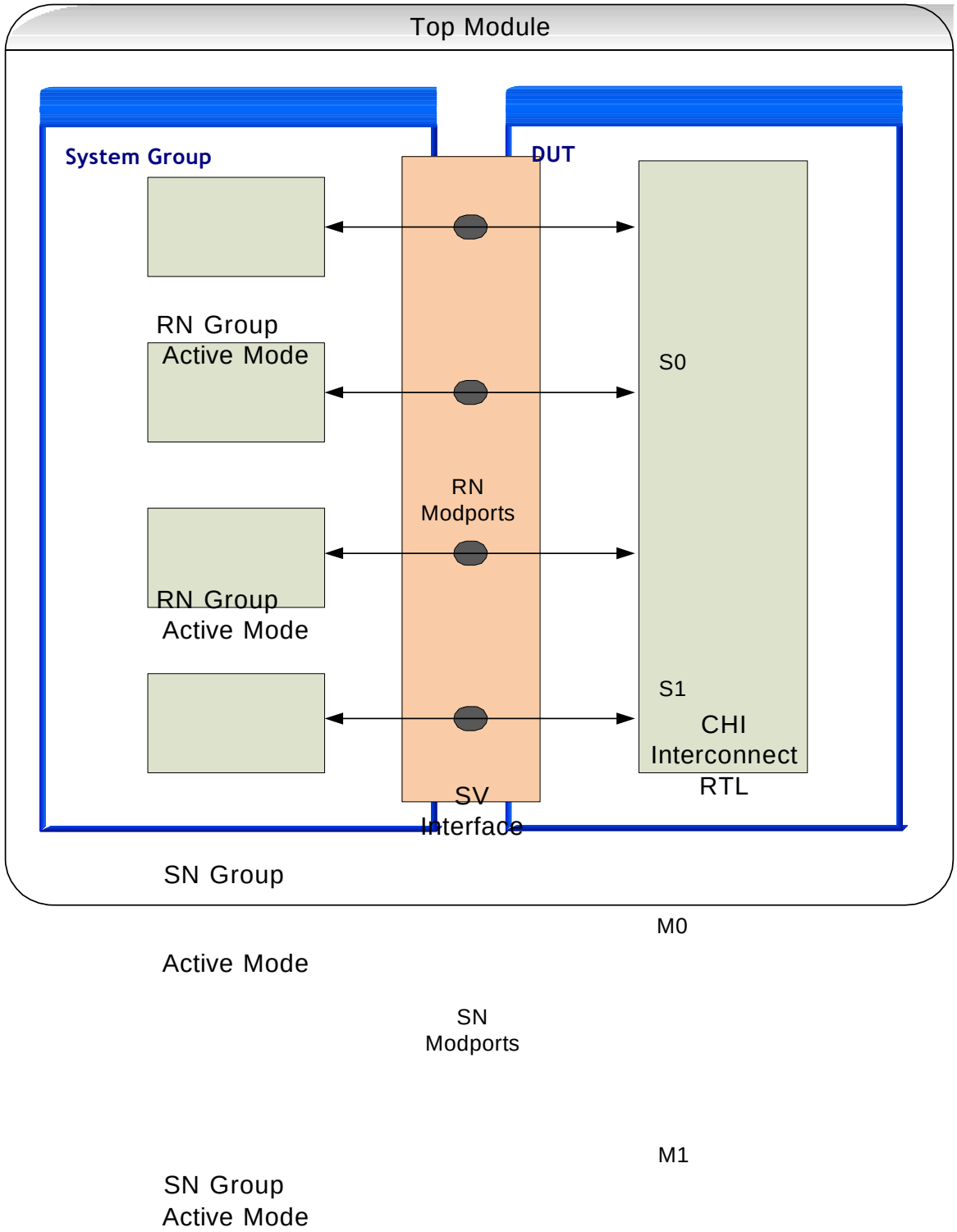
- chi_sys_cfg[0].num_rn = 2;
- chi_sys_cfg[0].num_sn = 2;

Port configuration settings:

- chi_sys_cfg.rn_cfg[0].is_active = 1;
- chi_sys_cfg.rn_cfg[1].is_active = 1;
- chi_sys_cfg.sn_cfg[0].is_active = 1;
- chi_sys_cfg.sn_cfg[1].is_active = 1;

Figure 5-1 shows the testbench setup.

Figure 5-1 Interconnect DUT with RN and SN VIP (Active Mode)



5.2 System DUT with Passive VIP

In this setup, DUT is a CHI system with multiple CHI RNs, SNs and interconnect. VIP is required to monitor DUT.

Assuming that the CHI System has *m* RNs and *s* SNs, configure the CHI System Env to have *m* RN groups and *s* SN groups, in passive mode. The passive RN and SN groups would perform passive functions such as protocol checking, coverage generation and transaction logging.

Implementation of this topology requires the setting of the following properties:

- Assuming instance name of system configuration is "sys_cfg"
- Assuming number of RN ports on interconnect = 2
- Assuming number of SN ports on interconnect = 2

System configuration settings:

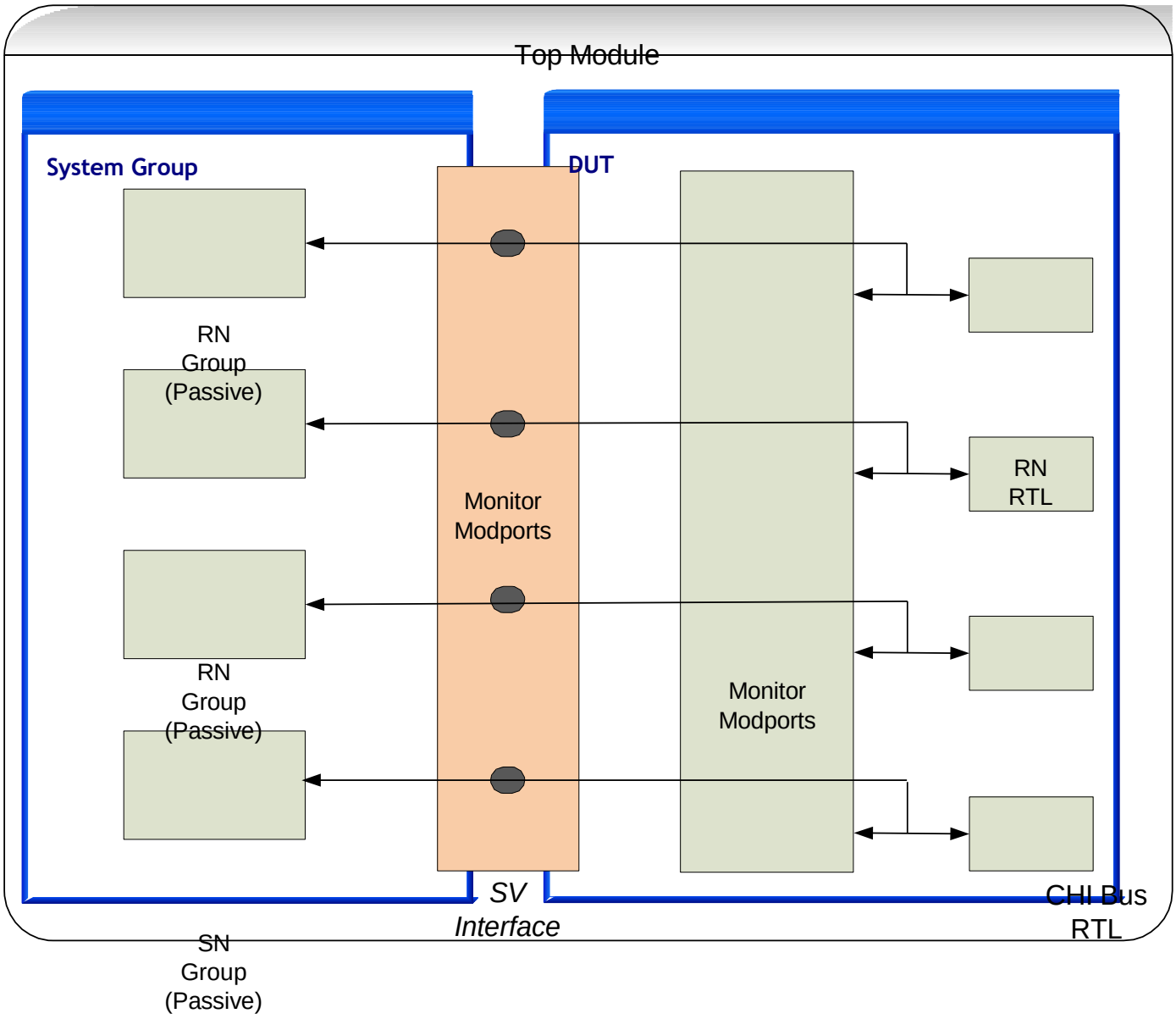
- chi_sys_cfg[0].num_rn = 2;
- chi_sys_cfg[0].num_sn = 2;

Port configuration settings:

- chi_sys_cfg.rn_cfg[0].is_active = 0;
- chi_sys_cfg.rn_cfg[1].is_active = 0;
- chi_sys_cfg.sn_cfg[0].is_active = 0;
- chi_sys_cfg.sn_cfg[1].is_active = 0;

Figure 5-2 shows this setup.

Figure 5-2 System DUT with Passive VIP



R
T
L

SN
RTL

RN

SN
Group
(Passive)

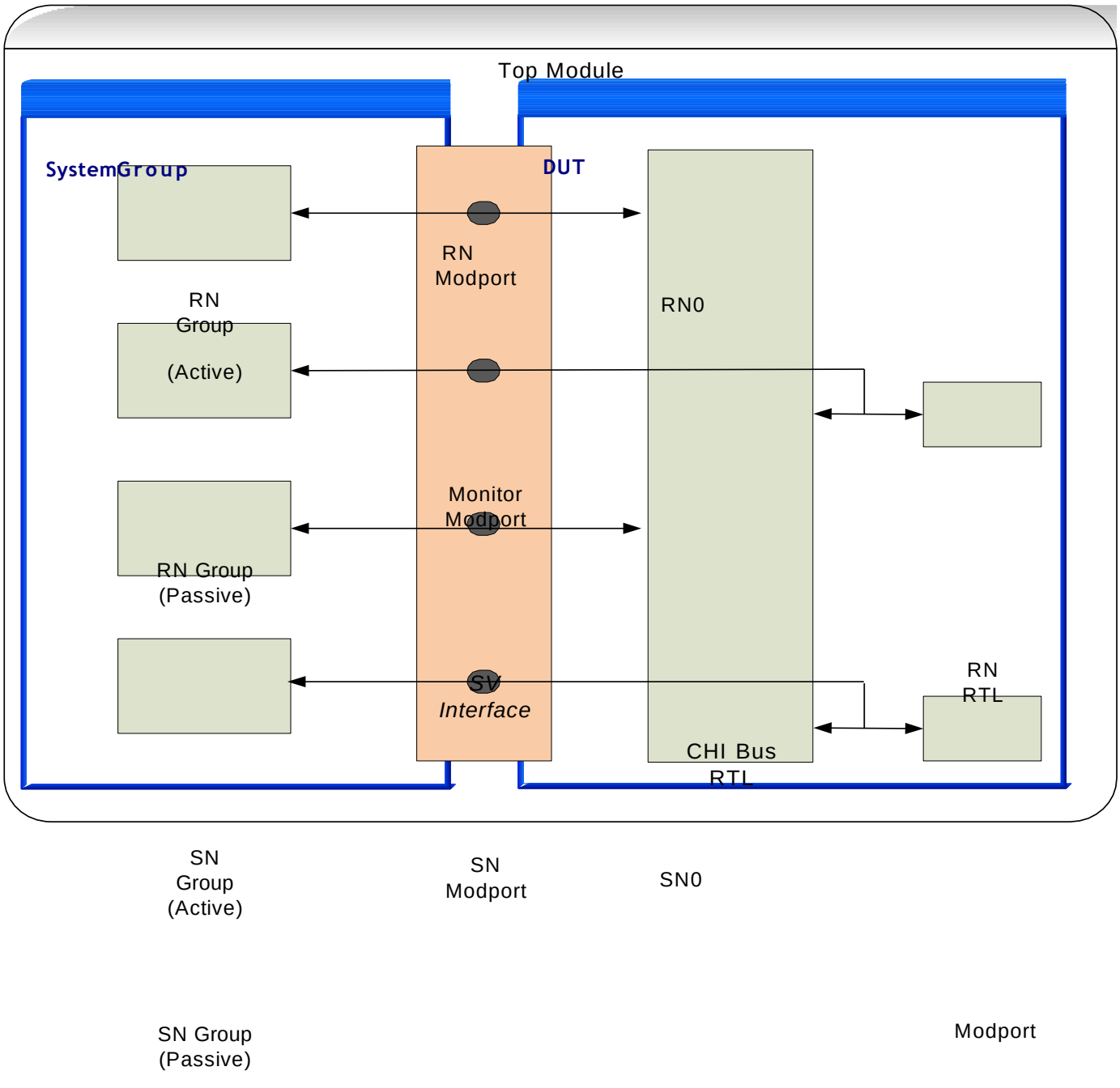
SN
RTL

5.3 System DUT with Mix of Active and Passive VIP

In this scenario, DUT is a system with multiple CHI RNs, SNs and interconnect. The VIP is required to provide background traffic on some ports, and to monitor on ports.

Assume that the CHI System DUT has two RN ports and two SN ports. VIP is required to provide background traffic to ports RN0 and SN0. All the ports need to be monitored. Configure the CHI System Group to have two RN groups and two SN groups. Configure the RN group connected to port RN0, and SN group connected to port SN0 as active. Configure the RN group connected to port RN1 and SN agent connected to port SN1 as passive. All the groups would continue to perform passive functions such as protocol checking and coverage.

Figure 5-3 System DUT with Mix of Active and Passive VIP



RTL

SN

Implementation of this topology requires the setting of the following properties:
Assuming instance name of system configuration is "sys_cfg".

System configuration settings:

- chi_sys_cfg[0].num_rn = 2;
- chi_sys_cfg[0].num_sn = 2;

Port configuration settings:

- chi_sys_cfg.rn_cfg[0].is_active = 1;
- chi_sys_cfg.rn_cfg[1].is_active = 0;
- chi_sys_cfg.sn_cfg[0].is_active = 1;
- chi_sys_cfg.sn_cfg[1].is_active = 0;

6
Using CHI Verification IP

This chapter describes how to install and run a getting started example and provides usage notes for CHI Verification IP.

6.1 SystemVerilog VMM Example Testbenches

This section describes SystemVerilog VMM example testbenches that show general usage for various applications. A summary of the examples is listed in Table 6-1

Table 6-1 SystemVerilog Example Summary

Example Name	Level	Description
tb_chi_svt_vmm_basic_sys	Basic	The example consists of the following: • A top-level module, which includes tests, instantiates interfaces, HDL interconnect wrapper, generates system clock, and runs the tests • A base test, which is extended to create a directed • The tests create a testbench environment, which in turn creates AMBA System Group. AMBA System Group creates CHI System Group • CHI System Group is configured with one RN Group, one SN Group.

The examples are located at:

$DESIGNWARE_HOME/vip/svt/amba_svt/latest/examples/sverilog/

6.2 Installing and Running the Examples

Below are the steps for installing and running example tb_chi_svt_vmm_basic_sys. The similar steps are also applicable for other examples:

1. Install the example using the following command line:

   % cd <location where example is to be installed>

   % mkdir design_dir <provide any name of your choice>

   % $DESIGNWARE_HOME/bin/dw_vip_setup -path ./design_dir -e amba_svt/tb_chi_svt_vmm_basic_sys -svtb

   The example would get installed under:

   <design_dir>/examples/sverilog/amba_svt/tb_chi_svt_vmm_basic_sys

2. Use either one of the following to run the testbench:

   a. Use the Makefile:

      One test is provided in the "tests" directory.

      i. ts.directed_test.sv

      To run test ts.directed_test.sv, do following:

      gmake USE_SIMULATOR=vcsvlog directed_test WAVES=1

      Invoke "gmake help" to show more options. b.

   Use the sim script:

      For example, to run test ts.directed_test.sv, do following:

      ./run_chi_svt_vmm_basic_sys -w directed_test vcsvlog

      Invoke "./run_chi_svt_vmm_basic_sys -help" to show more options.

For more details of installing and running the example, refer to the README file in the example, located at:

$DESIGNWARE_HOME/vip/svt/amba_svt/latest/examples/sverilog/tb_chi_svt_vmm_basic_sys/README

OR

<design_dir>/examples/sverilog/amba_svt/tb_chi_svt_vmm_basic_sys/README

A
Reporting Problems

A.1 Introduction

This chapter provides you an overview for working through and reporting SVT transactor issues encountered in the field. It describes the data you must submit when a problem is initially reported to Synopsys. After a review of the initial information, Synopsys may decide to request adjustments to the information being requested, which is the focus of the next section.

A.2 Initial Customer Information

To report any issues with SVT transactor, you may contact Synopsys Support Center. Follow these steps:

1. Before you contact technical support, be prepared to provide the following:
   - ✦ A description of the issue under investigation
   - ✦ A description of your verification environment
2. Create a value change dump (VCD) file
3. Generate a log file for the simulation

Providing this information will help ensure accurate diagnosis of the problem.

If the requested information is not sufficient to determine the cause of your issue, the Synopsys Support Center may request a simple testbench demonstrating the issue. This is the most effective way to debug issues, but if an example cannot be provided, Synopsys may request additional information that could include regenerating the log file using different settings.

If your testbench initializes the random seed (see 'srandom()' in the VCS manual) in the host simulator, then a seed that can be used to demonstrate the problem must be included in the testbench or provided as information for executing the testbench.