

Verification Continuum™

VC Verification IP

CHI

UVM User Guide

Version U-2022.12, December 2022



Copyright Notice and Proprietary Information

© 2022 Synopsys, Inc. All rights reserved. This Synopsys software and all associated documentation are proprietary to Synopsys, Inc. and may only be used pursuant to the terms and conditions of a written license agreement with Synopsys, Inc. All other use, reproduction, modification, or distribution of the Synopsys software or the associated documentation is strictly prohibited.

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.

Trademarks

Synopsys and certain Synopsys product names are trademarks of Synopsys, as set forth at <http://www.synopsys.com/company/legal/trademarks-brands.html>. All other product or company names may be trademarks of their respective owners.

Free and Open-Source Software Licensing Notices

If applicable, Free and Open-Source Software (FOSS) licensing notices are available in the product installation.

Third-Party Links

Any links to third-party websites included in this document are for your convenience only. Synopsys does not endorse and is not responsible for such websites and their practices, including privacy practices, availability, and content.

www.synopsys.com

Contents

Preface	9
About This Guide	9
Web Resources	9
Customer Support	9
Synopsys Statement on Inclusivity and Diversity	10
Chapter 1	
Introduction	11
1.1 Introduction	11
1.2 Prerequisites	12
1.3 References	12
1.4 Product Overview	12
1.5 Language and Methodology Support	13
1.6 Features Supported	13
1.6.1 Protocol Features	13
1.6.2 Verification Features	15
1.6.3 Methodology Features	15
Chapter 2	
Installation and Setup	17
2.1 Verifying the Hardware Requirements	17
2.2 Verifying Software Requirements	17
2.2.1 Platform/OS and Simulator Software	17
2.2.2 Synopsys Common Licensing (SCL) Software	18
2.2.3 Other Third Party Software	18
2.3 Preparing for Installation	18
2.4 Downloading and Installing	18
2.4.1 Downloading From the Electronic Software Transfer (EST) System (Download Center)	18
2.4.2 Downloading Using FTP with a Web Browser	20
2.5 Setting Up a Testbench Design Directory	20
2.6 What's Next?	20
2.6.1 Compiling CHI VIP Components within svt_amba_uvm_pkg	21
2.6.2 AXI VIP Components within CHI System Env	21
2.6.3 Licensing Information	21
2.6.4 Environment Variable and Path Settings	22
2.6.5 Determining Your Model Version	22
2.6.6 Integrating a VC VIP into Your Testbench	23
2.6.7 Include and Import Model Files into Your Testbench	29
2.6.8 Compile and Run Time Options	30

Chapter 3

General Concepts	31
3.1 Introduction to UVM	31
3.2 CHI VIP Architecture	32
3.2.1 CHI System Environment	32
3.2.2 RN Agent	32
3.2.3 SN Agent	33
3.2.4 System Monitor	34
3.2.5 CHI Interconnect	34
3.2.6 System Sequencer	34
3.3 CHI UVM User Interface	34
3.3.1 Configuration Objects	35
3.3.2 Transaction Objects	36
3.3.3 Analysis Port	37
3.3.4 Callbacks	38
3.3.5 Interfaces and Modports	38
3.3.6 Events	39
3.3.7 Overriding System Constants	39
3.3.8 Format Specification for Data, Byte Enable fields	40
3.3.9 TLM Generic Payload	43
3.3.10 Service Transactions	49
3.3.11 Delays	49
3.3.12 Reordering of Transactions	49
3.4 Reset Functionality	50
3.4.1 Dynamic Reset Support	51
3.5 Programming CHI System Address Ranges and Related Settings	51
3.5.1 Programming the HN Related Information	51
3.5.2 Programming MN Address Ranges	52
3.5.3 Programming HN Address Ranges	53
3.5.4 SN Index to HN Index Mapping	57
3.5.5 Recommended Programming Examples	57
3.6 Three SN-F striping	59
3.6.1 Overview	59
3.6.2 User Interface	59
3.6.3 SN-F Selection	61
3.6.4 Total Addressable Space	61
3.6.5 Address Aliasing	61
3.7 Connecting AXI Slaves to CHI System Monitor	62
3.7.1 Configuring AXI slaves	63
3.8 Connecting ACE-LITE Masters to CHI System Monitor	65
3.9 CHI Memory Within SN Agent	66
3.10 CHI Interface Clock Mode	66
3.10.1 Common Clock Mode	66
3.10.2 Multi-Clock Mode	67
3.11 CHI System Level Performance Metrics	68
3.11.1 Overview	68
3.11.2 User Interface	68
3.11.3 Performance Metrics	71
3.12 Interleaved Port Support	85
3.12.1 Active RN agent	85

3.12.2 Passive RN agent	85
3.12.3 Node Configuration Controls	85
3.12.4 Usage Examples	86

Chapter 4

Verification Features	87
4.1 Protocol Analyzer Support	87
4.1.1 Support for Native Dumping of FSDB	88
4.1.2 Interface Signal Grouping feature in Verdi	88
4.2 Native VERDI Performance Analyzer support	91
4.3 Built-in Performance Monitoring Unit Within CHI Agents	93
4.3.1 Measurement and Reporting of the Performance Metrics	93
4.3.2 Example Usage	95
4.4 Custom RN SAM (SYSTEM ADDRESS MAP)	97
4.5 Random Target ID Generation From RN	99
4.6 Target ID Remapping	99
4.7 Protocol Retry	100
4.8 Random SRC_ID Generation	101
4.9 Single Outstanding Request Per Address	102
4.10 Snoopable and Non-Snoopable Domains	102

Chapter 5

Verification Topologies	105
5.1 Interconnect DUT and RN or SN VIP	105
5.2 System DUT With Passive VIP	106
5.3 System DUT with Mix of Active and Passive VIP	108

Chapter 6

Protocol Features Usage And Reference	111
6.1 Transaction Ordering	111
6.1.1 User Interface	111
6.1.2 Protocol Checks	111
6.1.3 VIP Behavior	111
6.2 Outstanding Transactions	115
6.2.1 User Interface	115
6.2.2 VIP Behavior	117
6.3 Exclusive Access	119
6.3.1 User Interface	119
6.3.2 VIP Behavior	123
6.4 Target ID Remapping	125
6.4.1 User Interface	125
6.4.2 VIP Features	125

Chapter 7

Using CHI Verification IP	127
7.1 SystemVerilog UVM Example Testbenches	127
7.2 Installing and Running the Examples	128
7.2.1 Support for UVM version 1.2	129
7.3 How to Generate SN Response	129
7.4 Connecting an AXI slave to CHI Interconnect	130
7.5 Connecting an ACE-Lite Master to CHI Interconnect	131

7.6 Why the User Needs to Disable Auto Item Recording	132
7.7 Debug Features	133

Chapter 8

Using CHI B Verification IP	135
8.1 Enabling CHI Issue B	136
8.1.1 Compile Time Macros	136
8.1.2 Configuration Parameters/ APIS	136
8.1.3 Programming Example	140
8.2 Using CHI Issue B Compliant and Legacy VIP Components within Same Simulation	142
8.3 CHI-B Transactions	142
8.3.1 READNOTSHARED DIRTY	142
8.3.2 ROCI	142
8.3.3 ROMI	143
8.3.4 CleanSharedPersist	143
8.3.5 Atomic Transactions	144
8.4 DMT	144
8.5 DCT	144
8.6 Updates to Transaction Data Classes	145
8.6.1 DMT	145
8.6.2 DCT	145
8.6.3 RETTOSRC	146
8.6.4 DONOTGOTOSD	146
8.6.5 Atomic Transactions	146
8.7 Updates to the Signal Interface	147
8.7.1 Support for Fields With Variable Width	147
8.8 READNOTSHARED DIRTY	148
8.8.1 RN VIP	148
8.8.2 Interconnect VIP	148
8.9 READONCECLEANINVALID	148
8.9.1 RN VIP	148
8.9.2 Interconnect VIP	148
8.10 READONCEMAKEINVALID	148
8.10.1 RN VIP	148
8.10.2 INTERCONNECT VIP	148
8.11 CLEANSHARED PERSIST	149
8.11.1 RN VIP	149
8.11.2 Interconnect VIP	149
8.12 DMT	149
8.12.1 RN VIP	149
8.12.2 SN VIP	149
8.12.3 Interconnect VIP	150
8.13 DCT	150
8.13.1 RN VIP	150
8.14 RETTOSRC	150
8.14.1 RN VIP	150
8.15 DONOTGOTOSD	151
8.15.1 RN VIP	151
8.16 Atomic Transactions	151
8.16.1 RN VIP	151

8.16.2 Support for Atomic Transactions at SN-F VIP	151
8.17 Checks for the New Read Transactions	154
8.17.1 READNOTSHAREDDIRTY	154
8.17.2 READONCECLEANINVALID	154
8.17.3 READONCEMAKEINVALID	155
8.17.4 CLEANSHAREDPERSIST	155
8.18 Checks for Variable Width	156
8.18.1 Variable Address Width Support	156
8.18.2 Variable NODEID Width Support	156
8.19 Checks for DMT	156
8.20 Checks for DCT	157
8.21 Checks for RETTOSRC	157
8.22 Checks for DONOTGOTOSD	158
8.23 Checks for Atomic Transactions	158
8.24 Sequence Collection Updates	159
8.24.1 System Virtual Sequences	159
Chapter 9	
Using CHI C Verification IP	161
9.1 Enabling CHI-C Mode	161
9.1.1 User Interface	161
9.1.2 Use Model Details	162
9.2 CHI Issue C Features	162
9.2.1 Separate READ Data and COMP Response	162
9.2.2 Response After Receiving First Data Packet	164
9.2.3 Error Handling	165
9.2.4 Combined CompAck and WriteData	165
Chapter 10	
Using CHI D Verification IP	167
10.1 Overview of CHI D	167
10.1.1 User Interface	167
10.2 Features Supported for CHI Issue D	167
10.2.1 C Busy Feature	168
10.2.2 Increased TXN ID Width	170
10.2.3 MPAM Feature	170
10.2.4 Ordered Write Observation Flow Enhancements	172
10.2.5 CHI Issue A, B and C: Clarifications and Errata	175
10.2.6 Persistent CMO with Two-part Response	177
10.3 Unsupported Features	180
Chapter 11	
Using CHI E Verification IP	181
11.1 Overview of CHI E	181
11.1.1 User Interface	181
11.2 Features Supported for CHI Issue E	181
11.2.1 MakeReadUnique Transaction	182
11.2.2 Writes with Optional Data	183
11.2.3 Non-Forwarding of Data from SC state	184
11.2.4 Write Zero with No Data	184

11.2.5 SnpQuery Snoop Request	185
11.2.6 Exclusive Read Transactions	185
11.2.7 Combined Write and (P)CMO Transaction	186
11.2.8 Extending TxnID Width	189
11.2.9 Memory Tagging	189
11.2.10 DVM Updates	191
11.2.11 Handling of Response with NDERR	192
11.2.12 Two-part StashOnce Transaction	193
11.2.13 DBIDRespOrd Response	194
11.2.14 Direct Write-data Transfer (DWT)	195
11.2.15 SLC Replacement Hint	196
11.2.16 Replicated Channels in a Single Interface	197
11.2.17 Miscellaneous Updates	209
11.3 CHI E Protocol Checks	210
11.4 CHI E Functional Coverage Report	210
11.5 CHI E Unsupported Features	210
Appendix A	
Unsupported Features and Limitations	211
A.1 Unsupported Features	211
A.2 Known Issues and Limitations	212
Appendix B	
Reporting Problems	213
B.1 Introduction	213
B.2 Initial Customer Information	213

Preface

About This Guide

This guide contains the usage information related to AMBA CHI VIP compliant to 'ARM® AMBA® 5 CHI Architecture Specification - Issue A (ARM IHI 0050A)'.

For the information related to VIP usage and support for the new CHI Issue B features as specified in 'AMBA CHI VIP compliant to 'ARM® AMBA® 5 CHI Architecture Specification - Issue B (ARM IHI 0050B)', see CHI-B UVM User Guide (chi_b_svt_uvm_user_guide.pdf).

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

Web Resources

- ❖ Documentation through SolvNetPlus: <https://solvnetplus.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:

1. Go to <https://solvnetplus.synopsys.com> and open a case.
Enter the information according to your environment and your issue.
2. Send an e-mail message to support_center@synopsys.com.
Include the Product name, Sub Product name, and Tool Version in your e-mail so it can be routed correctly.
3. 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:
<https://www.synopsys.com/support/global-support-centers.html>

Synopsys Statement on Inclusivity and Diversity

Synopsys is committed to creating an inclusive environment where every employee, customer, and partner feels welcomed. We are reviewing and removing exclusionary language from our products and supporting customer-facing collateral. Our effort also includes internal initiatives to remove biased language from our engineering and working environment, including terms that are embedded in our software and IPs. At the same time, we are working to ensure that our web content and software applications are usable to people of varying abilities. You may still find examples of non-inclusive language in our software or documentation as our IPs implement industry-standard specifications that are currently under review to remove exclusionary language.

1

Introduction

This chapter gives an overview of CHI Verification IP and lists the supported features of the CHI UVM Verification IP. This chapter encompasses the following sections:

- ❖ [Introduction](#)
- ❖ [Prerequisites](#)
- ❖ [References](#)
- ❖ [Product Overview](#)
- ❖ [Language and Methodology Support](#)
- ❖ [Features Supported](#)

**Note**

Based on the AMBA Progressive Terminology updates, you must interpret the term Master as Manager and Slave as Subordinate in the VIP documentation and messages.

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 Universal Verification Methodology (UVM). 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 maximum reuse, scalability and modularity
- ❖ Proven verification approach and methodology
- ❖ Transaction-level models
- ❖ Self-checking tests
- ❖ Object oriented interface that allows OOP techniques

**Note**

CHI VIP supports Issue A, Issue B, and Issue C specifications. This document refers to the features specific to these revisions as CHI-A, CHI-B, CHI-C respectively. The features that are common across all revisions are referred as CHI.

1.2 Prerequisites

- ❖ Familiarize with CHI, object oriented programming, SystemVerilog, and the current version of UVM.

1.3 References

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

- ❖ Class Reference for VC Verification IP for AMBA® CHI (Issue A) is available at:
[\\$DESIGNWARE_HOME/vip/svt/amba_svt/latest/doc/class_ref/chi_svt_uvm_class_reference/html/index.html](#)
- ❖ Class Reference for VC Verification IP for AMBA® CHI (Issue B) is available at:
[\\$DESIGNWARE_HOME/vip/svt/amba_svt/latest/doc/class_ref/chi_b_svt_uvm_class_reference/html/index.html](#)
- ❖ Class Reference for VC Verification IP for AMBA® CHI (Issue C) is available at:
[\\$DESIGNWARE_HOME/vip/svt/amba_svt/latest/doc/class_ref/chi_c_svt_uvm_class_reference/html/index.html](#)
- ❖ Class Reference for VC Verification IP for AMBA® CHI (Issue D) is available at:
[\\$DESIGNWARE_HOME/vip/svt/amba_svt/latest/doc/class_ref/chi_d_svt_uvm_class_reference/html/index.html](#)
- ❖ Class Reference for VC Verification IP for AMBA® CHI (Issue E) is available at:
[\\$DESIGNWARE_HOME/vip/svt/amba_svt/latest/doc/class_ref/chi_e_svt_uvm_class_reference/html/index.html](#)

1.4 Product Overview

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

The VIP provides a CHI System Environment (Env) which comprises of the following agents and monitor:

- ❖ Request Node (RN) agents
- ❖ Slave Node (SN) agents
- ❖ CHI Interconnect Env
- ❖ CHI System Monitor

The RN and SN agents support all the functionality normally associated with active and passive UVM 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 and SN agents in the testbench.

For more information on AMBA System Env, see AMBA System UVM User guide.

1.5 Language and Methodology Support

Synopsys CHI VIP is qualified with the following languages and methodology:

- ❖ Languages
 - ◆ SystemVerilog
- ❖ Methodology
 - ◆ Qualified with UVM 1.1d and 1.2, UVM 1800.2-2017-1.0, and UVM 1800.2-2017-1.1.

1.6 Features Supported

1.6.1 Protocol Features

CHI VIP supports the following features:

- ❖ Support for RN-F, RN-I and SN-F node types
- ❖ Support for CHI Interconnect
- ❖ Support for ReadNoSnp, WriteNoSnpPtl, and WriteNoSnpFull transaction types
- ❖ Read-type transactions
- ❖ CopyBack transactions
- ❖ Write-type transactions
- ❖ Byte Enables and data transfer sizes
- ❖ Cache Maintenance transactions
- ❖ Barrier transactions
- ❖ DVM transactions
- ❖ Exclusive Accesses
- ❖ Protocol credit return
- ❖ Snoop request and response messages
- ❖ Cache state transitions at requesting RN-F
- ❖ Transaction ID (TxnID) and Data buffer ID (DBID)
- ❖ Request Retry
- ❖ Request Ordering
- ❖ Protocol and Link Flits
- ❖ Virtual channels
 - ◆ Request VC
 - ◆ Response VC
 - ◆ Snoop VC
 - ◆ Data VC
- ❖ Link layer Flow control
- ❖ Link activation or de-activation
- ❖ LINKACTIVE State Machine

- ❖ ACTIVE signal
- ❖ Data beat ordering
- ❖ CHI memory within SN agent

CHI VIP additionally supports the following features from Issue B specification:

- ❖ ReadNotSharedDirty
- ❖ ReadOnceCleanInvalid (ROCI)
- ❖ ReadOnceMakeInvalid (ROMI)
- ❖ CleanSharedPersist
- ❖ Atomic transactions
- ❖ Cache Stashing Transactions
- ❖ LPID, PcrdType, SnpAttr updated widths are supported.
 - ◆ Variable Node ID field width, as defined in the CHI-B specification, is supported. All *ID field widths in transactions data objects and FLIT interface signal fields support the same value.
- ❖ Variable Addr field width, as defined in the CHI-B specification, is supported in requests, and snoops.
- ❖ Variable widths are supported for Poison, DataCheck fields
- ❖ Direct Memory transfer (DMT)
- ❖ Direct Cache transfer (DCT)
- ❖ RetToSrc
- ❖ SharedClean State return
- ❖ DoNotGoToSD

CHI Issue C supports the following specification features :

- ❖ Separate Read Data and Comp Response (for Data sent to the Requester from Home or Slave).
 - ◆ CompAck generation and monitoring from RN agent before the reception of DataSepResp DAT flits.
- ❖ Response after receiving first Data packet.
 - ◆ Completion acknowledge response, CompAck, after receiving first read data packet, CompData or DataSepResp.
 - ◆ Snoop response with data, SnpRespData, must wait for receiving of all read data packets, CompData.
- ❖ Combined CompAck with WriteData.
- ❖ Error Handling

CHI Issue D supports the following specification features:

- ❖ C Busy
- ❖ Increased TXN ID Width

- ❖ Memory system Performance resource Partitioning and Monitoring (MPAM)
- ❖ Ordered Write Observation Flow Enhancements
- ❖ CHI Issue A, B and C: Clarifications and Errata
- ❖ Persistent CMO with Two-part Response

CHI Issue E supports the following specification features:

- ❖ MakeReadUnique Transaction
- ❖ Write Transactions with Optional Data
- ❖ Non-Forwarding of Data from SC state
- ❖ Write Zero Transactions with No Data
- ❖ SnpQuery Snoop Request
- ❖ Exclusive Read Transaction updates
- ❖ Combined Write and (P)CMO Transactions
- ❖ Extending TxnID Width
- ❖ Group ID Extension Field
- ❖ Memory Tagging
- ❖ DVM Updates
- ❖ Handling of Response with NDERR
- ❖ Two-part StashOnce Transaction
- ❖ DBIDRespOrd Response
- ❖ Direct Write-data Transfer
- ❖ SLC Replacement Hint
- ❖ Miscellaneous Updates

1.6.2 Verification Features

CHI VIP currently supports the following verification functions:

- ❖ Functional coverage
- ❖ Protocol Check coverage
- ❖ Verdi Protocol Analyzer
- ❖ Verdi Performance Analyzer
- ❖ Sequence collection
- ❖ Port-level checker
- ❖ System-level checker

1.6.3 Methodology Features

CHI VIP currently supports the following methodology functions:

- ❖ Active and Passive mode of RN and SN agents
- ❖ RN and SN agents support analysis ports in active and passive mode

- ❖ Bind Interface
- ❖ Callbacks
- ❖ Analysis Port
- ❖ TLM Generic Payload support

2

Installation and Setup

This chapter leads you through installing and setting up the AMBA CHI VIP. When you complete this checklist, the provided example testbench is operational and the AMBA CHI VIP is 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?”](#)

**Note**

If you encounter any problems with installing the AMBA CHI VIP, contact Synopsys Customer Support.

2.1 Verifying the Hardware Requirements

The AXI Verification IP requires a Solaris or Linux workstation configured as follows:

- ❖ 400 MB available disk space for installation
- ❖ 16 GB Virtual Memory recommended

2.2 Verifying Software Requirements

The AMBA CHI VIP is qualified for use with O-2018.12-1 version of platforms and simulators. This section lists software that the AMBA CHI requires.

2.2.1 Platform/OS and Simulator Software

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

2.2.2 Synopsys Common Licensing (SCL) Software

- ❖ The SCL software provides the licensing function for the AMBA CHI VIP. Acquiring the SCL software is covered here in the installation instructions in [Licensing Information](#).

2.2.3 Other Third Party Software

- ❖ **Adobe Acrobat:** AMBA CHI VIP documents are available in Acrobat PDF files. You can get Adobe Acrobat Reader for free from <http://www.adobe.com>.
- ❖ **HTML browser:** AMBA 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 the `DESIGNWARE_HOME` to the following absolute path where the Synopsys CHI VIP is installed:

```
setenv DESIGNWARE_HOME absolute_path_to_designware_home
```
2. Set your environment and `PATH` variables correctly, including the following:
 - ◆ `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 SolvNetPlus password is unknown or forgotten, go to <http://SolvNetPlus.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:

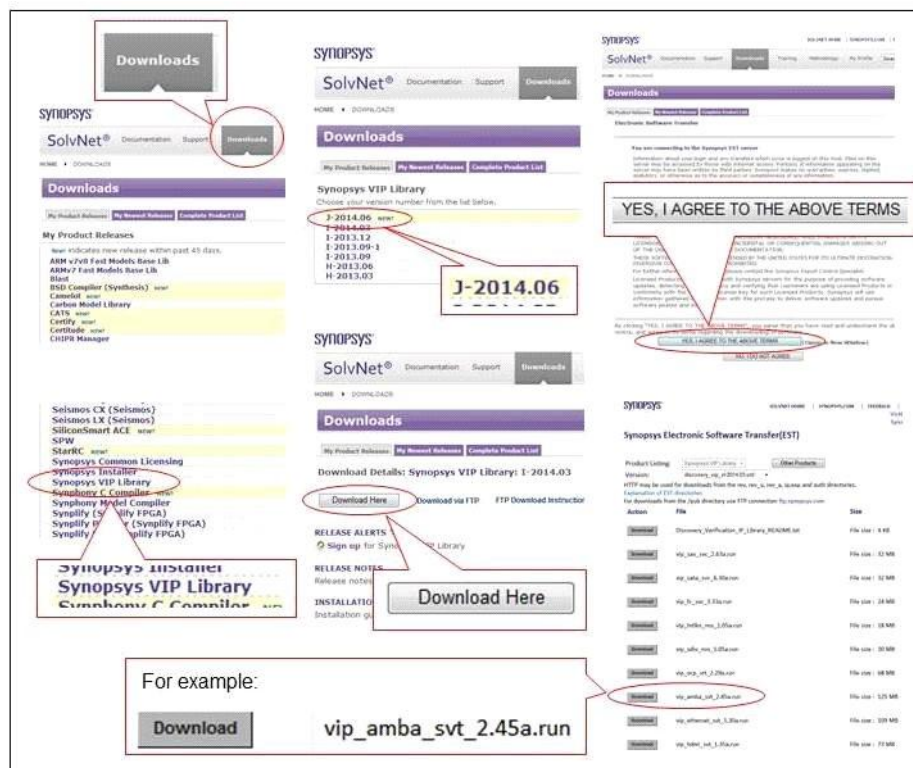
https://www.synopsys.com/apps/protected/support/EST-FTP_Accelerator_Help_Page.html

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

- a. Point your web browser to <http://SolvNetPlus.synopsys.com>.
- b. Enter your Synopsys SolvNetPlus Username and Password.

- c. Click Sign In button.
- d. Make the following selections on SolvNetPlus to download the .run file of the VIP (See Figure 2-1).
 - i. Downloads tab
 - ii. Synopsys VIP Library product releases
 - iii. <release_version>
 - iv. Download Here button
 - v. Yes, I Agree to the Above Terms button
 - vi. Download .run file for the VIP

Figure 2-1 SolvNetPlus Selections for VIP Download



- e. Set the DESIGNWARE_HOME environment variable to a path where you want to install the VIP.


```
% setenv DESIGNWARE_HOME VIP_installation_path
```
- f. Execute the .run file by invoking its filename. The VIP is unpacked and all files and directories are installed under the path specified by the DESIGNWARE_HOME environment variable. The .run file can be executed from any directory. The important step is to set the DESIGNWARE_HOME environment variable before executing the .run file.



Note

The Synopsys AMBA VIP suite includes VIP models for all AMBA interfaces (AHB, APB, AXI, CHI and ATB). You must download the VC VIP for AMBA suite to access the VIP models for AHB, APB, AXI, CHI and ATB.

2.4.2 Downloading Using FTP with a Web Browser

- Follow the above instructions through the product version selection step.
- Click the Download via FTP link instead of the Download Here button.
- Click the Click Here To Download button.
- Select the file(s) that you want to download.
- Follow browser prompts to select a destination location.

**Note**

If you are unable to download the Verification IP using above instructions, contact customer support 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 more information on `dw_vip_setup`, see [The `dw_vip_setup` Utility](#).

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_env_svt -svtb
```

The models provided with CHI VIP include:

- ❖ `chi_system_env_svt`
- ❖ `chi_rn_agent_svt`
- ❖ `chi_sn_agent_svt`

**Note**

UVM users are required to define the UVM macro `UVM_DISABLE_AUTO_ITEM_RECORDING`. CHI being a pipelined protocol (that is, previous transaction does not necessarily need to complete before starting new transaction), CHI VIP handles triggering the begin or end events and transaction recording.

**Note**

CHI VIP does not use the UVM automatic transaction begin or end event triggering and recording feature. If `UVM_DISABLE_AUTO_ITEM_RECORDING` is not defined, VIP issues a FATAL message.

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:

- ❖ [“Compiling CHI VIP Components within `svt_amba_uvm_pkg`”](#)

- ❖ “AXI VIP Components within CHI System Env”
- ❖ “Licensing Information”
- ❖ “Environment Variable and Path Settings”
- ❖ “Determining Your Model Version”
- ❖ “Integrating a VC VIP into Your Testbench”

2.6.1 Compiling CHI VIP Components within svt_amba_uvm_pkg

The 'svt_amba_uvm_pkg' (included through the file svt_amba.uvm.pkg) does not include the AMBA CHI VIP components by default. This also implies that the top-level VIP component AMBA System Env 'svt_amba_system_env' doesn't include CHI System Env 'svt_chi_system_env' and related components.

On the same lines, 'svt_amba_system_configuration' doesn't include any member attributes and APIs related to CHI VIP components.

To use CHI VIP components with 'svt_amba_uvm_pkg' and AMBA System Env 'svt_amba_system_env', following compile time macro needs to be defined: ``SVT_AMBA_INCLUDE_CHI_IN_AMBA_SYS_ENV`

So, any existing test bench that uses AMBA System Env with CHI VIP components, must define this compile macro to compile and run the test bench.

2.6.2 AXI VIP Components within CHI System Env

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 issues 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 prototype 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 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 be as follows:

```
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;
```

2.6.3 Licensing Information

The CHI UVM VIP uses the Synopsys Common Licensing (SCL) software to control its usage.

You can find general SCL information in the following location:

<http://www.synopsys.com/keys>

For more information on the order in which licenses are checked out for each VIP, refer to VC VIP AMBA Release Notes.

The licensing key must reside in files that are indicated by specific environment variables. For information about setting these licensing environment variables, see *Environment Variable and Path Settings* section.

2.6.3.0.1 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.

2.6.3.0.2 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.

2.6.4 Environment Variable and Path Settings

The following are environment variables and path settings required by the CHI UVM VIP verification models:

- ❖ `DESIGNWARE_HOME` – The absolute path to where the VIP is installed.
- ❖ `DW_LICENSE_FILE` – The absolute path to a file that contains the license keys for VIP product software or the `port@host` reference to this file.
- ❖ `SNPSLMD_LICENSE_FILE` – The absolute path to a file that contains the license keys for Synopsys 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.

2.6.4.1 Simulator-Specific Settings

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

2.6.5 Determining Your Model Version

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



Note

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
```

2.6.6 Integrating a VC VIP into Your Testbench

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

- ❖ [“Creating a Testbench Design Directory”](#)
- ❖ [“Setting Up a New VIP”](#)
- ❖ [“Installing and Setting Up More than One VIP Protocol Suite”](#)
- ❖ [“Updating an Existing Model”](#)
- ❖ [“Removing Synopsys VIP Models from a Design Directory”](#)
- ❖ [“Reporting Information About DESIGNWARE_HOME or a Design Directory”](#)
- ❖ [“The dw_vip_setup Utility”](#)

2.6.6.1 Creating a Testbench Design Directory

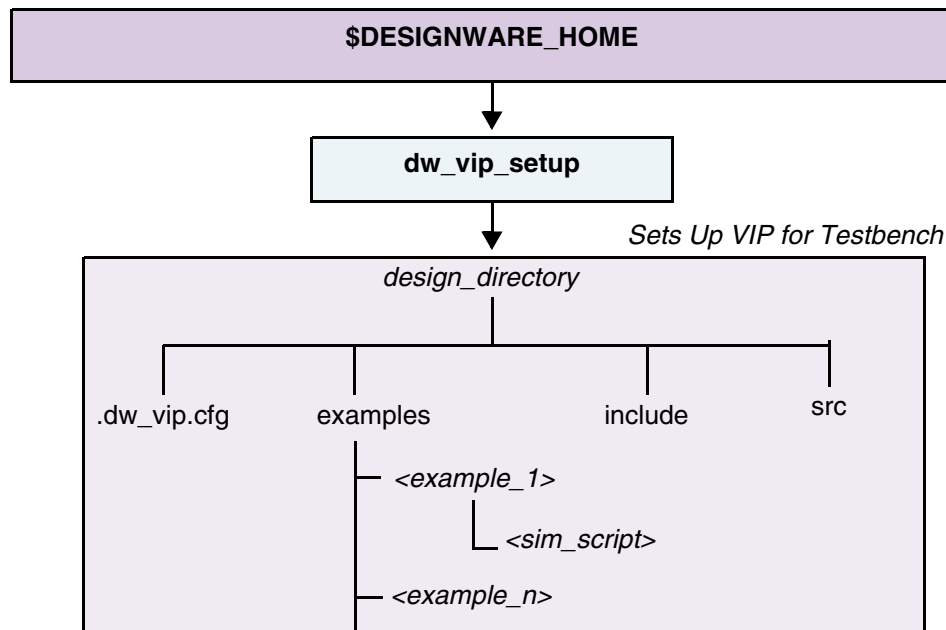
A *design directory* contains a version of the VIP that is set up and ready for use in a testbench. You use the `dw_vip_setup` utility to create design directories. For the full description of `dw_vip_setup`, refer to [The dw_vip_setup Utility](#).



Note

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 VIP in your testbench because it is isolated from the DESIGNWARE_HOME installation. When you want, you can use `dw_vip_setup` to update VIP in your design directory. [Figure 2-2](#) shows this process and the contents of a design directory.

Figure 2-2 Design Directory Created by 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 VIP). 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.

**Note**

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

2.6.6.2 Setting Up a New VIP

After you have installed the VIP, you must set up the VIP for project and testbench use. All VIP suites contain various components such as transceivers, masters, slaves, and monitors depending on the protocol. The setup process gathers together all the required component files you need to incorporate into your testbench required for simulation runs.

You have the choice to set up all of them, or only specific ones. For example, the CHI VIP contains the following components.

- ❖ chi_system_env_svt
- ❖ chi_rn_agent_svt
- ❖ chi_sn_agent_svt

You can set up either an individual component, or the entire set of components within one protocol suite. Use the Synopsys provided tool called `dw_vip_setup` for these tasks. It resides in `$DESIGNWARE_HOME/bin`.

To get help on `dw_vip_setup`, invoke the following:

```
% $DESIGNWARE_HOME/bin/dw_vip_setup --help
```

The following command adds a model `<model_svt>` to the directory `design_dir`.

```
% $DESIGNWARE_HOME/bin/dw_vip_setup -path /tmp/design_dir -add chi_system_env_svt -svlog
```

This command sets up all the required files in `/tmp/design_dir`.

The utility `dw_vip_setup` creates three directories under `design_dir` which contain all the necessary model files. Files for every VIP are included in these three directories.

- ❖ **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 "include/sverilog" is specified in simulator commands to locate model files.
- ❖ **src:** Synopsys-specific include files This directory "src/sverilog/vcs" must be included in the simulator command to locate model files.

Note that some components are “top level” and will setup the entire suite. You have the choice to set up the entire suite, or just one component such as a monitor.



Attention

There *must* be only one `design_dir` installation per simulation, regardless of the number of Synopsys Verification and Implementation IPs you have in your project. Do create this directory in `$DESIGNWARE_HOME`.

2.6.6.3 Installing and Setting Up More than One VIP Protocol Suite

All VIPs for a particular project must be set up in a single common directory once you execute the `*.run` file. You may have different projects. In this case, the projects can use their own VIP setup directory. However, all the VIPs used by that specific project must reside in a common directory.

The examples in this chapter call that directory as `design_dir`, but you can use any name.

In this example, assume you have the CHI suite set up in the `design_dir` directory. In addition to the CHI VIP, you require the Ethernet and USB VIP suites.

First, follow the previous instructions on downloading and installing the Ethernet VIP and USB suites.

Once installed, the Ethernet and USB suites must be set up in and located in the same `design_dir` location as AMBA. Use the following commands:

```
// First install CHI
%unix> $DESIGNWARE_HOME/bin/dw_vip_setup -path /tmp/design_dir
-add chi_system_env_svt -svlog

//Add Ethernet to the same design_dir as CHI
%unix> $DESIGNWARE_HOME/bin/dw_vip_setup -path /tmp/design_dir
-add ethernet_system_env_svt -svlog

// Add USB to the same design_dir as AMBA and Ethernet
%unix> $DESIGNWARE_HOME/bin/dw_vip_setup -path /tmp/design_dir
-add usb_system_env_svt -svlog
```

To specify other model names, consult the VIP documentation.

By default, all of the VIPs use the latest installed version of SVT. Synopsys maintains backward compatibility with previous versions of SVT. As a result, you may mix and match models using previous versions of SVT.

2.6.6.4 Updating an Existing Model

To add and update an existing model, do the following:

1. Install the model to the same location at which your other VIPs are present by setting the \$DESIGNWARE_HOME environment variable.
2. Issue the following command using `design_dir` as the location for your project directory.

```
%unix> $DESIGNWARE_HOME/bin/dw_vip_setup -path /tmp/design_dir
-add chi_rn_agent_svt
```

3. You can also update your `design_dir` by specifying the version number of the model.

```
%unix> dw_vip_setup -path design_dir -add chi_rn_agent_svt -v 3.50a -svlog
```

2.6.6.5 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.

2.6.6.6 Reporting Information About DESIGNWARE_HOME or a Design Directory

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

The following example lists the 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
```

2.6.6.7 The dw_vip_setup Utility

The `dw_vip_setup` utility:

- ❖ Adds, removes, or updates the VIP models in a design directory
- ❖ Adds example testbenches to a design directory, the 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

2.6.6.7.1 Setting Environment Variables

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

- ❖ `DESIGNWARE_HOME` – Points to where the VIP is installed

2.6.6.7.2 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] (<i>model</i> [-v[ersion] <i>version</i>]) ...	<p>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:</p> <ul style="list-style-type: none"> • <code>chi_system_env_svt</code> • <code>chi_rn_agent_svt</code> • <code>chi_sn_agent_svt</code> <p>The <code>-add</code> switch causes <code>dw_vip_setup</code> to build suite libraries from the same suite as the specified models, and to copy the other necessary files from <code>\$DESIGNWARE_HOME</code>.</p>
-r[emove] <i>model</i>	<p>Removes all versions of the specified model or models from the design. The <code>dw_vip_setup</code> program does not attempt to remove any include files used solely by the specified model or models. The model names are:</p> <ul style="list-style-type: none"> • <code>chi_system_env_svt</code> • <code>chi_rn_agent_svt</code> • <code>chi_sn_agent_svt</code>
-u[pdate] (<i>model</i> [-v[ersion] <i>version</i>]) ...	<p>Updates to the specified model version for the specified model or models. The <code>dw_vip_setup</code> script updates to the latest models when you do not specify a version. The model names are:</p> <ul style="list-style-type: none"> • <code>chi_system_env_svt</code> • <code>chi_rn_agent_svt</code> • <code>chi_sn_agent_svt</code> <p>The <code>-update</code> switch causes <code>dw_vip_setup</code> to build suite libraries from the same suite as the specified models, and to copy the other necessary files from <code>\$DESIGNWARE_HOME</code>.</p>

Table 2-1 Setup Program Switch Descriptions (Continued)

Switch	Description
-e [example] { <i>scenario</i> <i>model/scenario</i> } [-v[ersion] <i>version</i>]	<p>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.</p> <p>If you specify a <i>scenario</i> (or system) example testbench, the models needed for the testbench are included automatically and do not need to be specified in the command.</p> <p>Note: Use the -info switch to list all available system examples.</p>
-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] { <i>scenario</i> <i>model/scenario</i> }	Cleans the specified scenario/testbench in either the design directory (as specified by the -path switch) or the current working directory. This switch deletes <i>all files in the specified directory</i> , then restores all Synopsys created files to their original contents.
-i [nfo] <i>design</i> <i>home</i>	<p>When you specify the -info <i>design</i> 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.</p> <p>When you specify the -info <i>home</i> 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.</p> <p>The reports are printed to STDOUT.</p>
-h [elp]	Returns a list of valid dw_vip_setup switches and the correct syntax for each.
<i>model</i>	<p>AMBA CHI VIP models are:</p> <ul style="list-style-type: none"> • chi_system_env_svt • chi_rn_agent_svt • chi_sn_agent_svt <p>The <i>model</i> 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 <i>model</i> argument may also use a model list.</p> <p>You may specify a version for each listed <i>model</i>, using the -version option. If omitted, dw_vip_setup uses the latest version. The -update switch ignores <i>model</i> version information.</p>
-m [odel_list] <i>filename</i>	<p>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 <i>model</i> argument, can contain model version information. Each line in the file contains:</p> <p><i>model_name</i> [-v <i>version</i>] –or– # Comments</p>

Table 2-1 Setup Program Switch Descriptions (Continued)

Switch	Description
-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 ‘System Verilog VMM Example Testbenches’ and ‘Installing and Running the Examples’ sections.

2.6.7 Include and Import Model Files into Your Testbench

After you set up the models, you must include and import various files into your top testbench files to use the VIP. For more details, see *Compiling CHI VIP components within svt_amba_uvm_pkg* section.

Following is a code list of the includes and imports for components within *amba_system_env_svt*:

```
/* include uvm package before VIP includes, If not included elsewhere*/
`include "uvm_pkg.sv"

/* include AXI , AHB, CHI and APB VIP interface */
`include "svt_ahb_if.svi"
`include "svt_axi_if.svi"
`include "svt_apb_if.svi"
`include "svt_chi_if.svi"

/** Include the AMBA SVT UVM package */
`include "svt_amba.uvm.pkg"

/** Import UVM Package */
import uvm_pkg::*;

/** Import the SVT UVM Package */
import svt_uvm_pkg::*;

/** Import the AMBA VIP */
import svt_amba_uvm_pkg::*;
```

You must also include various VIP directories on the simulator command line. Add the following switches and directories to all compile scripts:

- ❖ +incdir+<design_dir>/include/verilog
- ❖ +incdir+<design_dir>/include/sverilog
- ❖ +incdir+<design_dir>/src/verilog/<vendor>
- ❖ +incdir+<design_dir>/src/sverilog/<vendor>

Supported vendors are VCS, MTI and NCV. For example:

```
+incdir+<design_dir>/src/sverilog/vcs
```

Using the previous examples, the directory `<design_dir>` would be `/tmp/design_dir`.

2.6.8 Compile and Run Time Options

Every Synopsys provided example has ASCII files containing compile and run time options. The examples for the model are located in:

```
$DESIGNWARE_HOME/vip/svt/<model>/latest/examples/sverilog/<example_name>
```

The files containing the options are:

- ❖ `sim_build_options` (contain compile time options common for all simulators)
- ❖ `sim_run_options` (contain run time options common for all simulators)
- ❖ `vcs_build_options` (contain compile time options for VCS)
- ❖ `vcs_run_options` (contain run time options for VCS)
- ❖ `mti_build_options` (contain compile time options for MTI)
- ❖ `mti_run_options` (contain run time options for MTI)
- ❖ `ncv_build_options` (contain compile time options for IUS)
- ❖ `ncv_run_options` (contain run time options for IUS)

These files contain both optional and required switches. For AXI VIP, following are the contents of each file, listing optional and required switches:

`vcs_build_options`

For more details, see *Compiling CHI VIP components within svt_amba_uvm_pkg* section.

Required: `+define+UVM_DISABLE_AUTO_ITEM_RECORDING`

Optional: `-timescale=1ns/1ps`

Required: `+define+SVT_<model>_INCLUDE_USER_DEFINES`

Required: `+define+`SVT_AMBA_INCLUDE_CHI_IN_AMBA_SYS_ENV`



Note

AMBA SVT VIP implementation does not depend on the macro `UVM_PACKER_MAX_BYTES`. However, if UVM pack or unpack operation needs to be performed on the transaction handle in your testbench, then `UVM_PACKER_MAX_BYTES` macro needs to be defined and set to an optimal value in your testbench. For example, if VIP title 1 needs `UVM_PACKER_MAX_BYTES` to be set to 8192, and VIP title 2 needs `UVM_PACKER_MAX_BYTES` to be set to 500000, you need to set `UVM_PACKER_MAX_BYTES` to 500000.

`vcs_run_options`

- Required: `+UVM_TESTNAME=$scenario`



Note

The “scenario” is the UVM test name you pass to VCS.

3

General Concepts

This chapter describes the usage of CHI VIP in an UVM environment, and it is an user interface. This chapter discusses the following topics:

- ❖ [Introduction to UVM](#)
- ❖ [CHI VIP Architecture](#)
- ❖ [CHI UVM User Interface](#)
- ❖ [Reset Functionality](#)
- ❖ [Programming CHI System Address Ranges and Related Settings](#)
- ❖ [Three SN-F striping](#)
- ❖ [Connecting AXI Slaves to CHI System Monitor](#)
- ❖ [Connecting ACE-LITE Masters to CHI System Monitor](#)
- ❖ [CHI Memory Within SN Agent](#)
- ❖ [CHI Interface Clock Mode](#)
- ❖ [CHI System Level Performance Metrics](#)
- ❖ [Interleaved Port Support](#)

3.1 Introduction to UVM

UVM 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 UVM environment, and its user interface.

**Note**

For description of attributes and properties of the objects mentioned in this chapter, see the Class Reference HTML.

This chapter assumes that you are familiar with SystemVerilog and UVM. For more information see the following:

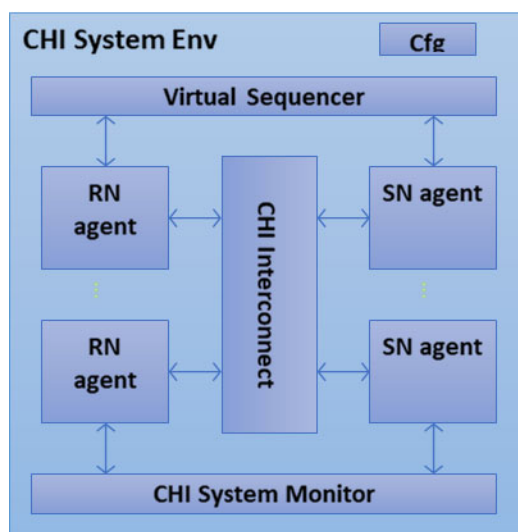
- ❖ For the IEEE SystemVerilog standard, see:
IEEE Standard for SystemVerilog – Unified Hardware Design, Specification, and Verification Language

- ❖ For an essential reference guide describing UVM as it is represented in SystemVerilog, along with a class reference, see:

Universal Verification Methodology (UVM) 1.0 User's Manual at:
<http://www.accellera.org/>.

3.2 CHI VIP Architecture

Figure 3-1 CHI VIP Architecture



3.2.1 CHI System Environment

The CHI System Env encapsulates the RN agents, SN Agents, Interconnect Env, System Monitor, and the CHI system configuration. CHI System Env also contains a virtual sequencer that contains references to all the sequencers within the contained agents. The number of configured RN and SN agents is based on the system configuration provided by you. In the build phase, the System Env builds the RN and SN agents. After the agents are built, they are configured by System Env by using the CHI Node configuration and information available in the system configuration.

3.2.2 RN Agent

The RN Agent encapsulates RN Transaction Sequencer, RN Protocol Driver, RN Protocol Monitor, TX Flit Sequencers, RN Link Driver, and RN Link Monitor. The RN Agent also contains a virtual sequencer that contains references to all of the non-virtual sequencer in this agent. The RN Agent can be configured to operate in active mode and passive mode. You can provide CHI sequences to the RN Sequencer.

The RN Agent is configured using a node configuration, which is available in the system configuration. The node configuration should be provided to the RN Agent 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 Agent, the RN Protocol driver gets sequences 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.

3.2.3 SN Agent

The SN Agent encapsulates SN Transaction Sequencer, SN Protocol Driver, SN Protocol Monitor, TX Flit Sequencers, SN Link Driver and SN Link Monitor. The SN Agent also contains a virtual sequencer that contains references to all of the non-virtual sequencer in this agent. The SN Agent can be configured to operate in active mode and passive mode. You can provide CHI response sequences to the SN Sequencer.

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

In the SN Agent, 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 Sequencer through `response_request_port`. The SN response sequence 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 agent 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.

3.2.3.1 Active and Passive Mode

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

Table 3-1 Agents 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.

Table 3-1 Agents in Active and Passive Mode (Continued)

Component behavior in active mode	Component behavior in passive mode
The Monitor components within the agent perform protocol checks only on sampled signals. That is, it does not perform checks on the signals that are driven by the agent. This is because when the agent 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 agent 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 agents 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 agents are the sampled delay values on the bus.

3.2.4 System Monitor

The System Monitor component is instantiated within the CHI System Env component. The System Monitor performs system-level checks across the CHI RN and SN ports within the system. The system monitor is enabled by setting the system configuration class member

```
svt_chi_system_configuration::system_monitor_enable.
```

For the list of system checks, see the CHI VIP Class reference HTML documentation.

3.2.5 CHI Interconnect

The CHI interconnect connects to RNs and SNs and routes transactions correctly. The configuration property `svt_chi_system_configuration::use_interconnect` must be set to use the interconnect. The configuration of the interconnect is specified in the property `ic_cfg` of the `svt_chi_system_configuration`. For a list of members that needs to be configured, see the documentation of `svt_chi_interconnect_configuration`. The RNs or SNs and the corresponding ports of the interconnect must be configured with compatible configuration properties. For more information on the procedure, see `tb_chi_svt_uvm_intermediate_sys` example. The `svt_chi_if` has sub-interfaces of type `ic_rn_if`, which connect to RNs and `ic_sn_if` which connect to SNs. These interfaces of the interconnect must be hooked up to corresponding RNs and SNs. For more information on the procedure, see `tb_chi_svt_uvm_intermediate_sys` example in the `env/vip_interconnect_sv_wrapper.sv` file.

3.2.6 System Sequencer

CHI System sequencer is a virtual sequencer within CHI System Env with references to virtual sequencers within RN and SN agents. The System Sequencer is created in the build phase of the System Env. The system configuration is provided to the System Sequencer. The System Sequencer can be used to synchronize between the sequencers in RN and SN Agents.

3.3 CHI UVM User Interface

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

- ❖ [Configuration Objects](#)
- ❖ [Transaction Objects](#)

- ❖ [Analysis Port](#)
- ❖ [Callbacks](#)
- ❖ [Interfaces and Modports](#)
- ❖ [Events](#)
- ❖ [Overriding System Constants](#)
- ❖ [Format Specification for Data, Byte Enable fields](#)
- ❖ [TLM Generic Payload](#)
- ❖ [Service Transactions](#)
- ❖ [Delays](#)
- ❖ [Reordering of Transactions](#)

3.3.1 Configuration Objects

Configuration data objects convey the system level and node level testbench configuration. The configuration of agents is done in the `build()` phase of environment or the testcase. For future modifications use `reconfigure()` method of the RN Agent, SN Agent, or System Env.

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. For example, 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. For example, a dynamic configuration parameter is a timeout value.

The configuration data objects contain built-in constraints, which are effective 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 env from the environment or the testcase. The system configuration mainly specifies:
 - ❖ Number of RN and SN agents
 - ❖ Node configurations for RN and SN agents
 - ❖ Virtual top level CHI interface
 - ❖ Address map
 - ❖ Timeout values
 - ❖ Number of HNs
 - ❖ HN index to node ID mapping for each of the HNs

- ❖ HN index to HN type mapping for each of the HNs
- ❖ HN to SN mapping
- ❖ MN node ID

Node configuration (`svt_chi_node_configuration`)

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

- ❖ Active or Passive mode of the RN or SN port agent
- ❖ Enable or disable protocol checks
- ❖ Enable or 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.

3.3.2 Transaction Objects

Transaction objects, which are extended from the `uvm_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 the 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 `uvm_sequence_item` base class and implement all methods specified by the UVM methodology 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 delay 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 the 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 agent 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 agent 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 or SN agent.

You should initialize the node configuration handle in the transaction using the node configuration available in the sequencer of the RN or SN agent. 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.



Note For more details on individual members of transaction classes, see the CHI VIP Class Reference HTML documentation.

3.3.3 Analysis Port

The Protocol and Link monitors within the RN and SN Agent provide an analysis port `item_observed_port`. After completing the transaction, the RN and SN Agents respectively write the completed `svt_chi_rn_transaction` and `svt_chi_sn_transaction` object to the analysis port of the Protocol monitor. On observation of a flit at the link layer, the RN and SN Agents write `svt_chi_flit`

object to the analysis port of Link monitor. This is applicable in active and passive mode of operation of the RN or SN agent. Apart from connecting to the scoreboard you can also use this port for various connections where a transaction object for the completed transaction is required.

3.3.4 Callbacks

Callbacks are an access mechanism that enable the insertion of the user-defined code and allow access to objects for scoreboarding and functional coverage. Each RN and SN Agent is associated with a callback class that contains a set of callback methods. These methods are called as part of the normal flow of procedural code.

The difference between callback methods and other methods are as follows:

- ❖ Callbacks are virtual methods with no code initially, thus they do not provide any functionality unless they are extended. The exception to this rule is that some of the callback methods for functional coverage already contain a default implementation of a coverage model.
- ❖ The callback class is accessible to you so the class can be extended and your code inserted, potentially including the testbench specific extensions of the default callback methods, and testbench specific variables and/or methods are used to control the various behavior the testbench is using the callbacks to support.
- ❖ Callbacks are called within the sequential flow at places where external access would be useful. In addition, the arguments to the methods include references to relevant data objects. For example, just before a monitor puts a transaction object into an analysis port is a good place to sample for functional coverage since the object reflects the activity that just happened on the pins. A callback at this point with an argument referencing the transaction object allows this exact scenario.
- ❖ It is recommended not to extend a callback while invoking callback methods. To avoid a loss of performance, callbacks are not executed by default. To execute callback methods, callback class must be registered with the component using ``uvm_register_cb` macro.

CHI VIP uses callbacks in the following three main applications:

- ❖ Access for functional coverage
- ❖ Access for scoreboarding
- ❖ Insertion of user-defined code



Note

For more information on callback classes supported by CHI VIP, see the Class Reference HTML documentation.

3.3.5 Interfaces and Modports

For details on CHI Interfaces, see

`$DESIGNWARE_HOME/vip/svt/amba_svt/latest/doc/class_ref/chi_svt_uvm_class_reference/html/interfaces.html` in class reference HTML documentation.

3.3.5.1 Bind Interfaces

CHI VIP supports bind interfaces for RN and SN components. Bind interface is an interface which contains directional signals for CHI. Users can connect DUT signals to these directional signals. Bind interfaces provided with VIP are `svt_chi_rn_bind_if` and `svt_chi_sn_bind_if`.

To use bind interface, it is recommended to instantiate the non-bind interface, and then connect the bind interface to the non-bind interface. VIP provides RN and SN connector modules to connect the VIP bind interface to the VIP non-bind interface. It is recommended to instantiate a connector module corresponding to each instance of VIP master and slave, and pass the bind interface and non-bind interface instance to this connector module. The connector module is parameterized, for more information on differences between Active and Passive connection, see the following files

`$DESIGNWARE_HOME/vip/svt/amba_svt/latest/sverilog/include/svt_chi_rn_bind_if.svi` and `$DESIGNWARE_HOME/vip/svt/amba_svt/latest/sverilog/include/svt_chi_sn_bind_if.svi` for description.

3.3.6 Events

Events are not yet supported. Once supported, see the Protocol Monitor and Link Monitor within RN and SN agents in the class reference HTML documentation for list of supported events.

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

These restrictions when overriding the default maximum footprint:

- It is recommended not to use a value of 0 for a `MAX_*_WIDTH` 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 is as follows:
 - Number of bytes should be equal to the entire cache line size of 64 byte irrespective of transaction data size.
- ❖ Byte enable (applicable only for Write transactions) is as follows:
 - Number of bits corresponds to the entire cache line size of 64 byte irrespective of transaction data size. Therefore, the size 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 are as follows:
 - ◆ 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) are as follows:
 - ◆ Number of bits should be equal to transaction data size
 - ◆ 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 conditions for the following examples:

- ❖ The data width of the DATFLIT is 16 bytes.
- ❖ The data size of the transaction is 32 bytes.

- ❖ For unaligned address example, address that is not aligned to data size, consider that the address as 'h3A.
- ❖ For aligned address example, address that is aligned to data size, consider the address as 'h20.
- ❖ Also assume that the 32 byte data from address 'h20 to 'h3F is:

```
{128'bdeadbeef_feedbeef_deadbeef_feedbeef,
128'bdeadbeef_feedbeef_deadbeef_feedbeef}
```

Normal Memory Read Transactions

Address aligned to data size is as follows:

Data provided corresponds to 32 bytes (256 bits) starting from 'h20 to 'h3F. Therefore,
`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 is as follows:

Data provided should correspond to 32 bytes (256 bits) starting from 'h20 to 'h3F. Therefore,
`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`

Normal Memory Write Transactions

Address aligned to data size are as follows:

- ❖ Data, byte enable provided should correspond to 32 bytes starting from 'h20 to 'h3F. Therefore,
`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 are as follows:

- ❖ Data, byte enable provided should correspond to 32 bytes starting from 'h20 to 'h3F. Therefore, `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}`

Device Memory Read Transactions

Address aligned to data size is as follows:

The valid data bytes is similar to the normal memory case as described in Address aligned to data size.

Address unaligned to data size is as follows:

- ❖ The `svt_chi_transaction::data = {data corresponding to 'h3F, ..., data corresponding to 'h3A, data observed or provided on invalid bytes corresponding from 'h20 to 'h39}`



Note

It is recommended to provide data for all the 32 bytes, including the invalid bytes from 'h20 to 'h39.

- The data in this example will be, `data = {128'bdeadbeef_feedxxxx_xxxxxxxxxxxxxxxxxx, 128'bxx}`



Note

The 'x' above indicates the data that correspond to invalid bytes.

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

Device Memory Write Transactions

Address aligned to data size is as follows:

The valid data bytes enable is similar to the normal memory scenario as described in Address aligned to data size.

Address unaligned to data size are as follows:

- ❖ 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 or provided on invalid bytes corresponding from 'h20 to 'h39}`

**Note**

It is recommended to provide the data, byte enable for all the 32 bytes, including the invalid bytes from 'h20 to 'h39.

- The data in this example will be: data = {128'bdeadbeef_feedxxxx_xxxxxxxx_xxxxxxxx, 128'bxxxxxxx_xxxxxxxx xxxxxxxx_xxxxxxxx}

**Note**

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}

**Note**

Similar scenario is observed in svt_chi_snoop_transaction::data, svt_chi_snoop_transaction::byte_enable

3.3.9 TLM Generic Payload

The CHI VIP supports TLM Generic Payload feature where the user can develop sequences based on the `uvm_tlm_generic_payload` transaction type. The CHI VIP then maps these Generic Payload sequences into CHI specific sequences.

3.3.9.1 Generating SNOOP Response

The CHI RN agent contains a reactive sequencer (named `rn_snp_xact_seqr`) to generate responses to SNOOP requests. By default, this reactive sequencer is configured to execute the `svt_chi_rn_snoop_response_sequence` sequence as its default `run_phase` sequence. However, without any user intervention, the master agent behaves as a fully-coherent RN and reply to SNOOP requests using a local cache model.

When a different user-defined SNOOP response is required, the response is modeled in a reactive sequence extended from the `svt_chi_rn_snoop_base_sequence` base sequence and its functionality is implemented in its `body()` method is as follows:

```
virtual task body();
    forever begin
        svt_chi_master_snoop_transaction req;
        wait_for_snoop_request(req);

        // Fill in snoop response in 'req' in zero-time
        // e.g. fully random response
        if (!req.randomize()) `uvm_fatal(...)

        // Execute the response
        `uvm_send(req)
    end
endtask
```

The appropriate instance of the master agent is configured to execute the user-defined SNOOP response sequence rather than the default response sequence:

```
uvm_config_db#(uvm_object_wrapper)::set("chi_env.master[2]",
    "rn_snp_xact_seqr.run_phase", "default_sequence",
    cust_chi_response_seq::type_id::get());
```

3.3.9.2 Generating TLM Generic Payload Stimulus

By default, the CHI stimulus is generated using `svt_chi_rn_transaction` sequence items in the CHI RN agent sequencer. While generating the bus-agnostic stimulus, it can be specified using `uvm_tlm_generic_payload` sequence items.

This functionality must be enabled by setting the

`svt_chi_node_configuration::use_tlm_generic_payload` to `TRUE` for the corresponding CHI RN before that node's `build_phase` is executed.

```
function void my_env::build_phase(uvm_phase phase);
    super.build_phase(phase);

    cfg.rn_cfg[1].use_tlm_generic_payload = 1;
    uvm_config_db#(svt_chi_system_configuration)::set(this,
                                                    "chi_env",
                                                    "cfg", cfg);

    chi_env = chi_system_env::type_id::create("chi_env", this);
endfunction
```

Enabling this functionality causes the instantiation of a `svt_chi_tlm_generic_payload_sequencer` in the `svt_chi_rn_agent::tlm_generic_payload_sequencer` property and the execution of a layering sequence on the CHI transaction sequencer. The layering sequence pulls generated TLM generic payload sequence items from the generic payload sequencer, maps them to one or more CHI RN transactions, and executes them on the driver. The layering sequence executes with a normal priority. It is still possible to execute normal CHI RN transaction sequences on the CHI transaction sequencer, in parallel with the TLM generic payload layering sequence.

The response from the execution of the generic payload item is annotated in the generic payload sequence item itself. It is valid only when the completed generic payload sequence item is returned by the `uvm_sequence::get_response()` method.

```
class my_gp_seq extends uvm_sequence#(uvm_tlm_generic_payload);
...
task body();
    `uvm_create(req);
    req.set_command(UVM_TLM_READ_COMMAND);
    req.set_address('h123456789);
    req.set_length(64);
    `uvm_send(req);
    get_response(rsp);
    if (rsp.is_response_ok()) begin
```

```

        // gp.m_data[] is now valid
    end
endtask
endclass

```

The TLM generic payload sequence items are mapped into one or more CHI RN transactions that implement the semantics of the Generic Payload transaction, as defined by the TLM 2.0 standard. It is not possible to generate all possible CHI RN transactions from generic payload stimulus.

By default, generic payload WRITE and READ commands are mapped to WRITENOSNP and READNOSNP 64-byte CHI transactions respectively. When different CHI transactions are required, the generic payload sequence item are recommended to be annotated with an instance of the `svt_amba_pv_extension` generic payload extension.

```
class my_gp_seq extends uvm_sequence#(uvm_tlm_generic_payload);
```

```

...
task body();
    svt_amba_pv_extension    pv;

    `uvm_create(req);
    pv = new("pv");
    req.set_extension(pv);
    ...
    pv.set_size(1);
    pv.set_length(64);

    `uvm_send(gp);
endtask
endclass

```

The various attributes of the AMBA PV extension can be set to specify the characteristics of the CHI transaction(s) used to implement the annotated generic payload transaction. If the annotation is available, then it is further annotated with the relevant response from the execution of the CHI transactions.

3.3.9.3 TLM Generic Payload Mapping to CHI RN Transactions

The operations described can be mapped using the generic payload transactions into one or more valid CHI transactions. If the generic payload transaction cannot be mapped to valid CHI transactions, an error is issued and the `UVM_TLM_INCOMPLETE_RESPONSE` status is returned. The generic payload transactions are mapped to the CHI transactions according to the generic payload semantics. Therefore, it is not possible to create all possible CHI transactions from generic payload stimuli.

`uvm_tlm_generic_payload` sequence items, optionally annotated with a `svt_amba_pv_extension`, are mapped to `svt_chi_rn_transaction` instances as follows:

1. If address bits [63:44] are not zeros, a `UVM_TLM_ADDRESS_ERROR_RESPONSE` is immediately returned.
2. More than one request may be necessary if a burst operation has more than one beat or covers more than one cache line in one beat.

3. If there is no `svt_chi_pv_extension`, the generic payload transaction is mapped to one or more `ReadNoSnp` or `WriteNoSnp` 64B requests.
4. The `svt_chi_pv_extension`, if present, must specify a size of 1, 2, 4, 8, 16, 32 or 64.

The field of the `uvm_tlm_generic_payload` sequence item and its optional `svt_chi_pv_extension` are mapped to the randomized `svt_chi_rn_transaction` as per the following table and all the other properties are randomized.

Table 3-2 Optional `svt_chi_pv_extension` are Mapped to the Randomized `svt_chi_rn_transaction`

Generic Payload	CHI PV Extension	CHI SVT Transaction
<code>get_command()</code>	<code>get_snoop()</code> <code>get_id()</code>	<code>xact_type</code> <code>txn_id</code>
<code>get_address()</code>		<code>addr</code>
	<code>get_size()</code>	<code>data_size</code>
<code>get_byte_enable()</code>		<code>byte_enable</code>
<code>get_data()</code>		<code>data</code>
randomized		<code>is_likely_shared</code> <code>mem_attr_is_early_wr_ack_allowed</code>
NORMAL		<code>mem_attr_mem_type</code>
NO_ORDERING_REQUIRED		<code>order_type</code>
	<code>is_cacheable</code>	<code>mem_attr_is_cacheable</code>
<code>get_command()</code>	<code>is_read_allocate()/is_write_allocate()</code>	<code>mem_attr_allocate_hint</code>
randomized		<code>is_likely_shared</code> <code>mem_attr_is_early_wr_ack_allowed</code>
NORMAL		<code>mem_attr_mem_type</code>
NO_ORDERING_REQUIRED		<code>order_type</code>
	<code>is_cacheable</code>	<code>mem_attr_is_cacheable</code>
<code>get_command()</code>	<code>is_read_allocate()/is_write_allocate()</code>	<code>mem_attr_allocate_hint</code>
randomized		<code>snp_attr_is_snoopable</code>
	<code>get_domain()</code>	<code>snp_attr_snp_domain_type</code>
randomized		<code>lpid</code> <code>exp_comp_ack</code>
	<code>is_exclusive()</code>	<code>is_exclusive</code>
randomized		<code>check_addr_overlap</code>

Table 3-2 Optional svt_chi_pv_extension are Mapped to the Randomized svt_chi_rn_transaction

Generic Payload	CHI PV Extension	CHI SVT Transaction
	get_qos()	qos
	is_non_secure()	is_non_secure_access
Randomized		p_crd_type
		req_rsvdc
		dat_rsvdc
	set_resp()	resp_err_status
	set_pass_dirty()	resp_pass_dirty
Ignored		current_state
		final_state
		dbid

Table 3-3 Mapping CHI Transaction Type

uvm_tlm_generic_payload::get_command()				
	amba_pv_extension			
	get_bar()	get_domain()	get_snoop()	CHI Type
UVM_TLM_READ_COMMAND				
	MEMORY SYNC			EOBARRIER
	RESPECT IGNORE	NONE SYSTEM	4'b0000	READNOSNP
			4'b0000	READONCE
		INNER OUTER	4'b0001	READSHARED
			4'b0010	READCLEAN
			4'b0011	Error
			4'b0111	READUNIQUE
			4'b1000	CLEANSHARED
			4'b1001	CLEANINVALID
			4'b1011	CLEANUNIQUE
			4'b1100	MAKEUNIQUE
			4'b1101	MAKEINVALID
			4'b111-	Not yet supported

Table 3-3 Mapping CHI Transaction Type

uvm_tlm_generic_payload::get_command()				
	amba_pv_extension			
	get_bar()	get_domain()	get_snoop()	CHI Type
UVM_TLM_WRITE_COMMAND				
	MEMORY SYNC			ECBARRIER
	RESPECT IGNORE	NONE SYSTEM	4'b-000	WRITENOSNP
		INNER OUTER	4'b-000	WRITEUNIQUE
			4'b-001	WRITEEVICT
			4'b-010	WRITECLEAN
			4'b-011	WRITEBACK
			4'b-100	EVICT

3.3.9.4 Connecting a TLM 2.0 Master

By default, TLM generic payload stimulus is generated using SystemVerilog sequences in the CHI RN agent generic payload sequencer. If the TLM generic payload transactions are created by an ARM FastModel or a TLM interconnect model written in SystemC, it is possible to connect the CHI master agent to a coherent TLM master or bridge.

This functionality is recommended to be enabled by setting the `svt_chi_node_configuration::use_pv_socket` to `TRUE` for the corresponding CHI RN prior to the node's `build_phase` execution.

```
function void my_env::build_phase(uvm_phase phase);
    super.build_phase(phase);

    cfg.master_cfg[1].use_pv_socket = 1;
    uvm_config_db#(svt_chi_system_configuration)::set(this,
                                                    "chi_env",
                                                    "cfg", cfg);

    chi_env = chi_system_env::type_id::create("chi_env", this);
endfunction
```

Note

Enabling this functionality implies the enabling of TLM generic payload stimulus (see the above section).

Enabling this functionality causes the instantiation of an `uvm_tlm_b_target_socket` interface in the `svt_chi_rn_agent::b_fwd` property and an `uvm_tlm_b_initiator_socket` interface in the `svt_chi_rn_agent::b_snoop` property. Furthermore, the default `run_phase` sequence for the SNOOP response sequencer is replaced with a reactive sequence which forwards all SNOOP transaction requests (translated to equivalent `uvm_tlm_generic_payload` transactions annotated with a `svt_amba_pv_extension`) to the backward `b_snoop` interface to be fulfilled by the coherent TLM master.

If the TLM master is not coherent, the CHI RN agent can be re-configured to handle SNOOP requests natively using its local cache model:

```
uvm_config_db#(uvm_object_wrapper)::set("chi_env.master[2]",
    "rn_snp_xact_seqr.run_phase", "default_sequence",
    svt_chi_rn_snoop_response_sequence::type_id::get());
```

3.3.10 Service Transactions

Service request classes are used to describe the external events of the normal protocol transaction and data flow, that are exclusively designed for the protocol to handle.

3.3.10.1 Link Activation or Deactivation Through Service Transactions

Class `svt_chi_link_service` is the service transaction class. Link activation and deactivation is supported through the following members of the class:

```
rand service_type_enum service_type;
rand int min_cycles_in_deactive;
```

A sequence can be created and submitted through the `svt_chi_link_service_sequencer` within the RN or SN agent to force the link to deactivate by setting the following:

```
service_type == DEACTIVATE;
min_cycles_in_deactivate == <value>;
```

The service request unblocks, after the link is in the de-active state, but prior to the expiration of the minimum timeout. If the user wants to ensure that the link should remain in de-active state for a number of cycles, then `min_cycles_in_deactive` is recommended to be set to a non-zero value. During the de-active state the user can force the link back to the active state by sending another service request with `service_type==ACTIVATE`.



Note

If a reactive use model is required, then the `svt_chi_status::is_link_active` property in the shared status object (which is available to all sequences) can be used to determine the current link state.

3.3.11 Delays

For information on the various delay attributes supported by CHI SVT VIP, see the attributes in the following groups in the HTML class reference documentation:

- ❖ `svt_chi_node_configuration - chi_config_delays`
- ❖ `svt_chi_flit - chi_flit_delays`

3.3.12 Reordering of Transactions

To know the user interface details on out of ordering support in CHI SVT VIP, see the attributes under following groups in HTML class reference documentation:

`svt_chi_node_configuration - chi_config_reordering`

3.3.12.1 Use Model

For a given CHI node configuration object (`svt_chi_node_configuration`), following attributes need to be programmed:

For DAT flits reordering the programming is as follows:

- ❖ Program the DAT flit reordering depth using the attribute `svt_chi_node_configuration::dat_flit_reordering_depth`
 - ◆ The maximum value of this attribute can be controlled through the user re-definable macro ``SVT_CHI_MAX_DAT_FLIT_REORDERING_DEPTH`
- ❖ Program the reordering algorithm to be used, for reordering the DAT flits across the transactions using the attribute `svt_chi_node_configuration::dat_flit_reordering_algorithm`
 - ◆ With this attribute set to `PRIORITIZED`, it is possible that a given transaction may not be able to access DAT VC for multiple times because of its low QoS value, compared to other transactions within reordering depth window. The attribute `svt_chi_node_configuration::xact_dat_vc_access_fail_max_count` can be configured to bail out such transactions in getting access to DAT VC.
- ❖ In case of RN, the reordering of DAT flits of Protocol transactions is not mixed with that of SNOOP transactions.

For RSP flits reordering the programming is as follows:

- ❖ Program the RSP flit reordering depth using the attribute `svt_chi_node_configuration::rsp_flit_reordering_depth`
 - ◆ The maximum value of this attribute can be controlled through the user re-definable macro ``SVT_CHI_MAX_DAT_FLIT_REORDERING_DEPTH`
- ❖ Program the reordering algorithm to be used for reordering the RSP flits across the transactions using the attribute `svt_chi_node_configuration::dat_flit_reordering_algorithm`
 - ◆ With this attribute set to `PRIORITIZED`, a given transaction will not be able to access RSP VC multiple times because of its low QoS value, compared to other transactions within the reordering depth window. The attribute `svt_chi_node_configuration::xact_rsp_vc_access_fail_max_count` can be configured to bail out such transactions in getting access to DAT VC.
- ❖ In case of RN, this controls the reordering of RSP flits of SNOOP transactions.



Note

The reordering of RSP flits of Protocol transactions is not performed.

Apart from ordered stream of WriteUnique transactions, only for Read type transactions the CompAck is generated as response after receiving the read data. So, it is unlikely that multiple outstanding transactions with only CompAck transmission is pending at the RN.

3.4 Reset Functionality

VIP components detect reset synchronously and asynchronously. For VIP to detect power on reset, it is required to observe a transition of reset signal to active low. VIP issues a FATAL message if it detects reset signal value of '0' at the start of simulation, that is, from zero simulation time.

3.4.1 Dynamic Reset Support

VIP supports dynamic reset, that is, handling reset assertion during simulation.

When a reset is detected, the following changes occur for the transactions in progress, that is, transactions which have started but not ended:

- ❖ They are ABORTED and removed from the internal transaction queue.
- ❖ They are written to the analysis port by all active and passive CHI VIP components.
- ❖ All status fields for these transactions, that is, `svt_chi_transaction::req_status`, `svt_chi_transaction::data_status` and `svt_chi_transaction::resp_status`, are also moved to ABORTED state.
- ❖ All protocol and link layer credit counts are reset to default values.

Transactions which receive a RETRYACK response and have not yet been retried will be aborted on reset assertion and will not be retried after reset de-assertion.

All outstanding CHI transactions that are still present in the internal queue and are attempting to be transmitted on the interface will continue to remain in the queue and resume after reset de-assertion.

At the link layer, the following signals will be de-asserted at the next clock edge after reset assertion is detected:

- ❖ TXSACTIVE
- ❖ TXLINKACTIVEREQ
- ❖ RXLINKACTIVEACK
- ❖ RX***LCRDV
- ❖ TX***FLITPEND
- ❖ TX***FLITV

In order to debug reset activity, you should run the simulation in UVM_HIGH verbosity. You can search for the following messages in the generated log:

- ❖ “Received reset”
- ❖ “Reset in progress. Waiting for ...”
- ❖ “Reset in progress. Marking transaction ...”

3.5 Programming CHI System Address Ranges and Related Settings

3.5.1 Programming the HN Related Information

The following information is essential for 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_interconnect`).

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 required to program SN-F index to HN-F index programming; however, this API is required to program SN index to HN-I index programming. For more details, refer to the section 'Three SN-F striping'.
- ❖ MN node ID programming:
 - ◆ Program using the attribute `svt_chi_system_configuration::misc_node_id`

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. Else, 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`).

The order of programming HN nodes (for node ID, HN interface type) should be such that all HN-F nodes should be programmed followed by all HN-I nodes. For more details, refer to programming examples in the User Guide and CHI UVM intermediate example.

From the upcoming release, if this programming order is not followed, then the simulation fails with ERRORS indicating invalid CHI system configuration (`svt_chi_system_configuration::do_is_valid()`).

3.5.2 Programming MN Address Ranges

The 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 applicable for MN as well. In such a case, the argument 'mn_idx' corresponds to HN index of HN-I. It is 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 its 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);
```

3.5.3 Programming HN Address Ranges

The following approaches are permitted for programming HN address ranges:

- ❖ Using APIs
- ❖ Using factory override for `svt_chi_system_configuration`

3.5.3.1 Programming Using APIs

The 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)
```

3.5.3.1.1 Programming HN-I Address Ranges

Use `svt_chi_system_configuration:: set_hn_addr_range()` with `'hn_idx'` that corresponds to the HN-I. It is 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 its 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);
```

3.5.3.1.2 Programming HN-F Address Ranges**Default Behavior: If you do not program the HN-F address ranges using**

`svt_chi_system_configuration::set_hn_addr_range()`, the following details are applicable.

By default, VIP does not 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. Else, the CHI system configuration will be considered as invalid and simulation fails. This is needed irrespective of whether VIP interconnect is enabled or not (`svt_chi_system_configuration::use_interconnect` setting).

3.5.3.1.3 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 is 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 its 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);
```

3.5.3.1.4 Address to HN Mapping Mechanism Used by 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

3.5.3.2 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.

You 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.

3.5.3.2.1 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;

    //Utility macro
    `svt_data_member_begin(cust_svt_chi_test_suite_system_configuration)
    `svt_data_member_end(cust_svt_chi_test_suite_system_configuration)

    function new (string name="cust_svt_chi_test_suite_system_configuration" );
    super.new(name);
    endfunction

    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
    end
```

```

        endfunction

    endclass
`endif //GUARD_CUST_SVT_CHI_TEST_SUITE_SYSTEM_CONFIGURATION_SV

```

3.5.3.2.2 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;

    //Utility macro
    `svt_data_member_begin(cust_svt_chi_system_configuration)
    `svt_data_member_end(cust_svt_chi_system_configuration)

    function new (string name="cust_svt_chi_system_configuration" );
        super.new(name);
    endfunction

/**
 * 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 established 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

```

3.5.4 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. For more details, refer to the section 'Three SN-F striping'.

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});

```

3.5.5 Recommended Programming Examples

The following 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});

```

3.6 Three SN-F striping

3.6.1 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 represented as $(2^{\text{integer value}})$. 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.

3.6.2 User Interface

3.6.2.1 Attributes

The following attributes that must be programmed in svt_chi_system_configuration to use this feature:

- ❖ Set the three_sn_f_striping_enable attribute to 1.
- ❖ Program the top address bit indices appropriately based on total addressable space across 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,
 - ◇ $\text{three_sn_f_striping_top_address_bit_1} = \text{three_sn_f_striping_top_address_bit_0} + 1$

3.6.2.2 APIs

The following APIs are present in the `svt_chi_system_configuration`. Refer to the class reference for more details.

- ❖ `get_three_sn_f_stripping_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_stripping_top_address_bit_1`, `svt_chi_system_configuration::three_sn_f_stripping_top_address_bit_1`.
- ❖ `get_three_sn_f_stripping_based_sn_f_idx()`
 - ◆ Returns the SN-F index that corresponds to the address input argument passed, based on three SN-F stripping.
- ❖ `is_three_sn_f_stripping_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 stripping 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.

3.6.2.3 Programing 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 participating SN-Fs should be 3 */
    num_sn = 3;
    foreach(sn_cfg[i]) sn_cfg[i].chi_interface_type = svt_chi_node_configuration::SN-F;

    /** Three SN stripping related settings */
    three_sn_f_stripping_enable = 1;
    three_sn_f_stripping_top_address_bit_0 = 42;
    three_sn_f_stripping_top_address_bit_1 = 43;
-----
endclass
```

3.6.3 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.

3.6.4 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 = $3 * (\text{addressable space per SN-F})$

3.6.4.1 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

3.6.4.2 Different Top Address Bit Field Values and Addressable Space Values

<code>top_bit1</code>	<code>top_bit0</code>	<code>size_per_SN-F</code>	<code>size_across_3_SN-Fs</code>
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

3.6.5 Address Aliasing

As described in previous 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 is possible that multiple addresses at RN-F can get mapped to same physical address in the DRAM.

3.6.5.1 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:

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

3.7 Connecting AXI Slaves to CHI System Monitor

When the CHI system monitor is enabled within a CHI system that is part of AMBA system environment (along with by including `svt_amba.uvm.pkg` and then importing `svt_amba_uvm_pkg::*`), CHI System Monitor is enhanced to perform slave transaction routing check and slave data integrity check when a transaction is routed to an AXI slave, that is connected to CHI interconnect:

- ❖ `svt_chi_system_err_checks` slave_transaction_routing_check is enhanced such that this check is performed on the AXI slave transaction from the AXI slaves that are mapped to CHI SN nodes through the API `svt_amba_system_configuration:: set_axi_slave_to_chi_sn_map()`
- ❖ `svt_chi_system_err_check::slave_data_integrity_check` is added such that the data of AXI slave transaction is compared with memory of AXI slaves that are mapped to CHI SN nodes through the API `svt_amba_system_configuration:: set_axi_slave_to_chi_sn_map()`

For additional information, see `amba_svt_uvm_class_reference` for the usage details of the API `svt_amba_system_configuration:: set_axi_slave_to_chi_sn_map()`.

Refer to `chi_svt_uvm_class_reference` for the description of the above mentioned checks.

For example:

```
class cust_svt_amba5_system_configuration extends svt_amba_system_configuration;
---
    num_of_axi_systems = 1;
    num_of_chi_systems = 1;
---
    create_sub_cfgs(num_of_axi_systems,num_ahb_systems,num_apb_systems,num_chi_systems);
---
    /** Number of CHI SNs should be sum of:
    *   o Number of external CHI SNs connected to CHI Interconnect   (3 in this example,
that are tied to HN-Fs)
    *   o Number of external AXI slaves connected to CHI interconnect (1 in this
example, that is tied to HN-I)
    *       - The corresponding sn_cfg, SN agent are dummy, but needed for CHI system
monitor to perform checks
    */
    num_chi_sn = 4;

    chi_sys_cfg[0].create_sub_cfgs(num_chi_rn,num_chi_sn,0,0,0,0,0,0);
---
    /** Map AXI slave with port ID 0 of AXI system 0 (associated to HN-I), to the
corresponding CHI SN slave index 3 of CHI system 0 */
    set_axi_slave_to_chi_sn_map(axi_sys_cfg[0].system_id, chi_sys_cfg[0].system_id, {0},
{3});
---
```

```
endclass
```

3.7.1 Configuring AXI slaves

3.7.1.1 Register address space:

The Register address space of the CHI Interconnect needs to be programmed as the register address space.

```
// E.g.: `TB_START_ADDR_MN, `TB_END_ADDR_MN corresponds to register
//         space start and end address respectively
<axi_sys_cfg>.set_addr_range(-1, `TB_START_ADDR_MN, `TB_END_ADDR_MN);
```

3.7.1.2 HN-I Mapped Address Space

For the AXI slave that corresponds to HN-I, the slave address range programmed should match with HN-I address ranges.

```
// E.g.: Slave 0 corresponds to HN-I mapped AXI slave.
//         `START_ADDR_AXI_SLV0, `END_ADDR_AXI_SLV0 indicate HN-I address ranges.
<axi_sys_cfg>.set_addr_range(0, `START_ADDR_AXI_SLV0, `END_ADDR_AXI_SLV0);
```

3.7.1.3 HN-F Mapped Address Space

3.7.1.3.1 HN-F Address Range is Based on XOR Function

For the transactions from HN-Fs to AXI slaves (that are mapped to SN-Fs), if the HN-Fs use either the XOR function based on number of HN-Fs OR three SN-F striping algorithm, following mechanism can be used to program the address ranges for such AXI slaves. Also it is required to enable overlapping addresses across such slaves:

```
// Enable overlapping addresses across the slaves
<axi_sys_cfg>.allow_slaves_with_overlapping_addr = 1;

// Map the entire address range of CHI address space except HN-I mapped range
// to each of the AXI slaves that are mapped to HN-Fs
// E.g.: slave 1 to num_slaves-1 in below example map to HN-Fs
for (int i=1; i<<axi_sys_cfg.num_slaves; i++) begin
  <axi_sys_cfg>.set_addr_range(i, (`END_ADDR_AXI_SLV0+1),
  ((1<<`SVT_CHI_MAX_ADDR_WIDTH)-1));
end
```

3.7.1.3.2 HN-F Address Range is Explicitly Specified

```
// For the transactions from HN-Fs to AXI slaves (that are mapped to
// SN-Fs), then map the HN address ranges to respective SN-F address range which is
// connected to AXI Slaves
// E.g.: HN-F[x] is mapped to AXI slave[x], and address range for HN-F[x] is
{[ `START_ADDR_AXI_SLVx: `END_ADDR_AXI_SLVx])
  // where the slave/HN-F idx is in the range 1 to 4
  axi_sys_cfg[0].set_addr_range(1, (`START_ADDR_AXI_SLV1), (`END_ADDR_AXI_SLV1));
  axi_sys_cfg[0].set_addr_range(2, (`START_ADDR_AXI_SLV2), (`END_ADDR_AXI_SLV2));
  axi_sys_cfg[0].set_addr_range(3, (`START_ADDR_AXI_SLV3), (`END_ADDR_AXI_SLV3));
  axi_sys_cfg[0].set_addr_range(4, (`START_ADDR_AXI_SLV4), (`END_ADDR_AXI_SLV4));
```

3.7.1.4 Configuring AXI Slaves To Match the CHI SN-F Nodes

Note that the following fields of the AXI slave port configuration should match with that of corresponding CHI SN-F node configuration. This can be programmed with the extended `svt_amba_system_configuration` class of the test bench.

❖ Interface type:

```
// Set AXI interface type: for HN-I mapped, it can be ACE-Lite/AXI4. For HN-F mapped,
// its AXI4. <axi_sys_cfg>.slave_cfg[i].axi_interface_type =
svt_axi_port_configuration::AXI4;
```

❖ Address width:

```
// Set addr_width to same as CHI
<axi_sys_cfg>.slave_cfg[i].addr_width = `SVT_CHI_MAX_ADDR_WIDTH;
```

❖ Data width:

```
// Set data_width to match Flit Data width of CHI
<axi_sys_cfg>.slave_cfg[i].data_width = 128;
```

❖ WYSIWYG enable:

```
// WYSIWYG MODE should be consistent across AXI and CHI.
// Below example it's enabled for both AXI and CHI systems.
<axi_sys_cfg>.slave_cfg[i].wysiwyg_enable = 1;
```

3.7.1.4.1 Configuring AXI Slaves for Three SN-F Striping

Consider the following example:

There are 4 external AXI4 slaves in the environment:

- ❖ One AXI4 slave is needed to map to HN-I
- ❖ Three AXI4 slaves are needed to map to HN-Fs for the three SN-F striping
- ❖ The AXI system ID is: <amba_sys_cfg>.<axi_sys_cfg>.system_id

As there are four external AXI4 slaves, four SNs should be present within the CHI system configuration, which needs to be mapped to these AXI4 slaves.

- ❖ Assume the HN-I index is 4
- ❖ Number of SNs needed are 4
- ❖ One SN should be mapped to HN-I. Also the SN index that is mapped to HN-I is 0.
- ❖ Three SNs (SN-Fs) should be part of three SN-F striping, so these SN indices are {1, 2, 3}.

Map HN-I to SN index 0:

```
// AXI4 SLAVE mapped to SN idx that corresponds to AXI slave
chi_sys_cfg.set_sn_to_hn_map(0, {8});
```

MAP AXI slave port_id's to CHI SN node_indices:

```
<amba_sys_cfg>.set_axi_slave_to_chi_sn_map(<amba_sys_cfg>.<axi_sys_cfg>.system_id,
<amba_sys_cfg>.<chi_sys_cfg>.system_id, {0,1,2,3}, {0,1,2,3});
```

Configure the SN-Fs and other related settings:

```
<chi_sys_cfg>.num_sn = 4;
foreach (<chi_sys_cfg>.sn_cfg[i]) begin
    <chi_sys_cfg>.sn_cfg[i].chi_interface_type = svt_chi_node_configuration::SN_F;
    // Other settings
end
<chi_sys_cfg>.sn_cfg[0].node_id = 2; // This corresponds to HN-I mapped AXI4 slave
<chi_sys_cfg>.sn_cfg[1].node_id = 4;
<chi_sys_cfg>.sn_cfg[2].node_id = 10;
<chi_sys_cfg>.sn_cfg[3].node_id = 12;
```



```

<chi_sys_cfg>.three_sn_f_stripping_enable = 1;
<chi_sys_cfg>.three_sn_f_stripping_top_address_bit_0 = 42;
<chi_sys_cfg>.three_sn_f_stripping_top_address_bit_1 = 43;

```

3.8 Connecting ACE-LITE Masters to CHI System Monitor

When the CHI system monitor is enabled within a CHI system that is part of an AMBA system environment (done by including `svt_amba_uvm.pkg` and then importing `svt_amba_uvm_pkg::*`), ACE-LITE masters can be directly connected to the CHI system monitor for performing coherency and data integrity checks. You can refer the documentation of `svt_amba_system_configuration::set_ace_lite_to_rn_i_map` for usage details. Each ACE-LITE port must have a corresponding RN_I node in the configuration and the ACE-LITE port must be mapped to the RN_I node using the above API.



Note

The RN_I interface signals need not be connected; only the configuration is required.

It is important to note that all hazard related checks in CHI System Monitor are disabled when an ACE-LITE port is connected directly using the above API. This is because these checks cannot be predictably done at the ACE-LITE interface due to the difference in timing of corresponding activity at the RN-I node.



Note

DVM and Barrier transactions are currently not supported when using the above API.

This is a sample code of the usage of this API in the CHI test suite environment

```

// ACE-LITE master 0 corresponds to RN-I node index 4. ACE-LITE master 1
// corresponds to RN-I node index 5. Comment out this line if transactions
// are to be sampled at the RN-I interface and not at ACE-LITE interface.
// Note that if this line is commented out, in the case of an interconnect
// RTL, the signals need to be connected to RN-I interface

set_ace_lite_to_rn_i_map(this.axi_sys_cfg[0].system_id,
this.chi_sys_cfg[0].system_id, {0,1}, {4,5});

// An ACE-Lite port is connected at node indices 4 and 5 of
// these RNs and the bridge sequence converts it to RN-I transactions. If
// Interconnect VIP is used, these nodes are in active mode, but with
// Interconnect RTL DUT, these are in passive mode. If the corresponding
// ACE-LITE port is mapped to these ports through a call to
// set_ace_lite_to_rn_i_map, then RN_I interface signals need not be
// connected at these interfaces as all sampling will be disabled at the

```

```
// RN-I interface, and the activity at the ACE-LITE interface is used by
// the system monitor. However, if the call to set_ace_lite_to_rn_map is
// not made, the system monitor uses activity at the RN_I interfaces and
// therefore these interfaces must be connected to the RN-I signals of
// the ACE-LITE to RN-I bridge in the RTL DUT Interconnect

    if ((i == 4) || (i == 5)) begin
`ifdef SVT_AMBA5_TEST_SUITE_VIP_DUT
        chi_sys_cfg.rn_cfg[i].is_active = 1;
`else
        chi_sys_cfg.rn_cfg[i].is_active = 0;
`endif
    end
```

3.9 CHI Memory Within SN Agent

The CHI SN agent 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 agent.

- ❖ In the active mode, the memory can be updated through SN response sequence. Refer to the sequence `svt_chi_sn_transaction_memory_sequence` in the `svt_chi_sn_transaction_sequence_collection` file and `tb_chi_svt_uvm_basic_sys` for the usage.
- ❖ In the passive mode, the SN agent updates the memory for write transactions that are completed successfully. To update the memory for the read 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.

3.10 CHI Interface Clock Mode

This section consists of the following subsections:

- ◆ [Common Clock Mode](#)
- ◆ [Multi-Clock Mode](#)

3.10.1 Common Clock Mode

This mode provides the following features:

- ◆ This is the default use model.
- ◆ Pass the common clock, `clk`, as an argument to the `svt_chi_if` instance of the top-level CHI interface.
- ◆ Pass the same `clk` from the top-level interface to all Request Node (RN), Slave Node (SN) sub-interfaces, and corresponding Interconnect port sub interfaces.

3.10.2 Multi-Clock Mode

This mode provides the following features:

- ◆ This is not the default use model. When different clocks are needed for each of RNs and SNs, you need to enable this mode.
- ◆ To enable this mode, define a compile-time macro, namely ``SVT_CHI_ENABLE_MULTI_CLOCK`.
- ◆ In this mode, 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 to the `svt_chi_if` instance of the top-level CHI interface instead of passing `clk` as an argument to `svt_chi_if`.
- ◆ The array of clocks should be declared as shown above, that is, from `[*_MAX_*-1:0]`.
- ◆ In this mode, `rn_clk[rn_idx]` then gets connected to the corresponding CHI RN sub interface, `svt_chi_if::chi_rn_if[rn_idx]`, and to the corresponding port on Interconnect, `svt_chi_if::chi_ic_rn_if[rn_idx]`.
- ◆ Similarly, `sn_clk[sn_idx]` then gets connected to the corresponding CHI SN sub interface, `svt_chi_if::chi_sn_if[sn_idx]`, and to the corresponding port on Interconnect, `svt_chi_if::chi_ic_sn_if[sn_idx]`.

For more details, see the documentation of the CHI top-level interface file, `svt_chi_if.svi`. Also, for the use model example, see `tb_chi_svt_uvm_basic_sys/top.sv` and its corresponding README file, `tb_chi_svt_uvm_basic_sys/top.sv`

3.10.2.1 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
endmodule
```

```

        end
    end // initial begin

    -----
endmodule

```

3.11 CHI System Level Performance Metrics

3.11.1 Overview

To measure the performance metrics of a CHI based system, it is important to track different performance metrics for each of the HNs that are present in the CHI interconnect.

For each HN within interconnect, the performance metrics can be broadly classified into the following categories:

- ❖ Data band width and Retry rate related throughput metrics
- ❖ Snoop filter and L3 cache related metrics
 - This is applicable only for HN-Fs
- ❖ Latency of a transaction initiated from RN and associated latencies

See the related sections for more details on each of these categories of metrics.

In order to involve the ACE-Lite masters connected to CHI interconnect through RN-I bridges, and AXI/ACE-Lite slaves connected to CHI interconnect through SN bridges, it is required to use the AMBA system ENV and related APIs described in *CHI SVT UVM User Guide*.

3.11.2 User Interface

3.11.2.1 Configuration Parameters

To control the tracking of performance metrics by CHI SVT VIP, following controls are added to `svt_chi_system_configuration`.

These are documented in the CHI SVT UVM class reference documentation, under the class `svt_chi_system_configuration` with the grouping 'CHI Performance metrics'.

bit attribute

```
svt_chi_system_configuration::perf_tracking_enable = 0
```

- ❖ To enable the tracking of the performance metrics.
- ❖ Applicable only when `system_monitor_enable` is set to 1
- ❖ When set to "1", the CHI RN, SN agents and the System monitor tracks the performance metrics.
- ❖ Setting this to "0" will not cause the VIP to track the performance metrics.
- ❖ When `svt_chi_system_configuration::enable_summary_tracing` is set to 1:
 - ◆ The performance metrics info is also dumped into the same file (`<chi_system_monitor_instance_hierarchy>.coh_snp_xact_summary_trace`) along with coherent and snoop transaction summary towards the end of simulation.
 - ◆ The coherent and snoop transaction summary with L3 missed is written to the file `<chi_system_monitor_instance_hierarchy>.l3_miss_coh_snp_xact_summary_trace`
- ❖ Not yet supported for VMM methodology.

- ❖ Type:Static
- ❖ Default value: 0

bit attribute

```
svt_chi_system_configuration::display_perf_summary_report = 0
```

- ❖ To control the display of the performance metrics summary
- ❖ Applicable only when both `system_monitor_enable` and `perf_tracking_enable` are set to 1
- ❖ When set to "1", The summary is displayed with NORMAL verbosity.
- ❖ When set to "0", The summary is displayed with HIGH verbosity.
- ❖ Not yet supported for VMM methodology.
- ❖ Type: Static
- ❖ Default value: 0

The following parameter is related to performance metrics tracking, but users don't need to set this explicitly for the purpose of performance metrics tracking.

bit attribute

```
svt_chi_system_configuration::slave_xact_to_rn_xact_correlation_enable = 0
```

- ❖ Controls correlation of Slave transactions to RN transactions by system monitor.
- ❖ Applicable only when `system_monitor_enable` is set to 1
- ❖ When `perf_tracking_enable` is set to 1, value of this attribute is ignored and is considered as enabled.
- ❖ Not yet supported for VMM methodology.
- ❖ Type: Static
- ❖ Default value: 0

3.11.2.2 Shared System Status

3.11.2.2.1 svt_chi_system_status and Related Classes

All the performance metrics that are collected for a given instance of CHI system ENV are encapsulated in a shared system status object of type `svt_chi_system_status`.

3.11.2.2.2 svt_chi_system_status

```
svt_chi_system_status attribute
```

```
svt_chi_system_env::shared_system_status
```

Shared status object used to convey the system level details such as performance metrics data.



Note

This object is to be treated as read-only for any accesses from outside the agent. Writing or modifying any of the attributes may lead to unexpected results from the VIP.

The `svt_chi_system_status` contains:

- ❖ An array of `svt_chi_system_hn_status` objects that capture the performance metrics on per HN basis.
- ❖ Average of some of the latencies that are computed across different HNs.
- ❖ Time scale info of the latency computations

The units of the band width computation; this is in MB/sec.

`SVT_CHI_SYSTEM_HN_STATUS`

`svt_chi_system_hn_status` attribute

`svt_chi_system_status::system_hn_status[]`

The dynamic array of `svt_chi_system_hn_status` objects, with one entry per HN in the CHI system.

The `svt_chi_system_hn_status` class contains:

- ❖ Various performance metrics related to latency, L3, and Snoop filter captured on per HN basis.
- ❖ Information of the HN including: HN node id, HN node index, and HN interface type

3.11.2.2.3 `svt_chi_hn_status`

The `svt_chi_system_hn_status` extends `svt_chi_hn_status`. The `svt_chi_hn_status` class contains the attributes that represent the throughput related performance metrics. As this is the base class to `svt_chi_system_hn_status`, all the attributes of this class are available through `svt_chi_system_hn_status` objects, which are updated on per HN basis.

All the HN related details are available through the following attributes of `svt_chi_hn_status`, under the group heading 'HN details' in the HTML class reference.

int attribute

`svt_chi_hn_status::hn_idx`

Indicates the `hn_idx` of the HN corresponding to this `svt_chi_hn_status` object

int attribute

`svt_chi_hn_status::hn_node_id`

Indicates the node ID the HN corresponding to this `svt_chi_hn_status` object

`svt_chi_address_configuration :: hn_interface_type_enum` attribute

`svt_chi_hn_status::hn_interface_type`

Indicates the HN interface type of the HN corresponding to this `svt_chi_hn_status` object

See the HTML class reference of these classes for more details.

3.11.2.3 Accessing Shared System Status Object From the Test Bench

When performance tracking (controlled by `svt_chi_system_configuration::perf_tracking_enable`), as well as the CHI system monitor (controlled by `svt_chi_system_configuration::system_monitor_enable`) are enabled, the CHI system monitor updates the shared system object of type `svt_chi_system_status` on a timely basis.

Further, CHI system monitor prints the content of shared system object in the report phase (controlled by `svt_chi_system_configuration::display_perf_summary_report`) of the simulation.

However, if you want to print or observe the performance metrics at a given point of time, the handle to shared system object can be obtained from the corresponding CHI system ENV through config DB.

Example:

The following are the steps to be followed in order to obtain and print the shared system object in a test, illustrated through the corresponding code snippet as follows:

- ❖ `chi_base_test` is the test that extends the `uvm_test`.
- ❖ The test contains the test env instance '`tb_env`', which further instantiates CHI system ENV as '`chi_env`'.
- ❖ The `shared_system_status` object from this CHI system ENV is retrieved through UVM config DB get operation as follows in the test's `end_of_elaboration_phase()`.
- ❖ The same handle can be used to monitor the performance metrics.

```
virtual function void chi_base_test::end_of_elaboration_phase(uvm_phase phase);
    svt_chi_system_status sys_status;
    `uvm_info("end_of_elaboration_phase", "Entered...", UVM_LOW)
    super.end_of_elaboration_phase(phase);
    svt_config_object_db#(svt_chi_system_status)::get(this.tb_env.chi_env, "",
"shared_system_status", sys_status);
    if (active_sys_status != null)
        `uvm_info("connect_phase", {"system status: \n", sys_status.sprint()}, UVM_LOW)
    `uvm_info("end_of_elaboration_phase", "Exiting...", UVM_LOW)
endfunction
```

3.11.2.4 Reporting Latencies, L3, and Snoop Filter Metrics for a Given Rn Transaction

CHI system monitor creates and updates system transactions that encapsulate a RN transaction, associated snoop transactions, slave transaction, L3 and SF metrics, and various associated latencies.

See [System Monitor Summary Report With Performance Metrics](#) for more details.

3.11.3 Performance Metrics

This section describes the different types of performance metrics that are measured by CHI system monitor through `svt_chi_system_status` and its sub objects.

All these metrics are measured on per HN basis.

3.11.3.1 Throughput

Throughput measures of the performance of the home nodes, within the CHI interconnect, in terms of different parameters like- rate of requests retry, data band width and so on.

The unit of the throughput measurement is 'MB/sec'.

3.11.3.1.1 Definitions

- ❖ Number of requests: All the requests except retried requests OR `PcrdReturn` type requests
- ❖ Number of retries: The number of requests that received `RETRYACK` response
- ❖ Retry rate from a given HN to RNs
 - ◆ $(\text{Number of retries to RNs} / \text{number of Requests from RNs}) * 100$
- ❖ Retry rate from Slaves (memory) to a given HN
 - ◆ $(\text{Number of retries from Slaves} / \text{number of Requests to Slaves}) * 100$

- ❖ Number of requests from RNs to a given HN for each possible value of QoS
- ❖ Number of retries from a given HN to RNs for every each possible value of QoS
- ❖ Number of requests from a given HN to Slaves (memory) for each possible value of QoS
- ❖ Number of retries from Slaves (memory) for each possible value of QoS
- ❖ Bandwidth Utilization: data band width utilization on Tx and Rx paths as per HN basis
 - ◆ (number of data bytes = number of DAT flits * DAT VC data size) / (simulation time)
 - ◆ This is reported in MB/sec units
 - ◆ At a given HN, the following data bandwidth metrics are computed:
 - ◇ Data bandwidth from RNs to a given HN
 - ◇ Data bandwidth from a given HN to RNs
 - ◇ Data bandwidth from a given HN to Slaves
 - ◇ Data bandwidth from Slaves to a given HN

3.11.3.1.2 Notes

- ❖ ACE-Lite masters connected to CHI interconnect through RN-I bridges:
 - ◆ The RETRY responses from HNs to ACE-Lite masters are not visible on the ACE-Lite interface. So the RETRY responses on this path are not tracked
 - ◆ The width of QoS field of ACE-Lite masters is assumed to be the same as that of RN-I bridge.
- ❖ AXI/ACE-Lite slaves connected to CHI interconnect through SN bridges:
 - ◆ The RETRY responses from SN bridges to HNs are not visible on the AXI/ACE-Lite slave interface. So the RETRY responses on this path are not tracked
 - ◆ The width of QoS field of AXI/ACE-Lite slave is assumed to be the same as that of SN bridge.
- ❖ Point of reference for bandwidth computation
 - ◆ The simulation time at which a given HN observes the very first request from any RN is considered as the starting point of reference for bandwidth computation between that HN and the RNs it is interacting with
 - ◆ The simulation time at which a given HN initiates the very first request to any SN is considered as the starting point of reference for bandwidth computation between that HN and the SNs it is interacting with
- ❖ Unit of bandwidth metrics is 'MB/sec'
- ❖ Effect of dynamic reset on data bandwidth computation
 - ◆ The duration of dynamic reset is excluded for data bandwidth computation as inactivity period

3.11.3.1.3 Details

The following are the throughput related parameters that are present in the class `svt_chi_hn_status`.

Within the class reference of `svt_chi_hn_status`:

- ❖ All throughput related metrics are grouped under the following heading: 'Throughput metrics'
- ❖ Further, each of the category of throughput metrics are grouped as follows:
 - ◆ QoS Throughput metrics
 - ◇ `int num_qos_reqs_from_rn [SVT_CHI_MAX_QOS_VALUE]`

- ❖ `int num_qos_reqs_to_sn [`SVT_CHI_MAX_QOS_VALUE]`
- ❖ `int num_qos_retries_from_sn [`SVT_CHI_MAX_QOS_VALUE]`
- ❖ `int num_qos_retries_to_rn [`SVT_CHI_MAX_QOS_VALUE]`
- ◆ **Bandwidth Throughput metrics**
 - ❖ `"real hn_to_rn_data_bw`
 - ❖ `"real hn_to_sn_data_bw`
 - ❖ `"real rn_to_hn_data_bw`
 - ❖ `"real sn_to_hn_data_bw`
- ◆ **Retry Rate Throughput metrics**
 - ❖ `real hn_retry_to_rn_req_rate`
 - ❖ `"real sn_retry_to_hn_req_rate`

The following are the details of these parameters. For other cross reference attributes mentioned below, see the HTML class reference.

3.11.3.1.4 QoS Throughput Metrics

int attribute

`svt_chi_hn_status::num_qos_reqs_from_rn [`SVT_CHI_MAX_QOS_VALUE]`

- ❖ Indicates the number of requests initiated from RNs to HN for each possible QoS values in the range `[0: `SVT_CHI_MAX_QOS_VALUE]`
- ❖ This does not include the retried transactions OR the `PcrdReturn` type transactions.
- ❖ The width of QoS of ACE-Lite masters is assumed to be same as connected RN-I bridge.

int attribute

`svt_chi_hn_status::num_qos_retries_to_rn [`SVT_CHI_MAX_QOS_VALUE]`

- ❖ Indicates the number of retry responses generated by a given HN to RNs for each possible QoS value in the range `[0: `SVT_CHI_MAX_QOS_VALUE]`
- ❖ This does not include the retried transactions OR the `PcrdReturn` type transactions.
- ❖ The width of QoS of ACE-Lite masters is assumed to be same as connected RN-I bridge.

int attribute

`svt_chi_hn_status::num_qos_reqs_to_sn [`SVT_CHI_MAX_QOS_VALUE]`

- ❖ Indicates the number of requests initiated from a given HN to SNs for each possible QoS value in the range `[0: `SVT_CHI_MAX_QOS_VALUE]`
- ❖ This does not include the retried transactions OR the `PcrdReturn` type transactions.
- ❖ The width of QoS of AXI/ ACE-Lite slaves is assumed to be same as connected SN bridge.

int attribute

`svt_chi_hn_status::num_qos_retries_from_sn [`SVT_CHI_MAX_QOS_VALUE]`

- ❖ Indicates the number of retry responses generated by SNs to a given HN for each possible QoS value in the range `[0: `SVT_CHI_MAX_QOS_VALUE]`

- ❖ This does not include the retried transactions OR the PcrdReturn type transactions.
- ❖ The width of QoS of AXI/ACE-Lite slaves is assumed to be same as connected SN bridge.

3.11.3.1.5 Bandwidth Throughput Metrics

real attribute

svt_chi_hn_status::rn_to_hn_data_bw

- ❖ Indicates the RNs to HN data bandwidth utilization in MB/sec from svt_chi_hn_status :: ref_time_for_rn_hn_bw
- ❖ This does not include any SnpData flits.
- ❖
$$\text{rn_to_hn_data_bw} = (\text{num_data_bytes_from_rn} / (\text{current simulation time} - \text{svt_chi_hn_status} :: \text{ref_time_for_rn_hn_bw} - \text{svt_chi_hn_status} :: \text{inactive_period})) * (\text{timeunit_factor})$$

real attribute

svt_chi_hn_status::hn_to_rn_data_bw

- ❖ Indicates the HN to RNs data bandwidth utilization in MB/sec from svt_chi_hn_status :: ref_time_for_rn_hn_bw
- ❖
$$\text{hn_to_rn_data_bw} = (\text{num_data_bytes_to_rn} / (\text{current simulation time} - \text{svt_chi_hn_status} :: \text{ref_time_for_rn_hn_bw} - \text{svt_chi_hn_status} :: \text{inactive_period})) * (\text{timeunit_factor})$$

real attribute

svt_chi_hn_status::hn_to_sn_data_bw

- ❖ Indicates the HN to SNs data bandwidth utilization from svt_chi_hn_status :: ref_time_for_hn_sn_bw
- ❖
$$\text{hn_to_sn_data_bw} = (\text{num_data_bytes_to_sn} / (\text{current simulation time} - \text{svt_chi_hn_status} :: \text{ref_time_for_hn_sn_bw} - \text{svt_chi_hn_status} :: \text{inactive_period})) * (\text{timeunit_factor})$$

real attribute

svt_chi_hn_status::sn_to_hn_data_bw

- ❖ Indicates the HN to SNs data bandwidth utilization in MB/sec from svt_chi_hn_status :: ref_time_for_hn_sn_bw
- ❖
$$\text{sn_to_hn_data_bw} = (\text{num_data_bytes_from_sn} / (\text{current simulation time} - \text{svt_chi_hn_status} :: \text{ref_time_for_hn_sn_bw} - \text{svt_chi_hn_status} :: \text{inactive_period})) * (\text{timeunit_factor})$$

3.11.3.1.6 Retry Rate Throughput Metrics

real attribute

svt_chi_hn_status::hn_retry_to_rn_req_rate

- ❖ Indicates the throughput metric for HN Retry to RN Request rate

- ❖ This doesn't include any RETRY responses to ACE-Lite master that is connected to interconnect through RN-I bridge within interconnect.
- ❖ $hn_retry_to_rn_req_rate = (num_retries_to_rn / num_reqs_from_rn) * 100$

real attribute

`svt_chi_hn_status::sn_retry_to_hn_req_rate`

- ❖ Indicates the throughput metric for SNs Retry to HN Request rate
- ❖ This doesn't include any RETRY responses from SN bridge within interconnect that is connected to AXI/ACE-Lite slave.
- ❖ $sn_retry_to_hn_req_rate = (num_retries_from_sn / num_reqs_to_sn) * 100$

3.11.3.2 L3 and Snoop Filter (SF) Metrics

Every HN-F within the CHI interconnect can contain a L3 cache and a Snoop Filter (SF). So, one of the performance metrics of a given HN-F would be the L3 and Snoop Filter metrics.

The L3 and SF metrics are governed by L3 and SF accesses and are measured in terms of L3 and SF hit/miss rate.

These metrics are measured on per HN-F basis.

3.11.3.2.1 Definitions

Every snoopable transaction received at a HN-F is considered for both L3, SF hit/miss rate calculation. For such transactions received at HN-F, both the L3 and SF lookup happens simultaneously.

3.11.3.2.2 L3 Performance Metrics

- ❖ L3 cache access:
 - ◆ The following request types result in L3 cache accesses:
 - ◇ All requests except write and CopyBack type
 - ◇ Write and CopyBack requests with memory attribute 'allocate hint' set to 1
- ❖ L3 cache miss:
 - ◆ When L3 access occurs for a given address at a given HN-F, then either of the following cases separately OR together is considered as L3 cache miss event:
 - ◇ A corresponding slave memory access from the same HN-F is initiated and completed before responding to initiating RN OR
 - ◇ Corresponding snoop requests are initiated from the same HN-F to peer RN-Fs and completed before responding to initiating RN
- ❖ L3 cache hit:
 - ◆ When L3 access occurs for a given address at a given HN-F, then both of the following conditions together is considered as L3 cache hit:
 - ◇ No corresponding slave memory access from the same HN-F is initiated and completed before responding to initiating RN
 - ◇ No corresponding snoop requests are initiated from the same HN-F to peer RN-Fs and completed before responding to initiating RN
- ❖ L3 cache hit rate:

- ◆ At a given HN-F: $(\text{total L3 cache hits} / \text{total L3 accesses}) * 100$
- ❖ L3 cache miss rate:
 - ◆ At a given HN-F: $(\text{total L3 cache misses} / \text{total L3 accesses}) * 100$

3.11.3.2.3 Snoop Filter Performance Metrics

- ❖ Snoop filter access:
 - ◆ At a given HN-F, every snoopable request received from RNs is considered as Snoop filter access for that address
- ❖ Snoop filter miss:
 - ◆ When snoop filter access occurs for a given address at a given HN-F, both the following conditions together is considered as Snoop filter miss:
 - ◇ No corresponding snoop requests are initiated from the same HN-F to peer RN-Fs and completed before responding to initiating RN.
 - ◇ Corresponding slave memory access from the same HN-F is initiated and completed before responding to initiating RN.
- ❖ Snoop filter hit:
 - ◆ When Snoop filter access occurs for a given address at a given HN-F, following event is considered as snoop filter hit:
 - ◇ Corresponding snoop requests are initiated from the same HN-F to peer RN-Fs and completed before responding to initiating RN
- ❖ Snoop filter hit rate:
 - ◆ At a given HN-F: $(\text{total Snoop filter hits} / \text{total Snoop filter accesses}) * 100$
- ❖ Snoop filter miss rate:
 - ◆ At a given HN-F: $(\text{total snoop filter misses} / \text{total snoop filter accesses}) * 100$

3.11.3.2.4 Notes

The following enumerated data types are defined to represent L3 and Snoop filter metrics for a given system transaction that is updated by CHI system monitor.

3.11.3.2.5 Updating SNOOP Filter Metrics for a Given System Transaction

```
typedef enum svt_chi_hn_status::sf_access_type_enum
```

This enum type is defined to count Snoop Filter metrics for the snoopable transactions received from RNs at a given HN-F

The following is the algorithm used for a given transaction:

- ❖ By default: `sf_access_type = SF_ACCESS_NA`
- ❖ If SF access is applicable and slave transaction association is applicable
 - ◆ Update: `sf_access_type = SF_ACCESS`
 - ◆ If snoops are generated
 - ◇ Update: `sf_access_type = SF_HIT`
 - ◆ If snoops are not generated and a slave transaction is associated before response to initiating RN is sent

❖ Update: `sf_access_type = SF_MISS`

`SF_MISS`: Indicates SF miss

`SF_HIT`: Indicates SF hit

`SF_ACCESS`: Indicates SF access

`SF_ACCESS_NA`: Indicates SF access is not applicable

3.11.3.2.6 Updating L3 Metrics for a Given System Transaction

```
typedef enum svt_chi_hn_status::l3_access_type_enum
```

This enum type is defined to measure L3 metrics for the transactions received from RNs at a given HN-F.

The following is the algorithm used for a given transaction:

- ❖ By default `l3_access_type = L3_ACCESS_NA`
- ❖ If L3 access is applicable and slave transaction association is applicable
 - ◆ Update: `l3_access_type value = L3_ACCESS`
 - ◆ For `ReadNoSnp`, `WriteNoSnp`, `WriteBack`, `WriteClean`, and `WriteEvict` transactions
 - ❖ If memory is accessed
 - Update: `l3_access_type = L3_MISS`
 - ❖ Else
 - Update: `l3_access_type = L3_HIT`
 - ◆ For `ReadOnce` transaction
 - ❖ If memory is accessed OR snoops are generated before responding to initiating RN
 - Update: `l3_access_type = L3_MISS`
 - ❖ If memory is not accessed AND no snoops are generated before responding to initiating RN
 - Update: `l3_access_type = L3_HIT`
 - ◆ For `ReadUnique` and `WriteUnique` transactions
 - ❖ If snoops are generated and snoops responses include the data before responding to initiating RN OR memory is accessed
 - Update: `l3_access_type = L3_MISS`
 - ❖ If (snoops are generated and snoops responses doesn't include the data before responding to initiating RN OR no snoops are generated before responding to initiating RN) AND memory is not accessed
 - Update: `l3_access_type = L3_HIT`
 - ◆ For `ReadClean` and `ReadShared` transactions
 - ❖ If snoops are generated and snoop responded with data OR memory is accessed
 - Update: `l3_access_type = L3_MISS`
 - ❖ If no snoops are generated before responding to initiating RN OR memory is not accessed
 - Update: `l3_access_type = L3_HIT`
 - ◆ For `CleanUnique`, `MakeUnique`, and `Evict` transactions
 - ❖ If memory is accessed AND snoops are generated before responding to initiating RN
 - Update: `l3_access_type = L3_MISS`

- ❖ If memory is not accessed AND snoops are generated before responding to initiating RN
Update: l3_access_type = L3_HIT
 - ◆ For CleanShared, MakeInvalid, and CleanInvalid transactions
 - ❖ If snoops are generated before responding to initiating RN AND snoops responses include the data before responding to initiating RN AND memory is accessed
Update: l3_access_type = L3_MISS
 - ❖ If snoops are generated before responding to initiating RN AND snoops responses doesn't include the data before responding to initiating RN AND memory is not accessed
Update: l3_access_type = L3_HIT _
- L3_MISS: Indicates L3 cache miss
 L3_HIT: Indicates L3 cache hit
 L3_ACCESS: Indicates L3 cache access
 L3_ACCESS_NA: Indicates L3 cache access is not applicable

3.11.3.2.7 Details

The following are the L3 and Snoop filter metrics that are present in the class `svt_chi_system_hn_status`, grouped under the header 'L3 cache and Snoop filter metrics' in the HTML class reference.

- ❖ `real l3_cache_hit_rate`
- ❖ `real l3_cache_miss_rate`
- ❖ `int num_l3_cache_accesses`
- ❖ `int num_l3_cache_hit_events`
- ❖ `int num_l3_cache_miss_events`
- ❖ `int num_snp_filter_accesses`
- ❖ `int num_snp_filter_misses`
- ❖ `real snp_filter_hit_rate`
- ❖ `real snp_filter_miss_rate`

The following are the L3 and SF hit and miss rate descriptions. For other parameters, see the HTML class reference.

real attribute

`svt_chi_system_hn_status::l3_cache_hit_rate`

- ❖ Indicates L3 cache hit rate at the HN-F.
- ❖ $\text{l3_cache_hit_rate} = (\text{num_l3_cache_hit_events} / \text{num_l3_cache_accesses}) * 100$

real attribute

`svt_chi_system_hn_status::l3_cache_miss_rate`

- ❖ Indicates L3 cache miss rate at the HN-F
- ❖ $\text{l3_cache_miss_rate} = (\text{num_l3_cache_miss_events} / \text{num_l3_cache_accesses}) * 100$

real attribute

`svt_chi_system_hn_status::snp_filter_hit_rate`

- ❖ Indicates snoop filter hit rate at the HN-F.
- ❖ $\text{snp_filter_hit_rate} = (\text{num_snp_filter_hits} / \text{num_snp_filter_accesses}) * 100$

real attribute

`svt_chi_system_hn_status::snp_filter_miss_rate`

- ❖ Indicates snoop filter miss rate at the HN-F.
- ❖ $\text{snp_filter_miss_rate} = (\text{num_snp_filter_misses} / \text{num_snp_filter_accesses}) * 100$

3.11.3.3 Latency Across Paths

When an RN initiates a request to interconnect, there are different aspects involved before the response is sent back to initiating RN by HN. The overall latency of the transaction at RN is composed of the different other latencies at interconnect, snooped RNs, accessed slaves. These metrics are used to measure these latencies.

All latency metrics are measured on per HN basis.

The time unit info of the latency metrics is available through `svt_chi_system_status::timeunit_string`

3.11.3.3.1 Definitions

- ❖ Transaction Latency: Latency is defined as the total time taken for the completion of a transaction
 - ◆ Latency of a transaction = end time of the transaction - begin time of the transaction
 - ◆ Average = total latency of transactions received from various RNs / number of completed RN transactions
- ❖ Associated Latencies:
 - ◆ Includes some or all of the following at the targeted HN within the interconnect:
 - ❖ Coherent request reception to Snoop request generation latency

Snoop request generation latency = Start Time of the first Snoop Request - End Time of the Coherent Request

Average = total snoop request generation latency / number of snoopable RN transactions received
 - ❖ Snoop response to Coherent response generation latency

Snoop response to coherent response generation latency = start time of coherent response - end time of last snoop response

Average = total snoop response to coherent response generation latency / number of snoopable RN transactions received
 - ❖ Request reception to Memory access request generation latency

Slave request generation latency = start time of slave transaction - start time of RN transaction

Average = total slave request generation latency / number of memory access transactions corresponding to RN transactions
 - ❖ Memory access response to response to RN generation latency

Memory access to RN response generation latency = end time of response to RN - end time of slave transaction

Average = total memory access to RN response generation latency / number of memory access transactions corresponding to RN transactions

- ❖ Coherent request reception to Coherent response generation post L3 cache hit latency
 Response generation latency after L3 hit = start time of response to RN - start time of request from RN
 Average = total response generation latency after L3 hit / number L3 hits
- ◆ At each of the snooped RNs:
 - ❖ Snoop response generation latency, i.e., Snoop transaction latency
 Snoop transaction latency = end time of snoop transaction - start time of snoop transaction
 Average = total snoop response generation latency / number of snoop requests observed at that snooped RN
- ◆ At the each of the accessed Slave:
 - ❖ Memory access response generation latency, i.e., Slave transaction latency
 Slave transaction latency = end time of memory transaction - start time of memory access transaction
 Average = total slave transaction latency / number of transactions observed at that slave

3.11.3.3.2 Notes

- ❖ The above described latencies are not yet supported for DVM, Barrier transactions
- ❖ The time unit info of the metrics is available through `svt_chi_system_status::timeunit_string`

3.11.3.3.3 Details

The following are the parameters defined in the class `svt_chi_system_hn_status` to capture latency information on a per HN basis. Note that `svt_chi_system_hn_status` extends `svt_chi_hn_status`.

Within the HTML class reference of the class `svt_chi_system_status`:

- ❖ All the attributes related to average latency metrics are grouped under the following heading: 'Averages of Latency related metrics'
- ❖ All the attributes related to total latency, average latency and related parameters are grouped under the following heading: 'Latency related metrics'

The following is the list of average latency metrics related attributes:

- ❖ `real average_transaction_latency`
- ❖ `real average_rsp_gen_latency_for_l3_hit`
- ❖ `real average_snoop_request_gen_latency`
- ❖ `real average_snoop_response_to_coh_response_gen_latency`
- ❖ `real average_rn_snoop_response_gen_latency []`
- ❖ `real average_slave_req_gen_latency`
- ❖ `real average_mem_access_to_coherent_response_gen_latency`
- ❖ `real average_slave_xact_latency []`

See the HTML class reference for more details.

real attribute

`svt_chi_system_hn_status::average_transaction_latency`

- ❖ Average latency for transactions initiated from RNs observed at a given HN

- ❖ $\text{average_transaction_latency} = (\text{svt_chi_hn_status} :: \text{total_xact_latency} / \text{svt_chi_hn_status} :: \text{num_completed_xacts})$

real attribute

`svt_chi_system_hn_status::average_rsp_gen_latency_for_l3_hit`

- ❖ For the coherent transactions initiated to the HN, Average Time taken by the HN to generate the response to initiating RN when L3 Cache is hit.
- ❖ This does not involve the retried and PCreditReturn type transactions.
- ❖ $\text{average_rsp_gen_latency_for_l3_hit} = (\text{total_rsp_gen_latency_for_l3_hit} / (\text{num_l3_cache_accesses} - \text{num_l3_cache_miss_events}))$

real attribute

`svt_chi_system_hn_status::average_snoop_request_gen_latency`

- ❖ Average Time taken by the HN to generate the first Snoop request after receiving a Coherent request from initiating RN.
- ❖ This does not involve the retried, non-coherent and PCreditReturn type transactions.
- ❖ $\text{average_snoop_request_gen_latency} = (\text{total_snoop_req_gen_latency} / \text{svt_chi_hn_status} :: \text{num_snoopable_xacts})$

real attribute

`svt_chi_system_hn_status::average_snoop_response_to_coh_response_gen_latency`

- ❖ For the coherent transactions initiated to the HN, indicates the average Time taken by the HN to generate the first Coherent response after receiving the last Snoop response from snooped RN.
- ❖ This does not involve the retried, non-coherent and PCreditReturn type transactions.
- ❖ $\text{average_snoop_response_to_coh_response_gen_latency} = (\text{total_snp_rsp_to_coh_rsp_gen_latency} / \text{svt_chi_hn_status} :: \text{num_snoopable_xacts})$

real attribute

`svt_chi_system_hn_status::average_rn_snoop_response_gen_latency[]`

- ❖ Average Time taken by the snooped RNs to generate the Snoop response after receiving a Snoop request
- ❖ for each of the snooped RNs: $\text{average_rn_snoop_response_gen_latency} [] = (\text{total_rn_snoop_rsp_gen_latency} [] / \text{num_rn_snoop_rsp} [])$

real attribute

`svt_chi_system_hn_status::average_slave_req_gen_latency`

- ❖ Average Time taken by the HN to generate a Slave(Memory) Request
- ❖ $\text{average_slave_req_gen_latency} = (\text{total_slave_req_gen_latency} / \text{num_slave_req} = 0;)$

real attribute

`svt_chi_system_hn_status::average_mem_access_to_coherent_response_gen_latency`

- ❖ For CHI RN transactions involving Memory Access that are initiated to the HN, indicates average Time taken by the HN to generate the first Coherent response to initiating RN after the associated Slave memory access transaction is complete.
- ❖ Not applicable in case the memory access transaction is complete after the HN transmits the response to initiating RN.
- ❖ $\text{average_mem_access_to_coherent_response_gen_latency} = (\text{total_mem_access_to_coh_rsp_gen_latency} / \text{num_mem_access_for_coh_xacts} = 0;)$

real attribute

`svt_chi_system_hn_status::average_slave_xact_latency[]`

Average Time consumed at each of the Slave Nodes

For each of the slaves: $\text{average_slave_xact_latency}[] = (\text{total_slave_xact_latency}[] / \text{num_xact_per_slave}[])$

3.11.3.3.4 System Monitor Summary Report With Performance Metrics

CHI system monitor reports the 'coherent and snoop transaction summary' for a given RN transaction. This is updated to further indicate the associated latencies, L3, SF metrics and associated slave transaction details when performance tracking is enabled.

See the section on '[Reporting Latencies, L3, and Snoop Filter Metrics for a Given Rn Transaction](#) on' for examples.

Example:

- ❖ MakeInvalid initiated from one of the RN-I ports, where the address is present in one of the snooped RN-Fs; and only that RN-F is snooped. So, this is a SF hit. As it's a snoopable transaction, L3 is still accessed.
- ❖ The associated latencies are also displayed.

```
*****
*****
XACT SUMMARY for {SYS_ID(0) OBJ_NUM(0) NODE_ID(0) TYPE(MAKEINVALID) TXN_ID(0)
ADDR(83c3edf4185) SIZE(SIZE_64BYTE) START_TIME(1503000000) END_TIME(2375500000)} :
*****
*****
INIT_STATE:I | FINAL_STATE:I | RESP_FINAL_STATE:I
Sequence Path: <sequence_name>.cmo_seq1[x].cmo_tran
XACT DATA:
-----
ASSOCIATED SNOOP TRANSACTIONS:
SNP XACT SUMMARY for {SYS_ID(0) OBJ_NUM(0) NODE_ID(6) TYPE(SNPMMAKEINVALID) TXN_ID(0)
ADDR(83c3edf4180) START_TIME(1503100000) END_TIME(1503900000)} :
INIT_STATE:UD | FINAL_STATE:I | PD:0 | IS_SHARED:0
SNOOP DATA:
.....
-----
```

INITIAL CACHE LINE CONTENTS:

PORT	NODE	INDEX	ADDR	STATUS	AGE	DATA
RN 0	7	---	83c3edf4185	INVALID		
RN 1	6	46	0000083c3edf4180	UD	0	0 cbad24dc_657e3d7c_cdb3fe76_c4ff54e4_50007508_6e1a5115_c8240a6e_c2dd0342_7111d715_faf090a4_e3357648_e8d787cd_6780afdc_b13a4dd2_90b88919_d626cd29
RN 2	5	---	83c3edf4185	INVALID		
RN 3	4	---	83c3edf4185	INVALID		

FINAL CACHE LINE CONTENTS:

PORT	NODE	INDEX	ADDR	STATUS	AGE	DATA
RN 0	7	---	83c3edf4185	INVALID		
RN 1	6	---	83c3edf4185	INVALID		
RN 2	5	---	83c3edf4185	INVALID		
RN 3	4	---	83c3edf4185	INVALID		

SYSTEM MONITOR L3/MEMORY CONTENT AFTER TRANSACTION END:

SYSTEM MONITOR HN PERFORMANCE METRICS - L3, SF, Transaction Latency :

HN details	:	hn_node_idx 3	hn_node_id 11	hn_type HN_F
L3, SF metrics	:	L3_ACCESS	SF_HIT	
Xact latency	:	xact_latency 872500.000000		
		snp_req_gen_latency 100.000000	snp_rsp_to_coh_rsp_gen_latency	
100.000000			snp_response_gen_latency[1] 800.000000	

Example 2:

- ❖ CleanInvalid is initiated from one of the RN-Fs, but no snoops are generated as none of the RN-Fs have this address in their caches; however, the content from L3 is written to memory before original request is complete.
- ❖ The associated slave transaction and related latencies are displayed.

```
*****
*****
```

```
XACT SUMMARY for {SYS_ID(0) OBJ_NUM(0) NODE_ID(4) TYPE(CLEANINVALID) TXN_ID(0)
ADDR(b4ccfa213ae) SIZE(SIZE_64BYTE) START_TIME(1502300000) END_TIME(2375950000)} :
```

```
*****
*****
```

```
INIT_STATE:I|FINAL_STATE:I|RESP_FINAL_STATE:I
```

```
Sequence Path: <sequence_name>.cmo_seq2[x].cmo_tran
```

XACT DATA:

INITIAL CACHE LINE CONTENTS:

PORT	NODE	INDEX	ADDR	STATUS	AGE	DATA
RN 0	7	---	b4ccfa213ae	INVALID		
RN 1	6	---	b4ccfa213ae	INVALID		
RN 2	5	---	b4ccfa213ae	INVALID		
RN 3	4	---	b4ccfa213ae	INVALID		

FINAL CACHE LINE CONTENTS:

PORT	NODE	INDEX	ADDR	STATUS	AGE	DATA
RN 0	7	---	b4ccfa213ae	INVALID		
RN 1	6	---	b4ccfa213ae	INVALID		
RN 2	5	---	b4ccfa213ae	INVALID		
RN 3	4	---	b4ccfa213ae	INVALID		

SYSTEM MONITOR L3/MEMORY CONTENT AFTER TRANSACTION END:

140ac2ca_e2fe529d_8c04075c_80598430_6db19bf6_0a99fcca_e99bfa18_820a832d_2955dcec_c58b62
c5_f17a9140_d583ae96_4e4a6c78_cfcde95f_f3a272ab_90e34b4e

SYSTEM MONITOR HN PERFORMANCE METRICS - L3, SF, Transaction Latency :

HN details : | hn_node_idx 4 | hn_node_id 12 | hn_type HN_F |
L3, SF metrics : | L3_MISS | SF_MISS |
Xact latency : | xact_latency 873650.000000 |
| slave_req_gen_latency 600.000000 | mem_access_to_coh_rsp_gen_latency
869300.000000 | slave_xact_latency 3750.000000 |

ASSOCIATED SLAVE TRANSACTION SUMMARY :

SN TRANSACTION SUMMARY for{SYS_ID(0) OBJ_NUM(0) NODE_ID(11) TYPE(WRITENOSNPPTL)
TXN_ID(d) ADDR(b4ccfa213ae) SIZE(SIZE_64BYTE) MEM_TYPE(NORMAL) START_TIME(1502900000)
END_TIME(1506650000)} :

node_idx	node_id	obj_num	xact_type	addr	id	data_size	mem_type	start_byte	end_byte	start_time	end_time
1	11	0	WRITENOSNPPTL	b4ccfa213ae	13	SIZE_64BYTE	NORMAL				
0	63	1502900000	1506650000								

DATA BYTE INDICES : ..62..60 ..58..56 ..54..52 ..50..48 ..46..44 ..42..40 ..38..36
..34..32 ..30..28 ..26..24 ..22..20 ..18..16 ..14..12 ..10.. 8 .. 6.. 4 .. 2.. 0

VALID WYSIWYG BE : 1 1 1 1_ 1 1 1 1_ 1 1 1 1_ 1 1 1 1_ 1 1 1 1_ 1 1 1 1_ 1 1 1 1_ 1 1
1 1_ 1 1 1 1_ 1 1 1 1_ 1 1 1 1_ 1 1 1 1_ 1 1 1 1_ 1 1 1 1

VALID WYSIWYG DATA:

140ac2ca_e2fe529d_8c04075c_80598430_6db19bf6_0a99fcca_e99bfa18_820a832d_2955dcec_c58b62
c5_f17a9140_d583ae96_4e4a6c78_cfcde95f_f3a272ab_90e34b4e

3.12 Interleaved Port Support

CHI RN VIP components support interleaved set of addresses. The RN components can be grouped together in interleaved group. RN components within an interleaved group will support a unique non-overlapping set of address ranges. For example, assume that there are 2 RNs within an interleaved group. RN 0 supports address range 0 to 63, RN 1 supports address range 64 to 127, RN 0 supports address range 128 to 191, RN 1 supports address range 192 to 255, and so on. What it means is that RN 0 would generate transactions with addresses 0 to 63, 128 to 191 and so on. The interconnect would snoop RN 0 for addresses 0 to 63, 128 to 191 and so on.

3.12.1 Active RN agent

In active mode, RN VIP checks if the coherent address generated by it falls in the interleaving address range. If the address does not lie in the interleaving address range for this RN, RN VIP generates error through `is_valid()` check. RN VIP will still transmit the transaction.

CHI node link monitor issues error if it observes snoop transaction for an address is issued to a RN which does not support that address range.

CHI node link monitor issues error if it observes snoop transaction issued within the same interleaved group.

3.12.2 Passive RN agent

In passive mode, Master VIP checks whether the coherent or snoop address observed on this port falls within the interleaving address range. If not, RN VIP generates protocol check error.

CHI node link monitor checks whether the coherent address observed on RN port falls within the interleaving address range. If not, node link monitor generates error.

CHI node link monitor issues error if it observes snoop transaction for an address is issued to a RN which does not support that address range.

CHI node link monitor issues error if it observes snoop transaction issued within the same interleaved group.

3.12.3 Node Configuration Controls

The following configuration members must be programmed to enable this feature. See, CHI Class Reference HTML for more details:

- ❖ `svt_chi_node_configuration::port_interleaving_enable`
- ❖ `svt_chi_node_configuration::port_interleaving_size`
- ❖ `svt_chi_node_configuration::port_interleaving_group_id`
- ❖ `svt_chi_node_configuration::port_interleaving_index`
- ❖ `svt_chi_node_configuration::dvm_sent_from_interleaved_port`
- ❖ `svt_chi_node_configuration::device_xact_sent_from_interleaved_port`
- ❖ `svt_chi_node_configuration::port_interleaving_for_device_xact_enable`

3.12.4 Usage Examples

Scenario 1: Two interleaved CHI RN (64 bytes Interleaving Size)

The following step will help in configuring the VIP:

1. `cust_svt_chi_system_configuration` class that extends `svt_chi_system_configuration` class, must be configured as below:

```
// rn[0]
    rn_cfg[0].port_interleaving_enable = 1;
    rn_cfg[0].port_interleaving_group_id = 2;
    rn_cfg[0].port_interleaving_index = 0;
    rn_cfg[0].port_interleaving_size = 64;
    rn_cfg[0].dvm_sent_from_interleaved_port = 0;
    rn_cfg[0].device_xact_sent_from_interleaved_port = 1; // Device traffic is
sent only on this port
// rn[1]
    rn_cfg[1].port_interleaving_enable = 1;
    rn_cfg[1].port_interleaving_group_id = 2;
    rn_cfg[1].port_interleaving_index = 1;
    rn_cfg[1].port_interleaving_size = 64;
    rn_cfg[1].dvm_sent_from_interleaved_port = 1; // DVM is sent/received only
on this port
```

The VIP will take care of the interleaving based on the above configuration settings. In the above use case 1, VIP will generate error under the following cases:

- ❖ `svt_chi_transaction::addr[6] == 0` address comes on `port_interleaving_index 1`
- ❖ `svt_chi_transaction::addr[6] == 1` comes on `port_interleaving_index 0`

4

Verification Features

4.1 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 from 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 a given CHI node configuration, if `svt_chi_node_configuration::pa_format_type` is set to `svt_xml_writer::FSDB` and

- ❖ `enable_xact_xml_gen` is set to 1, the Verdi PA transactions are linked to the respective node interface signals.
- ❖ `svt_chi_node_configuration::enable_fsm_xml_gen` is set to 1, the link activation/deactivation FSM in Verdi PA is linked to the respective node interface signals.

For Verdi documentation, refer to `Verdi_Transaction_and_Protocol_Debug.pdf`.

**Note**

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

4.1.1 Support for Native Dumping of FSDB

Native FSDB supported in CHI VIP.

- ❖ **FSDB 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`
 - ✧ `-debug_access`

For more information on how to set the FSDB dumping libraries, see Appendix B section in *Linking Novas Files with Simulators and Enabling FSDB Dumping* guide available at: `$VERDI_HOME/doc/linking_dumping.pdf`.
 - ◆ **New configuration parameter added** `svt_chi_system_configuration::pa_format_type` that controls the format of PA output file, is for FSDB generation in `svt_chi_system_configuration.sv`. It is a system configuration variable. Add the following setting in system configuration to enable the generation of 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.rn_cfg[0].enable_fsm_xml_gen = 1;
this.enable_xml_gen = 1;
this.pa_format_type = 1;
```

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 > Transaction and 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.

4.1.2 Interface Signal Grouping feature in Verdi

The CHI VIP supports Verdi Protocol Analyzer to pick the appropriate Interface Signal Grouping feature to Verdi nWave view window based CHI version (CHI A, B, C, D and E) used.

Interface Signal Grouping across different channel flits will be picked appropriately from extension.xml file based on the CHI Compile time Specification Macros `SVT_CHI_ISSUE_<B/C/D/E>_ENABLE`.

This figure shows how the enable macros are dumped in to the FSDB based on the CHI Compile time Specification Macros.

When `SVT_CHI_ISSUE_C_ENABLE` is defined.

Attribute	Value
<input type="checkbox"/> _MACRO_SVT_CHI_ISSUE_E_ENABLE	"not defined"
<input type="checkbox"/> _MACRO_SVT_CHI_ISSUE_D_ENABLE	"not defined"
<input type="checkbox"/> _MACRO_SVT_CHI_ISSUE_C_ENABLE	"defined"
<input type="checkbox"/> _MACRO_SVT_CHI_ISSUE_B_ENABLE	"defined"

When `SVT_CHI_ISSUE_E_ENABLE` is defined.

Attribute	Value
<input type="checkbox"/> _MACRO_SVT_CHI_ISSUE_E_ENABLE	"defined"
<input type="checkbox"/> _MACRO_SVT_CHI_ISSUE_D_ENABLE	"defined"
<input type="checkbox"/> _MACRO_SVT_CHI_ISSUE_C_ENABLE	"defined"
<input type="checkbox"/> _MACRO_SVT_CHI_ISSUE_B_ENABLE	"defined"

Each CHI interface flit signal is composed of several protocol fields, as defined under 'Flit packet definitions' of CHI specification.

It can be difficult to have visibility into each field just by looking at the CHI interface flit signals.

To enable users to have better visibility into individual protocol fields of each flit signal based on different CHI Spec versions, this feature is part of Verdi Protocol Analyzer and Verdi nWave.

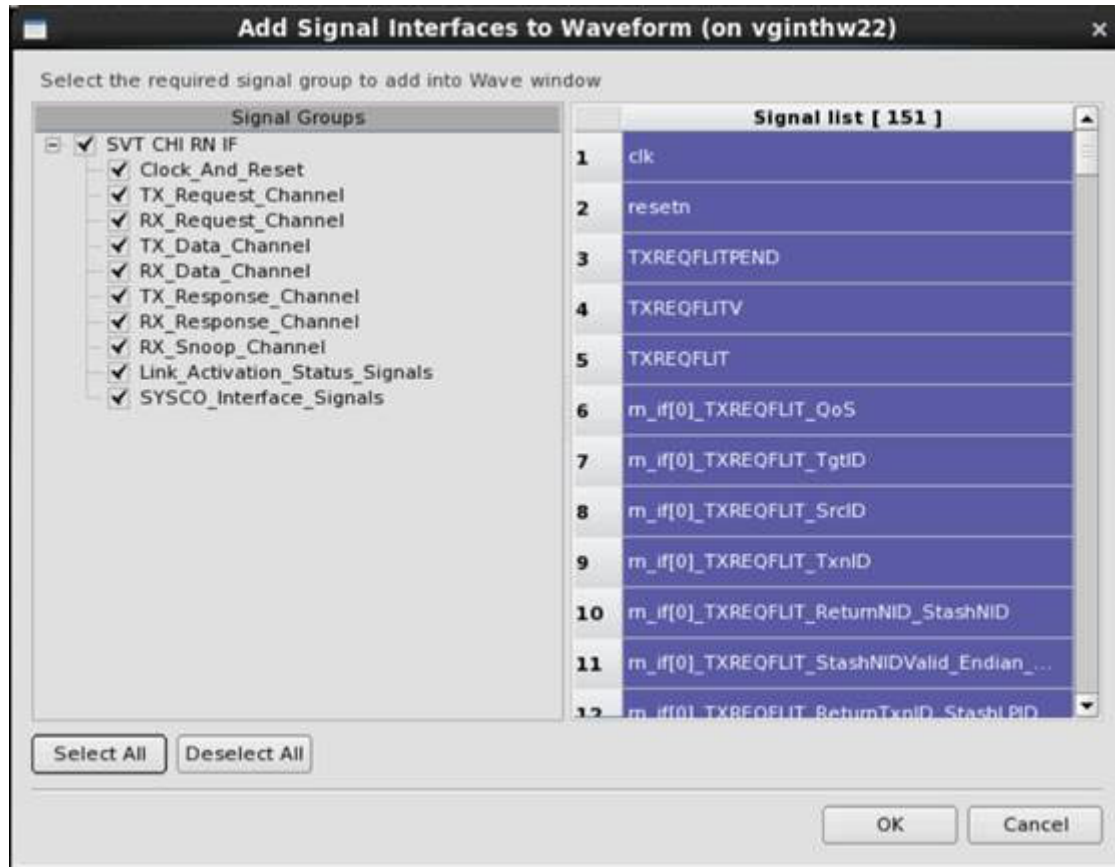
4.1.2.1 Steps to Use in Signal Grouping Feature with Verdi PA and nWave

These steps show how to add signals for a VIP node instance, and how Verdi PA adds the waveform to Verdi nWave, signal grouping based on different channels and subgrouping of flit signals in to protocol flit fields.

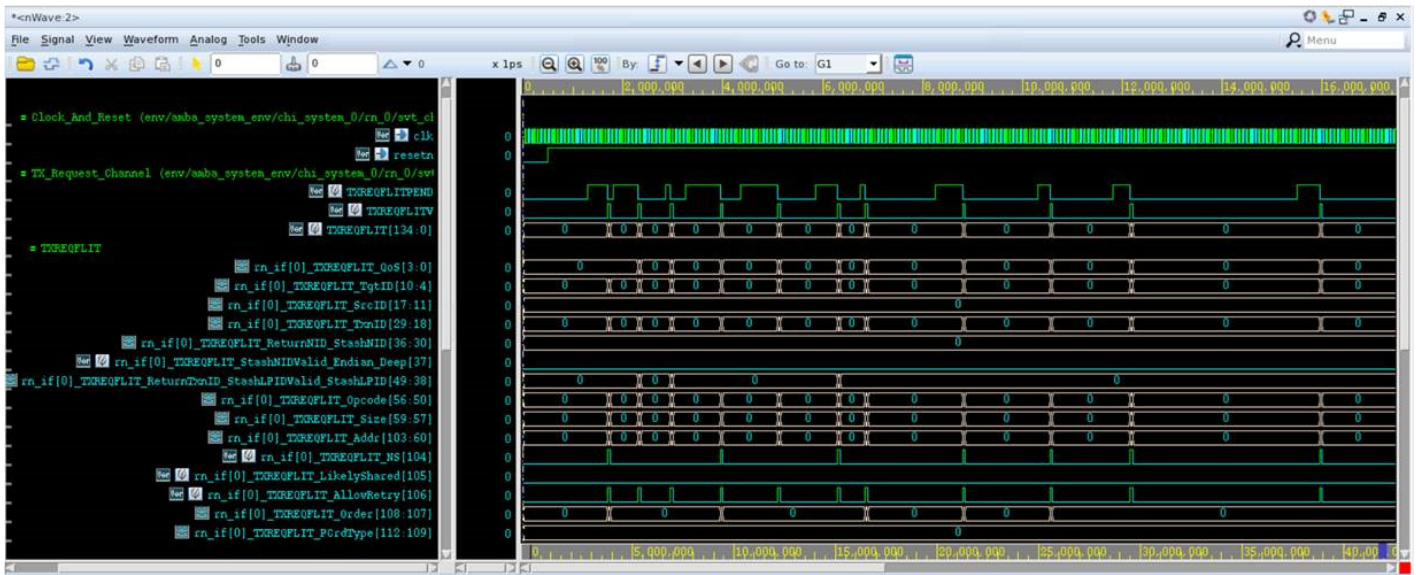
1. From Verdi Protocol Analyzer 'Hier. Tree' tab, right click on the desired CHI node instance (in this case rn_0) -> Select Add to Waveform option -> Select New Waveform (to add in a new nWave window) or Add to Wave * (to add in an existing nWave window).



2. Select the required signals and click OK.



3. Once the selected signals are loaded in the Verdi nWave view, you can see the related signals on expanding each signal group. Further, on expanding the flit packet definition, you can see that the flit signals have been grouped into readable protocol flit fields with their signal size. In this figure, you can see the TXREQ Channel signal grouping and the TXREQFLIT fields grouping.



4.1.2.2 Supported VIP Versions

vip_amba_svt_Q-2020.06 and later versions.

4.1.2.3 Supported Verdi Versions

Verdi 19.06, Verdi 20.03 or later versions.



Note

For earlier Verdi versions, Interface Signal Grouping is Compatible to CHI Issue D.

4.2 Native VERDI Performance Analyzer support

CHI VIP supports native VERDI performance analyzer from VERDI L-2016.06 release onwards.

Note that this requires enabling of native dumping of PA FSDB into VERDI as described in section [Support for Native Dumping of FSDB](#).

The following are the metrics supported:

- ❖ chi_cinst_request_read_bus_bandwidth_percentage
- ❖ chi_cinst_request_read_bus_bandwidth
- ❖ chi_cinst_request_read_byte_count
- ❖ chi_cinst_request_read_count
- ❖ chi_cinst_request_read_percentage
- ❖ chi_cinst_request_write_bus_bandwidth_percentage
- ❖ chi_cinst_request_write_bus_bandwidth
- ❖ chi_cinst_request_write_byte_count
- ❖ chi_cinst_request_write_count
- ❖ chi_cinst_request_write_percentage

- ❖ `chi_ctrans_avg_request_read_latency`
- ❖ `chi_ctrans_avg_request_write_latency`
- ❖ `chi_ctrans_max_request_read_latency`
- ❖ `chi_ctrans_max_request_write_latency`
- ❖ `chi_ctrans_min_request_read_latency`
- ❖ `chi_ctrans_min_request_write_latency`
- ❖ `chi_ctrans_request_read_byte_count`
- ❖ `chi_ctrans_request_read_count`
- ❖ `chi_ctrans_request_write_byte_count`
- ❖ `chi_ctrans_request_write_count`
- ❖ `chi_trans_CleanInvalid_latency`
- ❖ `chi_trans_CleanShared_latency`
- ❖ `chi_trans_CleanUnique_latency`
- ❖ `chi_trans_ECBarrier_latency`
- ❖ `chi_trans_EOBarrier_latency`
- ❖ `chi_trans_Evict_latency`
- ❖ `chi_trans_MakeInvalid_latency`
- ❖ `chi_trans_MakeUnique_latency`
- ❖ `chi_trans_ReadClean_latency`
- ❖ `chi_trans_ReadNoSnp_latency`
- ❖ `chi_trans_ReadOnce_latency`
- ❖ `chi_trans_ReadShared_latency`
- ❖ `chi_trans_ReadUnique_latency`
- ❖ `chi_trans_request_read_byte_count`
- ❖ `chi_trans_request_read_latency`
- ❖ `chi_trans_request_write_byte_count`
- ❖ `chi_trans_request_write_latency`
- ❖ `chi_trans_WriteBackFull_latency`
- ❖ `chi_trans_WriteBackPtl_latency`
- ❖ `chi_trans_WriteCleanFull_latency`
- ❖ `chi_trans_WriteCleanPtl_latency`
- ❖ `chi_trans_WriteEvictFull_latency`
- ❖ `chi_trans_WriteNoSnpFull_latency`
- ❖ `chi_trans_WriteNoSnpPtl_latency`
- ❖ `chi_trans_WriteUniqueFull_latency`
- ❖ `chi_trans_WriteUniquePtl_latency`

- ❖ `chi_ctrans_request_read_bus_bandwidth`
- ❖ `chi_ctrans_request_write_bus_bandwidth`

4.3 Built-in Performance Monitoring Unit Within CHI Agents

CHI VIP (`svt_chi_rn_agent`, `svt_chi_sn_agent`) monitors the performance of a system, based on configuration parameters.

For more information on properties related to Performance Analysis, see the Performance Analysis configuration parameters section in `svt_chi_node_configuration` class in the CHI Class Reference.

By default, performance monitoring is disabled. The CHI VIP agents must be configured to enable the performance monitoring. The CHI VIP RN and SN agent's node protocol monitor has the built-in Performance Monitoring Unit (PMU), that determines the performance metrics at that node. The VIP agent's PMU, verifies if the performance constraints are based on the configured values. In case of any observed violations of the performance constraints, associated protocol checks are triggered as failures (UVM_ERROR) by the VIP.

For more information on the protocol checks associated with the Performance Analysis, see the <TBUUpdated> group in the <Protocol checks page> in CHI UVM Class reference.

Following is an example for performance constraint violation:

```
UVM_ERROR
/remote/vgvip14/rayrv/rayrv_chi_svt_us01_perf_analysis_demo/vip/svt/src/svt_err_check_stats.sv(637) @ 100000: uvm_test_top.env.amba_system_env.chi_system[0].rn[0].prot_mon [register_fail] CHECK [effect=ERROR]: Executed and FAILED - ENABLED AMBA CHECK: perf_avg_max_write_xact_latency_check (CHI), Description: Monitor Check that the average latency of write transactions in a given interval is less than or equal to the configured max value - Configured average max latency constraint = 500.000000. Observed average latency = 1883.333333. Interval start time = 0.000000. Interval end time = 10000.000000
```

4.3.1 Measurement and Reporting of the Performance Metrics

The performance metrics are measured and reported for each interval specified.

4.3.1.1 Specifying Performance Monitoring Interval

The interval needs to be specified using the `svt_chi_node_configuration::perf_recording_interval` configuration attribute. The entire simulation time is divided into intervals, with the duration of each interval being the value specified using this configuration attribute.

4.3.1.2 Reporting the Performance Metrics

The performance metrics are reported for each of these intervals at the end of a given interval, as well as towards the end of the simulation.

To print the performance report with NORMAL verbosity in the simulation log interactively, the `svt_chi_system_configuration::display_perf_summary_report` configuration parameter must be set to 1. Otherwise, the performance report displays HIGH verbosity. The performance report is present in the log in the PERFORMANCE REPORT header.

4.3.1.3 Accessing the Node Level Performance Metrics From Testbench During Run-time

The performance metrics monitored by the VIP node monitor can be accessed from the APIs that are available in the VIP. The `svt_chi_rn_agent` and `svt_chi_sn_agent` has a `perf_status` member of type `svt_chi_node_perf_status` class. This class has the following APIs:

- ❖ `get_perf_metric()`
- ❖ `get_unit_for_latency_metrics()`
- ❖ `get_unit_for_throughput_metrics()`
- ❖ `start_performance_monitoring()`
- ❖ `stop_performance_monitoring()`
- ❖ `is_performance_monitoring_in_progress()`
- ❖ `get_num_performance_monitoring_intervals()`
- ❖ `get_num_completed_performance_monitoring_intervals()`
- ❖ `is_performance_monitoring_interval_complete()`

For more information on the API, see the `svt_chi_node_perf_status` class from the AMBA CHI class reference guide.

The API's can be accessed as follows:

For example, `<hierarchy to the CHI system env>.<chi_agent_instance>.perf_status.<API>`

The API `get_perf_metric()` can be used to retrieve the value of one of the following metrics that are monitored by the VIP. The API can also return the transaction handles corresponding to Min/Max Read/Write latencies and the transaction handles that violated the constraints on Min/Max Read/Write latencies:

- ❖ `AVG_READ_LATENCY`: Average latency of the read type transactions.
- ❖ `MIN_READ_LATENCY`: Minimum latency of the read type transactions.
- ❖ `MAX_READ_LATENCY`: Maximum latency of the read type transactions.
- ❖ `READ_THROUGHPUT`: Throughput of the read data virtual channel.
- ❖ `AVG_WRITE_LATENCY`: Average latency of the write type transactions.
- ❖ `MIN_WRITE_LATENCY`: Minimum latency of the write type transactions.
- ❖ `MAX_WRITE_LATENCY`: Maximum latency of the write type transactions.
- ❖ `WRITE_THROUGHPUT`: Throughput of the write data virtual channel.

Read type transactions: `ReadNoSnp`, `ReadOnce`, `ReadShared`, `ReadClean`, `ReadUnique`

Write type transactions: `WriteNoSnpFull`, `WriteNoSnpPtl`, `WriteUniqueFull`, `WriteUniquePtl`, `WriteBackFull`, `WriteBackPtl`, `WriteCleanFull`, `WriteCleanPtl`, `WriteEvictFull`,

The read data virtual channel definitions are as follows:

- ❖ For RN: `RXDAT VC`
- ❖ For SN: `TXDAT VC`

Write data virtual channel definitions are as follows:

- ❖ For RN: `TXDAT VC`
- ❖ For SN: `RXDAT VC`

`get_perf_metric()` can be used only if `svt_chi_node_configuration::perf_recording_interval` is set to 0 or -1.

When `svt_chi_node_configuration::perf_recording_interval` is set to 0:

`get_perf_metric()` returns the value of the metric computed from the start of simulation till the time the method is called.

When `svt_chi_node_configuration::perf_recording_interval` is set to -1:

To start and stop the performance monitoring the `start_performance_monitoring()` and `stop_performance_monitoring()` methods are called. These methods can be called multiple times in a single simulation, which establishes multiple user defined time windows for performance monitoring. The `perf_rec_interval` argument of `get_perf_metric()` determines the performance window for which the metric value needs to be retrieved.

4.3.2 Example Usage

```
int    num_completed_perf_intervals_for_rn;
string msg_id_str = "report_phase";
real   outvalue;
svt_chi_transaction out_xacts[$];
bit status;

//send the initial register configuration transactions, then start the perf monitoring
    status =
env.amba_system_env.chi_system[0].rn[0].perf_status.start_performance_monitoring();
//send the perf stimulus. Once done, stop the perf monitoring
    status =
env.amba_system_env.chi_system[0].rn[0].perf_status.stop_performance_monitoring();
//any time after stopping perf monitoring, access the metrics, transaction handles by
calling get_perf_metric()
    //-----
    // Get the number of perf intervals and retrieve values for RN
    //-----

    num_completed_perf_intervals_for_rn =
env.amba_system_env.chi_system[0].rn[0].perf_status.get_num_performance_monitoring_inte
rvals();    for (int i=0; i<num_completed_perf_intervals_for_rn; i++) begin
        `uvm_info(msg_id_str, $sformatf("perf_tracking:: interval %0d : rn[0] performance
metrics ::", i), UVM_LOW)

        `uvm_info(msg_id_str, $sformatf("perf_tracking:: interval %0d : rn[0] Latency
Unit: %0s Throughput unit:%0s ", i,
env.amba_system_env.chi_system[0].rn[0].perf_status.get_unit_for_latency_metrics(),
env.amba_system_env.chi_system[0].rn[0].perf_status.get_unit_for_throughput_metrics()),
UVM_LOW)

        outvalue =
env.amba_system_env.chi_system[0].rn[0].perf_status.get_perf_metric(svt_chi_node_perf_s
tatus::MAX_WRITE_LATENCY, out_xacts, 1, i);

        `uvm_info(msg_id_str, $sformatf("perf_tracking:: interval %0d : rn[0] max wr
latency %0f", i, outvalue), UVM_LOW)
```

```

        if (out_xacts.size())
            `uvm_info(msg_id_str, $sformatf("perf_tracking:: interval %0d : rn[0] max wr
latency xact %0s", i, `SVT_CHI_PRINT_PREFIX(out_xacts[0])), UVM_LOW)

            outvalue =
env.amba_system_env.chi_system[0].rn[0].perf_status.get_perf_metric(svt_chi_node_perf_s
tatus::MIN_WRITE_LATENCY, out_xacts, 1, i);

            `uvm_info(msg_id_str, $sformatf("perf_tracking:: interval %0d : rn[0] min wr
latency %0f", i, outvalue), UVM_LOW)

            if (out_xacts.size())
                `uvm_info(msg_id_str, $sformatf("perf_tracking:: interval %0d : rn[0] min wr
latency xact %0s", i, `SVT_CHI_PRINT_PREFIX(out_xacts[0])), UVM_LOW)

                outvalue =
env.amba_system_env.chi_system[0].rn[0].perf_status.get_perf_metric(svt_chi_node_perf_s
tatus::AVG_WRITE_LATENCY, out_xacts, 0, i);

                `uvm_info(msg_id_str, $sformatf("perf_tracking:: interval %0d : rn[0] avg wr
latency %0f", i, outvalue), UVM_LOW)

                outvalue =
env.amba_system_env.chi_system[0].rn[0].perf_status.get_perf_metric(svt_chi_node_perf_s
tatus::MAX_READ_LATENCY, out_xacts, 1, i);

                `uvm_info(msg_id_str, $sformatf("perf_tracking:: interval %0d : rn[0] max rd
latency %0f", i, outvalue), UVM_LOW)

                if (out_xacts.size())
                    `uvm_info(msg_id_str, $sformatf("perf_tracking:: interval %0d : rn[0] max rd
latency xact %0s", i, `SVT_CHI_PRINT_PREFIX(out_xacts[0])), UVM_LOW)

                    outvalue =
env.amba_system_env.chi_system[0].rn[0].perf_status.get_perf_metric(svt_chi_node_perf_s
tatus::MIN_READ_LATENCY, out_xacts, 1, i);

                    `uvm_info(msg_id_str, $sformatf("perf_tracking:: interval %0d : rn[0] min rd
latency %0f", i, outvalue), UVM_LOW)

                    if (out_xacts.size())
                        `uvm_info(msg_id_str, $sformatf("perf_tracking:: interval %0d : rn[0] min rd
latency xact %0s", i, `SVT_CHI_PRINT_PREFIX(out_xacts[0])), UVM_LOW)

                        outvalue =
env.amba_system_env.chi_system[0].rn[0].perf_status.get_perf_metric(svt_chi_node_perf_s
tatus::AVG_READ_LATENCY, out_xacts, 0, i);

                        `uvm_info(msg_id_str, $sformatf("perf_tracking:: interval %0d : rn[0] avg rd
latency %0f", i, outvalue), UVM_LOW)

                        outvalue =
env.amba_system_env.chi_system[0].rn[0].perf_status.get_perf_metric(svt_chi_node_perf_s
tatus::READ_THROUGHPUT, out_xacts, 0, i);

                        `uvm_info(msg_id_str, $sformatf("perf_tracking:: interval %0d : rn[0] rd
throughput %0f", i, outvalue), UVM_LOW)

                        outvalue =
env.amba_system_env.chi_system[0].rn[0].perf_status.get_perf_metric(svt_chi_node_perf_s
tatus::WRITE_THROUGHPUT, out_xacts, 0, i);

                        `uvm_info(msg_id_str, $sformatf("perf_tracking:: interval %0d : rn[0] wr
throughput %0f", i, outvalue), UVM_LOW)

```


end

For more information, see the following tests in `tb_chi_svt_uvm_basic_sys`:

- ❖ `chi_node_perf_metrics_test`
- ❖ `chi_node_perf_metrics_multiple_perf_intervals_test`

4.4 Custom RN SAM (SYSTEM ADDRESS MAP)

Custom RN SAM can be specified by extending `svt_chi_system_configuration` class, implementing the following virtual methods in the test bench, and then setting a UVM factory override for the `svt_chi_system_configuration` class with the extended version.



Note All the other system address map related details must be programmed.

The virtual methods include the following:

- ❖ Setting number of RNs, HNs, SNs.
- ❖ HN Node IDs.
- ❖ HN Interface Types.
- ❖ HN-SN Map.

For example,

In the following example, the system has three HNs (2 HN-Fs, 1 HN-I) with `node_id` and HN interface types configured.

```
/**
 * Abstract:
 * Class cust_svt_chi_system_configuration is used specify custom address to HN map
 * by overriding virtual function
 * get_hn_node_id_for_addr(), is_mapped_to_mn_addr_ranges().
 */

`ifndef GUARD_CUST_SVT_CHI_SYSTEM_CONFIGURATION_SV
`define GUARD_CUST_SVT_CHI_SYSTEM_CONFIGURATION_SV

class cust_svt_chi_system_configuration extends svt_chi_system_configuration;

    /** UVM Object Utility macro */
    `uvm_object_utils (cust_svt_chi_system_configuration)

    /** Class Constructor */
    function new (string name = "cust_svt_chi_system_configuration");
        super.new(name);
    endfunction

    /**
     * Indicate if the provided address is mapped to register address space, that is MN.
     * In case set_mn_addr_range() is not called to program the MN address ranges,
     * it's required to implement here. Especially when MN shares the node_id of
```

```

* one of the HN's (HN-I), get_hn_node_id_for_addr() doesn't indicate whether
* the hn node ID is MN or HN.
* In case set_mn_addr_range() is called to program the MN address ranges,
* this method is not required to be implemented.
*/
function bit is_mapped_to_mn_addr_ranges(bit[`SVT_CHI_MAX_ADDR_WIDTH-1:0] addr);
    if (addr inside {[`TB_START_ADDR_MN:`TB_END_ADDR_MN]})
        is_mapped_to_mn_addr_ranges = 1;
    else
        is_mapped_to_mn_addr_ranges = 0;
endfunction

/** Returns the node_id of the HN configured for this address. */
virtual function int get_hn_node_id_for_addr(bit [`SVT_CHI_MAX_ADDR_WIDTH-1:0] addr);
    string method_name = "cust_get_hn_node_id_for_addr";

    if (addr inside {[`TB_START_ADDR_MN:`TB_END_ADDR_MN]}) begin
//MN id configed through svt_chi_system_configuration::misc_node_id
        get_hn_node_id_for_addr = misc_node_id;
        `svt_debug(method_name, $sformatf("addr 'h%0h is mapped to MN with node_id 'd%0d",
            addr, get_hn_node_id_for_addr));
    end
else if (addr inside {[`TB_START_ADDR_HN_I:`TB_END_ADDR_HN_I]}) begin
    // node ID corresponding to hn_idx for which this addr range is applicable
    // hn_idx 2 corresponds to HN-I as programmed through set_hn_interface_type() and
    // set_hn_node_id()
    get_hn_node_id_for_addr = get_hn_node_id(2);
    `svt_debug(method_name, $sformatf("addr 'h%0h is mapped to HN-I with node_id
'd%0d,
                                corresponding to hn_idx 2", addr,
get_hn_node_id_for_addr));
    end
    else begin
        // XOR function for 2 HN-Fs
        // Stores to which HN-F the address belongs to, in the range [0:1]
        bit hn_f_idx;
        // Variable to aid in computing the hash function
        bit partial_addr = addr[6];

        // The HN-F address map when number of HN-Fs are 2:
        // hn_f_idx[0] = ^addr[(`SVT_CHI_MAX_ADDR_WIDTH-1):6]
        for (int lsb = 7; lsb <= (`SVT_CHI_MAX_ADDR_WIDTH-1); lsb+=1) begin
            partial_addr = (partial_addr ^ addr[lsb+:1]);
        end

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

        // HN indices 0, 1 are HN-Fs. Get the node_id of the HN-Fs as programmed through
        // set_hn_interface_type() and set_hn_node_id()
        get_hn_node_id_for_addr = get_hn_node_id(hn_f_idx);
        `svt_debug(method_name, $sformatf("addr 'h%0h is mapped to HN-F with node_id
'd%0d,
                                corresponding to hn_idx %0d", addr,
get_hn_node_id_for_addr, hn_f_idx));
    end
endfunction // get_hn_node_id_for_addr

```

```
endclass // cust_svt_chi_system_configuration

`endif
```

During the build_phase() of the test:

```
set_type_override_by_type
svt_chi_system_configuration::get_type(), cust_svt_chi_system_configuration::get_type()
;
```

4.5 Random Target ID Generation From RN

With Target ID Remapping expected at the Interconnect, active RN agent is updated to support, generating random target ID (svt_chi_rn_transaction::tgt_id) from the sequence. RN protocol layer driver processes such transactions without changing the tgt_id value in the transaction.

The following configuration parameter for the active RN agent is set to one, by considering the related aspects as described:

```
rand bit attribute
svt_chi_node_configuration::random_tgt_id_enable = 0
```

Applicable only for RN:

- ❖ When set to 1:
 - ◆ Can be set to 1 only when target ID remapping is expected at Interconnect through
svt_chi_system_configuration :: expect_target_id_remapping_by_interconnect set to 1.
 - ◆ Active RN: rand_mode is turned on for svt_chi_common_transaction :: tgt_id in pre_randomize method.
 - ◆ Passive RN: Expects svt_chi_common_transaction :: tgt_id can take any value
- ❖ When set to 0:
 - ◆ svt_chi_common_transaction :: tgt_id is expected to have valid value based on system address map and rand_mode of svt_chi_common_transaction :: tgt_id is turned off.
- ❖ Default value : 0
- ❖ Type : Static

4.6 Target ID Remapping

The RN agent is updated to capture the original tgt_id of the transaction as
svt_chi_transaction::original_tgt_id.

```
bit attribute
svt_chi_transaction::is_tgt_id_remapped = 0
```

Indicates if the tgt_id is remapped by Interconnect to a different value other than the tgt_id field sent in the request flit by RN.

With Target ID remapping expected at interconnect, and RN SAM specified, RN agent is updated to perform the checks on the correctness of remapped target ID based on RN SAM as well as communicating

node pairs as per Appendix B of CHI specification. These checks are performed when the following configuration attribute is set to 1:

```
bit attribute
```

```
svt_chi_system_configuration::rn_sam_specified_with_exp_tgt_id_remap_at_icn_enabled = 0
```

Indicates if the RN System Address Map (SAM) is specified for address to HN mapping when Target ID remapping is expected at Interconnect HNs (See the `expect_target_id_remapping_by_interconnect`).

- ❖ This can be set to 1 only when `expect_target_id_remapping_by_interconnect = 1`.
- ❖ When set to 1, VIP expects that the RN SAM specified (Address to HN&MN map, `misc_node_id`) in the CHI system configuration object is correct, and performs source/target HN type checks that are part of `_flit_*_check`.
- ❖ Type : Static
- ❖ Default value : 0

The following checks are added/updated:

- ❖ `expected_tgt_id_in_rn_xact_check` : Checks if the Target ID field in the transaction request is set to the node ID of the HN as per the RN SAM. When Target ID is not expected to be remapped at the Interconnect, RN must program the `tgt_id` of the request flit to the correct value as per the SAM
- ❖ `expected_remapped_tgt_id_in_response_check` : Checks if the Target ID is remapped correctly at the Interconnect. This is applicable only when target ID remapping is expected at the Interconnect and RN SAM is specified for address to HN mapping, that is,

```
svt_chi_system_configuration::expect_target_id_remapping_by_interconnect and
svt_chi_system_configuration::rn_sam_specified_with_exp_tgt_id_remap_at_icn_enabled
```

are set to 1.

4.7 Protocol Retry

RN agent is updated to accurately track and update the P-Credit counters along with Retry Protocol flow with target ID remapped.

Active RN agent is updated to control the sending of retried transaction with original/different values for `txn_id`, `qos`, `req_rsvdc`, and `tgt_id` (original/remapped), `trace_tag`, `slcrephint_replacement`, `slcrephint_unusedprefetch`.

```
rand bit attribute
```

```
svt_chi_transaction::is_retried_with_original_qos = 1
```

Indicates if the transaction is retried with original qos of original request or not. In case of active RN based on this field, RN driver decides to use original qos or any random qos for the retried transaction.

```
rand bit attribute
```

```
svt_chi_transaction::is_retried_with_original_rsvdc = 1
```

Indicates if the transaction is retried with original `req_rsvdc` of original request not. In case of active RN, based on this field, RN driver decides to use original `req_rsvdc` or any random `req_rsvdc` for the retried transaction. This is applicable only when `svt_chi_node_configuration::chi_spec_revision` is not set to `ISSUE_A`.

```
rand bit attribute
```

```
svt_chi_transaction::is_retried_with_original_txn_id = 1
```

Indicates if the transaction is retried with original `txn_id` of original request or not. In case of active RN, based on this field, RN driver decides to use original `txn_id` or any random `txn_id` for the retried transaction.

When `chi_spec_revision` is not set to `ISSUE_A`: for RN-SN back to back topology with no HNs, this is not applicable, and is always expected to be 1. In this case, the retried transaction must have same value for `return_txn_id` as that of original transaction, that is, `return_txn_id` must have same value as `txn_id`. This implies that the retried transactions `txn_id` and `return_txn_id` must be the same as original transaction.

rand bit attribute

```
svt_chi_transaction::is_retried_with_original_tgt_id = 1
```

Indicates if the transaction is retried with original `tgt_id` of original request or not. In case of active RN, based on this field, RN driver decides to use original `tgt_id` or the remapped `tgt_id` for the retried transaction, when

`svt_chi_system_configuration::expect_target_id_remapping_by_interconnect` is set to 1.

```
rand bit is_retried_with_original_trace_tag = 1;
```

Indicates if the transaction is retried with original `trace_tag` of original request or not. In case of active RN, the RN driver decides to use original `trace_tag` or any random `trace_tag` for the retried transaction based on this field. Applicable only when using the RN VIP with `chi_spec_revision` set to `svt_chi_node_configuration::ISSUE_B` or later.

These variables are applicable only when using the RN VIP with `chi_spec_revision` set to `svt_chi_node_configuration::ISSUE_E` or later.

- ❖ `rand bit is_retried_with_original_slcrephint_replacement = 1;`

Indicates if the transaction is retried with original `slcrephint_replacement` of original request or not. In case of active RN, based on this field, RN driver decides to use original `slcrephint_replacement` or any random `slcrephint_replacement` for the retried transaction. when

`svt_chi_node_configuration::slcrephint_mode` is set to `SLC_REP_HINT_DISABLED`, this field should be constraint to 1.

- ❖ `rand bit is_retried_with_original_slcrephint_unusedprefetch = 1;`

Indicates if the transaction is retried with original `slcrephint_unusedprefetch` of original request or not. In case of active RN, the RN driver decides to use original `slcrephint_unusedprefetch` or any random `slcrephint_unusedprefetch` for the retried transaction based on this field. When `svt_chi_node_configuration::slcrephint_mode` is set to `SLC_REP_HINT_DISABLED`, this field must be constraint to 1.

4.8 Random SRC_ID Generation

When RN agent is directly connected to SN DUT, in order to mimic multiple HNs generating traffic to the SN, active RN agent supports generating transactions with random `src_id` from the sequence. In this case, active RN agent does not modify the `src_id` generated from sequence within the transaction. Further, the RN agent supports correlating the responses to these different `src_ids` and tracks the transaction's progress. Passive RN agent also supports tracking multiple `src_ids` from the same RN.

rand bit attribute

```
svt_chi_node_configuration::random_src_id_enable = 0
```

Applicable only for RN.

- ❖ When set to 1:
 - ◆ This can be set to 1 only when `svt_chi_system_configuration :: num_rn = 1`, `svt_chi_system_configuration :: num_sn = 1` and `svt_chi_system_configuration :: num_hn = 0` together.
 - ◆ Active RN: `rand_mode` is turned on for `svt_chi_common_transaction :: src_id` in `pre_randomize` method.
 - ◆ Passive RN: Expects `svt_chi_common_transaction :: src_id` can take any value
- ❖ When set to 0:
 - ◆ `svt_chi_common_transaction :: src_id` is expected to have fixed value of RN's node ID, set through `svt_chi_node_configuration :: node_id`. The `rand_mode` of `svt_chi_common_transaction :: src_id` is turned off.
- ❖ Default value : 0
- ❖ Type : Static

4.9 Single Outstanding Request Per Address

Active RN agent has been updated such that, when Ordering rules are not applicable, a Read, Write, Dataless or Atomic request is issued only when there are no Outstanding Read, Write, Dataless or Atomic request to the same cacheline.

The following checks are added in the passive VIP RN:

- ❖ `new_req_before_completion_of_previous_read_write_xact_to_same_cacheline_check`: Checks if a Read, Write, Dataless or Atomic request is issued when there are no outstanding Read or Write transactions to the same cache line, unless both requests are Ordered.
 - ◆ Outstanding Read - refers to all Read as well as dataless transactions other than CMOs.
 - ◆ Outstanding Write - refers to all Write and Atomic transactions.
- ❖ `new_req_before_completion_of_previous_cmo_xacts_to_same_cacheline_check`: Checks if a request, other than Evict, WriteEvictFull and Prefetch, is issued when there are no outstanding CMO transactions to the same cache line.
- ❖ `cmo_xact_before_completion_of_previous_xacts_to_same_cacheline_check`: Checks if there are no outstanding Normal Non-cacheable or Device type writes, that are targeting an HN-I, which have not received a Comp Response, before a Barrier is issued by RN.

4.10 Snooperable and Non-Snooperable Domains

The following are the usage details for snooperable and non-snooperable domains' feature:

1. Enable `SVT_CHI_ISSUE_B_ENABLE` define to enable the behavior as complaint to ISSUE B specification version.
2. Set `svt_chi_node_configuration::enable_domain_based_addr_gen` to one, if the address must be generated as per the snooperable domains.
3. Use API to create snooperable domains `svt_chi_system_configuration::create_new_domain()`.
4. Use API to set the address for each snooperable domain created `svt_chi_system_configuration::set_addr_for_domain()`.

For example,

If there are 4 RNs in the chi system, then the domains can be created as:

```
create_new_domain(0,svt_chi_system_domain_item::SNOOPABLE,{0,1 });  
create_new_domain(1,svt_chi_system_domain_item::SNOOPABLE,{2,3 });
```

Each domain address can be set as:

```
set_addr_for_domain ( 0 , 0x0010_0000, 0x0010_1000); // 4K  
set_addr_for_domain ( 1 , 0x0010_2000, 0x0010_3000); // 4K
```


5

Verification Topologies

This chapter shows you from a high-level, how the CHI VIP can be used to test RN, SN, or Interconnect DUT. This chapter discusses the following topics:

- ❖ [Interconnect DUT and RN or SN VIP](#)
- ❖ [System DUT With Passive VIP](#)
- ❖ [System DUT with Mix of Active and Passive VIP](#)

5.1 Interconnect DUT and RN or 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 Env to have s RN agents and m SN agents, in active mode. The active RN agents will generate CHI transactions towards the interconnect SN ports, and active SN agents connected would respond to the transactions generated by interconnect RN ports. The RN and SN agents would also perform passive functions such as protocol checking, coverage generation and transaction logging.

Set the following properties to implement this topology:

- ❖ 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

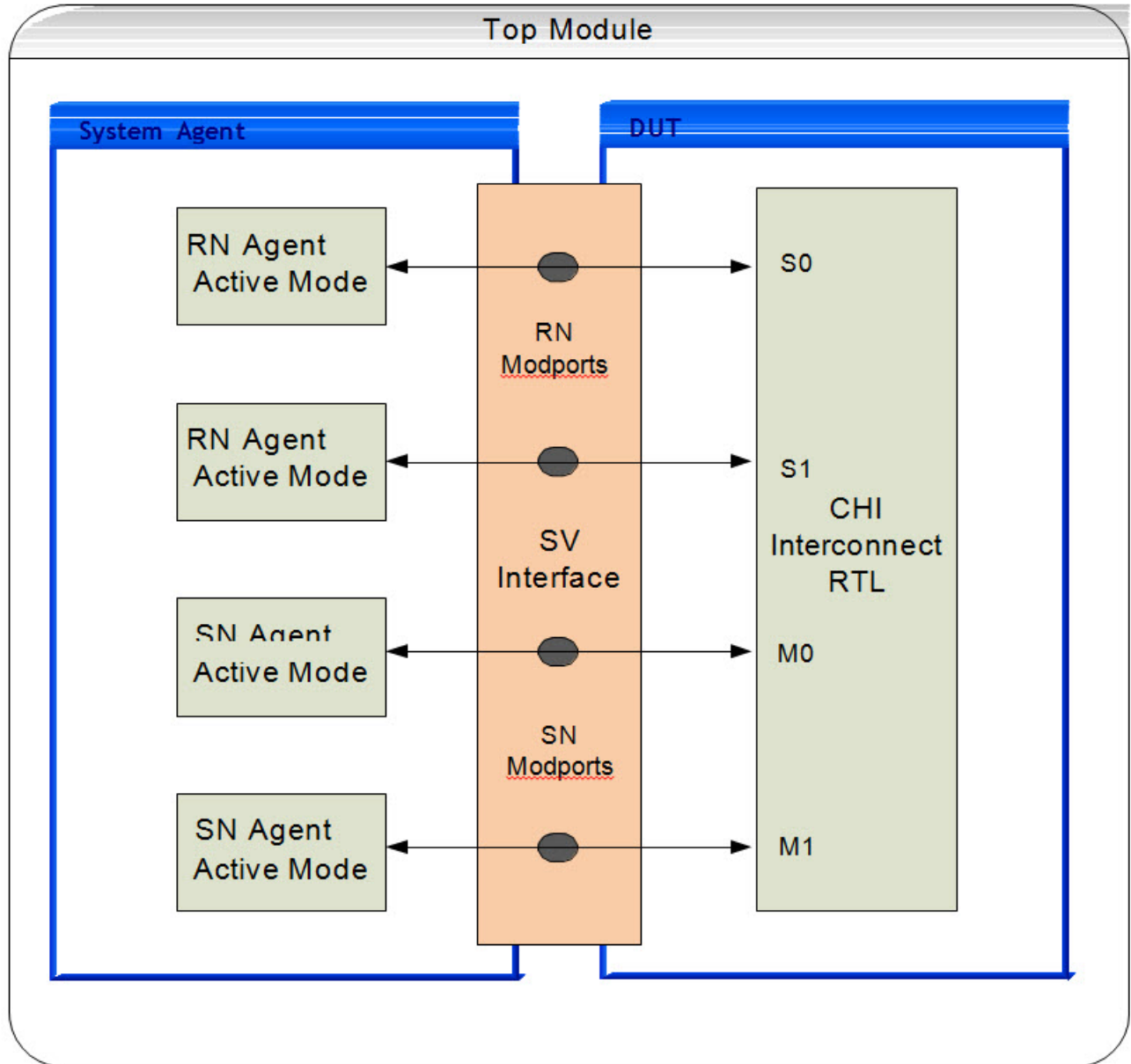
System configuration settings are as follows:

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

Port configuration settings are as follows:

- ❖ `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;`

This figure 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 agents and s SN agents, in passive mode. The passive RN and SN agents would perform passive functions for example, protocol checking, coverage generation, and transaction logging.

Set the following properties to implement this topology:

- ❖ 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

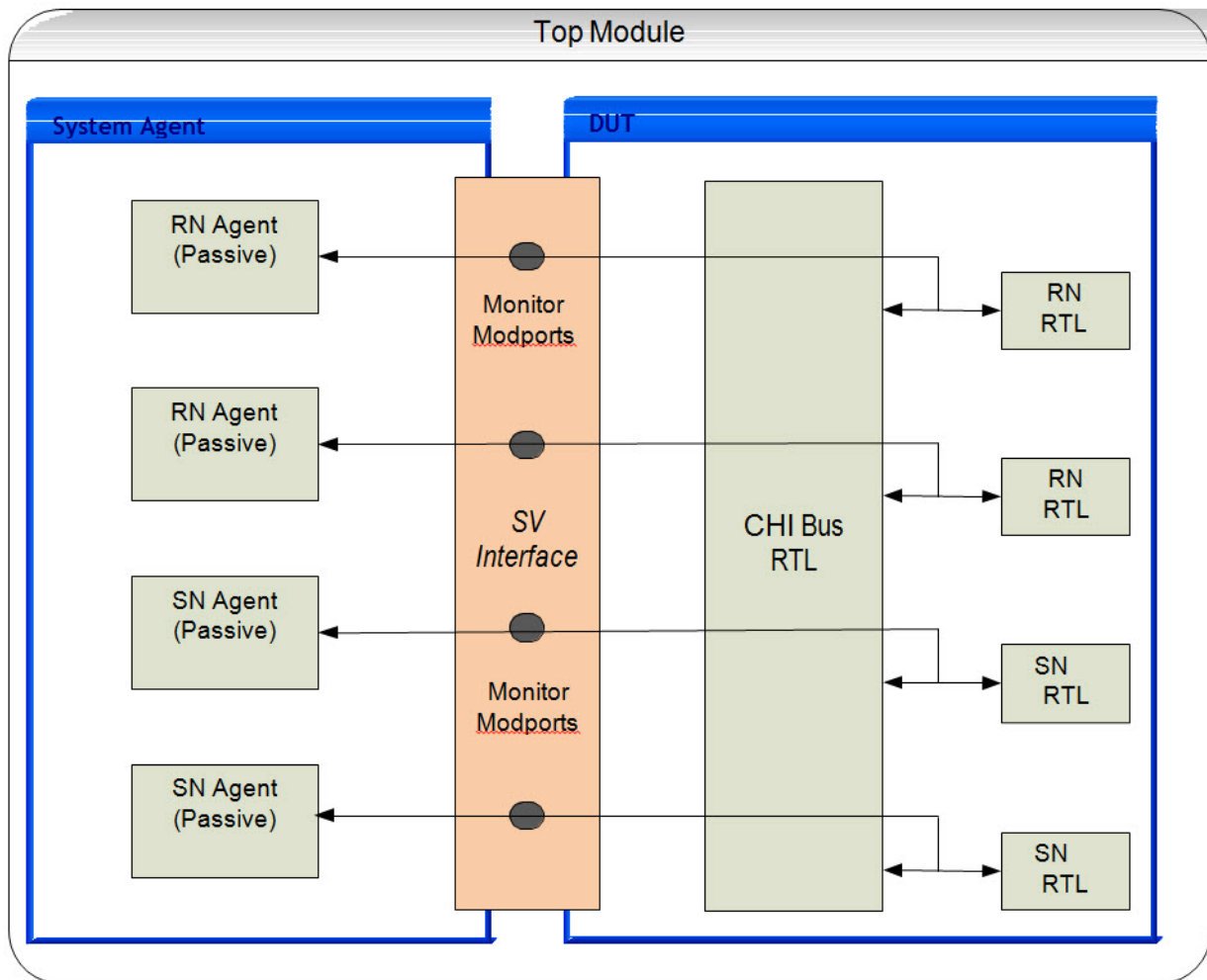
System configuration settings are as follows:

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

Port configuration settings are as follows:

- ❖ `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;`

This figure shows this setup.

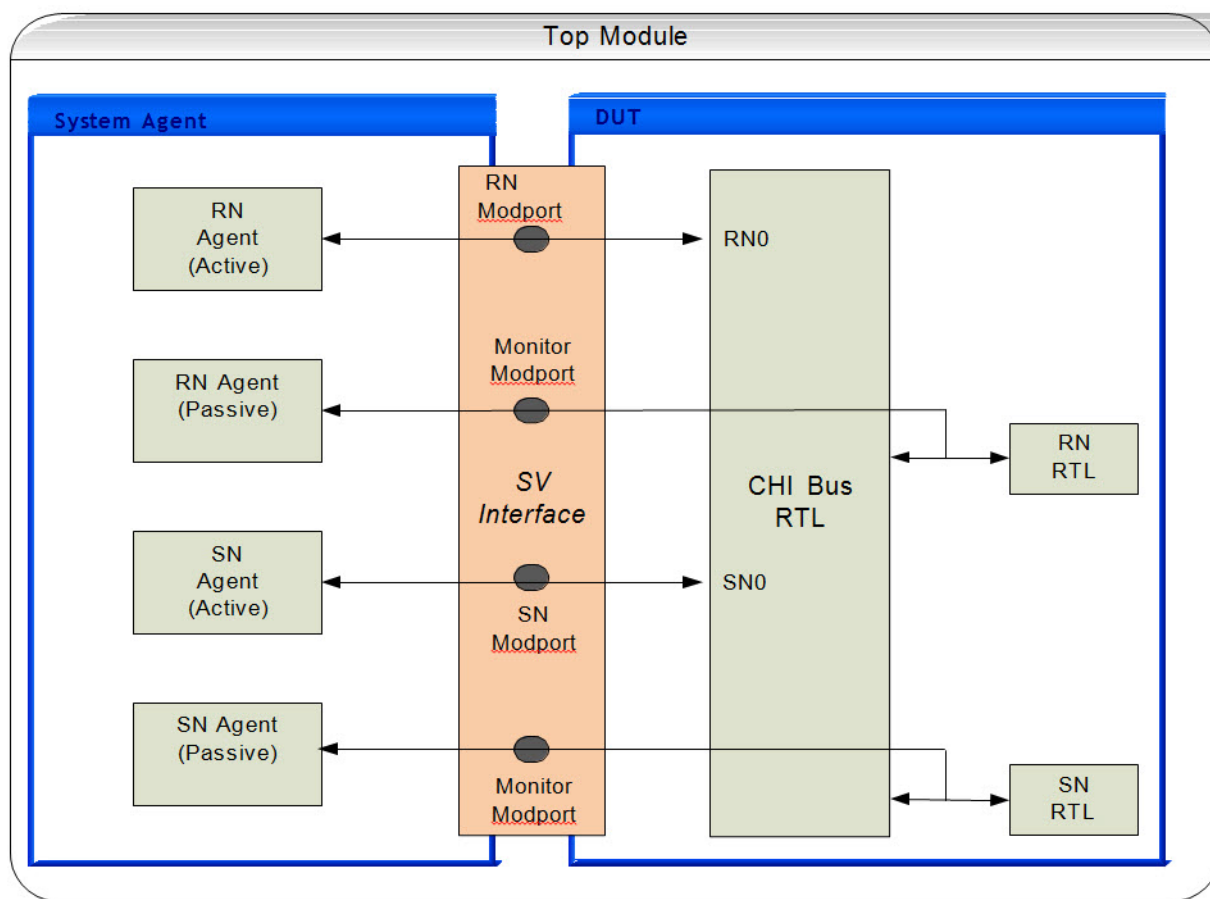
Figure 5-2 System DUT with Passive VIP

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 Env to have two RN agents and two SN agents. Configure the RN agent connected to port RN0, and SN agent connected to port SN0 as active. Configure the RN agent connected to port RN1 and SN agent connected to port SN1 as passive. All the agents would continue to perform passive functions such as protocol checking and coverage.

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



Set the following properties to implement this topology:

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

Protocol Features Usage And Reference

6.1 Transaction Ordering

6.1.1 User Interface

The field 'order_type' in the svt_chi_transaction class defines the ordering requirements for a transaction.

A programmable parameter, "request_ordering", has been provided in the svt_chi_system_single_node_rn_coherent_transaction_base_virtual_sequence to provide control to the user for setting the order_type field in the generated transaction.

When request_ordering is programmed to 1, the ordering_type field in the generated transaction will be set to REQUEST_ORDERING_REQUIRED/REQUEST_EP_ORDERING_REQUIRED.

When request_ordering is programmed to 0, the ordering_type field in the generated transaction will be set to NO_ORDERING_REQUIRED

If the request_ordering bit is not programmed, the ordering_type field in the generated transaction will be set randomly.

6.1.2 Protocol Checks

The check 'single_req_order_stream_check' has been defined in the passive RN agent in order to flag any violations detected with respect to the transaction request ordering. This check is enabled only when a single streaming ID is used, that is, when num_req_order_streams in svt_chi_node_configuration is set to 1 for the RN

Additionally, there are ordering related checks in the scoreboard which check if the Ordering rules have been violated for any given transaction.

6.1.3 VIP Behavior

RN maintains ordering between all transactions that require Ordering, which means that, if there are two transactions which require ordering, the second transaction is sent out by the RN only after the Ordering requirements of the first transaction have been met

Let us consider the following example scenarios to understand the behavior of the VIP RN:

Basic Scenario I:

Transaction 1 (Requires ordering), Transaction 2 (Does Not require Ordering), Transaction 3 (Requires Ordering)

Active RN:

- ❖ Request for Transaction 1 is sent to the Interconnect.
- ❖ Request for Transaction 2 is sent to the Interconnect without waiting for the Ordering condition to be met for transaction 1
- ❖ Request for Transaction 3 is queued and sent only after the ordering condition for transaction 1 is met.

Passive RN:

The Passive RN agent will check if Ordering is maintained between Transaction 1 and Transaction 3. If the request for Transaction 3 is seen before the Ordering condition of Transaction 1 is satisfied, the check 'single_streaming_order_check' will fail in the Passive RN.

Basic Scenario II:

Transaction 1 (requires ordering), Transaction 2 (requires ordering), Transaction 3 (does not require ordering)

Active RN:

- ◆ Request for Transaction 1 is sent to the interconnect
- ◆ Request for Transaction 2 is queued and sent only after the ordering condition for transaction 1 is met.
- ◆ Request for Transaction 3, although it doesn't require any ordering, is queued and sent only after the transmission of Transaction 2 request. (The spec permits the transmission of Transaction 3 before Transaction 2. But we do not want to re-order the transactions sent from the VIP RN. So, we are restricting this behavior)

Passive RN:

Passive RN will not fire any errors even if Transaction 3 is sent before Transaction 2 (as it is permitted by the specification). It will only check if ordering is maintained between Transaction 1 and Transaction 2.

Retry Scenario

Transaction 1 (requires ordering), Transaction 2 (requires ordering)

Active RN:

- ◆ Request for Transaction 1 is sent to the interconnect
- ◆ Request for Transaction 2 is queued and sent only after the ordering condition for transaction 1 is met.
- ◆ Retry response is received for Transaction 1. Transaction 2 remains queued.
- ◆ Transaction 1 is retried. Request for Transaction 2 remains queued in the RN till the Ordering requirement of the retry transaction is complete.

Passive RN:

The Passive RN expects to observe Transaction 2 only after the Transaction 1 is retried and, further, the ordering requirements of the retried Transaction are met. If Transaction 2 request is seen at any point before the Ordering condition is met for the Retried Transaction 1, the 'single_streaming_order_check' will fail.

Transaction Cancel Scenario

Transaction 1 (requires ordering), Transaction 2 (requires ordering)

Active RN:

- ◆ Request for Transaction 1 is sent to the interconnect
- ◆ Request for Transaction 2 is queued and sent only after the ordering condition for transaction 1 is met.
- ◆ Retry response is received for Transaction 1. Transaction 2 remains queued.
- ◆ Transaction 1 is cancelled and a PCRDRETURN transaction is scheduled. Request for Transaction 2 remains queued in the RN till the PCRDRETURN transaction is transmitted.

Passive RN:

single_streaming_order_check in the Passive RN will fail if the request for Transaction 2 is observed before the PCRDRETURN for Transaction 1.

Multiple Ordering Streams Scenario

Transaction 1 (requires Ordering, Stream ID 0), Transaction 2 (requires Ordering, Stream ID 1), Transaction 3 (requires Ordering, Stream ID 0)

Active RN:

- ◆ Request for Transaction 1 is sent to the interconnect
- ◆ Request for Transaction 2 is sent to the interconnect without waiting for the Ordering condition to be met for Transaction 1 as it has a different Stream ID
- ◆ Request for Transaction 3 is queued and is sent as soon as the Ordering condition is met for Transaction 1 irrespective of the status of Transaction 2.

Passive RN:

Passive RN will only perform Ordering checks on Transactions 1 and 3.
'single_streaming_order_check' will fail in case Transaction 3 is sent before the Ordering requirement for Transaction 1 is met.

Ordered WriteUnique Scenario

WriteUnique Transaction 1 (Requires Ordering, tgt_id 0), WriteUnique Transaction 2 (Requires Ordering, tgt ID 1), WriteUnique Transaction 3 (Requires Ordering, tgt ID 0)

1. When Optimized Streaming bit is not set in svt_chi_node_configuration for the RN

Active RN:

- ◆ Request for Transaction 1 is sent to the interconnect
- ◆ Request for Transaction 2 is queued and sent only after the ordering condition for transaction 1 is met.
- ◆ Request for Transaction 3 is queued and sent only after the ordering condition is met for Transaction 2.

Passive RN:

Passive RN will check if ordering is maintained between all three transactions. If any of the transaction requests is seen before the ordering requirement for the preceding transactions is met, the `single_streaming_order_check` will fail.

2. When Optimized Streaming bit is set in `svt_chi_node_configuration` for the RN

Active RN:

- ◆ Request for Transaction 1 is sent to the interconnect.
- ◆ Request for Transaction 2 is sent to the interconnect without waiting for the Ordering condition to be met for Transaction 1 as it has a different Target ID.
- ◆ Request for Transaction 3 is queued and is sent as soon as the Ordering condition is met for Transaction 1 irrespective of the status of Transaction 2.

Passive RN:

The Passive RN will only check if ordering is maintained between Transaction 1 and Transaction 3 as Transaction 2 has a different Target ID.

Multiple Transaction Retry And Cancel Scenario

Transaction 1 (requires ordering) , Transaction 2 (does not require ordering), Transaction 3 (requires ordering)

Active RN:

- ◆ Request for Transaction 1 is sent to the interconnect
- ◆ Request for Transaction 2 is sent without waiting for the Ordering condition to be met for Transaction 1.
- ◆ Transaction 3 is queued as the request ordering condition for Transaction 1 is not yet met
- ◆ Retry response is received for Transaction 1.
- ◆ Retry response is received for Transaction 2.
- ◆ Transaction 1 is cancelled and a `PCRDRETURN` transaction is scheduled.
- ◆ Request for Transaction 3 remains queued in the RN and is sent out as soon as the `PCRDRETURN` transaction is transmitted, irrespective of the status of Transaction 2.

Passive RN:

In this scenario, the Ordering checks in the Passive RN are disabled after `PCRDRETURN` is received as it is cannot be determined, at that point in time, as to which transaction was canceled because there are two transactions that received a retry response.

Once Transaction 2 is retried, the sum of the number of `PCREDTURN` and the number of retry transactions will be equal to the number of transactions that received a `RETRY` response and so, we re-enable the checks from that point on.

In other words, the checks are disabled when a `PCRDRETURN` is received until the following condition is satisfied: (number of `PCRDRETURN` + number of retry transactions = number of transactions that received a Retry response)

6.2 Outstanding Transactions

6.2.1 User Interface

You can either set the overall maximum number of outstanding transactions or separate maximum number of outstanding transactions for various transaction type categories, for a given node in the system.

The following controls have been provided in `svt_chi_node_configuration`, to let users program the maximum number of outstanding transactions for a node.

❖ `int`

```
svt_chi_node_configuration:: num_outstanding_xact =
`SVT_CHI_MAX_NUM_OUTSTANDING_XACT
```

- ◆ Specifies the number of outstanding transactions a node can support.
- ◆ Must be set to -1 if separate outstanding transaction limit is to be defined for the various transaction types
- ◆ Default value: ``SVT_CHI_MAX_NUM_OUTSTANDING_XACT`
- ◆ Permitted values: -1, 1 to ``SVT_CHI_MAX_NUM_OUTSTANDING_XACT`

❖ `int`

```
svt_chi_node_configuration:: num_outstanding_read_xact = -1
```

- ◆ Specifies the number of outstanding read transactions a node can support.
- ◆ Must be set when `#num_outstanding_xact = -1`
- ◆ Default value: -1
- ◆ Permitted values: -1, 1 to ``SVT_CHI_MAX_NUM_OUTSTANDING_XACT`

❖ `int`

```
svt_chi_node_configuration:: num_outstanding_write_xact = -1
```

- ◆ Specifies the number of outstanding write transactions a node can support.
- ◆ Must be set when `#num_outstanding_xact = -1`
- ◆ Default value: -1
- ◆ Permitted values: -1, 1 to ``SVT_CHI_MAX_NUM_OUTSTANDING_XACT`

❖ `int`

```
svt_chi_node_configuration:: num_outstanding_cmo_xact = -1
```

- ◆ Specifies the number of outstanding Cache Maintenance transactions a node can support.
- ◆ Must be set only when `#num_outstanding_xact = -1`
- ◆ Default value: -1
- ◆ Permitted values: -1, 1 to ``SVT_CHI_MAX_NUM_OUTSTANDING_XACT`

❖ `int`

```
svt_chi_node_configuration:: num_outstanding_control_xact = -1
```

- ◆ Specifies the number of outstanding control (DVM,Barrier) transactions a node can support.
- ◆ Must be set only when `#num_outstanding_xact = -1`
- ◆ Default value: -1

- ◆ Permitted values: -1, 1 to `SVT_CHI_MAX_NUM_OUTSTANDING_XACT

The VIP provides support to track the current number of transactions outstanding at a given node. For this purpose, outstanding transaction counters have been implemented in status object of type `svt_chi_protocol_status`, which in turn is encapsulated in a shared status object of type `svt_chi_status`.

- ❖ `svt_chi_protocol_status` attribute
- ❖ `svt_chi_status::prot_status`

Status object used to store details specific to the protocol layer within a node.

The following are the outstanding transaction counters implemented in the `svt_chi_protocol_status` class

- ❖ `int`
`svt_chi_protocol_status::current_outstanding_xact_count`
 - ◆ Indicates the number of total outstanding transactions at a given node
- ❖ `int`
`svt_chi_protocol_status::current_outstanding_read_xact_count`
 - ◆ Indicates the number of total outstanding Read transactions at a given node
- ❖ `int`
`svt_chi_protocol_status::current_outstanding_write_xact_count`
 - ◆ Indicates the number of total outstanding Write and Copyback transactions at a given node
- ❖ `int`
`svt_chi_protocol_status::current_outstanding_cmo_xact_count`
 - ◆ Indicates the number of total outstanding CMO transactions at a given node
- ❖ `int`
`svt_chi_protocol_status::current_outstanding_control_xact_count`
 - ◆ Indicates the number of total outstanding DVM and Barrier transactions at a given node

In addition to these, the following attributes are also provided in the protocol status object

- ❖ `int`
`svt_chi_protocol_status::pending_retry_xact_count`
 - ◆ Indicates the number of transactions that have received a retry response but have not been retried yet
This will also include retry transactions that might have been cancelled using `PCrdReturn`
- ❖ `int`
`svt_chi_protocol_status::num_p_crd_return_count`
 - ◆ Indicates the number of `PCrdReturn` transactions observed at the node.
(`pending_retry_xact_count` - `num_p_crd_return_counter`) gives the actual pending retry transaction count

Refer the HTML class reference for more details.

6.2.1.1 Accessing the Outstanding Transaction Counters From the Test

The node agent updates the outstanding transaction counters in `shared_status.prot_status` of type `svt_chi_protocol_status`, at the start and end of every transaction.

If users want to print/observe the outstanding transaction counts at a node, at any given point in time, the handle to shared status object and, hence, the protocol status object can be obtained from the corresponding node agent that is instantiated in the CHI system ENV.

Example:

The following are the steps to be followed in order to obtain and print the shared status object in a test, illustrated through the corresponding code snippet below.

- ❖ `chi_base_test` is the test that extends the `uvm_test`.
- ❖ The test contains the test env instance '`tb_env`', which further instantiates CHI system ENV as '`chi_env`'.
- ❖ CHI system ENV, '`chi_env`', contains handles to all the RN and SN agents (`rn[$]` & `sn[$]`)
- ❖ The `shared_system_status` object from the respective RN/SN agent is retrieved through UVM config DB get operation as follows in the test's `end_of_elaboration_phase()`.
- ❖ The object "`prot_status`", of type `svt_chi_protocol_status`, is instantiated in '`shared_status`' and can be used to monitor the outstanding transaction counters for that node.

```
class chi_base_test extends uvm_test;

    svt_chi_status rn0_shared_status;
    .
    .
    virtual function end_of_elaboration_phase(uvm_phase phase);
        svt_config_int_db#(svt_chi_status)::get(this.tb_env.chi_env.chi_system[0].rn[0], "",
        "shared_status", rn0_shared_status);
    endfunction

    virtual task main_phase(uvm_phase phase);
        wait(rn0_shared_status.prot_status.current_outstanding_xact_count ==
        cfg.chi_sys_cfg[0].rn_cfg[0].num_outstanding_xact);
        `uvm_info("main_phase", "Transaction overflow condition reached for RN0", UVM_LOW)
    endtask

endclass
```

6.2.2 VIP Behavior

6.2.2.1 Max Outstanding Transactions

The maximum number of transactions that can be outstanding at a node is programmed through either `svt_chi_node_configuration::num_outstanding_xact` or `svt_chi_node_configuration::num_outstanding*_xact`.

When `svt_chi_node_configuration::num_outstanding_xact` is set to a value other than -1, `svt_chi_node_configuration::num_outstanding*_xact` must not be set to any value other than the default (-1).

When `svt_chi_node_configuration::num_outstanding_xact` is set to -1, `svt_chi_node_configuration::num_outstanding*_xact` must be set to appropriate values by the user.

Overflow Scenario:

- ❖ CASE I: `svt_chi_node_configuration::num_outstanding_xact != -1`

When a new transaction request is scheduled at the active RN agent, it waits till `svt_chi_protocol_status::current_outstanding_xact_count` is less than the programmed `svt_chi_node_configuration::num_outstanding_xact`, and then transmits the transaction request.

When a transaction request is received by the passive RN agent or the SN agent, a warning is flagged if `svt_chi_protocol_status::current_outstanding_xact_count` is already equal to or greater than the programmed `svt_chi_node_configuration::num_outstanding_xact`, after which, the received transaction request is processed normally.

- ❖ CASE II: `svt_chi_node_configuration::num_outstanding_xact == -1 & svt_chi_node_configuration::num_outstanding_*_xact != -1`

When a new transaction request is scheduled at the active RN agent, it waits till `svt_chi_protocol_status::current_outstanding_<type>_count` is less than the programmed `svt_chi_node_configuration::num_outstanding_<type>_count` and then transmits the transaction request.

When a transaction request is received by the passive RN agent or the SN agent, a warning is flagged if `svt_chi_protocol_status::current_outstanding_<type>_xact_count` is already equal to or greater than the programmed `svt_chi_node_configuration::num_outstanding_<type>_xact`, after which, the received transaction request is processed normally.

6.2.2.2 Current Outstanding Transactions Count

`svt_chi_status::current_outstanding_xact_count` as well as `svt_chi_status::current_outstanding_*_xact_count` are updated by the agents whenever a new transaction starts or an ongoing transaction completes.

Transaction Retry/cancel Scenario:

The outstanding transaction counters remain unaltered when a retry response is received for a transaction.

In case of an active RN agent, the counters are updated only when the retry transaction is either cancelled or completed.

In the passive RN as well as the active/passive SN agents, the counters are updated when the retry transaction is completed. In case the retry transaction is canceled using a `PCRDRETURN`, the overall outstanding transactions counter is updated immediately, but the individual transaction type based counters are not, as it might not always be possible to determine which transaction was canceled using the `PCRDRETURN`.

For example, when there are multiple pending retry transactions and a `PCRDRETURN` is received, it will not be possible to know which of the pending retries was dropped, immediately. We will be able to determine which transaction was canceled only when the response for all of the pending retries have been received, that is, when the sum of the number of transactions retried and the number of `PCRDRETURN` observed is equal to the number of transactions that received a retry response.

Therefore, we update the individual outstanding transaction counters, for the observed `PCRDRETURN`s, when the following condition becomes true:

```
svt_chi_protocol_status::pending_retry_xact_count ==
svt_chi_protocol_status::num_p_crd_return_count
```

End Of Test:

The RN/SN agents, in the check phase, check if the overall outstanding transaction counter in the protocol status object, is not zero. If any of the counters is not zero, a warning is indicated to the users stating that the run phase of the test is complete whilst there are still outstanding transactions at the RN/SN agent.

6.3 Exclusive Access

6.3.1 User Interface

6.3.1.1 Configuration

The following controls have been provided in `svt_chi_node_configuration`, to let users control the exclusive access support for a node.

rand bit attribute <code>svt_chi_node_configuration::exclusive_access_enable = 0</code>
This bit controls the generation and support of exclusive sequence in RN and SN. <ul style="list-style-type: none"> ❖ Active RN: Enables generation of exclusive access transactions. ❖ SN, Passive RN: Indicates whether the SN, Passive RN supports exclusive access or not. ❖ Default value: 1'b0 ❖ type: Static

rand bit attribute <code>svt_chi_node_configuration::exclusive_monitor_enable = 1</code>
This bit controls enabling CHI RN exclusive monitor and applicable only when <code>svt_chi_node_configuration :: exclusive_access_enable</code> bit is set to 1. <ul style="list-style-type: none"> ❖ When set to 1, exclusive monitor is enabled in CHI RN and in case of RN-F, when the response is EXOK, cache is updated based on exclusive monitor state in addition to coherent response. ❖ When set to 0, exclusive monitor is disabled in CHI RN and in case of RN-F, cache is updated solely based on coherent response. ❖ Default value: 1'b0 ❖ type: Static

```
rand int attribute
svt_chi_node_configuration::max_num_exclusive_access =
`SVT_CHI_MAX_NUM_EXCLUSIVE_ACCESS
```

- ❖ Active RN: The maximum number of active exclusive transactions that will be initiated by the RN.
- ❖ Passive RN: The maximum number of exclusive transactions that can be active at any given point in time. A warning is flagged if the number of active exclusive transactions exceeds this value.
- ❖ SN: The maximum number of active exclusive transactions that can be active at a given point in time. In case of Passive SN, attempts to exceed this max number results in a warning being flagged. In case of an Active SN, exceeding this number results in a failed exclusive access read response of OK instead of EXOK
- ❖ min val: 0
- ❖ max val: Value defined by macro `SVT_CHI_MAX_NUM_EXCLUSIVE_ACCESS. Default value is 4.
- ❖ type: Static

Note: if it is set to '0' value then there are no restrictions on maximum number of active exclusive transactions. VIP will drop any exclusive sequence initiated post the maximum allow exclusive access limit is reached. Currently max_num_exclusive_access supported value is only 4

6.3.1.2 Transaction

The Field 'is_exclusive' in the svt_chi_transaction class defines the exclusive nature of a transaction.

```
rand bit attribute
svt_chi_base_transaction::is_exclusive = 0
```

This field defines the exclusive bit of:

- The Transaction (svt_chi_transaction :: xact_type) AND
- The Request flit (svt_chi_flit :: flit_type = svt_chi_flit:REQ)

Value of 0 indicates that the corresponding transaction is a normal transaction.

Value of 1 indicates that the corresponding transaction is an exclusive type transaction.

The Exclusive bit must only be used with the following transactions:

- ReadShared
- ReadClean
- CleanUnique
- ReadNoSnp
- WriteNoSnp

VIP provides a mechanism to track exclusive transaction status and exclusive monitor status for a coherent exclusive transaction at any given time, through the following attributes.

excl_access_status_enum attribute

svt_chi_transaction::excl_access_status = EXCL_ACCESS_INITIAL

Represents the status of coherent exclusive access. Following are the possible status types:

- EXCL_ACCESS_INITIAL : Initial state of the transaction before it is processed by RN
- EXCL_ACCESS_PASS : CHI exclusive access is successful
- EXCL_ACCESS_FAIL : CHI exclusive access is failed

A combination of excl_access_status and excl_mon_status can be used to determine the reason for failure of exclusive store. Please refer to subsequent sections for more details.

excl_mon_status_enum attribute

svt_chi_transaction::excl_mon_status = EXCL_MON_INVALID

Represents the status of RN exclusive monitor. The following are the possible status types:

- EXCL_MON_INVALID: RN exclusive monitor does not monitor the exclusive access on the cache line associated with the transaction
- EXCL_MON_SET: RN exclusive monitor is set for exclusive access on the cache line associated with the transaction
- EXCL_MON_RESET: RN exclusive monitor is reset for exclusive access on the cache line associated with the transaction

A combination of excl_access_status and excl_mon_status can be used to determine the reason for failure of exclusive store. Please refer to subsequent sections for more details.

excl_xact_drop_cond_enum attribute

svt_chi_transaction::excl_xact_drop_cond = EXCL_MON_FAILURE_COND_DEFAULT_VALUE

This enum represents the value for conditions under which the RN coherent exclusive transactions are dropped. Conditions are based on the values of excl_mon_status and excl_access_status. Combination of these attributes causes the transaction to drop.

The following are the possible status types:

- EXCL_MON_FAILURE_COND_DEFAULT_VALUE: Default value.
- EXCL_MON_RESET_ACCESS_FAIL_XACT_DROPPED: Set if the exclusive transaction is dropped as the excl_mon_status is reset and the exclusive_access_status is failed
- EXCL_MON_SET_ACCESS_FAIL_XACT_DROPPED: Set if the exclusive transaction is dropped as the excl_mon_status is set and the exclusive_access_status is failed as unexpected INVALID cache line status encountered
- EXCL_MON_SET_ACCESS_PASS_XACT_DROPPED: Set if the exclusive transaction is dropped as when RN has a cacheline for which CU is generated, excl_mon is set for the same cacheline address and the cacheline state is unique, which means this is a RN speculative transaction to a cacheline present in its own cache. In such scenario, do a silent write into the cache and drop the transaction. Need not issue this transaction.
- EXCL_MON_RESET_SNOOP_INVALIDATION: Set to indicate that exclusive sequence needs to restart as the exclusive monitored RN cache has been invalidated by the incoming invalidating snoop request from different lpid or due to the normal coherent store[CU] transactions.
- EXCL_MON_RESET_STORE_WITHOUT_LOAD_XACT_DROPPED: Set if the exclusive transaction is dropped as the excl_mon_status is invalid and exclusive_access_status is failed. this occurs when store without load is issued by the RN.
- EXCL_MON_INVALID_MAX_EXCL_ACCESS_XACT_DROPPED: Set if the exclusive transaction is dropped when the maximum number of active exclusive acessses exceeds 4.

See the HTML class reference for more details.

6.3.1.3 Exclusive Transaction Attributes Use Case

Some of the transaction attributes use case are explained below.

- ❖ If an Exclusive load is issued from an RN and it completes with an EXOK response, then the excl_mon_status field is set (EXCL_MON_SET). For the subsequent Exclusive Store, if the RN gets an EXOK response, then excl_access_status in the Exclusive Store is set to pass (EXCL_ACCESS_PASS), indicating that the exclusive sequence completed successfully.
- ❖ If an Exclusive load is issued from an RN and it completes with an OK response, then excl_mon_status field is set (EXCL_MON_SET). For the subsequent Exclusive Store, if the RN gets an EXOK response, then excl_access_status in the Exclusive Store is set to fail (EXCL_ACCESS_FAIL), indicating that the exclusive sequence failed.
- ❖ If an Exclusive load is issued from an RN and it completes with an OK response, then excl_mon_status is reset (EXCL_MON_RESET). The subsequent Exclusive Store is dropped by the RN, and the excl_access_status field in the dropped Exclusive Store is set to fail (EXCL_ACCESS_FAIL), while excl_xact_drop_cond is set to 'EXCL_MON_RESET_ACCESS_FAIL_XACT_DROPPED'.
- ❖ An exclusive load is initiated from RN1 which completes with an EXOK response. The excl_mon_status field is set (EXCL_MON_SET). Before the corresponding Exclusive Store is issued, an exclusive load-store pair is initiated from a different RN, say RN2, to the same address. Invalidating snoop request corresponding to the Exclusive store from RN2 will cause invalidation of

the cacheline at RN1. If the cacheline is invalidated, the Exclusive Store corresponding to the Exclusive load from RN1 is dropped as the monitor is reset due to snoop invalidation. The `excl_access_status` field in the dropped Exclusive store is set to fail (`EXCL_ACCESS_FAIL`) while `excl_xact_drop_cond` is set to `'EXCL_MON_RESET_SNOOP_INVALIDATION'`.

6.3.2 VIP Behavior

The following are the details on the supported behavior of different VIP components:

VIP supports only a load/read followed by store/write exclusive sequence. When an LP initiates an Exclusive Read or Exclusive Load transaction, the agents register the corresponding exclusive sequence only after an EXOK response is observed for the Exclusive Read/Load. The VIP agents contain an Exclusive Monitor which is used to monitor the address location of the exclusive sequence. An Exclusive sequence is expected to complete successfully, if there are no intervening Stores/Writes to the address location that is being monitored for the current exclusive sequence.



Note

The terms Exclusive load and Exclusive store refer to exclusive requests sent to snoopable memory locations. The terms Exclusive read and Exclusive write refer to exclusive requests sent to a non-snoopable memory location.

6.3.2.1 RN Agent

Exclusive transactions are monitored by an exclusive monitor at the RN and SN agents. Both active and passive RN components support coherent and non-coherent exclusive transactions.

If the active RN gets an exclusive store/write request from the sequence, without there having been a corresponding exclusive load/read previously, it drops the exclusive store/write request. If the passive RN observes an exclusive store/write request without a corresponding exclusive load/read, it flags a warning.

The active RN agent sends a new Exclusive Load/Read request to the Interconnect only when the number of outstanding Exclusive sequences is less than `svt_chi_node_configuration::max_num_exclusive_access`. If the number of outstanding exclusive sequences is equal to `svt_chi_node_configuration::max_num_exclusive_access`, the new Exclusive load request is dropped by the active RN. Subsequent store/write will be treated as store/write without load/read.

After successful exclusive store/write, the exclusive monitor is reset. Any exclusive store/write to the same address post this will be treated as exclusive store/write with exclusive load/write.

Passive RN checks if valid exclusive transactions are issued. It ensures the exclusive response received for exclusive transactions are as expected.

In the passive RN, if the number of outstanding Exclusive sequences becomes greater than `svt_chi_node_configuration::max_num_exclusive_access`, a warning is issued after which the outstanding transactions are processed normally.



Note

Outstanding exclusive sequence here refers to the Exclusive sequences wherein the Exclusive Load is complete but the corresponding Exclusive Store is not yet initiated.

6.3.2.2 SN Agent

Exclusive monitor at the SN agent receives the exclusive traffic and responds with appropriate error types. SN agent supports exclusive transactions to non-snoopable memory locations for exclusive monitoring. If

exclusive write transactions are received by the SN agents without a preceding exclusive read transaction, the active/passive SN agent flags a warning. The active SN, after flagging the warning, drops the Write Data and sends an OK response.

6.3.2.3 CHI Interconnect VIP

CHI Interconnect VIP instantiates CHI exclusive monitor and responds with appropriate error types. Interconnect VIP supports both coherent and non-coherent exclusive transaction monitoring.

For non-coherent exclusive transactions received from an RN, that complete with an EXOK response, the Interconnect VIP sends corresponding non-exclusive transactions to the SN. If a non-coherent exclusive transaction issued by an RN completes with a OK response, the Interconnect VIP does not initiate any corresponding SN transactions.

If the Interconnect observes an exclusive store/write request without a corresponding exclusive load/read, it flags a warning and respond with OK response.

6.3.2.4 CHI System Monitor

Instantiates the CHI exclusive monitor to perform checks on the exclusive transactions seen in the system. Both coherent and non-coherent exclusive transactions are monitored by the system monitor.

If the System monitor observes an exclusive store/write request without a corresponding exclusive load/read, it flags a warning and expects OK response.

Refer to the 'Protocol checks' tab in the class reference to know more about the system level as well as the protocol level checks related to Exclusive access.

For Protocol level checks - Under the group 'protocol', refer to the sub-group. 'This group contains checks for Exclusive Access'

For System level checks - Refer to the group 'chi system'

6.3.2.5 Sequence Collection

New sequences have been added to exercise coherent, non-coherent exclusive sequences. Refer to the 'Sequences' tab in the class reference HTML for sequence for more details.

- ❖ For System Virtual sequences related to Exclusive access, refer to the group 'CHI_EXCLUSIVE_ACCESS'
- ❖ For RN transaction sequences related to Exclusive access, refer to the group 'CHI_RN_EXCLUSIVE'

6.3.2.6 Coverage

Added new covergroups to cover exclusive transaction type and error response type received by each of the exclusive transactions.

For more information, see the Covergroups tab in the class reference HTML.

6.3.2.7 Unsupported Features and Limitations

- ❖ Exclusive Store without a corresponding Exclusive load is not supported. Such transaction are dropped by active RN, passive RN will flag it as a warning if it sees exclusive store/write without corresponding exclusive load/store.
- ❖ At the Interconnect, there is no way to limit the maximum number of exclusive access threads that can be active for a given HN.

- ❖ However, RN/SN agents have a limit on the exclusive accesses that can be active at any given time, which can be programmed using the `svt_chi_node_configuration::max_num_exclusive_access` attribute. This mechanism helps limit the number of outstanding exclusive access sequences from a given RN/SN, in order to tune the system performance.
- ❖ First exclusive store after system reset can be successful. This is not supported.
- ❖ RN-D and RN-I nodes do not support Exclusive accesses.
- ❖ Additional address comparison is not supported At PoC monitor while comparing address and LPID information for an exclusive sequence, subset of address bits and LPID bits can be used for comparison. Currently full address bits and LPID bits are used for comparison.

6.4 Target ID Remapping

If the System Address Map (SAM) is present in the interconnect, it is possible for the Interconnect to remap the target ID field that is programmed in the requests sent by the Request Nodes (RNs). The source ID set in the response flit might not match the target ID field that was received in the request. In such cases, the RN correlates the responses received from the interconnect to their respective request and, also determines the subsequent flits that is delivered to the Interconnect, for the corresponding transaction, with the updated target ID field.

6.4.1 User Interface

The following parameters added to `svt_chi_system_configuration`, indicates if the interconnect supports target ID remapping.

For more information, see the `svt_chi_system_configuration` class in the CHI SVT UVM Class Reference Guide.

Table 6-1

Attribute	bit <code>svt_chi_system_configuration:: expect_target_id_remapping_by_interconnect</code>
Default	0
Static	Yes
Description	<p>Specifies if the <code>target_ID</code> field set in the RN request is expected to be remapped by the Interconnect or not.</p> <p>Must be set to 1 if the Interconnect supports target ID remapping and the System Address Map programmed in the system configuration does not match the System Address Map of the Interconnect.</p> <p>Setting can be avoided, if the System Address map programmed in the system configuration is similar to the System Address Map of the Interconnect, even if the Interconnect supports Target ID remapping.</p> <p>Setting can be avoided, if the Interconnect does not support target ID remapping. Also, in such cases the System Address Map programmed in the system configuration must match the System Address Map of the Interconnect.</p>

6.4.2 VIP Features

When `svt_chi_system_configuration:: expect_target_id_remapping_by_interconnect` is configured to 1, the RN agent expects the interconnect to remap the target ID field that is received in the request. When the interconnect sends a response to the request flit, the VIP RN selects the `src_id` from the response flit and overwrites the `tgt_id` field in the corresponding transaction handle to this value.

Therefore, any subsequent flits that are issued by the RN for this transaction, will have the updated `tgt_id` field. Additionally, the following features are impacted when `svt_chi_system_configuration::expect_target_id_remapping_by_interconnect` is set:

6.4.2.1 Streaming Ordered WriteUniques

Active RN agent presumes that all WriteUniques are targeted to an HN-F and, hence, does not use the optimized version of the Streaming Ordered WriteUnique flow.

Passive RN does not expect the corresponding RN agent to use the Optimized Streaming Ordered WriteUnique flow and throws an error if it observes two outstanding Ordered WriteUniques even if their target IDs do not match.

6.4.2.2 Barrier

Active RN presumes that the potential target of any Normal Non-cacheable, and Device memory request is targeted to an HN-I. The Active RN issues a Barrier request only when all non-cacheable, and device memory Write requests that it delivered prior to the Barrier receives a Comp Response.

Passive RN flags an error if a Barrier request is received in the presence of outstanding normal non-cacheable, and device memory Write requests that did not receive a Comp Response.

Active RN initiates a transaction that is to be ordered by the Barrier (a post-barrier transaction) only after a Comp Response is received for all outstanding write requests, irrespective of the target type (as it assumes that all Write requests will get remapped to an HN-F).

6.4.2.3 Communicating Nodes Checks

The source/target HN type checks, that are part of `<req/rsp/dat/snp>_flit*_check`, are disabled.

7

Using CHI Verification IP

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

This chapter discusses the following topics:

- ❖ [SystemVerilog UVM Example Testbenches](#)
- ❖ [Installing and Running the Examples](#)
- ❖ [How to Generate SN Response](#)
- ❖ [Connecting an AXI slave to CHI Interconnect](#)
- ❖ [Connecting an ACE-Lite Master to CHI Interconnect](#)
- ❖ [Why the User Needs to Disable Auto Item Recording](#)
- ❖ [Debug Features](#)

7.1 SystemVerilog UVM Example Testbenches

This section describes SystemVerilog UVM example testbenches that show general usage for various applications. A summary of the examples is listed in this table.

Table 7-1 SystemVerilog Example Summary

Example Name	Level	Description
tb_chi_svt_uvm_basic_sys	Basic	<p>The example consists of the following:</p> <ul style="list-style-type: none">• 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 and a random test.• The tests create a testbench environment, which in turn creates AMBA System Env. AMBA System Env creates CHI System Env.• CHI System Env is configured with one RN agent and one SN agent.

Table 7-1 SystemVerilog Example Summary (Continued)

Example Name	Level	Description
tb_chi_svt_uvm_basic_progr m_sys	Basic	<p>This example demonstrates the usage of program block. It consists of the following:</p> <ul style="list-style-type: none"> • A top-level module, which includes the user program, tests, instantiates interfaces, HDL interconnect wrapper, generates system clock, and runs the tests. • The <code>chi_basic_tb.sv</code> file that contains the user program in the example • A base test, which is extended to create a directed and a random test. • The tests create a testbench environment, which in turn creates AMBA System Env. AMBA System Env creates CHI System Env. • CHI System Env is configured with one RN agent and one SN agent.
tb_chi_svt_uvm_intermediate _sys	Intermediate	<p>This example demonstrates Multiple RN Agents connected to multiple SN agents through the CHI Interconnect VIP. In this example 4 RN-F Agents and 4 RN-I Agents are connected to 2 SN-F Agent through the CHI Interconnect VIP.</p> <p>This example also demonstrates the use of System Monitor.</p>
tb_chi_svt_uvm_advanced_sys	Advanced	Not yet supported

The examples are located at:

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

7.2 Installing and Running the Examples

Steps for installing and running example `tb_chi_svt_uvm_basic_sys` is as follows:

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_uvm_basic_sys -svtb
```

The example is installed in the following location:

```
<design_dir>/examples/sverilog/amba_svt/tb_chi_svt_uvm_basic_sys
```

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

- a. Use the following Makefile:

One test is provided in the `tests` directory.

i. `ts.base_test.sv`

To run test `ts.base_test.sv`, do following:

```
gmake USE_SIMULATOR=vcsvlog base_test WAVES=1
```


Invoke `gmake help` to show more options.

- b. Use the following sim script:

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

```
./run_chi_svt_uvm_basic_sys -w base_test vcsvlog
```

Invoke `./run_chi_svt_uvm_basic_sys -help` for more options.

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

```
$DESIGNWARE_HOME/vip/svt/amba_svt/latest/examples/sverilog/tb_chi_svt_uvm_basic_sys/README
```

OR

```
<design_dir>/examples/sverilog/amba_svt/tb_chi_svt_uvm_basic_sys/README
```



Note Similar steps are also applicable for other examples.

7.2.1 Support for UVM version 1.2

While using UVM 1.2, note the below requirements:

- ❖ When using VCS version H-2013.06-SP1 and lower versions, you must define the `USE_UVM_RESOURCE_CONVERTER` macro. This macro is not required to be defined with VCS version I-2014.03-SP1 and higher versions.
- ❖ It is not required to define the `UVM_DISABLE_AUTO_ITEM_RECORDING` macro.

7.3 How to Generate SN Response

SN responses are generated by the CHI SN Agent component. The SN monitor contains a blocking peek port named `response_request_imp` which gets updated with any requests that target the SN. The SN agent connects this peek port to the SN sequencer which has a blocking peek port `response_request_port`. SN sequences must obtain a handle to the request through this port in the sequence, fill in the response information, and then submit this response to the driver through the sequencer or driver communication. The following code segment demonstrates a simple random response generator:

```
virtual task body();
  forever begin
    // Wait until a request is available response_request_port.get(req_resp);
    $cast(req, req_resp);
    // Return with an unconstrained random response
    `uvm_rand_send(req)
  end
endtask: body
```

**Note**

The SN sequence can customize the response as per the requirement, prior to sending it to the driver. The only restriction is that the request handle received from the blocking peek port must be updated and sent to the driver in zero time.

7.4 Connecting an AXI slave to CHI Interconnect

The CHI Interconnect can be connected to an AXI slave through a bridge as shown below. There must be an SN agent corresponding to each AXI slave that the CHI interconnect communicates with. The CHI interconnect routes transactions to the corresponding SN. A bridge sequence in the corresponding SN converts the CHI transactions to AXI transactions and initiates it on an AXI master. It is recommended to connect the AXI master to the user's AXI slave.

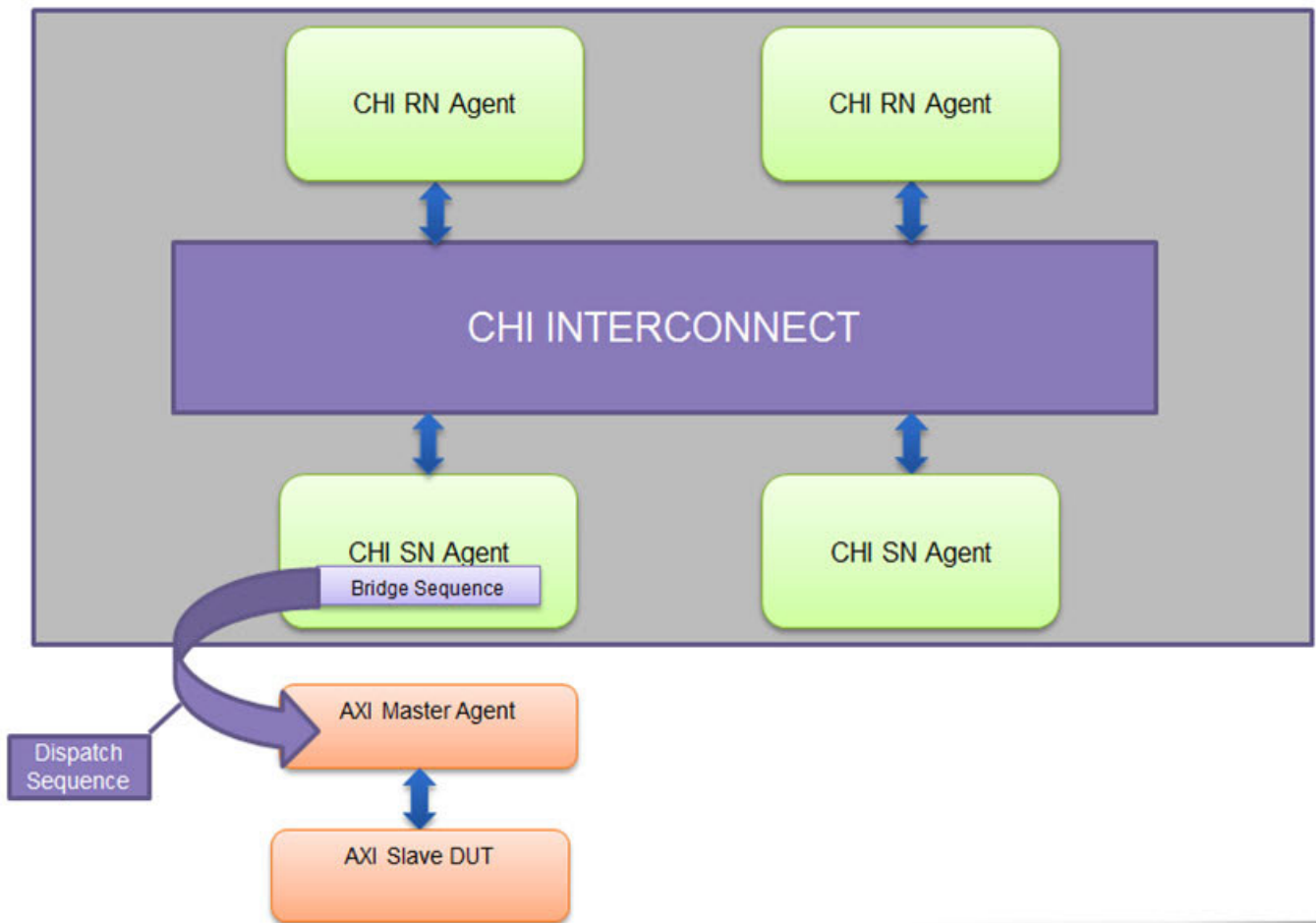
Perform the following steps to connect the AXI Slave to CHI Interconnect:

1. Create an AXI system env to instantiate bridge related components. In the following figure, the AXI system env will encapsulate the AXI Master Agent.
2. Configure the AXI master agent with compatible configuration properties as that of the AXI Slave DUT/VIP.
3. Connect the interface signals of AXI master agent to the AXI slave DUT/VIP.
4. Run a specific bridge sequence on the SN agent corresponding to this AXI slave. This sequence is the `svt_chi_sn_xact_to_axi_master_xact_sequence`, which is shipped with the VIP.
5. It is recommended to set `axi_master_sequencer` property of the `svt_chi_sn_xact_to_axi_master_xact_sequence`.

The following code snippet shows the above steps (4) and (5):

```
svt_chi_sn_xact_to_axi_master_xact_sequence sn_xact_to_axi_xact_seq =
new("sn_xact_to_axi_xact_seq");
sn_xact_to_axi_xact_seq.axi_master_sequencer =
env.amba_system_env.axi_system[1].master[0].sequencer;
sn_xact_to_axi_xact_seq.start(env.amba_system_env.chi_system[0].sn[0].sn_xact_seqr, null,
-1, 1);
```

For the entire list of updates to be done in connecting an AXI slave through a bridge sequence, see the `tb_chi_svt_uvm_intermediate_sys` example and the README.

Figure 7-1 Connecting an AXI Slave to CHI Interconnect

7.5 Connecting an ACE-Lite Master to CHI Interconnect

The CHI Interconnect can be connected to an ACE-Lite Master through a bridge as displayed in the following figure. There must be an RN agent corresponding to each ACE-Lite Master that the CHI interconnect communicates with. The CHI interconnect routes transactions to the corresponding SN. A bridge sequence in the corresponding ACE-Lite Slave converts the ACE-Lite transactions to CHI transactions and initiates it on RN. The ACE-Lite Slave must be connected to the user's ACE-Lite Master. Perform the following steps to connect an ACE-Lite Master to CHI Interconnect are as follows:

1. Create an AXI System Env to instantiate bridge related components. In the following figure, the AXI System Env encapsulates the ACE-Lite Slave Agent.
2. Configure the ACE-Lite Slave agent with compatible configuration properties as that of the ACE-Lite Master DUT or VIP.
3. Connect the interface signals of ACE-Lite Slave agent to the ACE-Lite master DUT or VIP.
4. Run a specific bridge sequence on the ACE-Lite Slave agent corresponding to the ACE-Lite Master. The bridge sequence is `svt_axi_slave_xact_to_chi_rn_xact_sequence`, which is shipped with the VIP.

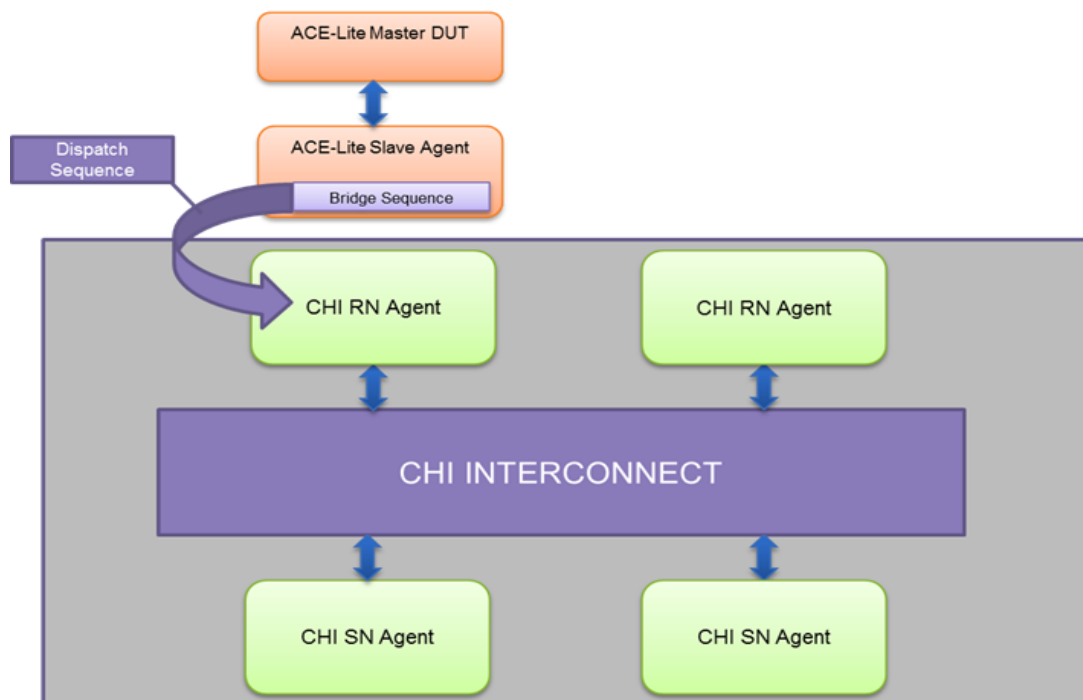
- Must set property `rn_xact_seqr` of the `svt_axi_slave_xact_to_chi_rn_xact_sequence`.

The following code snippet shows the above steps (4) and (5):

```
svt_axi_slave_xact_to_chi_rn_xact_sequence axi_slave_xact_to_chi_rn_xact_seq =
new("axi_slave_xact_to_chi_rn_xact_seq");
axi_slave_xact_to_chi_rn_xact_seq.rn_xact_seqr =
env.amba_system_env.chi_system[0].rn[4].rn_xact_seqr;
axi_slave_xact_to_chi_rn_xact_seq.start(env.amba_system_env.axi_system[2].slave[0].sequencer, null, -1, 1);
```

For the complete list of updates to be performed while connecting an ACE-Lite Master through a bridge sequence, see the `tb_chi_svt_uvm_intermediate_sys` example and the README.

Figure 7-2 Connecting an ACE-Lite Master to CHI Interconnect



7.6 Why the User Needs to Disable Auto Item Recording

If you are using AHB UVM, AXI UVM, or CHI UVM Verification IP, it is recommended to define a macro named `UVM_DISABLE_AUTO_ITEM_RECORDING`. This section describes why this macro needs to be defined, and what are its implications if a user defined driver and sequencer also exist in the same environment.

CHI, AXI, and AHB protocols are pipelined protocols, where the driver needs to initiate the next transaction before the previous transaction completes. Thus, the VIP driver indicates `seq_item_port.item_done()` much before the transaction is completed on the bus, so that the sequencer can provide next sequence item to the driver. the driver does not wait for a transaction to complete before calling `seq_item_port.item_done()`.

For CHI or AXI, `seq_item_port.item_done()` is called after the driver accepts a transaction from the sequencer. For AHB, `seq_item_port.item_done()` is called when penultimate beat address of the current

transaction is accepted by the slave. The VIP explicitly marks end of transaction when the transaction is completed on the interface, instead of letting UVM do it. Hence, VIP needs to define `UVM_DISABLE_AUTO_ITEM_RECORDING` macro.

If the environment contains a user defined driver or sequencer, and the macro `UVM_DISABLE_AUTO_ITEM_RECORDING` is defined, user needs to make sure that the driver explicitly marks end of transaction when transaction actually completes on the interface. For example, for a non- pipelined protocol, user can call `req.end_tr()` in the driver code after calling `seq_item_port.item_done()`, assuming that `seq_item_port.item_done()` is called only after the transaction is complete. Alternatively, user can call `req.end_tr()` in the corresponding sequence, after the sequence unblocks based on `seq_item_port.item_done()`.

For pipelined protocol, you are required to wait till the transaction is completed on the bus, before calling the `req.end_tr()`.

Code snippet for the driver (assuming non-pipelined protocol) is as follows:

```
seq_item_port.item_done();
req.end_tr();
```

Code snippet for the sequence (assuming non-pipelined protocol) is as follows:

```
`uvm_do(req);
req.end_tr();
```



Note

VIPs for pipelined and non-pipelined protocols are designed to work appropriately when `UVM_DISABLE_AUTO_ITEM_RECORDING` macro is defined.

7.7 Debug Features

The following are the debug capabilities provided by CHI Verification IP. Refer to HTML class reference for more details.

❖ 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, AXI slave 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, AXI slave transaction summary as and when its available with UVM_FULL verbosity.
- ◆ `svt_chi_system_configuration::enable_summary_tracing` Enables interactive writing of coherent and snoop transaction summary, SN transaction summary, AXI slave transaction summary into respective separate files as and when its available.

❖ CHI RN agent:

- ◆ `svt_chi_node_configuration::enable_pl_reporting` Enables interactive display of coherent and snoop transaction tracing summary when these transaction are complete with UVM_FULL verbosity
- ◆ `svt_chi_node_configuration::enable_pl_tracing` Enables interactive writing of coherent and snoop transaction tracing summary with corresponding flit info to separate files when these transactions are complete

8

Using CHI B Verification IP

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

This chapter discusses the following topics:

- ❖ [Enabling CHI Issue B](#)
- ❖ [Using CHI Issue B Compliant and Legacy VIP Components within Same Simulation](#)
- ❖ [CHI-B Transactions](#)
- ❖ [DMT](#)
- ❖ [DCT](#)
- ❖ [Updates to Transaction Data Classes](#)
- ❖ [Updates to the Signal Interface](#)
- ❖ [READNOTSHAREDDIRTY](#)
- ❖ [READONCECLEANINVALID](#)
- ❖ [READONCEMAKEINVALID](#)
- ❖ [CLEANSHAREDPERST](#)
- ❖ [DMT](#)
- ❖ [DCT](#)
- ❖ [RETTOSRC](#)
- ❖ [DONOTGOTOSD](#)
- ❖ [Atomic Transactions](#)
- ❖ [Checks for the New Read Transactions](#)
- ❖ [Checks for Variable Width](#)
- ❖ [Checks for DMT](#)
- ❖ [Checks for DCT](#)
- ❖ [Checks for RETTOSRC](#)
- ❖ [Checks for DONOTGOTOSD](#)
- ❖ [Checks for Atomic Transactions](#)

❖ [Sequence Collection Updates](#)

8.1 Enabling CHI Issue B

8.1.1 Compile Time Macros

Table 8-1 **Macros**

Macro	Description
<code>`SVT_CHI_ISSUE_B_ENABLE</code>	This macro must be defined to use the CHI-B features of the VIP. Controls the interface signals and other structural changes needed.
<code>`SVT_CHI_NODE_CFG_DEFAULT_CHI_SPEC_REVISION</code>	This macro must be set to <code>ISSUE_B</code> if the <code>svt_chi_node_configuration::chi_spec_revision</code> is to be set to <code>ISSUE_B</code> for all the VIP nodes in the system. If set to <code>ISSUE_A</code> , the <code>chi_spec_revision</code> enum will be set to <code>ISSUE_A</code> .
<code>`SVT_CHI_IC_CFG_DEFAULT_CHI_SPEC_REVISION</code>	This macro must be set to <code>ISSUE_B</code> if the <code>svt_chi_interconnect_configuration::chi_spec_revision</code> is to be set to <code>ISSUE_B</code> for the VIP Interconnect. If set to <code>ISSUE_A</code> , the <code>chi_spec_revision</code> enum will be set to <code>ISSUE_A</code> .

8.1.2 Configuration Parameters/APIs

The following APIs have been added to `svt_chi_system_configuration` to support DMT:

Table 8-2 **APIs**

Configuration APIs
<pre>function void svt_chi_system_configuration::set_hn_dmt_enable (bit dmt_enable [])</pre> <ul style="list-style-type: none"> Sets the 'DMT Enable' for each of the HN indices. DMT is applicable only when compile time macro is defined, and all the RN&SN agents are programmed with <code>svt_chi_node_configuration::chi_spec_revision=svt_chi_node_configuration::ISSUE_B</code>. CHI Interconnect VIP doesn't support DMT feature. So, setting the DMT as enabled for any of the HNs with <code>use_interconnect</code> set to 1 is not valid. <p><code>dmt_enable</code> - An array containing the 'DMT Enable' setting of the each of the HNs. The size of this array must be equal to <code>num_hn</code>. <code>node_id[0]</code> corresponds to <code>node_id</code> of HN 0 and so on.</p> <p>Note: The order of programming 'DMT Enable' should be such that all the HN-F node 'DMT Enable' should be programmed first, followed by HN-I node 'DMT Enable'. DMT is not applicable for HN-I, but for the sake of consistency with other APIs related to HN programming, the size of the <code>dmt_enable[]</code> array must be equal to <code>num_hn</code>. The DMT enable settings are used by RN, SN agents and CHI System Monitor. The default value of DMT enable for all the HNs can be controlled through the macro <code>`SVT_CHI_ENABLE_DMT</code> (whose default value is 0).</p>

Table 8-2 APIs

Configuration APIs	
<pre>function bit svt_chi_system_configuration::is_dmt_enabled (int hn_idx)</pre>	<p>Returns the 'DMT Enable' setting for a given HN index</p> <p>hn_idx - Home Node index to be used in the lookup</p>
<pre>function void svt_chi_system_configuration::set_hn_dct_enable (bit dct_enable [])</pre> <ul style="list-style-type: none"> Sets the 'DCT Enable' for each of the HN indices. DCT is applicable only when compile time macro is defined, and all the RN&SN agents are programmed with <code>svt_chi_node_configuration :: chi_spec_revision=svt_chi_node_configuration :: ISSUE_B</code>. CHI Interconnect VIP doesn't support DCT feature. So, setting the DCT as enabled for any of the HNs with <code>use_interconnect</code> set to 1 is not valid. <p>dct_enable - An array containing the 'DCT Enable' setting of the each of the HNs. The size of this array must be equal to <code>num_hn.node_id[0]</code> corresponds to <code>node_id</code> of HN 0 and so on.</p> <p>Note: The order of programming 'DCT Enable' should be such that all the HN-F node 'DCT Enable' should be programmed first, followed by HN-I node 'DCT Enable'. DCT is not applicable for HN-I, but for the sake of consistency with other APIs related to HN programming, the size of the <code>dmc_enable[]</code> array must be equal to <code>num_hn</code>. The DCT enable settings are used by RN, SN agents and CHI System Monitor. The default value of DCT enable for all the HNs can be controlled through the macro <code>`SVT_CHI_ENABLE_DCT</code> (whose default value is 0).</p>	
<pre>function bit svt_chi_system_configuration::is_dct_enabled (int hn_idx)</pre>	<p>Returns the 'DCT Enable' setting for a given HN index</p> <p>hn_idx - Home Node index to be used in the lookup</p>


8.1.2.1 CHI Node Agents**Table 8-3 Node Agents**

Attribute	
bit svt_chi_node_configuration::chi_spec_revision	
Default	svt_chi_node_configuration::ISSUE_A
Static	Yes

Table 8-3 Node Agents

Attribute	
Description	<p>When this attribute is set to 'ISSUE_B' and the compile macro <code>`SVT_CHI_ISSUE_B_ENABLE</code> is defined, the CHI VIP RN/SN node agents support CHI Issue B features.</p> <p>There are other configuration parameters specific to each of the CHI Issue B feature, which are considered only when this attribute is set to 'ISSUE_B'.</p> <p>When this attribute is set to 'ISSUE_A' but the compile macro <code>`SVT_CHI_ISSUE_B_ENABLE</code> is defined, the CHI RN/SN node agents do not support CHI Issue B. However, the interface signals will be as per CHI Issue B specification for such nodes.</p> <p>When this attribute is set to 'ISSUE_A' and the compile macro <code>`SVT_CHI_ISSUE_B_ENABLE</code> is not defined, the CHI RN/SN node agents do not support CHI Issue B and the interface signals will be as per the CHI Issue A specification for such nodes.</p> <p>Unsupported:</p> <p>If this represents the Interconnect VIP's RN/SN connected node configuration, indicates whether the external RN/SN agent connected to that specific interconnect port supports CHI Issue B or not.</p>
Validity	Valid to set to <code>ISSUE_B</code> : Only when the compile macro <code>`SVT_CHI_ISSUE_B_ENABLE</code> is defined.
<code>bit svt_chi_node_configuration::addr_width</code>	
Default	<code>`SVT_CHI_MAX_ADDR_WIDTH</code>
Static	Yes
Description	<p>Indicates the width of the address field for the RN/SN node.</p> <p>For a CHI-B node, this can take any value between 44 and <code>`SVT_CHI_MAX_ADDR_WIDTH</code></p> <p>For a CHI-A node, this parameter (as well as <code>`SVT_CHI_MAX_ADDR_WIDTH</code>) are expected to be fixed to 44.</p> <p>The width of the address field in the request/snoop channel is always fixed to <code>`SVT_CHI_MAX_ADDR_WIDTH</code>.</p> <p>If <code>svt_chi_node_configuration::addr_width</code> is different from <code>`SVT_CHI_MAX_ADDR_WIDTH</code>, only the bits specified by <code>addr_width</code> are valid and the remaining (MSB) ones must be ignored or tied off in the interface connections.</p>
Validity	Valid to set to a value other than 44 only when: the compile macro <code>`SVT_CHI_ISSUE_B_ENABLE</code> is defined and spec revision <code>svt_chi_node_configuration::chi_spec_revision</code> is set to <code>ISSUE_B</code> .
<code>bit svt_chi_node_configuration::node_id_width</code>	
Default	<code>`SVT_CHI_MAX_NODE_ID_WIDTH</code>
Static	Yes

Table 8-3 Node Agents

Attribute	
Description	<p>Indicates the width of the node ID related fields at the RN/SN node.</p> <p>For a CHI-B node, this can take any value between 7 and `SVT_CHI_MAX_NODE_ID_WIDTH`</p> <p>For a CHI-A node, this parameter (as well as `SVT_CHI_MAX_NODE_ID_WIDTH`) are expected to be fixed to 7.</p> <p>The width of the NodeID fields (src_id, tgt_id, return_nid, home_nid and fwd_nid) in the request/snoop/data/response channels is always fixed to `SVT_CHI_MAX_NODE_ID_WIDTH`.</p> <p>If svt_chi_node_configuration::node_id_width is different from `SVT_CHI_MAX_NODE_ID_WIDTH`, only the bits specified by node_id_width are valid and the remaining (MSB) ones must be ignored or tied off in the interface connections.</p>
Validity	<p>Valid to set to a value other than 44 only when: the compile macro `SVT_CHI_ISSUE_B_ENABLE` is defined and spec revision svt_chi_node_configuration::chi_spec_revision is set to ISSUE_B.</p>
enum svt_chi_node_configuration::atomic_transactions_enable	
Default	0
Static	Yes
Description	<p>Should be set to 1 to enable atomic ops</p> <p>Permitted values: 0,1</p>
Validity	<p>Valid to set to 1: Only when the compile macro SVT_CHI_ISSUE_B_ENABLE is defined and svt_chi_node_configuration::chi_spec_revision == ISSUE_B</p>
To provide control to set the number of outstanding atomic transactions, following control is added to svt_chi_node_configuration.	
int svt_chi_node_configuration:: num_outstanding_atomic_xact	
Default	-1
Static	Yes
Description	<p>Specifies the number of outstanding Atomic transactions a node can support.</p> <p>Must be set only when #num_outstanding_xact = -1</p> <p>When set:</p> <p>If the number of outstanding Atomic transactions is equal to this number, the RN will refrain from initiating any new Atomic transactions until the number of outstanding Write transactions is less than this parameter.</p> <p>If the number of outstanding Atomic transactions is equal to this number, the SN will not acknowledge until the number of outstanding Atomic transactions becomes less than this parameter.</p> <p> Note Permitted values: -1, 1 to `SVT_CHI_MAX_NUM_OUTSTANDING_XACT`</p>
Validity	<p>Valid to be set: only when the num_outstanding_xact is set to -1 and when the compile macro SVT_CHI_ISSUE_B_ENABLE is defined and svt_chi_node_configuration::chi_spec_revision is set to ISSUE_B</p>

Following attribute has been added to the Node Protocol status class:

Table 8-4 Node Protocol Status Class

Attribute	Description
int attribute svt_chi_protocol_status::current_outstanding_atomic_xact_count	Indicates the number of total outstanding Atomic transactions at a given node

8.1.2.2 Interconnect VIP

The interconnect needs to know which of the nodes connected to interconnect are legacy and CHI-B compliant. The `rn_connected_node_cfg[]`, `sn_connected_node_cfg[]` within `svt_chi_interconnect_configuration` are of type `svt_chi_node_configuration`. So, the `svt_chi_node_configuration::chi_spec_revision` can be used to know whether the connected external RN, SN nodes are enabled with CHI Issue B or not.

However, for the interconnect VIP, it's required to program whether it is CHI-B compliant or not. For this purpose, a new attribute `svt_chi_interconnect_configuration::chi_spec_revision` has been defined.

Table 8-5 Interconnect VIP

Attributes	
bit svt_chi_interconnect_configuration:: chi_spec_revision	
Default	svt_chi_node_configuration::ISSUE_A
Static	Yes
Description	<p>When this attribute is set to 'ISSUE_B' and the compile macro `SVT_CHI_ISSUE_B_ENABLE` is defined, the CHI interconnect VIP supports CHI Issue B features.</p> <p>There are other configuration parameters specific to each of the CHI Issue B feature, which are considered only when this attribute is set to 'ISSUE_B'.</p> <p>When this attribute is set to 'ISSUE_A' but the compile macro `SVT_CHI_ISSUE_B_ENABLE` is defined, the CHI interconnect VIP does not support CHI Issue B. However, the interface signals will be as per CHI Issue B specification for each of the ports of the interconnect VIP that are connected to external RN/SN components.</p> <p>When this attribute is set to 'ISSUE_A' and the compile macro `SVT_CHI_ISSUE_B_ENABLE` is not defined, the CHI interconnect VIP does not support CHI Issue B features. The interface signals will be as per the CHI Issue A specification for each of the ports of the interconnect VIP that are connected to external RN/SN components.</p>
Validity	Valid to set to ISSUE_B: only when the compile macro `SVT_CHI_ISSUE_B_ENABLE` is defined.

8.1.3 Programming Example

The following code snippet is a programming example:

```
/**
 * Programing chi_spec_revision which, when set to 'ISSUE_B', enables the nodes to
 * support CHI Issue B spec (CHI Issue B) features.
 * In this testbench, these RN, SN agents of this system configuration are
 * in active mode.
```

```

* cfg is of type svt_amba_system_configuration or it's derived type.
* cfg.chi_sys_cfg[sys] is the CHI system configuration
*/
foreach(cfg.chi_sys_cfg[sys])begin
    foreach(cfg.chi_sys_cfg[sys].rn_cfg[rn]) begin
        cfg.chi_sys_cfg[sys].rn_cfg[rn]. chi_spec_revision =
svt_chi_node_configuration::ISSUE_B;
    end
    foreach(cfg.chi_sys_cfg[sys].sn_cfg[sn]) begin
        cfg.chi_sys_cfg[sys].sn_cfg[sn].chi_spec_revision =
svt_chi_node_configuration::ISSUE_B;
    end
end

/** If the CHI interconnect is enabled within CHI sys cfg
*   chi_ic_cfg is the handle to IC VIP cfg
*/
if (cfg.chi_sys_cfg[sys].use_interconnect == 1) begin
    chi_ic_cfg = cfg.chi_sys_cfg[sys].ic_cfg;
    /**
    * Program the chi_spec_revision and readspec_enable attributess
    * for CHI Interconnect VIP
    */
    chi_ic_cfg.chi_spec_revision = svt_chi_node_configuration::ISSUE_B;

    /** Program chi_spec_revision for RN, SN connected ports of IC VIP */
    foreach(chi_ic_cfg.rn_connected_node_cfg[j]) begin
        chi_ic_cfg.rn_connected_node_cfg[j]. chi_spec_revision =
svt_chi_node_configuration::ISSUE_B;
    end
    foreach(chi_ic_cfg.sn_connected_node_cfg[j]) begin
        chi_ic_cfg.sn_connected_node_cfg[j]. chi_spec_revision =
svt_chi_node_configuration::ISSUE_B;
    end

    /** If required, enable readspec along with snoop filter */
    chi_ic_cfg.readspec_enable = 1;
    chi_ic_cfg.snoop_filter_enable = 1;

end

```

If `svt_chi_node_configuration::chi_spec_revision` is to be set to 'ISSUE_B' for all the RN/SN nodes, users can define the compile time macro `SVT_CHI_NODE_CFG_DEFAULT_CHI_SPEC_REVISION` to `ISSUE_B`. To set

`chi_spec_revision` for the Interconnect to 'ISSUE_B', users can define the macro `SVT_CHI_IC_CFG_DEFAULT_CHI_SPEC_REVISION` to `ISSUE_B`.

8.2 Using CHI Issue B Compliant and Legacy VIP Components within Same Simulation

There are changes to interface signal widths from CHI to CHI-B. These changes need to be controlled using compile time macros. The following requirements determine the CHI VIP component used along with CHI Issue B specification compliant VIP component.

For example, consider an use case where there is a CHI system environment with 2 RN-F VIP agents. One of the RN-F is CHI Issue B compliant, while the other RN-F is legacy CHI compliant.

As CHI-B agent has additional signals for FLIT, the compile time macros need to be defined to make sure that the signal widths are compliant with the CHI Issue B. However, this makes the Interface signals of the legacy RN-F agent also to be that of CHI-B compliant RN-F agent.

It is required to tie off the signals accordingly for the legacy RN-F agent in such case, along with taking care of programming the CHI-B specific configuration parameters appropriately.

These details are explained in detail, wherever they are applicable, in the subsequent sections.

8.3 CHI-B Transactions

8.3.1 READNOTSHARED DIRTY

This transaction is used to obtain the snapshot of current value of the cache line in a state other than SharedDirty. This request has been added in order to support MESI caches.

Only applicable in case of RN-Fs.

The final cache state of the Requester can be UC, SC or UD.

CompAck must be asserted in the request.

Permitted responses to the corresponding Snoop Request (SnpNotSharedDirty):

- ❖ The Snoopee must return a copy to the interconnect if the line is in Dirty state.
- ❖ The Snoopee must transition to either SD, SC or I state.

8.3.2 ROCI

ROCI (ReadOnceCleanInvalid) is used as a Read request to a Snoopable address region to obtain a snapshot of the coherent data. It is recommended that a cached copy is invalidated but not mandatory. If a Dirty copy is invalidated, then it must be written back to memory. The received data is not cached in a coherent state at the Requester.

ROCI Transaction instead of ReadOnce is used in cases where the data is still valid but expected to be not required in near future. Use of ROCI by an application improves cache efficiency by reducing cache pollution.

Snoop corresponding to this request is SnpUnique. Home is permitted to send SnpOnce instead of SnpUnique. Home is not permitted to send Fwd type snoop.

Other characteristics of a ROCI transaction are:

- ❖ Permitted but not required to assert ExpCompAck in the request.

- ❖ Permitted to assert Order field in the request.
- ❖ A ROCI transaction is permitted to use DMT if ExpCompAck is asserted in the Request.
- ❖ A ROCI transaction with ExpCompAck de-asserted can use DMT only if Order field is also de-asserted.

Communicating node pair: RN-F, RN-D, RN-I -> ICN(HN-F)

8.3.3 ROMI

ROMI (ReadOnceMakeInvalid) is used as a Read request to a Snoopable address region to obtain a snapshot of the coherent data. It is recommended, but not required that all cached copies are invalidated. If a Dirty copy is invalidated, it does not need to be written back to memory. The received data is not cached in a coherent state at the Requester.

ROMI instead of ROCI or ReadOnce is used to obtain a snapshot of data value when the application knows that the cached data is not going to be used again and thus can free up the caches and also by dropping dirty data avoids an unnecessary write back to memory.

Snoop corresponding to this request is SnpUnique. Home is permitted to send SnpOnce instead of SnpUnique. Home is not permitted to send Fwd type snoop.

The following are other characteristics of a ROMI transaction:

- ❖ Permitted but not required to assert ExpCompAck in the request.
- ❖ Permitted to assert Order field in the request.
- ❖ A ROMI transaction is permitted to use DMT if ExpCompAck is asserted in the Request.
- ❖ A ROMI transaction with ExpCompAck de-asserted can use DMT only if Order field is also de-asserted.

Communicating node pair: RN-F, RN-D, RN-I -> ICN(HN-F)

8.3.4 CleanSharedPersist

CleanSharedPersist a new cache maintenance transaction is added in this specification. Completion response to a CleanSharedPersist request ensures that all cached copies are changed to a non-dirty state and any dirty cached copy is written back to Point of Persistence.

Data is not included with the completion response.

Transaction structure as well as transaction flow is similar to CleanShared, except that Comp response from Home to the Requester must be sent only after Comp from downstream is received by Home.

The permitted and required initial and final cache states at the Request node are same as for CleanShared.

Communicating node pairs:

RN-F, RN-D, RN-I -> ICN(HN-F, HN-I)

ICN(HN-F) -> SN-F

ICN(HN-I) -> SN-I

The expected Snoop corresponding to CleanSharedPersist request is SnpCleanShared.

Both Snoopable and Non-Snoopable SnpAttr are permitted.

8.3.5 Atomic Transactions

Atomic transactions allow for a Requester to send to the interconnect a transaction with a memory address and an operation to be performed on that location. Such a transaction moves the operation closer to where the data resides and is useful for atomically executing an operation and updating the location in a performance efficient manner. Without availability of such transactions, atomic operations have to be executed using a sequence of memory accesses. These accesses might rely on exclusive reads and writes.

Atomic transactions can be broadly classified into four categories: AtomicStore, AtomicLoad, AtomicSwap and AtomicCompare.

AtomicStore and AtomicLoad can in turn be classified into eight opcodes each, based on the Atomic operation to be performed.

Snoop corresponding to Atomic transactions is SnpUnique.

While the flow for AtomicStore is exactly the same as that of WriteUnique transactions, in case of AtomicLoad, AtomicSwap and AtomicCompare, a CompData response containing the initial data value for the given address is sent instead of a Comp response. CompAck should not be sent for Atomic transactions.

Interconnect is permitted to Snoop the requester node as well. A new field called “SnoopMe” has been introduced. If the SnoopMe field is asserted and the Interconnect determines that the requester node has a copy of the cache, it must send a SnpUnique to the requester.

Interconnect is also permitted to pass the Atomic transaction downstream and let the Slave nodes perform the atomic operation locally.

Communicating node pairs:

- ❖ RN-F, RN-D, RN-I -> ICN(HN-F, HN-I)
- ❖ ICN(HN-F) -> SN-F
- ❖ ICN(HN-I) -> SN-I

8.4 DMT

Direct Memory Transfer (DMT) defines the feature which permits Slave node to send Data directly to the Requester.

DMT can be used for following transactions:

- ❖ ReadShared, ReadClean, ReadUnique, ReadNotSharedDirty.
- ❖ ReadNoSnp, ReadOnce, ROMI, ROCI with a CompAck.
- ❖ ReadNoSnp, ReadOnce, ROMI, ROCI without a CompAck and with no ordering.

When DMT is used the Requester irrespective of Request type receives CompData_UC from the Slave. The cache state in the response is UC instead of I.

8.5 DCT

Direct Cache Transfer (DCT) defines the feature which permits a Snooped RN to send Data directly to the Requester.

DCT can be used for following transactions:

- ❖ ReadShared, ReadClean, ReadUnique, ReadNotSharedDirty, ROCI, ROMI.

8.6 Updates to Transaction Data Classes

As there are new fields in the various flit types along with expansion of existing fields of various flit types, the transaction data classes have been updated in terms of:

- ❖ Updating the enumerated data types for the field widths that are expanded/shrunk including new enumerated values
- ❖ New attributes in the transaction classes for the new fields

The definition of the macro ``SVT_CHI_ISSUE_B_ENABLE` will make the updated CHI-B enumerated data types to be available. Whereas in case this macro is not defined, the existing CHI Issue A definitions are intact.

For the new attributes within the data classes and the enumerated data types, they will be available irrespective of the macro define, however they will be applicable based on the macro and any associated configuration attributes in terms of the functionality.

As the development of CHI-B support is feature by feature, the data classes will be added with the new attributes as and when required accordingly. However, for the existing attributes that need updates in terms of additional widths, the updates will be done to pack the data class into FLIT signal, and then to unpack the FLIT signal into data class attributes.

8.6.1 DMT

For DMT, following fields are added to the common transaction class (`svt_chi_common_transaction`)

- ❖ `return_nid`: Return NID. Identifies the node to which the SN-F sends the CompData response
- ❖ `return_txn_id`: Return TxnID. Identifies the value the SN-F must use in the TxnID field of the CompData response. It can be either the TxnID generated by Home for this transaction or the TxnID in the Request packet from the Requester that originated the transaction.
- ❖ `home_nid`: Home identifier associated with the original request. The Requester uses the value in this field to determine the TgtID of the CompAck to be sent in response to CompData.



Note

The “`is_dmt_used`” field is added to the coherent transaction class (`svt_chi_transaction`) to indicate whether Direct memory transfer is used or not.

8.6.2 DCT

For DCT, following new fields have been added to the snoop transaction class (`svt_chi_snoop_transaction`):

- ❖ `fwded_compdata` : Contains data, corresponding to a forward type Snoop, that is sent to the initiating RN in case of DCT .
- ❖ `fwded_read_data_resp_err_status[]` : Contains the resperr field that is sent in the fwded compdata flits corresponding to the forward type snoop
- ❖ `fwded_read_data_rsvdc[]` : Contains the RSVDC values sent in the fwded compdata flits corresponding to the forward type Snoop.
- ❖ `num_fwded_dat_flits` : Indicates the number of forward CompData received/transmitted corresponding to the forward type Snoop.

- ❖ `is_dct_used` : Indicates if forward CompData is applicable for the forward type Snoop. In case of active RN, this flag will be set when a forward type Snoop request is received and initial cacheline state is UC/SC/UD/SD. In case of passive RN, this flag will be set when forward CompData is seen corresponding to a forward type Snoop request.

Following field has been added to the coherent transaction class (`svt_chi_transaction`):

- ❖ `is_dct_used` : Indicates if DCT was used for the coherent transaction. Will be set in `svt_chi_rn_transaction`, when an RN receives Compdata flits, corresponding to a coherent RN transaction, whose SrcID correspond to another RN.

Following field has been added to the common transaction class (`svt_chi_common_transaction`):

- ❖ `fwd_nid` : Field that contains the Node ID of the RN to which CompData must be forwarded by the snooped RN.
- ❖ `fwd_txn_id` : Field that contains the value of Txn ID that must be programmed in the CompData which is to be forwarded by the snooped RN.
- ❖ `home_nid` : The HomeNID field that must be programmed in the CompData which is to be forwarded by the snooped RN.
- ❖ `fwd_state_final_state` : Field that contains the Final state value that is programmed in the Resp field of the CompData that must be forwarded by the snooped RN.
- ❖ `fwd_state_pass_dirty` : Field that contains the Pass Dirty value that is programmed in the Resp field of the CompData that must be forwarded by the snooped RN.

8.6.3 RETTOSRC

For RetToSrc, following field has been added to the common transaction class (`svt_chi_common_transaction`):

- ❖ `ret_to_src` : The RetToSrc field that is programmed in the Snoop request sent by the Interconnect.

8.6.4 DONOTGOTOSD

For DoNotGoToSD, following field has been added to the common transaction class (`svt_chi_common_transaction`):

- ❖ `do_not_go_to_sd` : The DoNotGoToSD field that is programmed in the Snoop request sent by the Interconnect.

8.6.5 Atomic Transactions

The following field has been added to the base transaction class (`svt_chi_base_transaction`):

- ❖ `bit snoopme` : Indicates if the requester must be snooped in case of Atomic transactions.
- ❖ `bit endian` : Indicates if the atomic data is in Little/Big Endian format.
It is just an indication that data, for the given address, is stored and presented in Little Endian/Big Endian format. this information is only used by the ALU and does not affect any other component. Endian would likely only be set to '1' for an atomic request coming from a CPU core operating in big endian mode.

Following new fields will be added to the transaction class (`svt_chi_transaction`):

- ❖ `bit[(`SVT_CHI_MAX_ATOMIC_LD_ST_DATA_WIDTH-1):0] atomic_store_load_txn_data` : Data that must be sent as Write Data in AtomicStore and AtomicLoad transactions. User must program the bits appropriately based on the data size, which should be one of 1, 2, 4, or 8 bytes

- ❖ `bit[(`SVT_CHI_MAX_ATOMIC_DATA_WIDTH-1):0] atomic_compare_data` : Data that must be sent as Compare Data in AtomicCompare transactions. User must program the bits corresponding to (outbound data size/2), which should be one of 1, 2, 4, 8 or 16 bytes
- ❖ `bit[(`SVT_CHI_MAX_ATOMIC_DATA_WIDTH-1):0] atomic_swap_data` : Data that must be sent as Swap Data in AtomicSwap and AtomicCompare transactions. For AtomicSwap, user must program the bits corresponding to the outbound data size, which should be one of 1, 2, 4, or 8 bytes. For AtomicCompare, user must program the bits corresponding to (outbound data size/2), which should be one of 1, 2, 4, 8 or 16 bytes.
- ❖ `bit[(`SVT_CHI_MAX_ATOMIC_DATA_WIDTH-1):0] atomic_returned_initial_data` : Data containing the original value of the addressed location that is returned by the Completer for Atomic transactions. Only applicable when `xact_type` is set to `ATOMICLOAD`, `ATOMICCOMPARE` or `ATOMICSWAP`. User must not program this field. This field will be populated by the agent after all CompData flits corresponding to the Atomic transaction are received.



Note Only the bits corresponding to the inbound data size must be considered.

8.7 Updates to the Signal Interface

8.7.1 Support for Fields With Variable Width

As there are additional fields and updates to widths of few existing fields of various flit types, the signal interface for various *FLIT signals have been updated to ensure that these are as per CHI Issue B specific.

If a given field width can take a variable value, users can redefine the width macro corresponding to the field to the desired value. It should be noted that the width macros can be redefined only if all the nodes in the system are CHI-B compliant. In other words, the widths should be redefined only when

``SVT_CHI_ISSUE_B_ENABLE` is defined and `svt_chi_node_configuration::chi_spec_revision` for all the VIP nodes is set to `ISSUE_B`.

Eg: The width of 'addr' field can take any value ranging from 44 to 52 in a CHI-B system. If all the nodes in the system are CHI-B compliant and the address width must be specified as 48, the macro `"SVT_CHI_MAX_ADDR_WIDTH"` must be defined to 48.

If the widths vary across the different nodes in the system, then the field width macro must be set to the maximum width used across the nodes and the corresponding field width parameter in the respective node configurations must be set to match the exact width of the field at each of the nodes.

Eg: The width of the addr field is to be 46 in RN0 but 48 in RN1. In such a case, `"SVT_CHI_MAX_ADDR_WIDTH"` must be set to 48. `svt_chi_node_configuration::chi_address_width` must be set to 46 for RN0 and 48 for RN1.

The request flit interface width takes the address width macro into account. So, for RN0 the two most significant bits corresponding to the address field must be tied off.

Similarly, if width of Node ID fields in RN0 is 7 while it is 9 in RN1, the macro `"SVT_CHI_MAX_NODE_ID_WIDTH"` should be defined to 9 and `svt_chi_node_configuration::chi_node_id_width` must be set to 7 for RN0.

8.8 READNOTSHARED DIRTY

8.8.1 RN VIP

- ❖ Requester issues `ReadNotSharedDirty` when the initial cache line state is either I or UCE.
- ❖ The final cache state of the Requester will be UC, SC or UD.
- ❖ `CompAck` is asserted in the request.
- ❖ Permitted responses to the corresponding Snoop Request (`SnpNotSharedDirty`):
- ❖ The Snoopee returns a copy to the interconnect if the line is in Dirty state.
- ❖ The Snoopee transitions to either SD, SC or I state.

8.8.2 Interconnect VIP

The Interconnect, when it receives a `ReadNotSharedDirty` transaction, generates snoops of the type `SnpNotSharedDirty` to CHI Issue B compliant RNs and `SnpShared` to legacy RNs, if any.

The final `CompData` response that the Interconnect generates has a final cache state of UC, UD or SC based on the Snoop Responses that it received.

8.9 READONCECLEAN INVALID

8.9.1 RN VIP

Requester issues `ReadOnceCleanInvalid` when the initial cache line state is I.

The final cache state of the requester will be I.

The requester expects to receive either a `Comp_UC` response or `Comp_I` in response to the request.

8.9.2 Interconnect VIP

The Interconnect, upon receiving a `ReadOnceCleanInvalid` transaction, generates Snoops of the type `SnpUnique`.

The final `Comp` response that the Interconnect generates has a final cache state of I.

8.10 READONCEMAKE INVALID

8.10.1 RN VIP

Requester issues `ReadOnceMakeInvalid` when the initial cache line state is I.

The final cache state of the requester is I.

The requester expects to receive either a `Comp_UC` response or `Comp_I` in response to the request.

8.10.2 INTERCONNECT VIP

The Interconnect, upon receiving a `ReadOnceMakeInvalid` transaction, generates Snoops of the type `SnpUnique`.

The final Comp response that the Interconnect generates has a final cache state of I.

8.11 CLEANSHAREDPERSIST

8.11.1 RN VIP

Requester issues `CleanSharedPersist` when the initial cache line state is I, Sc or UC.

The final cache state of the requester will be I, UC or SC.

The requester expects to receive either a `Comp_UC`, `Comp_SC` or `Comp_I` response in response to the request.

8.11.2 Interconnect VIP

The Interconnect, upon receiving a `CleanSharedPersist` transaction, generates snoops of the type `SnpCleanShared`.

Both Snoopable and Non-Snoopable `SnpAttr` are permitted

8.12 DMT

8.12.1 RN VIP

DMT can be used for following transactions:

- ❖ `ReadShared`, `ReadClean`, `ReadUnique`, `ReadNotSharedDirty`, `ReadSpec`.
- ❖ `ReadNoSnp`, `ReadOnce`, `ROMI`, `ROCI` with a `CompAck`.
- ❖ `ReadNoSnp`, `ReadOnce`, `ROMI`, `ROCI` without a `CompAck` and with no ordering.

The initiating RN, after receiving the `CompData` response, checks for the `HomeNID` field in order to associate the `CompData` with the coherent transaction.

If the `SrcID` field is the same as the `HomeNID` field, in the `CompData` flit, it would mean that DMT was not used and the response has been generated by the HN and the field “`is_dmt_used`” is set to 0.

If `HomeNID` and `SrcID` fields are different and the `SrcId` is one of the Sn node Id's, it'd mean that DMT was used and the field “`is_dmt_used`” is set to 1.

Compack response from RN if applicable will have the `TgtID` and `TxnID` mapped to `HomeNID` and `DBID` of the `CompData` response respectively.

If `HomeNID` is different from the original `TgtId` of the request flit it means `target_id` remapping has happened and the related aspects needs to be taken care.

8.12.2 SN VIP

Upon reception of `ReadNoSnp` from HN the SN compares the `SrcID` with the `ReturnNID`, if both the values are same it indicates that DMT was not used (`is_dmt_used=0`) else DMT was used(`is_dmt_used=1`).

If the `order` field is set to 1 in the request SN sends `ReadReceipt` to the HN.

SN sends `CompData` with `TgtID` and `TxnID` mapped to `ReturnNID` and `ReturnTxnId` respectively,

Also the `HomeNID` and `DBID` will be mapped to `SrcId` and `TxnId` of the `ReadNoSnp` request it received.

8.12.3 Interconnect VIP

The Home node in response to a Read request sends a ReadNoSnp to the slave node with two additional ID fields (ReturnNID and ReturnTxnID). Setting of these two values determines if Home wants to use DMT for this particular Read request.

Interconnect VIP currently does not support DMT.

The value of the ReturnNID and ReturnTxnID is set to be the same as SrcID and TxnID respectively of the incoming read request from the RN.

8.13 DCT

8.13.1 RN VIP

Whenever an RN receives a Forward type Snoop request, and it has a copy of the line cached, it sends 2 responses: CompData to the initiating RN and SnpResp*_Fwded_* to the HN-F. In such cases, 'is_dct_used' in the snoop transaction handle is set to 1. In case the line is not cached at the snooped RN, it would simply send SnpResp_I to the HN-F and, so, 'is_dct_used' would be set to 0.

The snooped RN looks at the FwdNID and FwdTxnID fields to determine the TgtID and TxnID of the CompData flits. The response in the CompData will indicate the state in which the data is being returned (UC/UD/SC/SD). The fwdstate in SnpResp*_fwded response is set to the response field in the CompData.

The cache state transitions of the Snoopee and the cache state in the responses sent by the snoopee will be as per section 4.7.8 in the CHI-B specification. If more than one transition/response state are allowed, snooped RN generates a random transition/response so that all different possible scenarios prescribed by the spec are exercised.

The initiating RN, after receiving the CompData response, checks for the HomeNID field in order to associate the CompData with the coherent transaction.

If the HomeNID field is the same as the SrcID field, in the CompData flit, it would mean that DMT & DCT were not used and the response has been generated by the HN.

If HomeNID and SrcID fields are different, it'd mean that DCT or DMT was used. The initiating RN, then, looks at the SrcID field and checks if it corresponds to another RN or SN node in the system. If it corresponds to an RN, the is_dct_used flag in the coherent transaction handle is set.



Attention

Interconnect VIP currently does not support DCT.

8.14 RETTOSRC

8.14.1 RN VIP

If RetToSrc bit is set to 1 in a non-forwarding type Snoop request and the snooped RN has an initial cacheline state of SC, then it must always send SnpRespData response to the HN. If RetToSrc is set to zero, then snooped RN must only send SnpResp in such a scenario (as per the existing CHI-A implementation).

In case RetToSrc is set to 1 in a forward type snoop request, the snooped RN always sends SnpRespData*_fwded if the line is cached at the RN, else it sends SnpResp_I.

To know if data was sent in the Snoop response, users can refer to the read-only property "snp_rsp_datatransfer" that is present in the Snoop transaction object.

Interconnect VIP currently does not support RetToSrc.

8.15 DONOTGOTOSD

8.15.1 RN VIP

If DoNotGoToSD is set to 1 in a given Snoop request, then the snooped RN does not transition to Shared Dirty state. This means that the final state of the Snooped RN is not set to SD.

8.16 Atomic Transactions

8.16.1 RN VIP

VIP RN issues Atomic transaction requests from the following states: I, UC, UD, SC and SD.

RN does not currently support performing the Atomic operation locally and so always transmits the Atomic transaction to the Interconnect.

When the SnoopMe bit is de-asserted in the Atomic request:

- ❖ If the initial cacheline state is UC/SC, the RN sends out an Evict transaction to invalidate the cacheline before sending out the Atomic transaction to the Interconnect.
- ❖ If the initial cacheline state is UD/SD, the RN sends out a WriteBackFull transaction to write the dirty data to the memory and invalidate the copy, before sending out the Atomic transaction to the Interconnect.

When the SnoopMe bit is asserted in the Atomic request or when the initial cacheline state is I, the RN sends out the Atomic transaction to the interconnect before performing any further actions.

Once the DBIDResp is received from the Interconnect, the RN packs the relevant following transaction attributes into the NonCopyBackWriteData based on the atomic transaction type:

`atomic_store_load_txn_data` (for AtomicStore/Load transactions), `atomic_swap_data` (for AtomicSwap/Compare transactions) and `atomic_compare_data` (for AtomicCompare transactions).

The RN does not wait for the reception of Comp/CompData before sending out NonCopyBackWriteData.

In case of AtomicLoad, AtomicSwap and AtomicCompare transactions, once all the CompData flits are received, RN packs the valid byte lanes into the transaction attribute `atomic_returned_initial_data`.



Note VIP SN Agents currently do not support Atomic transactions.

8.16.2 Support for Atomic Transactions at SN-F VIP

The VIP supports atomic transactions in SN VIP. This is applicable for:

- ❖ Active SN VIP
- ❖ Passive SN VIP
- ❖ CHI system Monitor

8.16.2.1 Limitations:

- ❖ No functional coverage is supported for this feature.

- ❖ These features have not been verified for SN Atomic transactions:
 - ◆ Direct data transfer (DMT, DWT)
 - ◆ Poison and Datacheck

8.16.2.2 User Interface

To enable atomic transactions at SN VIP, the node configuration member `atomic_transactions_enable` must be set to 1.

8.16.2.3 Transaction Class Properties

No new transaction attributes are added. However, the following existing atomic specific transaction attributes are used:

- ❖ `svt_chi_common_transaction::atomic_returned_initial_data_status`
- ❖ `svt_chi_common_transaction::atomic_write_data_status`
- ❖ `svt_chi_transaction::atomic_write_data_resp_err_status`
- ❖ `svt_chi_transaction::atomic_read_data_resp_err_status`
- ❖ `svt_chi_transaction::atomic_write_dat_rsvdc`
- ❖ `svt_chi_transaction::atomic_read_dat_rsvdc`
- ❖ `svt_chi_transaction::num_atomic_write_dat_flits`
- ❖ `svt_chi_transaction::num_atomic_returned_initial_dat_flits`
- ❖ `svt_chi_transaction::atomic_transaction_type`
- ❖ `svt_chi_transaction::atomic_comp_resp_err`
- ❖ `svt_chi_transaction::atomic_dbid_resp_err`
- ❖ `svt_chi_transaction::atomic_read_data_trace_tag`
- ❖ `svt_chi_transaction::atomic_write_data_trace_tag`
- ❖ `svt_chi_transaction::atomic_store_load_txn_data`
- ❖ `svt_chi_transaction::atomic_write_data`
- ❖ `svt_chi_transaction::atomic_read_data`
- ❖ `svt_chi_transaction::atomic_returned_initial_data`
- ❖ `svt_chi_transaction::atomic_swap_data`
- ❖ `svt_chi_transaction::atomic_compare_data`
- ❖ `svt_chi_transaction::atomic_store_load_txn_tag`
- ❖ `svt_chi_transaction::atomic_swap_tag`
- ❖ `svt_chi_transaction::atomic_compare_tag`
- ❖ `svt_chi_transaction::atomic_returned_initial_tag`
- ❖ `svt_chi_transaction::atomic_write_tag`
- ❖ `svt_chi_transaction::atomic_read_tag`
- ❖ `svt_chi_transaction::atomic_write_data_tag_op`
- ❖ `svt_chi_transaction::atomic_read_data_tag_op`

For more information, see the CHI HTML class reference of the above-mentioned properties.

8.16.2.4 VIP Components

Active SN Agent:

Active SN agent is updated to support the reception and processing of atomic transactions and subsequently sending the required responses as specified by the SN response sequence.

For Atomic Load/Swap/Compare, the transaction attribute `atomic_compdata_order_policy` controls the flow order of `DBIDRESP` and `COMPDATA`. These types are supported:

- ❖ `DBIDRESP_WITH_COMPDATA`
- ❖ `DBIDRESP_BEFORE_COMPDATA`
- ❖ `DBIDRESP_AFTER_COMPDATA`

For Atomic Store, the transaction attribute `xact_rsp_msg_type` controls the flow order of `Comp` and `DBIDResp`. These types are supported:

- ❖ `RSP_MSG_COMP`
- ❖ `RSP_MSG_DBIDRESP`
- ❖ `RSP_MSG_COMPDDBIDRESP`

The inbuilt SN response sequence `svt_chi_sn_transaction_memory_sequence` is updated to support atomic transaction types.

The sequence invokes the following APIs added in `svt_chi_memory` to get the initial data from the memory and compute/write the resultant data into the memory:

- ❖ `svt_chi_memory::put_atomic_resultant_data_to_mem`: Computes the resultant data based on atomic write data (`svt_chi_transaction::atomic_write_data`) and atomic read data (`svt_chi_transaction::atomic_returned_initial_data`) once the transaction is complete, and writes the resultant data to memory.
- ❖ `svt_chi_memory::get_atomic_read_data_from_mem_to_transaction`: Gets the data from memory into the transaction atomic read data fields (`svt_chi_transaction::atomic_read_data` and `svt_chi_transaction::atomic_returned_initial_data`). In case of AtomicStore, even though `atomic_returned_initial_data` is not applicable (as there is no Read data sent on the interface), this method still populates the member so it can be used for computing the resultant data in `put_atomic_resultant_data_to_mem`.

Passive SN Agent:

Passive SN agent is updated to receive and process atomic transactions. Also, the passive SN is updated to update the resultant atomic data to the SN memory.

8.16.2.5 System Monitor

System monitor is updated to process slave atomics transaction and perform master slave association.

8.16.2.6 Protocol Checks

- ❖ `expected_xact_type_check` is updated to support atomics at SN VIP.
- ❖ `req_flit_atomic*_check` is updated for `return_txn_id` and `return_nid` fields related check.

8.17 Checks for the New Read Transactions

8.17.1 READNOTSHAREDDIRTY

- ❖ `is_valid` checks: These checks are applicable only for active modes.
 - ◆ Node support check: Supported node type `RN_F`
 - ◆ `ExpCompAck` must be asserted in the Request.
- ❖ System monitor checks:
 - ◆ Valid snoop association: The Interconnect, upon receiving a `ReadNotSharedDirty` transaction, generates Snoops of the type `SnpNotSharedDirty` to CHI Issue B compliant RNs and `SnpShared` to legacy RNs, If any.
- ❖ Node protocol monitor checks
 - ◆ Valid final cache state check:

Check name: `readnotshareddirty_valid_final_cache_state_check`

The Final cache state for `READNOTSHAREDDIRTY` transaction must be valid (SC/UC/UD). This check is valid only when the compile time macro ``SVT_CHI_ISSUE_B_ENABLE` is defined and `svt_chi_node_configuration::chi_spec_revision` is set to `svt_chi_node_configuration::ISSUE_B`.
 - ◆ Valid `RespErr` value check:

Check name: `readnotshareddirty_associated_compdata_packets_legal_resperr_check`

The Exclusive Okay response must only be given for a transaction that has the `Excl` attribute set.
 - ◆ Valid cache state in `compdata` field check:

Check name: `readspec_associated_compdata_packets_legal_cache_state_check`

The cache state in the associated `compdata` packets of a `ReadNotSharedDirty` transaction should indicate a valid value when the `RespErr` field does not indicate any errors

8.17.2 READONCECLEANINVALID

- ❖ `Is_valid` checks:
 - ◆ `Data_size` check: Data size should be 64 bytes
 - ◆ `Likelyshared` bit check: `Likelyshared` must not be asserted in the request
 - ◆ `Cachebale`: `cacheable` bit should be set to 1
 - ◆ `Device` bit: Should be set to 0.
 - ◆ `EWA` bit: Should be set to 1
 - ◆ `SnpAttr` bit: Should be set to 1
 - ◆ `Exclusive` bit check: `Exclusive` attribute must not be asserted in the Request
- ❖ System monitor checks:

Valid snoop association: The Interconnect, upon receiving a `ROCI` transaction must generate Snoops of the type `SnpUnique`
- ❖ Node protocol monitor checks
 - ◆ Valid final cache state check:

The Final cache state for ROCI transaction must be Invalid. This check is valid only when the compile time macro ``SVT_CHI_ISSUE_B_ENABLE` is defined along with `svt_chi_node_configuration::chi_spec_revision` is set to `ISSUE_B`.

◆ Valid Resperr check:

The associated compdata packet's `resp_err` field should not take EXOK value for ROCI transaction.

8.17.3 READONCEMAKEINVALID

❖ Is_valid checks:

- ◆ Data_size check: Data size should be 64 bytes
- ◆ Likelyshared bit check: Likelyshared must not be asserted in the request
- ◆ Allocatte bit: Should be set to 0
- ◆ Cachebale: cacheable bit should be set to 1
- ◆ Device bit: Should be set to 0.
- ◆ EWA bit: Should be set to 1
- ◆ SnpAttr bit: Should be set to 1
- ◆ Exclusive bit check: Exclusive attribute must not be asserted in the Request

❖ System monitor checks:

Valid snoop association: The Interconnect, upon receiving a ROMI transaction must generate Snoops of the type SnpUnique

❖ Node protocol monitor checks

◆ Valid final cache state check:

The Final cache state for ROMI transaction must be Invalid. This check is valid only when the compile time macro ``SVT_CHI_ISSUE_B_ENABLE` is defined along with `svt_chi_node_configuration::chi_spec_revision` is set to `ISSUE_B`.

◆ Valid Resperr check:

The associated compdata packet's `resp_err` field should not take EXOK value for ROMI transaction.

8.17.4 CLEANSHAREDPERSIST

❖ Is_valid checks:

- ◆ Data_size check: Data size should be 64 bytes
- ◆ Likelyshared bit check: Likelyshared must not be asserted in the request
- ◆ Device bit: Should be set to 0.
- ◆ EWA bit: Should be set to 1
- ◆ Exclusive bit check: Exclusive attribute must not be asserted in the Request

❖ System monitor checks:

Valid snoop association: The Interconnect, upon receiving a CleanSharedPersist transaction must generate Snoops of the type SnpCleanShared

❖ Node protocol monitor checks

◆ Valid final cache state check:

The Final cache state for CleanSharedPersist transaction must take a valid supported value. This check is valid only when the compile time macro ``SVT_CHI_ISSUE_B_ENABLE` is defined along with `svt_chi_node_configuration::chi_spec_revision` is set to `ISSUE_B`.

◆ Valid Resperr check:

The associated comp packet's `resp_err` field should not take EXOK value for CleanSharedPersist transaction.

8.18 Checks for Variable Width

8.18.1 Variable Address Width Support

❖ `is_valid` checks:

- ◆ ``SVT_CHI_MAX_ADDR_WIDTH` must be 44 if there are any CHI-A nodes in the system
- ◆ ``SVT_CHI_MAX_ADDR_WIDTH` must be between 44 and 52 if there only CHI-B nodes in the system
- ◆ `svt_chi_node_configuration::addr_width` must always be less than or equal to ``SVT_CHI_MAX_ADDR_WIDTH` and also greater than or equal to 44

8.18.2 Variable NODEID Width Support

❖ `is_valid` checks:

- ◆ ``SVT_CHI_NODE_ID_WIDTH` must be 7 if there are any CHI-A nodes in the system
- ◆ ``SVT_CHI_NODE_ID_WIDTH` must be between 7 and 11 if there only CHI-B nodes in the system
- ◆ `svt_chi_node_configuration::node_id_width` must always be less than or equal to ``SVT_CHI_NODE_ID_WIDTH` and also greater than or equal to 7

8.19 Checks for DMT

❖ RN Checks:

These checks are applicable only when the marco `SVT_CHI_ISSUE_B_ENABLE` is defined, `chi_spec_revision` is set to `ISSUE_B` and DMT is enabled for atleast one of the HN-Fs.

- ◆ `valid_transaction_supporting_dmt_check`: DMT is applicable only for the following transactions: Readclean, Readshared, Readunique, RNSD, Readnosnp, Readonce, ROCI and ROMI
- ◆ `valid_ordering_and_compack_combination_for_dmt_check`: DMT for Readnosnp, Readonce, ROCI and ROMI can be used only for the valid combinations of order and compack
- ◆ `valid_exclusive_access_for_dmt_check`: DMT cannot be used when exclusive access is set
- ◆ `dmt_used_by_hn_with_dmt_enabled_check`: DMT can be used by an HN only when DMT is enabled for that HN

❖ SN Checks:

These checks are applicable only when the marco `SVT_CHI_ISSUE_B_ENABLE` is defined, `chi_spec_revision` is set to `ISSUE_B` and DMT is enabled for atleast one of the HN-Fs.

- ◆ `valid_order_type_for_non_dmt_check`: If DMT is used order field can take value of `REQ_ACCEPTED` is case of optimized DMT, If DMT is not used and order field is set to `REQ_ACCEPTED` for readnosnp from HN to SN
- ◆ `valid_return_txn_id_check`: If DMT is used the `RetrunNID` is the `NodeID` of the Home, If DMT is not used the `RetrunNID` is the `NodeID` of the Requester
- ◆ `dmt_used_by_hn_with_dmt_enabled_check`: DMT can be used by an HN only when DMT is enabled for that HN
- ❖ System monitor checks:

These checks are applicable only when the macro `SVT_CHI_ISSUE_B_ENABLE` is defined and DMT is enabled for at least one of the HN-Fs.

 - ◆ `valid_xacts_for_optimized_dmt_check`: Checks optimized DMT is issued only for `READNOSNOOP` and `READONCE` transactions with `ExpCompAck` asserted or with unordered `READNOSNOOP` and `READONCE` transactions
 - ◆ `no_snoop_resp_dirty_with_dmt_check`: Checks that DMT is not used if any of the snoop transactions returns dirty data
 - ◆ `dmt_after_snoop_xacts_completion_check`: If snoop request needs to be send the Home must wait to send DMT request till all the required snoop responses are received

8.20 Checks for DCT

- ❖ Node Protocol Monitor checks:

These checks are applicable only when the macro `SVT_CHI_ISSUE_B_ENABLE` is defined and `svt_chi_node_configuration::chi_spec_revision` is set to `ISSUE_B`.

 - ◆ `rsp_flit_snprespfwded_check`: Checks that the fields of `SNPRESFWDDED` flits are set as per the legal values specified in the CHI-B spec.
 - ◆ `rsp_flit_snprespdatafwded_check`: Checks that the fields of `SNPRESPDATAFWDED` flits are set based on the legal values specified in the CHI-B spec.
 - ◆ `snoop_flit_snpreq_check`: This check has been updated to perform field value checks for all Forward type Snoop requests
 - ◆ `expected_snoop_xact_type_check`: This check ensures that a forward type Snoop is received only from an HN-F for which DCT has been enabled through `svt_chi_system_configuration::set_hn_dct_enable()`.
 - ◆ `associate_read_dat_flit_with_xact_check`: This check has been updated to handle `CompData` received from peer RN-Fs.
- ❖ System Monitor Checks:

These checks are applicable only when the macro `SVT_CHI_ISSUE_B_ENABLE` is defined

 - ◆ `only_one_forward_snoop_per_coherent_transaction_check`: Checks that, when DCT is used, only one Forward type Snoop is issued by the Interconnect, for a given coherent transaction

8.21 Checks for RETTOSRC

- ❖ Node Protocol Monitor checks:

RetToSrc related checks are applicable only when the macro `SVT_CHI_ISSUE_B_ENABLE` is defined and `svt_chi_node_configuration::chi_spec_revision` is set to `ISSUE_B`.

- ◆ `snoop_flit_snpreq_check`: Checks if RetToSrc is set to one of the legal values listed in the CHI-B spec, for a given Snoop request type.
- ◆ `snoop_flit_snpdvmop_check`: Checks if RetToSrc is set to zero for a SnpDVMOp request.
- ❖ System monitor checks:
These checks are applicable only when the macro `SVT_CHI_ISSUE_B_ENABLE` is defined.
 - ◆ `only_one_snoop_with_rettosrc_check`: Checks that not more than one Snoop request issued by the Interconnect, for a given coherent transaction, has RetToSrc asserted.

8.22 Checks for DONOTGOTOSD

- ❖ Node Protocol Monitor checks:
DoNotGoToSD related checks are applicable only when the macro `SVT_CHI_ISSUE_B_ENABLE` is defined and `svt_chi_node_configuration::chi_spec_revision` is set to `ISSUE_B`.
 - ◆ `snoop_flit_snpreq_check`: Checks if DoNotGoToSD is set to one of the legal values listed in the CHI-B spec, for a given Snoop request type.
 - ◆ `snoop_flit_snpdvmop_check`: Checks if DoNotGoToSD is set to zero for a SnpDVMOp request.
 - ◆ `snponce_associated_response_data_packets_legal_cache_state_check`: Checks that final state in the Snoop response for a SnpOnce request is not set to SD if DoNotGoToSD is set to 1 in the request.
 - ◆ `snpclean_associated_response_data_packets_legal_cache_state_check`: Checks that final state in the Snoop response for a SnpClean request is not set to SD if DoNotGoToSD is set to 1 in the request.
 - ◆ `snpshared_associated_response_data_packets_legal_cache_state_check`: Checks that final state in the Snoop response for a SnpShared request is not set to SD if DoNotGoToSD is set to 1 in the request.
 - ◆ `snpnotshareddirty_associated_response_data_packets_legal_cache_state_check`: Checks that final state in the Snoop response for a SnpNotSharedDirty request is not set to SD if DoNotGoToSD is set to 1 in the request.

8.23 Checks for Atomic Transactions

- ❖ Node Protocol Monitor checks:
Atomic transactions related checks are applicable only when the macro `SVT_CHI_ISSUE_B_ENABLE` is defined and `svt_chi_node_configuration::chi_spec_revision` is set to `ISSUE_B`.
 - ◆ `atomic_associated_response_data_packets_legal_cache_state_check`: Checks that the associated response and data packets have valid cache state value for an Atomic transaction. The cache state in the associated response and data packets of an Atomic transaction should indicate a valid value when the `RespErr` field does not indicate any errors. This check is valid only when the compile time macro `SVT_CHI_ISSUE_B_ENABLE` is defined along with `svt_chi_node_configuration::chi_spec_revision` is set to `ISSUE_B`.
 - ◆ `atomic_associated_response_data_packets_legal_resperr_check`: Checks that the associated response and data packets have valid `resp_err` value for Atomic transaction

- ◆ `req_flit_atomicstore_check`: Checks that all the fields of an AtomicStore transaction request are set to valid values.
- ◆ `req_flit_atomicload_check`: Checks that all the fields of an AtomicLoad transaction request are set to valid values.
- ◆ `req_flit_atomiccompare_check`: Checks that all the fields of an AtomicCompare transaction request are set to valid values.
- ◆ `req_flit_atomicswap_check`: Checks that all the fields of an AtomicStore transaction request are set to valid values.
- ❖ System monitor checks:

These checks are applicable only when the macro `SVT_CHI_ISSUE_B_ENABLE` is defined.

 - ◆ `coherent_read_atomic_hazard_check`: Checks that hazard condition between a coherent read/ dataless request and an atomic request to the same cacheline are correctly handled by HN.
 - ◆ `coherent_atomic_atomic_hazard_check`: Checks that hazard condition between a coherent atomic request and another atomic request to the same cacheline are correctly handled by HN.
 - ◆ `coherent_atomic_copyback_hazard_check`: Checks that hazard condition between a coherent atomic request and a Copyback request to the same cacheline are correctly handled by HN.
 - ◆ `coherent_write_atomic_hazard_check`: Checks that hazard condition between a coherent atomic request and a Write request to the same cacheline are correctly handled by HN.
 - ◆ `comp_or_compdata_response_after_snoop_xacts_completion_for_atomic_transaction_check`: The Completer must wait for all snoop responses before sending the Comp response in case of Atomic store or CompData response in case of Atomic load, swap and compare transactions.
 - ◆ `atomic_returned_initial_data_integrity_check`: Checks that the returned initial data is consistent with the system monitor memory view when an Atomic LOAD/SWAP/COMPARE completes at an RN.

In case there are associated snoops with data transfer the system monitor compares the returned initial data with the corresponding bytes in the snoop data.

In case there are no associated snoops or none of the associated Snoop requests received a data response, the system monitor compares the returned initial data with the corresponding bytes in the memory.

8.24 Sequence Collection Updates

Sequences added to support the new Read transaction types.

8.24.1 System Virtual Sequences

System virtual sequences and RN transaction sequences for new transaction types

- ❖ `svt_chi_system_single_node_readspec_virtual_sequence`
 - ◆ This sequence initiates ReadSpec transaction from the RN-F node specified with `node_index`, which can be a random node or a specific node configured by the user. Before sending each ReadSpec transaction, cachelines of peer node are initialized to random, valid states.
 - ◆ Parameters used to control the sequence are:
 - ◇ `Node_index`
 - ◇ `Sequence_length`
- ❖ `svt_chi_system_single_node_readnotshareddirty_virtual_sequence`

- ◆ This sequence initiates `ReadNotSharedDirty` transaction from the RN-F node specified with `node_index`, which can be a random node or a specific node configured by the user. Before sending each `ReadNotSharedDirty` transaction, cachelines of peer node are initialized to random, valid states.
- ◆ Parameters used to control the sequence are:
 - ◇ `Node_index`
 - ◇ `Sequence_length`
- ❖ `svt_chi_system_single_node_readoncecleaninvalid_virtual_sequence`
 - ◆ This sequence initiates `ReadOnceCleanInvalid` transaction from the RN-F node specified with `node_index`, which can be a random node or a specific node configured by the user. Before sending each `ReadOnceCleanInvalid` transaction, cachelines of peer node are initialized to random, valid states.
 - ◆ Parameters used to control the sequence are:
 - ◇ `Node_index`
 - ◇ `Sequence_length`
- ❖ `svt_chi_system_single_node_readoncemakeinvalid_virtual_sequence`
 - ◆ This sequence initiates `ReadOnceMakeInvalid` transaction from the RN-F node specified with `node_index`, which can be a random node or a specific node configured by the user. Before sending each `ReadOnceMakeInvalid` transaction, cachelines of peer node are initialized to random, valid states.
 - ◆ Parameters used to control the sequence are:
 - ◇ `Node_index`
 - ◇ `Sequence_length`
- ❖ `svt_chi_system_single_node_cleansharedpersist_virtual_sequence`
 - ◆ This sequence initiates `CleanSharedPersist` transaction from the RN-F node specified with `node_index`, which can be a random node or a specific node configured by the user. Before sending each `CleanSharedPersist` transaction, cachelines of peer node are initialized to random, valid states.
 - ◆ Parameters used to control the sequence are:
 - ◇ `Node_index`
 - ◇ `Sequence_length`

9

Using CHI C Verification IP

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

This chapter discusses the following topics:

- ❖ [Enabling CHI-C Mode](#)
- ❖ [CHI Issue C Features](#)
- ❖ [Response After Receiving First Data Packet](#)
- ❖ [Error Handling](#)
- ❖ [Combined CompAck and WriteData](#)
- ❖ [Protocol Checks](#)

9.1 Enabling CHI-C Mode

There are backward incompatible changes in CHI Issue C spec compared to earlier versions:

- ❖ Opcode field width of DAT VC is increased from 3 to 4.
- ❖ There are new opcodes on REQ, RSP VC 's introduced, which correspond to reserved values in case of earlier versions of CHI Specifications.

9.1.1 User Interface

To use all the CHI Issue C features including earlier CHI spec versions to Issue C, it is required to define the compile time macro ``SVT_CHI_ISSUE_C_ENABLE`. Once this macro is defined, all the VC signal widths will be as per CHI Issue C spec.

Once the ``SVT_CHI_ISSUE_C_ENABLE` compile macro is defined, to configure the CHI VIP components to work with CHI Issue C features, it is required to program the following attribute for each of the agents:

```
svt_chi_node_configuration::chi_spec_revision = svt_chi_node_configuration::ISSUE_C
```

Once ``SVT_CHI_ISSUE_C_ENABLE` macro is defined, still it is possible to use only features of only Issue B OR only ISSUE A by setting above configuration attribute accordingly.

Recap from CHI Issue B VIP User guide: Note that, to use all the CHI Issue B features including earlier CHI spec versions to Issue B, users are required to define the compile time macro ``SVT_CHI_ISSUE_B_ENABLE`.

Once this macro is defined, all the VC signal widths will be as per CHI Issue B spec. To use only CHI Issue A features, users are not required to define any of the above macros.

9.1.2 Use Model Details

Once ``SVT_CHI_ISSUE_C_ENABLE` macro is defined, all the VIP interface signal widths will be as per CHI Issue C spec. In case any of the RTL components that are connected are not yet compliant to CHI Issue C spec, it is required to tie off the required interface signals appropriately in the test bench.

9.2 CHI Issue C Features

9.2.1 Separate READ Data and COMP Response

9.2.1.1 Overview

For all read transactions that can have CompData response, except:

- ❖ Atomic transactions
- ❖ Exclusive Reads
- ❖ Reads with `((order_type != NO_ORDERING_REQUIRED) && (exp_comp_ack == 0))`

This feature permits the following:

- ❖ In response to such read requests from RN: ICN generating `dat_vc_flit_opcode = DATASEPRES` together with ICN generating `rsp_vc_flit_opcode = RESPSEPDATA`
- ❖ Corresponding to such read requests from RN, ICN generating DMT flow read request with `req_vc_flit_opcode = READNOSNPSEP` to SN: SN generating `dat_vc_flit_opcode = DATASEPRES` to RN together with ICN generating `rsp_vc_flit_opcode = RESPSEPDATA` to RN

9.2.1.2 Configuration Attributes

```
svt_chi_system_configuration::set_hn_sep_rd_data_sep_rsp_enable
(bit sep_data_rsp_enable[])
```

Indicates whether a given HN supports separate data and separate response mechanism for the applicable reads.

When `set_hn_dmt_enable(bit dmt_enable[])` is also set to 1 for the same HN, it's expected that only in that case, data is expected from SN. Otherwise data is expected from HN only. In either case, RSP is expected from HN when `sep_data_rsp_enable` is 1 for the same HN.

Table 9-1 Transaction Attributes

Attributes
<p>bit</p> <p><code>svt_chi_transaction::is_respsepdata_datasepresp_flow_used = 1'b0</code></p> <p>This field indicates whether Seperate Data and Seperate Home Response flow is used or not</p> <ul style="list-style-type: none"> ❖ This should not be programmed by user. ❖ Applicable for both RN and SN. ❖ This attribute along with <code>is_dmt_used</code> indicates the following: <ul style="list-style-type: none"> ◆ <code>is_dmt_used = 0, is_respsepdata_datasepresp_flow_used = 0</code>: normal CompData flow without DMT ◆ <code>is_dmt_used = 1, is_respsepdata_datasepresp_flow_used = 0</code>: normal CompData flow with DMT ◆ <code>is_dmt_used = 0, is_respsepdata_datasepresp_flow_used = 1</code>: home sent data, home sent response ◆ <code>is_dmt_used = 1, is_respsepdata_datasepresp_flow_used = 1</code>: slave sent data, home sent response
<p>rand bit</p> <p><code>svt_chi_transaction::is_compack_after_respsepdata_and_all_datasepresp = 1'b1</code></p> <p>Applicable only for active RN configured with <code>svt_chi_node_configuration :: chi_spec_revision >= svt_chi_node_configuration :: ISSUE_C</code>.</p> <ul style="list-style-type: none"> ❖ Applicable for all Read transactions that has <code>exp_comp_ack=1</code>, except read transactions with Exclusives <ul style="list-style-type: none"> ◆ Applicable when Seperate Read Data and Seperate Home response are observed. ❖ In all other cases, this is not applicable. ❖ When set to 1 for a Read transaction with <code>exp_comp_ack=1</code>, CompAck is generated only after receiving RespSepData and all DataSepResp flits. ❖ When set to 0 for a Read transaction with <code>exp_comp_ack=1</code>, CompAck is generated after receiving RespSepData. In this case, CompAck is generated as per <code>is_compack_after_all_compdata</code>.

Table 9-1 Transaction Attributes

Attributes
<pre>rand bit svt_chi_transaction::is_compack_after_all_compdata = 1'b1</pre> <p>Applicable only for active RN configured with <code>svt_chi_node_configuration :: chi_spec_revision >= svt_chi_node_configuration :: ISSUE_C</code>.</p> <ul style="list-style-type: none"> ❖ Applicable for all Read transactions that has <code>exp_comp_ack=1</code> ❖ Default value is set to 1 ❖ When set to 0 CompAck can be sent after <ul style="list-style-type: none"> ◆ Receiving first CompData for unordered Reads. ◆ Receiving first DataSepResp for unordered Reads. ◆ Receiving RespSepData for ordered Reads

9.2.2 Response After Receiving First Data Packet

9.2.2.1 Overview

CHI system monitor supports the following scenarios, where a Home Node is permitted to:

- ❖ Send read data to Requester after receiving first read data packet from a Slave
- ❖ Send read data to Requester after receiving first snoop data packet from a snooped Requester

RN agent supports the following scenarios, related to Response after receiving first Data packet:

- ❖ Completion acknowledge response, CompAck, after receiving first read data packet, CompData or DataSepResp.
- ❖ Snoop response with data, SnpRespData, must wait for receiving of all read data packets, CompData.

Table 9-2 Transaction Attributes

Attributes
<pre>rand bit svt_chi_transaction::is_compack_after_respsepdata_and_all_datasepresp = 1'b1</pre> <p>Applicable only for active RN configured with <code>svt_chi_node_configuration :: chi_spec_revision >= svt_chi_node_configuration :: ISSUE_C</code>.</p> <ul style="list-style-type: none"> ❖ Applicable for all Read transactions that has <code>exp_comp_ack=1</code>, except read transactions with exclusives. ❖ Applicable when Seperate Read Data and Seperate Home response are observed. ❖ In all other cases, this is not applicable. ❖ When set to 1 for a Read transaction with <code>exp_comp_ack=1</code>, CompAck is generated only after receiving RespSepData and all DataSepResp flits. ❖ When set to 0 for a Read transaction with <code>exp_comp_ack=1</code>, CompAck is generated after receiving RespSepData. In this case, CompAck is generated as per <code>is_compack_after_all_compdata</code>.
<pre>rand bit svt_chi_transaction::is_compack_after_all_compdata = 1'b1</pre> <p>Applicable only for active RN configured with <code>svt_chi_node_configuration :: chi_spec_revision >= svt_chi_node_configuration :: ISSUE_C</code>.</p> <ul style="list-style-type: none"> ❖ Applicable for all Read transactions that has <code>exp_comp_ack=1</code> ❖ Default value is set to 1 ❖ When set to 0 CompAck can be sent after <ul style="list-style-type: none"> ◆ Rceiving first CompData for unordered Reads. ◆ Receiving first DataSepResp for unordered Reads. ◆ Receiving RespSepData for ordered Reads

9.2.2.2 Protocol Checks

For more information on the checks are added, see the HTML class reference:

- ❖ `svt_chi_protocol_err_check::valid_snp_response_check`

9.2.3 Error Handling

The `respErr` field requirements for DataSepResp, RespSepData and NCBWrDataCompAck flits are supported.

9.2.4 Combined CompAck and WriteData

CHI Issue C spec permits sending of combined WriteData and CompAck from the RN for a WriteUnique transaction, instead of sending them separately.

VIP RN supports sending NonCopyBackWrDataCompAck flits instead of NonCopyBackWriteData flits for WriteUniqueFull, and WriteUniquePtl transactions whose ExpCompAck is set to 1.

9.2.4.1 Transaction Attributes

Following is the transaction attribute that must be programmed to 1 if NCBWrDataCompAck flits are to be sent in place of NCBWrData:

Table 9-3 Transaction Attributes

Attribute
<pre>rand bit attribute svt_chi_transaction::is_ncbwrdatacompack_used_for_write_xact = 0</pre> <ul style="list-style-type: none"> ❖ This field is defined only when the compile time macro SVT_CHI_ISSUE_C_ENABLE is set. This field can be programmed to 1 only when <code>svt_chi_node_configuration::chi_spec_revision</code> is set to <code>ISSUE_C</code>. ❖ This field indicates that NCBWRDATACOMPACT shall be transmitted over DAT channel instead of NCBWRDATA and CompAck in response to the write type transactions when this flag is asserted. ❖ Applicable for WriteUnique transactions with ExpCompAck asserted. ❖ The Write Data and CompAck flits are transmitted only once Comp or CompDBID response is received. ❖ If DBIDResp is received first, this field will be overridden to zero and the NonCopyBackWriteData and CompAck flits will be transmitted separately. ❖ When set to 1, VIP active RN drives NCBWrDataCompAck once it receives the following responses: <ul style="list-style-type: none"> ◆ Comp followed by DBIDResp , OR, ◆ CompDBIDResp. ❖ When set to 0, VIP active RN always drives NCBWrData and CompAck separately. ❖ For all transaction types other than WriteUnique, or, for WriteUniques with ExpCompAck set to 0, this attribute is not applicable and must be set to 0.

9.2.4.2 Protocol Checks

Following checks have been added in the passive RN to check the validity of the received combined WriteData and CompAck response:

- ❖ `svt_chi_link_err_check::data_flit_ncbwrdatacompack_check`: Checks that the data VC message fields are set to valid values for a NCBWrDataCompAck flit.
- ❖ `svt_chi_protocol_err_check:: valid_ncbwrdatacompack_flit_for_xact_check`: Checks that NCBWrDataCompAck is used only in case of WriteUnique transactions with ExpCompAck asserted.
- ❖ `svt_chi_protocol_err_check:: ncbwrdatacompack_after_comp_and_dbid_check`: Checks that NCBWrDataCompAck for a WriteUnique transaction is sent only after the reception of both Comp and DBIDResp ,or, CompDBID response. In case separate Comp and DBID responses are seen, this check additionally checks that NCBWrDataCompAck flits are sent only in case Comp was received before the DBIDResp flit.

10

Using CHI D Verification IP

This chapter describes the support for CHI D protocol.

This chapter discusses the following topics:

- ❖ [Overview of CHI D](#)
- ❖ [Features Supported for CHI Issue D](#)
- ❖ [Unsupported Features](#)

10.1 Overview of CHI D

The support for CHI D protocol is based on the ARM IHI 0050D (ID080717) specification.

10.1.1 User Interface

To use all the CHI Issue D features including earlier CHI spec versions to Issue D, it is required to define the compile time macro ``SVT_CHI_ISSUE_D_ENABLE`. Once this macro is defined, all the VC signal widths will be as per CHI Issue D spec. Once the ``SVT_CHI_ISSUE_D_ENABLE` compile macro is defined, to configure the CHI VIP components to work with CHI Issue D features, it is required to program the following attribute for each of the VIP agents: `svt_chi_node_configuration::chi_spec_revision = svt_chi_node_configuration::ISSUE_D`. Once ``SVT_CHI_ISSUE_D_ENABLE` macro is defined, still it is possible to use only features of only Issue C OR Issue B OR only ISSUE A by setting above configuration attribute accordingly.



Note

For details on the transaction members, configuration members and protocol checks, refer to the CHI Class Reference at the following location:
`$DESIGNWARE_HOME/vip/svt/amba_svt/latest/doc/class_ref/chi_d_svt_uvm_class_reference/html/index.html`

10.2 Features Supported for CHI Issue D

The following sections describe the features supported by VIP in CHI D protocol.

10.2.1 C Busy Feature

The Completer Busy indication is a mechanism for the Completer of a transaction to indicate its current level of activity. This signaling provides additional information to a Requester on how aggressive it can be in generating speculative activity to improve performance.

- ❖ Cbusy is a 3-bit field applicable in appropriate RSP and DAT flits.
- ❖ CBusy field is applicable in all responses received by an RN or an HN with or without data, except the following responses for which cbusy must be zero.
 - ◆ Compack
 - ◆ CopyBackWrData
 - ◆ NonCopyBackWrData
 - ◆ NCBWrDataCompAck
 - ◆ WriteDataCancel
- ❖ Cbusy does not care for the responses RspLCrdReturn and DatLCrdReturn.

10.2.1.1 Transaction Attributes

This attribute is added in `svt_chi_flit` class to drive the cbusy value on the interface in appropriate response and data flits

- ❖ `svt_chi_flit::cbusy`

HTML Page:

[\\$DESIGNWARE_HOME/vip/svt/amba_svt/latest/doc/class_ref/chi_d_svt_uvm_class_reference/html/class_svt_chi_flit.html](#)

Following attribute is added in `svt_chi_transaction` class to configure or hold the cbusy response and data:

- ❖ `svt_chi_transaction::response_cbusy`
- ❖ `svt_chi_transaction::data_cbusy[]`

HTML Page:

[\\$DESIGNWARE_HOME/vip/svt/amba_svt/latest/doc/class_ref/chi_d_svt_uvm_class_reference/html/class_svt_chi_transaction.html](#)

Following attributes are added in `svt_chi_snoop_transaction` to configure or hold the cbusy response and data in snoop transactions

- ❖ `svt_chi_snoop_transaction::snp_response_cbusy`
- ❖ `svt_chi_snoop_transaction::snp_data_cbusy[]`
- ❖ `svt_chi_snoop_transaction::fwded_data_cbusy`

HTML Page:

[\\$DESIGNWARE_HOME/vip/svt/amba_svt/latest/doc/class_ref/chi_d_svt_uvm_class_reference/html/class_svt_chi_snoop_transaction.html](#)

10.2.1.2 Link Layer Checks

Flit level checks are implemented in `svt_chi_link_err_ckeck` for response and data flits for which `cbusy` field is not applicable and must be set to zero. For the following response and data flits, `cbusy` field is not applicable and must be zero.

This is the list of existing related checks which have been updated:

- ❖ `svt_chi_link_err_check::rsp_flit_compack_check`
- ❖ `svt_chi_link_err_check::data_flit_copybackwrdata_check`
- ❖ `svt_chi_link_err_check::data_flit_noncopybackwrdata_chec`
- ❖ `svt_chi_link_err_check::data_flit_ncbwrdatacompack_check`
- ❖ `svt_chi_link_err_check::data_flit_writedatacancel_check`

HTML Page:

`$DESIGNWARE_HOME/vip/svt/amba_svt/latest/doc/class_ref/chi_d_svt_uvm_class_reference/html/protocolChecks.html`

10.2.1.3 Protocol Layer Checks

None.

10.2.1.4 Covergroups

These are the basic covergroups that have been added:

- ❖ `svt_chi_node_protocol_monitor_issue_d_def_cov_callback::trans_chi_d_rn_cbusy_indication_on_rsp_channel`
- ❖ `svt_chi_node_protocol_monitor_issue_d_def_cov_callback::trans_chi_d_rn_cbusy_indication_on_dat_channel`
- ❖ `svt_chi_node_protocol_monitor_issue_d_def_cov_callback::trans_chi_d_rn_cbusy_indication`
- ❖ `svt_chi_node_protocol_monitor_issue_d_def_cov_callback::trans_chi_d_hn_cbusy_indication_from_rn_perspective`

HTML Page:

`$DESIGNWARE_HOME/vip/svt/amba_svt/latest/doc/class_ref/chi_d_svt_uvm_class_reference/html/level1_covergroups.html`

10.2.1.5 Limitations

- ❖ `Cbusy` feature is not validated for the stash type transactions and associated snoop types and a single read transaction with separate comp and data responses as CHI INTERCONNECT VIP does not support this flow.
- ❖ `Cbusy` feature is not supported for stash type snoops with `datapull` set to 1 as CHI VIP components do not support stash type snoops with `datapull` set to 1.
- ❖ RN, SN agent Trace files for `cbusy` feature are not supported.
- ❖ Performance metrics related `cbusy` feature are not supported.
- ❖ Functional coverage for this feature is not yet supported completely apart from the above mentioned covergroups.

10.2.2 Increased TXN ID Width

TxnID with is increased to 10 bits in CHI Issue D specification.

- ❖ TXNID width is increased to 10 bits from 8 bits.
- ❖ CHI components RN, SN and interconnect VIP now supports TXNID and DBID values 0 to 1023 when `SVT_CHI_ISSUE_D_ENABLE` macro defined and `svt_chi_node_configuration::chi_spec_revison` is set to `ISSUE_D`. This also means that there can be 1024 outstanding transactions at RN/interconnect.
- ❖ The re-definable macros related to outstanding transactions ``SVT_CHI_MAX_NUM_OUTSTANDING_XACT`, ``SVT_CHI_MAX_NUM_OUTSTANDING_SNOOP_XACT` can now be defined between 1 to 1024 when ``SVT_CHI_ISSUE_D_ENABLE` macro defined and `svt_chi_node_configuration::chi_spec_revison` is set to `ISSUE_D`.
- ❖ Constraints and `is_valid` checks are updated related to `svt_chi_node_configuration::num_outstanding_xact` and `svt_chi_node_configuration::num_outstanding_snoop_xact` such that `svt_chi_node_configuration::num_outstanding_xact` and `svt_chi_node_configuration::num_outstanding_snoop_xact` can be programmed between 1 and 1024 when ``SVT_CHI_ISSUE_D_ENABLE` macro defined and `svt_chi_node_configuration::chi_spec_revisionion >= ISSUE_D`.
- ❖ If the TxnID value in the received request is greater than 255 when ``SVT_CHI_ISSUE_D_ENABLE` macro defined and `svt_chi_node_configuration::chi_spec_revisionion <= ISSUE_C`, CHI active SN, Passive SN and passive RN will throw an error.

10.2.2.1 Covergroups

These are the basic covergroups that have been added:

- ❖ `svt_chi_node_protocol_monitor_issue_d_def_cov_callback::trans_chi_d_rn_req_txn_id`
- ❖ `svt_chi_node_protocol_monitor_issue_d_def_cov_callback::trans_chi_d_rn_snp_req_txn_id`
- ❖ `svt_chi_node_protocol_monitor_issue_d_def_cov_callback::trans_chi_d_sn_req_txn_id`

HTML Page:

`$DESIGNWARE_HOME/vip/svt/amba_svt/latest/doc/class_ref/chi_d_svt_uvm_class_reference/html/level1_covergroups.html`

10.2.2.2 Limitations

- ❖ Functional coverage for this feature is not yet supported completely apart from the covergroups mentioned in the covergroups section.

10.2.3 MPAM Feature

The Memory system Performance resource Partitioning And Monitoring (MPAM) feature is introduced to efficiently utilize memory resources among users and to monitor the utilization of those resources

An optional 11bit wide MPAM field is added to REQ and SNP channels to support this feature. The width of MPAM is either 0 or 11BITS.

To enable the MPAM feature, user should define the macro `+define+SVT_CHI_MPAM_WIDTH_ENABLE`.

When this macro is set mpam field in REQ/SNP flit will be packed/unpacked while sending/receiving. If the macro is not defined mpam width will be zero.

10.2.3.1 Node Configuration Attribute

The MPAM feature can be enabled and configured by following node configurations parameters:

- ❖ svt_chi_node_configuration::enable_mpam
- ❖ svt_chi_node_configuration::mpam_partid_width
- ❖ svt_chi_node_configuration::mpam_perfmongroup_width

HTML Page:

[\\$DESIGNWARE_HOME/vip/svt/amba_svt/latest/doc/class_ref/chi_d_svt_uvm_class_reference/html/class_svt_chi_node_configuration.html](#)

10.2.3.2 Transaction Attributes

The following transaction class members are added for this feature:

- ❖ svt_chi_common_transaction::mpam_partid
- ❖ svt_chi_common_transaction::mpam_perfmongroup
- ❖ svt_chi_common_transaction::mpam_ns

HTML Page:

[\\$DESIGNWARE_HOME/vip/svt/amba_svt/latest/doc/class_ref/chi_d_svt_uvm_class_reference/html/class_svt_chi_common_transaction.html](#)



Note

- As CHI D protocol does not support widths other than 9 and 1 for mpam_partid and mpam_perfmongroup fields respectively, svt_chi_node_configuration::mpam_partid_width and mpam_perfmongroup_width. Have fixed values and cannot be modified by the user.
- If in RN svt_chi_node_configuration::enable_mpam=1 and while in SN the svt_chi_node_configuration::enable_mpam=0, then the mpam field values which SN will see will be zero.
- For an RN, it is not permitted to support MPAM only for one of the REQ or SNP channels.

10.2.3.3 FLIT Level Checks

The following existing checks are updated as mentioned below:

1. svt_chi_link_err_check::req_flit_<xact_type>_check
 - a. When svt_chi_node_configuration::enable_mpam is set to 0, then MPAM fields mpam_partid, mpam_perfmongroup and mpam_ns must take value 0 in request message.
 - b. MPAM fields mpam_partid, mpam_perfmongroup and mpam_ns must take value 0 for the xact_type PCRDRETURN and DVMOp.
2. svt_chi_link_err_check::snoop_flit_snp_req_check
 - a. When svt_chi_node_configuration::enable_mpam is set to 0, then MPAM fields mpam_partid, mpam_perfmongroup and mpam_ns must take value 0 in snoop message.
 - b. When svt_chi_node_configuration::enable_mpam is set to 1, then MPAM fields mpam_partid and mpam_perfmongroup must take value 0 and mpam_ns must be same as SNP.NS in snoop message for all non-stash type snoops.
3. svt_chi_link_err_check::snoop_flit_snpdvmop_check
 - a. MPAM fields mpam_partid, mpam_perfmongroup and mpam_ns must take value 0 for the SnpDVMOp.

HTML Page:

\$DESIGNWARE_HOME/vip/svt/amba_svt/latest/doc/class_ref/chi_d_svt_uvm_class_reference/html/protocolChecks.html

10.2.3.4 Limitations

- ❖ Functional coverage for this feature is not yet supported

10.2.4 Ordered Write Observation Flow Enhancements

10.2.4.1 OWO WriteUnique CompAck Timing Relaxation

These updates have been done to incorporate relaxation on OWO WriteUnique CompAck Timing.

- ❖ CompAck response for an Ordered Write Observation WriteUnique transaction (svt_chi_transaction::order_type set to REQ_ORDERING_REQUIRED and svt_chi_transaction::exp_comp_ack set to 1) can be sent when both of the following conditions are true:
 - ◆ Comp responses are received for all earlier OWO WriteUnique transactions which are required to be ordered with respect to the current OWO WriteUnique transaction.
 - ◆ DBIDResp or CompDBIDResp or Comp response for the current OWO WriteUnique is received.
- ❖ For an OWO WriteUnique transaction, when both the above conditions are met and the Comp for the current transaction is not received then the Requester must not wait for receiving of the Comp before sending Compack.
- ❖ The above rules are applicable only when macro `SVT_CHI_ISSUE_D_ENABLE` is enabled and svt_chi_node_configuration::chi_spec_revision is set to ISSUE_D or later.

10.2.4.1.1 Agent Level Updates

RN in Active Mode:

When macro `SVT_CHI_ISSUE_D_ENABLE` is defined, svt_chi_node_configuration::chi_spec_revision is set to ISSUE_D or later and svt_chi_node_configuration::streaming_ordered_writeunique_enable is set to 1:

- ❖ CompAck for OWO WriteUnique transactions is sent out after the reception of any one of the responses Comp or DBIDResp or CompDBIDResp for Current OWO WriteUnique is received and all previous OWO WriteUnique transactions have received Comp.
- ❖ From the above rule, for OWO WriteUnique transactions Compack can be sent out after Comp/DBIDResp/CompDBIDResp is received. Hence, if the first received response Comp/DBIDResp/CompDBIDResp has a trace tag value of 1, then trace tag value in Compack is set to 1. Otherwise, svt_chi_system_configuration::loopback_trace_tag is set to 1, trace tag value in Compack is set to a value same as trace tag value in Comp/DBIDResp/CompDBIDResp, or a random value is set for trace tag in Compack.

RN in Passive Mode:

Passive RN will detect the early Compack for OWO WriteUnique flow only when the macro `SVT_CHI_ISSUE_D_ENABLE` is defined, svt_chi_node_configuration::chi_spec_revision is set to ISSUE_D or later and svt_chi_node_configuration::streaming_ordered_writeunique_enable is set to 1.

10.2.4.1.2 Interconnect VIP Updates

Interconnect VIP is updated to expect and associate CompAck flits for an OWO WriteUnique at any point, once DBIDResp, Comp or CompDBIDResp is sent out when macro `SVT_CHI_ISSUE_D_ENABLE` is defined and `svt_chi_interconnect_configuration::chi_spec_revision` is set to `ISSUE_D` or later.

Updates to related checks regarding OWO WriteUnique CompAck Timing Relaxation have been done.

10.2.4.1.3 Protocol Checks

These protocol checks have been added:

- ❖ `svt_chi_protocol_err_check::valid_compack_rsp_flit_check`
- ❖ `svt_chi_protocol_err_check::expected_compack_check`
- ❖ `svt_chi_protocol_err_check::trace_tag_validity_check`

These new protocol checks are added in Passive RN:

- ❖ `svt_chi_protocol_err_check::owo_writeunique_compack_timing_check`
- ❖ `svt_chi_protocol_err_check::ncbwrdatacompack_flit_timing_check`

HTML Page:

`$DESIGNWARE_HOME/vip/svt/amba_svt/latest/doc/class_ref/chi_d_svt_uvm_class_reference/html/protocolChecks.html`

10.2.4.2 OWO IN WRITENOSNP

- ❖ ExpCompAck can be asserted for WriteNoSnp* transactions only when Order type is set to `REQ_ORDERING_REQUIRED`.
- ❖ The Ordered Write Observation in WriteNoSnp* transactions can be obtained in the following manner:
 - ◆ `svt_chi_transaction::order_type` field must be set to `REQ_ORDERING_REQUIRED`
 - ◆ `svt_chi_transaction::exp_comp_ack` field must be set to 1
 - ◆ The Completer sends the DBIDRESP and Comp responses in any order or can send the combined response `COMPDBIDRESP` or the `RETRY` response.
 - ◆ The Requester will send the Write data once the DBIDRESP response is received.
 - ◆ The Requester will send the CompAck response at appropriate time according to the below described Compack rules.
 - ◆ The Requester can combine the NCBWrData and Compack responses and transmit as single response `NCBWrDataCompack`.
 - ◆ The Requester can cancel the write data and can send `WRITEDATACANCEL` and Compack independently. This is possible in case of WriteNoSnpPtl transactions with `mem_attr_mem_type` is set to Normal.
- ❖ CompAck response for an ordered WriteNoSnp* transaction can be sent when both the following conditions are true:
 - ◆ Comp responses are received for all earlier WriteNoSnp* transactions which are required to be ordered with respect to the current WriteNoSnp* transaction.
 - ◆ DBIDResp or CompDBIDResp or Comp response for the current OWO WriteNoSnp* is received.

For an OWO WriteNoSnp* transaction when both the above conditions are met and the Comp for the current transaction is not received then the Requester must not wait for receiving of the Comp before sending CompAck.

- ❖ The OWO in WriteNoSnp* transactions feature described above is applicable only when `SVT_CHI_ISSUE_D_ENABLE` macro is defined and `svt_chi_node_configuration::chi_spec_revision` is set to `ISSUE_D` or later and `svt_chi_node_configuration::streaming_ordered_writenosnp_enable` is set to 1.

10.2.4.2.1 Node Configuration Attributes

In `svt_chi_node_configuration` new configuration attributes are added for OWO WriteNoSnp transactions similar to existing OWO WriteUnique transaction.

Following are the node configurations parameters:

- ❖ `svt_chi_node_configuration::streaming_ordered_writeunique_enable`
- ❖ `svt_chi_node_configuration::streaming_ordered_writenosnp_enable`
- ❖ `svt_chi_node_configuration::optimized_streaming_ordered_writenosnp_enable`
- ❖ `svt_chi_node_configuration::num_req_order_streams`

HTML Page:

`$DESIGNWARE_HOME/vip/svt/amba_svt/latest/doc/class_ref/chi_d_svt_uvm_class_reference/html/class_svt_chi_node_configuration.html`



Note

`svt_chi_transaction::xact_flow_category_type` is updated to reflect the new OWO WriteNoSnp transaction flows.

WriteDataCancel is permitted only for WriteUniquePtl, WriteUniquePtlStash and WriteNoSnpPtl transactions (with `mem_attr_mem_type` set to Normal).

The `is_writedatacancel_used_for_write_xact` flag is constrained to zero in transaction data classes for transactions other than the ones mentioned above.

10.2.4.2.2 Interconnect VIP Updates

Interconnect VIP is updated to respond to the OWO WriteNoSnp transactions and also to associate the new possible flits like NCBWrDataCompack and Compack with the WriteNoSnp transactions when macro `SVT_CHI_ISSUE_D_ENABLE` is defined and `svt_chi_interconnect_configuration::chi_spec_revision` is set to `ISSUE_D` or later.

10.2.4.3 Flit Level Checks

- ❖ The following existing flit level checks are updated to be performed on the OWO WriteNoSnp requests:
 - ◆ `svt_chi_link_err_check::req_flit_writenosnpfull_check`
 - ◆ `svt_chi_link_err_check::req_flit_writenosnpptl_check`

10.2.4.4 Protocol Checks

- ❖ The following existing protocol checks are updated to be performed for OWO WriteNoSnp transactions:
 - ◆ `svt_chi_protocol_err_check::valid_compack_rsp_flit_check`
 - ◆ `svt_chi_protocol_err_check::expected_compack_check`

- ◆ svt_chi_protocol_err_check::trace_tag_validity_check
- ◆ svt_chi_protocol_err_check::valid_ncbwrdatacompact_flit_for_xact_check
- ◆ svt_chi_protocol_err_check::single_rn_optimized_streaming_order_check
- ❖ The following protocol checks are added in Passive RN
 - ◆ svt_chi_protocol_err_check::owo_writenosnp_compact_timing_check
 - ◆ svt_chi_protocol_err_check::ncbwrdatacompact_flit_timing_check

10.2.4.5 Coverage

- ❖ Only these covergroups have been added: -
svt_chi_node_protocol_monitor_issue_d_def_cov_callback::trans_chi_d_rn_owo_writeunique_flow_type
- ❖ svt_chi_node_protocol_monitor_issue_d_def_cov_callback::trans_chi_d_rn_owo_writenosnp_flow_type

HTML Page:

\$DESIGNWARE_HOME/vip/svt/amba_svt/latest/doc/class_ref/chi_d_svt_uvm_class_reference/html/level1_covergroups.html

10.2.4.6 Limitations

- ❖ Functional coverage for this feature is not yet supported completely.
- ❖ Propagation of CMO transactions to Slave nodes is currently not supported by the Interconnect VIP. Hence no updates done to Interconnect VIP

10.2.5 CHI Issue A, B and C: Clarifications and Errata

Updates for the ERRATA items requirements and related checks are done except for the items listed in the 'Unsupported Features' section.

#2 A CMO intended for a particular address must not be sent before all previous requests, targeting the same cacheline, that can allocate data in the requester cache (ReadShared, ReadClean, ReadUnique, ReadNotSharedDirty, CleanUnique, MakeUnique) are complete.

The above rule applicability based on the request type:

- ❖ ReadShared, ReadClean, ReadUnique, ReadNotSharedDirty, CleanUnique, MakeUnique: Applicable
- ❖ ReadNoSnp, ReadOnce* with mem_attr_is_cacheable bit set to 1: IMPLEMENTATION DEFINED, and the ARM CPU cores will not consider these transactions for issuing a CMO to same cache line as ReadNoSnp and ReadOnce* will not cache data in requester cache. Therefore, VIP will also be implemented in compliance to ARM CPU cores.
- ❖ For AtomicLoad, AtomicCompare and AtomicSwap with mem_attr_cacheable bit set to 1: IMPLEMENTATION DEFINED, and the ARM CPU cores will not consider these transactions for issuing a CMO to same cache line as ReadNoSnp and ReadOnce* will not cache data in requester cache, hence VIP will also be implemented in compliance to ARM CPU cores.
- ❖ All other transaction types: Not Applicable

10.2.5.1 Active RN Updates

- ❖ Relaxed the conditions for a new CMO transaction in adding transaction to active RN queue and getting request channel lock such that a new CMO transaction be added to the active queue if there are no outstanding transactions (ReadShared, ReadClean, ReadUnique, ReadNotSharedDirty, CleanUnique, MakueUnique) targeted to the same cacheline.

10.2.5.2 Interconnect VIP

Propagation of CMO transactions to Slave nodes is currently not supported by the Interconnect VIP. Hence no updates done to Interconnect VIP.

10.2.5.3 Passive RN Updates:

- ❖ `svt_chi_protocol_err_check::cmo_xact_before_completion_of_previous_xacts_to_same_cacheline_check` has been updated based on the above rule.

#3 UD_PD state is permitted on DataSepResp.

- ◆ Applicable for Read transactions ReadShared, ReadNotSharedDirty, ReadUnique with separate comp and data responses. Legal cache state checks corresponding to these transactions `read*_associated_compdata_packets_legal_cache_state_check` are aligned with errata.
- ◆ For a ReadShared transaction SD_PD in CompData is permitted but SD_PD in DataSepResp response is not supported. The legal cache state check `readshared_associated_compdata_packets_legal_cache_state_check` has been updated accordingly.

#4 Reasserting that ExpCompAck must be deasserted in non-OWO writes.

- ◆ This requirement is already supported.

#5 Reiterating that DataCheck is a link layer property, and when enabled in DAT channel is valid on all data bytes irrespective of validity of individual data bytes.

- ◆ Default value of `svt_chi_node_configuration::datacheck_computation_logic` updated to `COMPUTE_DATACHECK_ON_ENTIRE_DATA`

#7 When RespSepData includes NDE error, all corresponding DataSepResp packets must be marked with NDE error.

- ◆ New protocol check `matching_resp_err_nderr_in_respsepdata_datasepresp_check` added.

#8 Additional requirement for SYSCOREQ and SYSCOACK signals.

- ◆ AMBA SVT CHI VIP follows the above rule.

#9 Additional requirement for SACTIVE signals.

- ◆ AMBA SVT CHI VIP follows the above rule.

10.2.5.4 Unsupported Features

1. #1 Errata item, 'Interconnect interface of a link is permitted to loop back incoming RXSACTIVE onto outgoing TXSACTIVE'. This includes updates to CHI interconnect VIP and this feature is not yet supported.
2. #6 Errata item, DataPull feature is not supported by CHI VIP.
3. #10 Errata item is not yet supported.

4. Interconnect VIP: Propagation of CMO transactions to Slave nodes is currently not supported by the Interconnect VIP

10.2.6 Persistent CMO with Two-part Response

10.2.6.1 CleanSharedPersistSep transaction

The support for these features is added in following components:

- ❖ Active and Passive RN VIP
- ❖ CHI system monitor
- ❖ Interconnect Full slave VIP
- ❖ Interconnect VIP

This new transaction type opcode is added in to Req VC in when CHI-D feature is enabled.

10.2.6.2 Node Configuration Attribute

The following attribute has been added to svt_chi_node_configuration:

- ❖ svt_chi_node_configuration::cleansharedpersistsep_xact_enable

Note: Applicable for RN only

HTML Page:

[\\$DESIGNWARE_HOME/vip/svt/amba_svt/latest/doc/class_ref/chi_d_svt_uvm_class_reference/html/class_svt_chi_node_configuration.html](#)

10.2.6.3 Interconnect Configuration Attribute

- ❖ svt_chi_interconnect_configuraton:: cleansharedpersistsep_xact_enable

If this variable is set to 1, HN receives this transaction from RN and the ICN full slave component of the interconnect issues appropriate responses, if applicable. The default value of this configuration attribute will be controlled through user re-definable macro SVT_CHI_NODE_CFG_DEFAULT_CLEANSHAREDPERSISTSEP_XACT_ENABLE . If the configuration variable cleansharedpersistsep_xact_enable is set to 1 for RN and the configuration variable for the interconnect is set to 0, then HN will drop this transaction.

10.2.6.4 Transaction Class Attributes

Following Attributes are added to the transaction class (svt_chi_transaction) to indicate the responses received for CleanSharedPersistSep transactions.

- ❖ svt_chi_transaction::is_persist_received
- ❖ svt_chi_transaction::is_comppersist_received
- ❖ svt_chi_transaction::is_comp_received (existing parameter)

Following controlling attributes are added to the transaction class (svt_chi_transaction)

- ❖ svt_chi_transaction::req_to_persist_flit_delay
- ❖ svt_chi_transaction::req_to_comppersist_flit_delay
- ❖ svt_chi_transaction::req_to_comp_flit_delay (existing parameter)

HTML Page:

[\\$DESIGNWARE_HOME/vip/svt/amba_svt/latest/doc/class_ref/chi_d_svt_uvm_class_reference/html/class_svt_chi_transaction.html](#)

Following attributes have been added in the SN transaction class (svt_chi_sn_transaction) to allow users to specify the delay between the request and first response issued by the ICN full slave VIP for a CleanSharedPersistSep transaction.

- ❖ svt_chi_sn_transaction::req_to_persist_flit_delay
- ❖ svt_chi_sn_transaction::req_to_comppersist_flit_delay

Following new enum literals are added to the response type enum 'xact_type_rsp_msg' in the SN transaction class (svt_chi_sn_transaction) to allow users to control the responses sent by the ICN Full slave for a CleanSharedPersistSep transaction. These are:

- ❖ RSP_MSG_COMP_PERSIST - Configuring xact_sp_msg_type to this will enable the interconnect component to send COMP followed by PERSIST.
- ❖ RSP_MSG_PERSIST_COMP: Configuring xact_sp_msg_type to this will enable the interconnect component to send PERSIST followed by COMP.
- ❖ RSP_MSG_COMPPERSIST: Configuring xact_sp_msg_type to this will enable the interconnect component to send COMPPERSIST.

HTML Page:

\$DESIGNWARE_HOME/vip/svt/amba_svt/latest/doc/class_ref/chi_d_svt_uvm_class_reference/html/class_svt_chi_sn_transaction.html

10.2.6.5 Component Level Updates

10.2.6.5.1 Active RN Updates

- ❖ VIP RN will have the capability to receive and process either two responses (COMP and PERSIST, in any order) or a single response (COMP_PERSIST) as applicable for CleanSharedPersistSep transaction. RN will also be able to handle retry responses if received for a CleanSharedPersist transaction.
- ❖ After receiving the responses from the interconnect for this transaction, appropriate checks will be performed.
- ❖ The cache state will remain unchanged at the end of a CleanSharedPersistSep transaction.

10.2.6.5.2 Passive RN Updates

- ❖ Passive RN will be able to monitor this transaction. This includes the ability to receive CleanSharedPersistSep transaction requests and also associate the subsequent responses from the Interconnect or SN with the CleanSharedPersistSep request
- ❖ Passive RN also perform checks on the request flit as well as the responses seen for the transaction.

10.2.6.5.3 Interconnect VIP Updates

- ❖ Interconnect VIP is updated to receive CleanSharedPersistSep transaction.
- ❖ The snoop applicable for this transaction is SnpCleanShared.
- ❖ After receiving the CleanSharedPersistSep, interconnect issues snoops to RNs, and if any of the associated snoops results in dirty data, then interconnect will write this dirty data in memory before responding to the CleanSharedPersistSep request.

10.2.6.5.4 ICN Full Slave VIP Updates

- ❖ The ICN full slave will be able to receive and process CleanSharedPersistSep transaction. It will be able to issue COMPPERSIST, {COMP and PERSIST} responses for this transaction.

- ❖ Just as is the case for the existing transaction types that are supported by the VIP, the responses sent by the ICN full slave for a `CleanSharedPersist` transaction can be controlled by programming appropriate fields of the IC SN transaction handle in the IC SN transaction response sequence.
- ❖ New literals have been added in `xact_rsp_msg_type_enum` to support different responses that the interconnect full slave component can issue in response to a `CLEANSHAREDPERISTSEP` request.
- ❖ The ICN full slave component will also be able respond to the received request after a user specified delay. This delay is applicable when the first response sent for this transaction. These delay attributes are added in `svt_chi_transaction` class.
 - ◆ If `xact_rsp_msg_type` is programmed to `RSP_MSG_COMP_PERSIST` or `RSP_MSG_PERSIST_COMP`, then two responses `COMP` and `PERIST` will be transmitted, and if some delay is specified for any of these responses, for only the response which is sent first, the corresponding request to response flit delay parameter will take effect.
 - ◆ You can specify the delay between Request and `PERSIST` response by setting the variable `req_to_persist_flit_delay` in the IC SN transaction handle to an appropriate value.
 - ◆ The delay between Request and `COMP` response can be specified by setting the variable `req_to_comp_flit_delay` in the IC SN transaction handle to an appropriate value.
 - ◆ The delay between Request and `COMPPERSIST` can be specified by setting the variable `req_to_comppersist_flit_delay` in the IC SN transaction handle to an appropriate value.
 - ◆ All these delays should lie between the min and max values which are specified through macros ``SVT_CHI_< MIN\MAX >REQTO< COMP/COMPPERSIST/PERSIST >_delay`.

10.2.6.5.5 CHI System Monitor

- ❖ CHI system monitor is updated to perform protocol checks like `coherent_snoop_type_match_check`, to make sure that correct snoop was generated for this transaction.
- ❖ System monitor will associate `CleanSharedPersistSep` transaction with the `SnpcleanShared` snoop transaction only and performs all the checks relevant to coherent-snoop association.
- ❖ Since VIP SN does not support this transaction, the system monitor currently does not expect to see `CleanSharedPersistSep` transactions at the downstream SN nodes. Therefore, it does not perform any master slave association or relevant checks in case `CleanSharedPersistSep` is seen at an SN.

10.2.6.6 Link Layer Checks

The following link layer checks have been added and will be constructed only when `svt_chi_nod_configuration::chi_spec_revision >= ISSUE_D` and `svt_chi_node_configuration::cleansharedpersistsep_xact_enable` is set to 1

- ❖ `svt_chi_link_err_check::req_flit_cleansharedpersistsep_check`
- ❖ `svt_chi_link_err_check::rsp_flit_comppersist_check`
- ❖ `svt_chi_link_err_check::rsp_flit_persist_check`

10.2.6.7 Protocol Layer Checks

The following protocol layer checks have been added and will be constructed only when `svt_chi_nod_configuration::chi_spec_revision >= ISSUE_D` and `svt_chi_node_configuration::cleansharedpersistsep_xact_enable` is set to 1

- ❖ `svt_chi_prot_err_check::valid_persist_flit_type_for_xact_check`

- ❖ svt_chi_prot_err_check::valid_comppersist_flit_type_for_xact_check
- ❖ svt_chi_prot_err_check::valid_response_combinations_for_cleansharedpersistsep_check
- ❖ svt_chi_prot_err_check::cleansharedpersistsep_associated_response_legal_resperr_check
- ❖ svt_chi_prot_err_check::cleansharedpersistsep_associated_response_legal_cache_state_check
- ❖ svt_chi_prot_err_check::single_outstanding_req_per_txn_id_check
- ❖ svt_chi_prot_err_check::new_req_before_completion_of_previous_cmo_xacts_to_same_cacheline_check
- ❖ svt_chi_prot_err_check::expected_rsp_flit_for_xact_check
- ❖ svt_chi_prot_err_check::trace_tag_validity_check
- ❖ expected_xact_type_check

10.2.6.8 System Level Check

The following system level existing checks have been enhanced:

- ❖ svt_chi_system_err_check::coherent_snoop_type_match_check

HTML Page:

\$DESIGNWARE_HOME/vip/svt/amba_svt/latest/doc/class_ref/chi_d_svt_uvm_class_reference/html/protocolChecks.html

10.2.6.9 Limitations

- ❖ This feature is not supported in SN (active/passive).
- ❖ VIP RN does not support CleanSharedPersistSep transaction to be issued to HN-I nodes.
- ❖ Coverage is not yet supported for this feature

10.3 Unsupported Features

- ❖ Icache Invalidation
- ❖ Interface Parity
- ❖ Deep Persistent Cache Maintenance
- ❖ Dvm Early Comp

11

Using CHI E Verification IP

This chapter describes the support for CHI E protocol. The topics discussed in this chapter are:

- ❖ [Overview of CHI E](#)
- ❖ [Features Supported for CHI Issue E](#)
- ❖ [CHI E Functional Coverage Report](#)
- ❖ [CHI E Unsupported Features](#)

11.1 Overview of CHI E

The support for the CHI E protocol is based on ARM CHI Issue E specification ARM IHI 0050E.a.

11.1.1 User Interface

To use all the CHI Issue E features including earlier CHI spec versions to Issue E, it is required to define the compile time macro ``SVT_CHI_ISSUE_E_ENABLE`. Once this macro is defined, all the VC signal widths will be as per CHI Issue E specification. Once the ``SVT_CHI_ISSUE_E_ENABLE` compile macro is defined, to configure the CHI VIP components to work with CHI Issue E features, it is required to program the following attribute for each of the VIP agents: `svt_chi_node_configuration::chi_spec_revision = svt_chi_node_configuration::ISSUE_E`. Once ``SVT_CHI_ISSUE_E_ENABLE` macro is defined, still it is possible to use only features of only Issue D OR Issue C OR only ISSUE B by setting above configuration attribute accordingly.

11.2 Features Supported for CHI Issue E

The following sections describe the features supported by VIP in CHI E protocol. Each feature lists

1. Configuration Attribute
2. Transaction Class attributes
3. Protocol checks and
4. Limitations in the VIP for the corresponding feature, if they are applicable.

11.2.1 MakeReadUnique Transaction

'MakeReadUnique' is a new transaction type which is used by a Requester to change a cached copy of the line from Shared state to Unique state. 'MakeReadUnique' transaction also Supports exclusive access.

11.2.1.1 Configuration Attributes

MakeReadUnique transactions are enabled in VIP without any additional node or system configuration attributes provided, `SVT_CHI_ISSUE_E_ENABLE` macro is defined and `svt_chi_node_configuration::chi_spec_revision` is set to `ISSUE_E`.

Care must be taken to ensure that the Interconnect Snoop Filter settings are programmed correctly using the following system configuration API:

- ❖ `svt_chi_system_configuration::set_hn_snoop_filter_enable()`

HTML Page:

[\\$DESIGNWARE_HOME/vip/svt/amba_svt/latest/doc/class_ref/chi_e_svt_uvm_class_reference/html/configuration/class_svt_chi_system_configuration.html](#)

To enable exclusive feature for MakeReadUnique, exclusive access needs to be enabled using following node configuration:

- ❖ `svt_chi_node_configuration::exclusive_access_enable`

This custom configuration attribute is added to enable specific custom SNPQUERY related check at the RN VIP

- ❖ `svt_chi_node_configuration::enable_custom_snp_query_check`

HTML Page:

[\\$DESIGNWARE_HOME/vip/svt/amba_svt/latest/doc/class_ref/chi_e_svt_uvm_class_reference/html/configuration/class_svt_chi_node_configuration.html](#)

This interconnect configuration property is added to specify when SNPQUERY must be sent by the interconnect for a given exclusive MakeReadUnique transaction:

- ❖ `svt_chi_interconnect_configuration::snpquery_policy_for_excl_makereadunique`

HTML Page:

[\\$DESIGNWARE_HOME/vip/svt/amba_svt/latest/doc/class_ref/chi_e_svt_uvm_class_reference/html/configuration/class_svt_chi_interconnect_configuration.html](#)

11.2.1.2 Transaction Attributes

Following transaction type enum is added to VIP transaction class to support MakeReadUnique transaction

- ❖ `svt_chi_common_transaction::MAKEREADUNIQUE`

These transaction properties are added in `svt_chi_transaction` class for MakeReadUnique transactions:

- ❖ `svt_chi_transaction::makereadunique_read_data`
- ❖ `svt_chi_transaction::makereadunique_read_poison`
- ❖ `svt_chi_transaction::makereadunique_read_datacheck`
- ❖ `svt_chi_transaction::invalidating_type_snoop_received_while_xact_is_outstanding`
- ❖ `svt_chi_transaction::non_invalidating_type_snoop_received_while_xact_is_outstanding`
- ❖ `svt_chi_transaction::snoop_filter_precision_info`

HTML page:

[\\$DESIGNWARE_HOME/vip/svt/amba_svt/latest/doc/class_ref/chi_e_svt_uvm_class_reference/html/transaction/class_svt_chi_transaction.html](#)

This transaction property is added in the snoop transaction class, `svt_chi_snoop_transaction`

- ❖ `svt_chi_snoop_transaction::is_outstanding_makereadunique_to_same_cacheline`

HTML page:

[\\$DESIGNWARE_HOME/vip/svt/amba_svt/latest/doc/class_ref/chi_e_svt_uvm_class_reference/html/transaction/class_svt_chi_snoop_transaction.html](#)

11.2.1.3 Limitations

The following flows are not supported by the Interconnect VIP:

- Separate read data and response flow.
- DMT
- DCT

11.2.2 Writes with Optional Data

`WriteEvictorEvict` is a new copyback write transaction with optional data transfer. When a requester evicts a clean line from its cache it signals to the Home, its intention to send a copy of Clean data by sending `WriteEvictorEvict` instead of `Evict`. The Home in response to `WriteEvictorEvict` transaction is permitted to refuse receiving of the data by sending just a `Comp` response instead of `CompDBIDResp` response. Communicating node pairs for this transaction is RN-F to Interconnect (HN-F).

`WriteEvictorEvict` transaction is enabled in the VIP without any additional node or system configuration attributes provided ``SVT_CHI_ISSUE_E_ENABLE` macro is defined and `svt_chi_node_configuration::chi_spec_revison` is set to `ISSUE_E`.

The following configuration attribute is added under `svt_chi_interconnect_configuration` class to control the response to `WriteEvictorEvict` transaction when CHI Interconnect is used.

- ❖ `svt_chi_interconnect_configuration::writes_with_optional_data_xact_response_type`

HTML page:

[\\$DESIGNWARE_HOME/vip/svt/amba_svt/latest/doc/class_ref/chi_e_svt_uvm_class_reference/html/configuration/class_svt_chi_interconnect_configuration.html](#)



Note

`WriteEvictorEvict` transaction is applicable and generated only from Request node with interface type RN-F.

11.2.2.1 Transaction Attributes

Following new transaction type is added to VIP transaction class to support `WriteEvictorEvict` transaction

- ❖ `svt_chi_common_transaction::WRITEEVICTOREVICT`

HTML page:

[\\$DESIGNWARE_HOME/vip/svt/amba_svt/latest/doc/class_ref/chi_e_svt_uvm_class_reference/html/transaction/class_svt_chi_common_transaction.html](#)

11.2.3 Non-Forwarding of Data from SC state

This feature is a relaxation on when a snoop is required to forward or return data in response to a snoop. The relaxation is to permit an RN-F to not forward a line from SharedClean state while it continues to keep the line in SharedClean state.

These changes are introduced in CHI Issue E specifications:

1. For `SnpOnceFwd`, `SnpCleanFwd`, `SnpNotSharedDirtyFwd` and `SnpSharedFwd` snoops the Snoopee with cache line in SC state is permitted to respond with `SnpResp_SC`, that is can keep the line in SC state and not return data with the snoop response.
2. For both forwarding and non-forwarding snoops, a Snoopee with cache line in SC state is permitted to consider `RetToSrc` value in a Snoop as a hint when the `RetToSrc` bit is set. A Snoopee is permitted to respond with `SnpResp_SC` and not return data with snoop response when the `RetToSrc` bit is set to one.



Note Currently the first change mentioned is not supported by CHI SVT VIP. The second change is supported only for non-forward type snoops by the VIP.

11.2.3.1 Configuration Attributes

The following node configuration is added in VIP for this feature:

- ❖ `svt_chi_node_configuration::fwd_data_from_sc_state_when_rettosrc_set`

HTML page:

`$DESIGNWARE_HOME/vip/svt/amba_svt/latest/doc/class_ref/chi_e_svt_uvm_class_reference/html/configuration/class_svt_chi_node_configuration.html`

11.2.3.2 Limitations

Currently CHI SVT VIP does not support the following requirement:

For forward type snoops, the Snoopee with cache line in SC state is permitted to respond with `SnpResp_SC`, that is can keep the line in SC state and not return data with the snoop response.

11.2.4 Write Zero with No Data

This revision of the specification permits sending Write message without data bytes when the data value is zero which was not permitted in earlier specification revisions. This new variation of write is only applicable in Non-CopyBack write transactions. These are the new write transaction types, which are write message without write data flits:

- ❖ `WriteNoSnpZero`
- ❖ `WriteUniqueZero`

11.2.4.1 Configuration Attributes

These new transaction types are supported when ``SVT_CHI_ISSUE_E_ENABLE` macro is defined and `svt_chi_node_configuration::chi_spec_revision` is set to `ISSUE_E` or later. Therefore, additional system or node configurations are not required to enable them.

11.2.4.2 Transaction Attributes

The following new transaction type is added to VIP transaction class to support `WriteNoSnpZero` and `WriteUniqueZero` transactions:

- ❖ `svt_chi_common_transaction::WRITENOSNPZERO`
- ❖ `svt_chi_common_transaction::WRITEUNIQUEZERO`

HTML page:

[\\$DESIGNWARE_HOME/vip/svt/amba_svt/latest/doc/class_ref/chi_e_svt_uvm_class_reference/html/transaction/class_svt_chi_common_transaction.html](#)

11.2.5 SnpQuery Snoop Request

`SnpQuery` is a new snoop request that probes the state of a cache line at a Request Node. A home can send a `SnpQuery` snoop without any corresponding request from a requester. The snoop response must include precise state of the cache line at the targeted Snoopee. `SnpQuery` snoop must not change the state of the line at the Snoopee.

11.2.5.1 Configuration Attributes

No additional node or system configuration attributes are required to enable `SnpQuery` request type. It is enabled by default when ``SVT_CHI_ISSUE_E_ENABLE` macro is defined and `svt_chi_node_configuration::chi_spec_revison` is set to `ISSUE_E`.

11.2.5.2 Transaction Attributes

This new transaction type is added to VIP transaction class to support `SnpQuery` snoop request:

- ❖ `svt_chi_common_transaction::SNPQUERY`

HTML page:

[\\$DESIGNWARE_HOME/vip/svt/amba_svt/latest/doc/class_ref/chi_e_svt_uvm_class_reference/html/transaction/class_svt_chi_common_transaction.html](#)



Note Interconnect VIP supports generation of `SnpQuery` transaction only in case of `MakeReadUnique` transaction as described in Section 11.2.1.

11.2.6 Exclusive Read Transactions

A new read transaction type `ReadPreferUnique` and corresponding new non-forwarding snoop, `SnpPreferUnique` and forwarding snoop, `SnpPreferUniqueFwd` are added in CHI Issue E specification to improve execution efficiency of an exclusive sequence. Based on the specification, response of `OKAY` in the returned data response must not be taken as a failure of `ReadPreferUnique`. Failure of exclusive sequence is only determined from corresponding `MakeReadUnique (Excl)` or `CleanUnique (Excl)` transaction completion.

11.2.6.1 Configuration Attributes

These configuration parameters are added to enable tracking of an ongoing exclusive transaction at the snooped RN:

- ❖ `svt_chi_node_configuration::snppreferunique_interpretation_policy`
- ❖ `svt_chi_node_configuration::snppreferuniquefwd_interpretation_policy`

HTML page:

\$DESIGNWARE_HOME/vip/svt/amba_svt/latest/doc/class_ref/chi_e_svt_uvm_class_reference/html/configuration/class_svt_chi_node_configuration.html

11.2.6.2 Transaction Attributes

This new transaction type is added to VIP transaction class to support `ReadPreferUnique` transaction:

- ❖ `svt_chi_common_transaction::READPREFERUNIQUE`

These new snoop transaction types are added to the VIP transaction class to support the new snoop requests

- ❖ `svt_chi_common_transaction::SNPPREFERUNIQUE`
- ❖ `svt_chi_common_transaction::SNPPREFERUNIQUEFWD`

HTML page:

\$DESIGNWARE_HOME/vip/svt/amba_svt/latest/doc/class_ref/chi_e_svt_uvm_class_reference/html/transaction/class_svt_chi_common_transaction.html

A new property is defined in `svt_chi_snoop_transaction` class to detect an ongoing exclusive transaction at the snoopee.

- ❖ `svt_chi_snoop_transaction::is_ongoing_exclusive_detected`

HTML page:

\$DESIGNWARE_HOME/vip/svt/amba_svt/latest/doc/class_ref/chi_e_svt_uvm_class_reference/html/transaction/class_svt_chi_snoop_transaction.html

11.2.6.3 Limitations

The following flows are not be supported by the Interconnect VIP:

- a. DMT
- b. DCT

11.2.7 Combined Write and (P)CMO Transaction

Combined write and (P) CMO transactions are a group of new transactions introduced in CHI Issue E specification to avoid the need to serialize the write transaction followed by the CMO transaction to the same address. These transactions are useful when CMO reaches a point in the system where a write operation must be complete before CMO transaction can be initiated. The cache maintenance transactions include both Persistent and non-Persistent variants.

11.2.7.1 Configuration Attributes

Combined Write and (P)CMO transactions are enabled in VIP without any additional node or system configuration attributes provided, ``SVT_CHI_ISSUE_E_ENABLE` macro is defined and `svt_chi_node_configuration::chi_spec_revison` is set to `ISSUE_E`.

Following configuration attributes related to Streaming Ordered Write Observation flow support for Combined Non-Copyback Write and (P)CMO type transactions are added to

`svt_chi_node_configuration` class:

- ❖ `svt_chi_node_configuration::streaming_ordered_combined_writeunique_cmo_enable`
- ❖ `svt_chi_node_configuration::optimized_streaming_ordered_combined_writeunique_cmo_enable`

- ❖ `svt_chi_node_configuration::streaming_ordered_combined_writenosnp_cmo_enable`
- ❖ `svt_chi_node_configuration::optimized_streaming_ordered_combined_writenosnp_cmo_enable`

HTML page:

[\\$DESIGNWARE_HOME/vip/svt/amba_svt/latest/doc/class_ref/chi_e_svt_uvm_class_reference/html/configuration/class_svt_chi_node_configuration.html](#)

In `svt_chi_system_configuration`, the following configuration attribute is added to guard custom checks related to Combined Write and (P)CMO type transactions:

- ❖ `svt_chi_system_configuration::custom_combined_write_cmo_check_enable`

HTML page:

[\\$DESIGNWARE_HOME/vip/svt/amba_svt/latest/doc/class_ref/chi_e_svt_uvm_class_reference/html/configuration/class_svt_chi_system_configuration.html](#)

11.2.7.2 Transaction Attributes

The following attributes are added:

- ❖ `svt_chi_transaction::is_compcmo_received`
- ❖ `svt_chi_transaction::writecmo_compcmo_resp_err`

HTML page:

[\\$DESIGNWARE_HOME/vip/svt/amba_svt/latest/doc/class_ref/chi_e_svt_uvm_class_reference/html/transaction/class_svt_chi_transaction.html](#)

New transaction types which are added in VIP to support Combined Write and (P)CMO transaction in VIP are listed in the table. These are the Combined Write and (P)CMO transaction types which are supported by from RN-F agent, Interconnect VIP, ICN Full Slave and System Monitor.

- ❖ `WRITENOSNPFULL_CLEANSHARED`
- ❖ `WRITENOSNPFULL_CLEANINVALID`
- ❖ `WRITEUNIQUEFULL_CLEANSHARED`
- ❖ `WRITEBACKFULL_CLEANSHARED`
- ❖ `WRITEBACKFULL_CLEANINVALID`
- ❖ `WRITECLEANFULL_CLEANSHARED`
- ❖ `WRITENOSNPPTL_CLEANSHARED`
- ❖ `WRITENOSNPPTL_CLEANINVALID`
- ❖ `WRITEUNIQUEPTL_CLEANSHARED`

Table 11-1 Combined Write and (P)CMO transactions & corresponding transaction types in VIP transaction class

Opcode[6]	Opcode[5:0]	Write Type	CMO Type	<code>svt_chi_common_transaction::xact_type</code>
1	0x10	WriteNoSnpFullCMO	CleanShared	<code>WRITENOSNPFULLCMO_CLEANSHARED</code>
1	0x11	WriteNoSnpFullCMO	CleanInvalid	<code>WRITENOSNPFULLCMO_CLEANINVALID</code>

Opcode[6]	Opcode[5:0]	Write Type	CMO Type	svt_chi_common_transaction::xact_type
1	0x12	WriteNoSnPFullCMO	CleanSharedPersistSep	WRITENOSNPFULLCMO_CLEANSHAREDPERERSISTSEP
1	0x14	WriteNoSnPFullCMO	CleanShared	WRITEUNIQUEFULLCMO_CLEANSHARED
1	0x16	WriteNoSnPFullCMO	CleanSharedPersistSep	WRITEUNIQUEFULLCMO_CLEANSHAREDPERERSISTSEP
1	0x18	WriteCleanFullCMO	CleanShared	WRITEBACKFULLCMO_CLEANSHARED
1	0x19	WriteCleanFullCMO	CleanInvalid	WRITEBACKFULLCMO_CLEANINVALID
1	0x1A	WriteCleanFullCMO	CleanSharedPersistSep	WRITEBACKFULLCMO_CLEANSHAREDPERERSISTSEP
1	0x1C	WriteCleanFullCMO	CleanShared	WRITECLEANFULLCMO_CLEANSHARED
1	0x1E	WriteCleanFullCMO	CleanSharedPersistSep	WRITECLEANFULLCMO_CLEANSHAREDPERERSISTSEP
1	0x20	WriteNoSnPPtlCMO	CleanShared	WRITECLEANFULLCMO_CLEANSHARED
1	0x21	WriteNoSnPPtlCMO	CleanInvalid	WRITENOSNPPTLCMO_CLEANINVALID
1	0x22	WriteNoSnPPtlCMO	CleanSharedPersistSep	WRITENOSNPPTLCMO_CLEANSHAREDPERERSISTSEP
1	0x24	WriteUniquePtlCMO	CleanShared	WRITEUNIQUEPTLCMO_CLEANSHARED
1	0x26	WriteUniquePtlCMO	CleanSharedPersistSep	WRITEUNIQUEPTLCMO_CLEANSHAREDPERERSISTSEP



Note All the transaction types listed above are not supported by the VIP. Please refer 'Limitations' for details on unsupported transactions

HTML page:

[\\$DESIGNWARE_HOME/vip/svt/amba_svt/latest/doc/class_ref/chi_e_svt_uvm_class_reference/html/transaction/class_svt_chi_common_transaction.html](#)

These are the new response message types added in the VIP for ICN Slave VIP to configure the response for Combined Write and (P)CMO

- ❖ svt_chi_sn_transaction::RSP_MSG_COMP_DBIDRESP_COMPCMO
- ❖ svt_chi_sn_transaction::RSP_MSG_DBIDRESP_COMP_COMPCMO

- ❖ svt_chi_sn_transaction::RSP_MSG_DBIDRESP_COMPPCMO_COMP
- ❖ svt_chi_sn_transaction::RSP_MSG_COMP_COMPPCMO_DBIDRESP
- ❖ svt_chi_sn_transaction::RSP_MSG_COMPPCMO_COMP_DBIDRESP
- ❖ svt_chi_sn_transaction::RSP_MSG_COMPPCMO_DBIDRESP_COMP
- ❖ svt_chi_sn_transaction::RSP_MSG_COMPPCMO_COMPDBIDRESP

**Note**

Above response types are not applicable for SN VIP as it does not support CMOs. They are applicable to ICN Slave VIP only.

11.2.7.3 Limitations

RN-I, RN-D VIP agents do not support Combined Write and (P)CMO transaction types

The following CHI-E combined Write and (P)CMO transaction types are not supported from HN to SN VIP Agents:

- ❖ WRITENOSNPFULL_CLEANSHARED
- ❖ WRITENOSNPFULL_CLEANINVALID
- ❖ WRITENOSNPFULL_CLEANSHAREDPERSTISEP
- ❖ WRITENOSNPPTL_CLEANSHARED
- ❖ WRITENOSNPPTL_CLEANINVALID
- ❖ WRITENOSNPPTL_CLEANSHAREDPERSTISEP

11.2.7.4 Combined Write+(P)CMO Transactions at SN-F VIP

CHI SN-F VIP supports write+(p)cmo transactions. These transaction types are supported:

- ❖ WRITENOSNPFULL_CLEANSHARED
- ❖ WRITENOSNPFULL_CLEANINVALID
- ❖ WRITENOSNPFULL_CLEANSHAREDPERSTISEP
- ❖ WRITENOSNPPTL_CLEANSHARED
- ❖ WRITENOSNPPTL_CLEANINVALID
- ❖ WRITENOSNPPTL_CLEANSHAREDPERSTISEP

Support for the above transactions is applicable to the following VIP components:

- ❖ Active SN-F VIP
- ❖ Passive SN-F VIP
- ❖ CHI System Monitor

11.2.7.4.1 Limitations

- ❖ No functional coverage will be supported for this feature
- ❖ These features have not been validated for SN-F write +(p)cmo transactions:
 - ◆ Direct data transfer
 - ◆ Poison and Datacheck

- ❖ AMBA Multi-chip system monitor does not support this feature.

11.2.7.4.2 Compile Macro

You must define the compile time macro ``SVT_CHI_ISSUE_E_ENABLE` to enable this feature.

11.2.7.4.3 Node Configuration

To enable `write+cleansharedpersistsep` transactions at SN-F VIP, the node configuration member `svt_chi_node_configuration::cleansharedpersistsep_xact_enable` must be set to 1.

Additionally, `svt_chi_node_configuration::cspsep_from_rn_to_sn_enable` is added to allow you to program `svt_chi_transaction::return_nid` for `WRITENOSNPFULL_CLEANSHAREDPERSTISTSEP` and `WRITENOSNPPTL_CLEANSHAREDPERSTISTSEP` transactions from RN to SN-F. The default value is set to 0.

When this configuration is set to 0, the CHI RN VIP driver sets the `svt_chi_transaction::return_nid` field same as that of `svt_chi_transaction::src_id`. When this configuration is set to 1, the `svt_chi_transaction::return_nid` field would be randomized to a value other than the SN-F `node_id`.

11.2.7.4.4 Transaction Class Properties

There are no new transaction attributes added for this feature.

These flow combinations are supported and controlled through `svt_chi_sn_transaction::xact_rsp_msg_type_enum`:

WriteNoSnP*CMO:

- ❖ `RSP_MSG_{COMPDBIDRESP/COMPCMO}_{COMPCMO/COMPDBIDRESP}`
Covers all the 2 different possible sequences of sending `COMPDBIDRESP` and `COMPCMO` response flits
- ❖ `RSP_MSG_{COMP/DBIDRESP/DBIDRespOrd/COMPCMO}_{DBIDRESP/COMPCMO/COMP}_{COMPCMO/COMP/DBIDRESP/DBIDRespOrd}`
Covers all the 12 different possible sequences of sending `COMP`, `DBIDRESP` and `COMPCMO` response flits
- ❖ `RSP_MSG_RETRYACK`

WriteNoSnP*PCMO:

- ❖ Response flow combinations possible when `COMPDBIDResp` and separate `CompCMO`, `PERSIST` responses are used
- ❖ Response flow combinations possible when `COMPDBIDResp` and `CompPERSIST` responses are used
- ❖ Response flow combinations possible when separate `COMP`, `DBIDResp/DBIDRespOrd` and separate `CompCMO`, `PERSIST` responses are used
- ❖ Response flow combinations possible when separate `COMP`, `DBIDResp/DBIDRespOrd` and `CompPERSIST` responses are used
- ❖ `RSP_MSG_RETRYACK`

11.2.7.4.5 Active SN Agent

Active SN-F agent supports the reception and processing of `write+(p)cmo` transactions and subsequently sending the required responses as specified by the SN-F response sequence. The applicable protocol checks

are performed. The inbuilt SN-F response sequence `svt_chi_sn_transaction_memory_sequence` is updated to support these transaction types.

11.2.7.4.6 Passive SN Agent

Passive SN agent is updated to receive, and process write+(p)cmo transactions and perform the appropriate checks.

11.2.7.4.7 CHI System Monitor

CHI System monitor is updated to associate RN transaction with SN-F transactions and perform the data integrity and tag integrity checks.

```

SYSTEM TRANSACTION SUMMARY: SLAVE XACT ASSOCIATED POST COHERENT XACT COMPLETE
*****
*****
XACT SUMMARY for {SYS_ID(0) OBJ_NUM(0) NODE_ID(0) SRC(RN_I, IDX(0), ID(0)), TGT(SN_F,
IDX(0), ID(6)) TYPE(WRITENOSNPFULL_CLEANSHPERSISTSEP) TXN_ID('d379) QOS('d4)
ADDR(7f5d71d60e2) SIZE(SIZE_64BYTE) NS(1) RETRY_ALLOWED(1) DATACHECK('h0) COMPACK(0)
ALLOCATE(0) SNPATTR('b00) CACHEABLE(1) MEM_TYPE(NORMAL) EWA(1) PGROUP_ID('d241) DEEP(0)
REQUEST_TAGOP(TAG_TRANSFER) DO_DWT(0) DATA_TAGOP(TAG_TRANSFER) START_TIME(1250000)
END_TIME(2400000) REQ_ACCEPT_TIME(2400000)} IS_DMT_USED(0) IS_DCT_USED(0)
IS_DWT_USED(0):
-----
INIT_STATE:I|CURRENT_STATE:I|FINAL_STATE:I|RESP_FINAL_STATE:I
RESPONSE_RESP_ERR_STATUS:NORMAL_OKAY
COMPCMO_RESP_ERR_STATUS:NORMAL_OKAY
DATA_RESP_ERR_STATUS[]:
    [0]:DATA_ERROR
    [1]:DATA_ERROR
    [2]:NORMAL_OKAY
    [3]:DATA_ERROR
Sequence Path: svt_chi_rn_transaction_xact_type_sequence.req
Sequencer Name: uvm_test_top.env.active_env.rn[0].rn_xact_seqr
Is Auto Generated: 0
REQUEST_TAGOP: TAG_TRANSFER
DATA_TAGOP: TAG_TRANSFER
XACT DATA:
e480da26_8519e4f6_a29e260e_4683c014_cdfbf8ec_8da0d7d8_4b0558c5_628fd1b9_b96b9e54_035583c
c_654d3762_aefd53f7_d35bcafd_5eaac1c0_1f2665a9_08738115
XACT TAG      : a_b_1_1
-----

INITIAL CACHE LINE CONTENTS:
PORT| NID | INDEX |   ADDR   | STATUS | SECURE | AGE |   DATA
| TAG STATUS | TAG
-----
FINAL CACHE LINE CONTENTS:
PORT| NID | INDEX |   ADDR   | STATUS | SECURE | AGE |   DATA
| TAG STATUS | TAG
-----
SYSTEM MONITOR L3/MEMORY CONTENT AFTER TRANSACTION END:
e480da26_8519e4f6_a29e260e_4683c014_cdfbf8ec_8da0d7d8_4b0558c5_628fd1b9_b96b9e54_035583c
c_654d3762_aefd53f7_d35bcafd_5eaac1c0_1f2665a9_08738115
-----
MEMORY TAGS IN SYSTEM MONITOR L3/MEMORY AFTER TRANSACTION END:

```



```

-----
ASSOCIATED SLAVE TRANSACTION SUMMARY:
  SN TRANSACTION SUMMARY for{SYS_ID(0) OBJ_NUM(0) NODE_ID(6) SRC(RN_I, IDX(0), ID(0)),
  TGT(SN_F, IDX(0), ID(6)) TYPE(WRITENOSNPFULL_CLEANSHAREDPERERSISTSEP) TXN_ID('d379)
  QOS('d4) ADDR(7f5d71d60e2) SIZE(SIZE_64BYTE) NS(1) RETRY_ALLOWED(1) DATACHECK('h0)
  COMPACT(0) ALLOCATE(0) SNPATTR('b00) CACHEABLE(1) MEM_TYPE(NORMAL) EWA(1)
  PGROUP_ID('d241) DEEP(0) REQUEST_TAGOP(TAG_TRANSFER) DO_DWT(0) DATA_TAGOP(TAG_TRANSFER)
  START_TIME(1250000) END_TIME(2400000) REQ_ACCEPT_TIME(1250000)} :
node_idx| node_id |obj_num| xact_type | addr | id| data_size
|mem_type|start_byte|end_byte|start_time|end_time
0| 6| 0|WRITENOSNPFULL_CLEANSHAREDPERERSISTSEP|007f5d71d60e2|379|
SIZE_64BYTE| NORMAL| 0| 63| 1250000| 2400000
RESPONSE_RESP_ERR_STATUS:NORMAL_OKAY
DATA_RESP_ERR_STATUS[]:
[0]:DATA_ERROR
[1]:DATA_ERROR
[2]:NORMAL_OKAY
[3]:DATA_ERROR
REQUEST_TAGOP: TAG_TRANSFER
DATA_TAGOP: TAG_TRANSFER
DATA BYTE INDICES : ..62..60 ..58..56 ..54..52 ..50..48 ..46..44 ..42..40 ..38..36
..34..32 ..30..28 ..26..24 ..22..20 ..18..16 ..14..12 ..10.. 8 .. 6.. 4 .. 2.. 0
VALID WYSIWYG BE : 1 1 1 1_ 1 1 1 1_ 1 1 1 1_ 1 1 1 1_ 1 1 1 1_ 1 1 1 1_ 1 1 1 1_ 1 1
1 1_ 1 1 1 1_ 1 1 1 1_ 1 1 1 1_ 1 1 1 1_ 1 1 1 1_ 1 1 1 1_ 1 1 1 1
VALID WYSIWYG DATA:
e480da26_8519e4f6_a29e260e_4683c014_cdfbf8ec_8da0d7d8_4b0558c5_628fd1b9_b96b9e54_035583c
c_654d3762_aefd53f7_d35bcafd_5eaac1c0_1f2665a9_08738115
VALID WYSIWYG TAG : a_b_1_1

```

The System Monitor summary also displays the SN transaction.

```

SN TRANSACTION SUMMARY for{SYS_ID(0) OBJ_NUM(2) NODE_ID(6) SRC(RN_I, IDX(0), ID(0)),
TGT(SN_F, IDX(0), ID(6)) TYPE(WRITENOSNPFULL_CLEANSHAREDPERERSISTSEP) TXN_ID('d2071)
QOS('d14) ADDR(7e3efdc7726) SIZE(SIZE_64BYTE) NS(0) RETRY_ALLOWED(1) DATACHECK('h0)
COMPACT(0) ALLOCATE(0) SNPATTR('b00) CACHEABLE(1) MEM_TYPE(NORMAL) EWA(1)
PGROUP_ID('d89) DEEP(0) REQUEST_TAGOP(TAG_UPDATE) DO_DWT(0) DATA_TAGOP(TAG_UPDATE)
START_TIME(1950000) END_TIME(4250000) REQ_ACCEPT_TIME(3050000)} :
node_idx| node_id |obj_num| xact_type | addr | id| data_size
|mem_type|start_byte|end_byte|start_time|end_time
0| 6| 2|WRITENOSNPFULL_CLEANSHAREDPERERSISTSEP|007e3efdc7726|2071|
SIZE_64BYTE| NORMAL| 0| 63| 1950000| 4250000
RESPONSE_RESP_ERR_STATUS:NORMAL_OKAY
DATA_RESP_ERR_STATUS[]:
[0]:DATA_ERROR
[1]:DATA_ERROR
[2]:NORMAL_OKAY
[3]:DATA_ERROR
REQUEST_TAGOP: TAG_UPDATE
DATA_TAGOP: TAG_UPDATE
DATA BYTE INDICES : ..62..60 ..58..56 ..54..52 ..50..48 ..46..44 ..42..40 ..38..36
..34..32 ..30..28 ..26..24 ..22..20 ..18..16 ..14..12 ..10.. 8 .. 6.. 4 .. 2.. 0
VALID WYSIWYG BE : 1 1 1 1_ 1 1 1 1_ 1 1 1 1_ 1 1 1 1_ 1 1 1 1_ 1 1 1 1_ 1 1 1 1_ 1 1
1 1_ 1 1 1 1_ 1 1 1 1_ 1 1 1 1_ 1 1 1 1_ 1 1 1 1_ 1 1 1 1_ 1 1 1 1
VALID WYSIWYG DATA:
72c8ca26_db5b865f_475efee9_8648174d_29b64839_9b20dfc8_a19e278b_ad339239_f848b744_46f1a22
9_827177db_bfa6ee9b_1cafe97a_e6227950_5868456c_529cd3ea
VALID WYSIWYG TAG : d_1_2_8

```


VALID WYSIWYG TAG UPDATE (TU) : 1_1_1_1

11.2.7.4.8 Protocol Checks

The `svt_chi_protocol_err_check::expected_xact_type_check` is updated to support write+(p)cmo transactions at SN-F VIP. The existing checks applicable for write+(p)cmo transactions would be performed.

The following system monitor checks are updated for these transactions:

- ❖ `slave_data_integrity_check`
- ❖ `slave_tag_integrity_check`

11.2.7.4.9 Debug Features

Combined Write+(P)CMO Transactions at SN-F VIP are supported in Verdi Protocol Analyzer.

11.2.8 Extending TxnID Width

TxnID width is increased to 12 bits in CHI Issue E specification. Even though the TxnID width supported is 12 bits, the maximum number of outstanding transactions from a single RN is limited to 1024.

11.2.8.1 Configuration Attributes

CHI components RN, SN, and interconnect VIP now support `Txn_id`, `return_txn_id` and `DBID` values 0 to 4095. This feature is enabled when ``SVT_CHI_ISSUE_E_ENABLE` macro defined and `svt_chi_node_configuration::chi_spec_revison` is set to `ISSUE_E`.

The re-definable macros related to outstanding transactions ``SVT_CHI_MAX_NUM_OUTSTANDING_XACT`, ``SVT_CHI_MAX_NUM_OUTSTANDING_SNOOP_XACT` are restricted to 1024. Therefore, the maximum number of outstanding transactions from a single RN is limited to 1024 as in CHI-E specification.

HTML page:

`$DESIGNWARE_HOME/vip/svt/amba_svt/latest/doc/class_ref/chi_e_svt_uvm_class_reference/html/transaction/class_svt_chi_common_transaction.html`

11.2.9 Memory Tagging

Memory tagging is a mechanism which is used to check the correct usage of data held in memory. When a memory location is allocated for a particular use it can also be assigned a memory tag. This memory tag is held alongside that data in memory and is referred to as the Allocation Tag. When the memory location is later accessed, the Requester uses both the address of the location and the tag value that it believes is associated with the location. This tag is referred to as the 'Physical Address Tag' or 'Physical Tag'. For any access where tag checking is enabled, the Physical Tag is checked against the Allocation Tag. The access always progresses as normal and the result of the tag check determines whether an error condition is signaled.

11.2.9.1 Configuration Attributes

The following node configuration attribute is added:

- ❖ `svt_chi_node_configuration::mem_tagging_enable`
- ❖ `svt_chi_node_configuration::return_tags_if_available_when_read_req_tag_op_is_invalid`

HTML page:

[\\$DESIGNWARE_HOME/vip/svt/amba_svt/latest/doc/class_ref/chi_e_svt_uvm_class_reference/html/configuration/class_svt_chi_node_configuration.html](#)

The following interconnect configuration attribute is added:

- ❖ `svt_chi_interconnect_configuration::return_tags_if_available_when_read_req_tag_op_is_invalid`

HTML page:

[\\$DESIGNWARE_HOME/vip/svt/amba_svt/latest/doc/class_ref/chi_e_svt_uvm_class_reference/html/configuration/class_svt_chi_interconnect_configuration.html](#)

Following member and APIs are provided in `svt_chi_system_configuration` related to Memory Tagging support at the Home nodes:

- ❖ `svt_chi_system_configuration::update_tags_in_memory_when_tagop_in_write_is_transfer`
- ❖ `svt_chi_system_configuration::set_hn_mem_tagging_enable`
- ❖ `svt_chi_system_configuration::set::is_mem_tagging_enabled`

HTML page:

[\\$DESIGNWARE_HOME/vip/svt/amba_svt/latest/doc/class_ref/chi_e_svt_uvm_class_reference/html/configuration/class_svt_chi_system_configuration.html](#)

11.2.9.2 Transaction Attributes

These fields are added to `svt_chi_common_transaction`:

- ❖ `svt_chi_common_transaction::tag_group_id`

HTML page:

[\\$DESIGNWARE_HOME/vip/svt/amba_svt/latest/doc/class_ref/chi_e_svt_uvm_class_reference/html/transaction/class_svt_chi_common_transaction.html](#)

The following fields are added to `svt_chi_transaction` class:

- ❖ `svt_chi_transaction::req_tag_op`
- ❖ `svt_chi_transaction::data_tag_op`
- ❖ `svt_chi_transaction::atomic_write_data_tag_op`
- ❖ `svt_chi_transaction::atomic_read_data_tag_op`
- ❖ `svt_chi_transaction::rsp_tag_op`
- ❖ `svt_chi_transaction::tag`
- ❖ `svt_chi_transaction::tag_update`
- ❖ `svt_chi_transaction::atomic_store_load_txn_tag`
- ❖ `svt_chi_transaction::atomic_swap_tag`
- ❖ `svt_chi_transaction::atomic_compare_tag`
- ❖ `svt_chi_transaction::atomic_returned_initial_tag`
- ❖ `svt_chi_transaction::makereadunique_read_tag`
- ❖ `svt_chi_transaction::allocate_in_cache_data_for_tag_fetch_readunique`
- ❖ `svt_chi_transaction::tag_match_resp`
- ❖ `svt_chi_transaction::tag_match_resp_err_status`

- ❖ svt_chi_transaction::initial_tag_state
- ❖ svt_chi_transaction::current_tag_state
- ❖ svt_chi_transaction::final_tag_state
- ❖ svt_chi_transaction::is_tag_match_received

HTML page:

\$DESIGNWARE_HOME/vip/svt/amba_svt/latest/doc/class_ref/chi_e_svt_uvm_class_reference/html/transaction/class_svt_chi_transaction.html

The following fields are added in svt_chi_snoop_transaction:

- ❖ svt_chi_snoop_transaction::data_tag_op
- ❖ svt_chi_snoop_transaction::rsp_tag_op
- ❖ svt_chi_snoop_transaction::snr_rsp_is_tag_shared
- ❖ svt_chi_snoop_transaction::fwded_tag_op
- ❖ svt_chi_snoop_transaction::fwded_tag
- ❖ svt_chi_snoop_transaction::data_pull_tag_op
- ❖ svt_chi_snoop_transaction::data_pull_tag
- ❖ svt_chi_snoop_transaction::initial_tag_state
- ❖ svt_chi_snoop_transaction::final_tag_state

HTML page:

\$DESIGNWARE_HOME/vip/svt/amba_svt/latest/doc/class_ref/chi_e_svt_uvm_class_reference/html/transaction/class_svt_chi_snoop_transaction.html

11.2.9.3 Limitations

- ❖ Memory Tags and Tag related operations are not visible in the cache visualization view.
- ❖ Memory Tagging is only supported by the Interconnect VIP when L3 is disabled. Hence, this feature has not been validated with L3 enabled at the Interconnect.

11.2.10 DVM Updates

New TLBI and iCache Invalidation DVM Operations have been introduced in the CHI Issue E specification in order to support ARM v8.4 instruction set. Also, for TLBI operations that are invalidated by VA or IPA, there is a provision to specify a range of addresses.

11.2.10.1 Configuration Attributes

The following node configuration attribute is added in VIP:+9

- ❖ svt_chi_node_configuration::dvm_version_support

HTML Page:

\$DESIGNWARE_HOME/vip/svt/amba_svt/latest/doc/class_ref/chi_e_svt_uvm_class_reference/html/configuration/class_svt_chi_node_configuration.html

11.2.10.2 Transaction Attributes

The following fields are added to `svt_chi_common_transaction` class. Therefore, this is common to `svt_chi_transaction` as well as `svt_chi_snoop_transaction`

- ❖ `svt_chi_common_transaction::dvm_range`
- ❖ `svt_chi_common_transaction::dvm_num`
- ❖ `svt_chi_common_transaction::dvm_scale`
- ❖ `svt_chi_common_transaction::dvm_ttl`
- ❖ `svt_chi_common_transaction::dvm_tg`

HTML page:

`$DESIGNWARE_HOME/vip/svt/amba_svt/latest/doc/class_ref/chi_e_svt_uvm_class_reference/html/transaction/class_svt_chi_common_transaction.html`

11.2.10.3 Limitations

DVM domains and the two optional pins to the HN interface related to the Broadcasting of DVMs are not supported by the VIP.

11.2.11 Handling of Response with NDERR

CHI Issue E specification introduced these updates for handling of Non-data Error for various transaction types:

1. For an allocating transaction when the start state is I, the Requester must not allocate the received data. If the request was sent from a non-I state, then the Requester must leave the cached copy unchanged. In both cases the cache state must not be changed.
2. For a de-allocating transaction, the Requester must continue as normal. De-allocating transactions are `WriteBack`, `WriteEvictFull`, `Evict`, `WriteEvictOrEvict`.
3. For other transactions that do not change allocation, the Requester must not upgrade cache state but is permitted to downgrade.

11.2.11.1 Configuration Attributes

A new node configuration `nderr_resp_policy` is added in the VIP so that users can choose to enable this new feature for a given VIP RN agent.

- ❖ `svt_chi_node_configuration::nderr_resp_policy`

HTML page:

`$DESIGNWARE_HOME/vip/svt/amba_svt/latest/doc/class_ref/chi_e_svt_uvm_class_reference/html/configuration/class_svt_chi_node_configuration.html`

11.2.11.2 Limitations

The rules in the CHI-E specification with respect to the handling of NDERR response for Forwarding type snoops are not yet supported.

Passive RN does not yet support passive cache state tracking. Therefore, there are no checks corresponding to the rules in the CHI-E specification with respect to how an RN must handle Non-data error response received for a coherent transaction.

11.2.12 Two-part StashOnce Transaction

`StashOnceSepUnique` and `StashOnceSepShared` are two new transactions of `StashOnce` flavor introduced in CHI-E. The new transaction characteristics are the same as corresponding `StashOnceUnique` and `StashOnceShared` transactions except that they send two responses: `Comp` and `StashDone`, to the requester instead of single response.

11.2.12.1 Configuration Attributes and APIs

Following node configuration variable needs to be set in order to enable cache stashing and the new transaction types on respective VIP components

- ❖ `svt_chi_node_configuration::cache_stashing_enable`

HTML page:

[\\$DESIGNWARE_HOME/vip/svt/amba_svt/latest/doc/class_ref/chi_e_svt_uvm_class_reference/html/configuration/class_svt_chi_node_configuration.html](#)

Following system configuration APIs are applicable to cache stashing and the new transaction types. They need to be used with respective hn index number to configure cache stashing features on the respective HN VIP.

- ❖ `svt_chi_system_configuration::svt_chi_system_configuration::set_hn_stash_enable(bit stash_enable [])`
- ❖ `svt_chi_system_configuration::set_hn_stash_data_pull_enable(bit stash_data_pull_enable [])`
- ❖ `svt_chi_system_configuration::set_hn_ord_stash_data_pull_enable()`

HTML page:

[\\$DESIGNWARE_HOME/vip/svt/amba_svt/latest/doc/class_ref/chi_e_svt_uvm_class_reference/html/configuration/class_svt_chi_system_configuration.html](#)

11.2.12.2 Transaction Attributes

Following new transaction types are added to VIP transaction class for `StashOnceSepUnique` and `StashOnceSepShared` transactions. These new transaction types are applicable to `svt_chi_transaction::xact_type` attribute.

- ❖ `svt_chi_common_transaction::STASHONCESEPUNIQUE`
- ❖ `svt_chi_common_transaction::STASHONCESEPSHARED`

Following transaction class attributes are applicable for two-part stash transactions.

- ❖ `svt_chi_common_transaction::stash_lpid`
- ❖ `svt_chi_common_transaction::stash_lpid_valid`
- ❖ `svt_chi_common_transaction::stash_nid`
- ❖ `svt_chi_common_transaction::stash_nid_valid`
- ❖ `svt_chi_common_transaction::stash_group_id`

HTML page:

[\\$DESIGNWARE_HOME/vip/svt/amba_svt/latest/doc/class_ref/chi_e_svt_uvm_class_reference/html/transaction/class_svt_chi_common_transaction.html](#)

Following transaction class attribute of svt_chi_snoop_transaction class can be used to configure datapull in stash type snoops.

- ❖ svt_chi_snoop_transaction::data_pull

Following response types of the common transaction class (svt_chi_common_transaction) are applicable to flit opcode attribute, svt_chi_flit::opcode, of CHI response type flit.

- ❖ svt_chi_common_transaction::STASHDONE
- ❖ svt_chi_common_transaction::COMPSTASHDONE

HTML page:

[\\$DESIGNWARE_HOME/vip/svt/amba_svt/latest/doc/class_ref/chi_e_svt_uvm_class_reference/html/transaction/class_svt_chi_common_transaction.html](#)

11.2.12.3 Limitations

Interconnect VIP does not support Two-part StashOnce transactions.

Coherent to snoop association related aspects in the system monitor have not been tested as the StashOnceSep* feature has been validated only with the ICN Full slave VIP.

11.2.13 DBIDRespOrd Response

DBIDRespOrd Response is sent to signal to the Requester that resources are available to accept the WriteData response. It also indicates that the Completer provides certain transaction ordering guarantees.

11.2.13.1 Configuration Attributes

DBIDRespOrd is enabled by default when the CHI-E configurations mentioned under the section "11.1.1 User Interface" are enabled.

11.2.13.2 Transaction Attributes

Following attribute specifies if DBIDRespOrd response is received for a given transaction by RN

- ❖ svt_chi_transaction::is_dbidrespord_received:

HTML page:

[\\$DESIGNWARE_HOME/vip/svt/amba_svt/latest/doc/class_ref/chi_e_svt_uvm_class_reference/html/transaction/class_svt_chi_transaction.html](#)

This attribute is applicable to snoop transactions

- ❖ svt_chi_snoop_transaction::force_data_pull_to_zero

HTML page:

[\\$DESIGNWARE_HOME/vip/svt/amba_svt/latest/doc/class_ref/chi_e_svt_uvm_class_reference/html/transaction/class_svt_chi_snoop_transaction.html](#)

11.2.13.3 Limitations

- ❖ Coverage is not yet supported for this feature

11.2.14 Direct Write-data Transfer (DWT)

Direct Write-data Transfer (DWT) allows write data to be passed directly from the Requester to the Slave reducing the use of the DAT channel and removing the need for the Home Nodes to hold a copy of the write data.

DWT is permitted only in writes that are Non-CopyBack. DWT is also not permitted in Non-CopyBack writes that are OWO writes. DWT is applicable only in WriteNoSnPFull, WriteNoSnPtl and Write requests combined with CMO or PCMO from Home Node to Slave Node.

A DoDWT field is added to the Request channel to support DWT. The DoDWT field is only applicable in WriteNoSnPFull, WriteNoSnPtl and in Combined Write requests from Home to Slave.

11.2.14.1 Configuration API

You can enable DWT on each HN indices using following API:

- ❖ `svt_chi_system_configuration:: set_hn_dwt_enable(bit dwt_enable[])`

HTML page:

[\\$DESIGNWARE_HOME/vip/svt/amba_svt/latest/doc/class_ref/chi_e_svt_uvm_class_reference/html/configuration/class_svt_chi_system_configuration.html](#)

Following node configuration can enable DWT from RN. But this feature is disabled by default as there are dependency on other configurations related to number of RN, HN and SN. Refer the HTML page description on this variable before enabling DWT from RN.

- ❖ `svt_chi_node_configuration::allow_dwt_from_rn_when_hn_is_absent`

HTML page:

[\\$DESIGNWARE_HOME/vip/svt/amba_svt/latest/doc/class_ref/chi_e_svt_uvm_class_reference/html/configuration/class_svt_chi_node_configuration.html](#)

11.2.14.2 Transaction Attributes

This attribute specifies DoDWT field of applicable transactions. This attribute is applicable in WriteNoSnPFull, WriteNoSnPtl and Write requests combined with CMO or PCMO from HN to SN.

- ❖ `svt_chi_common_transaction:: do_dwt`

This transaction class variable is asserted by RN and SN VIPs to flag a DWT transaction.

- ❖ `svt_chi_transaction:: is_dwt_used`

Following are the existing transaction class attributes which are applicable to DWT

- ❖ `svt_chi_common_transaction:: return_nid`

- ❖ `svt_chi_common_transaction::return_txn_id`

HTML page:

`$DESIGNWARE_HOME/vip/svt/amba_svt/latest/doc/class_ref/chi_e_svt_uvm_class_reference/html/transaction/class_svt_chi_common_transaction.html`

11.2.14.3 Protocol Checks

Protocol Layer Checks:

Following are the new protocol layer checks introduced for DWT

- ❖ `valid_transaction_supporting_dwt_check`
- ❖ `dwt_used_by_hn_with_dwt_enabled_check`
- ❖ `valid_ordering_and_compack_combination_for_dwt_check`

System monitor checks:

Following are the new system monitor checks introduced for DWT

- ❖ `comp_after_snoop_xacts_completion_with_dwt_check`
- ❖ `no_snoop_resp_dirty_with_dwt_check`

HTML page:

`$DESIGNWARE_HOME/vip/svt/amba_svt/latest/doc/class_ref/chi_e_svt_uvm_class_reference/html/protocolChecks.html`

11.2.14.4 Limitations

Functional Coverage is not supported for all the features.

11.2.15 SLC Replacement Hint

This feature is to forward cache replacement hints from the Requesters to the caches in the interconnect (System Level Cache). Typically, RN has the best knowledge of utility of a cache line. A system level cache that is informed of this knowledge can use it to bias its own replacement algorithms and manage replacement in a more efficient manner.

11.2.15.1 Configuration Attributes and APIs

The node configuration variable is added under node configuration for this feature

`svt_chi_node_configuration::slcrephint_mode`

HTML page:

`$DESIGNWARE_HOME/vip/svt/amba_svt/latest/doc/class_ref/chi_e_svt_uvm_class_reference/html/configuration/class_svt_chi_node_configuration.html`

11.2.15.2 Transaction Attributes

Following transaction class variable are applicable to this feature

- ❖ `svt_chi_common_transaction::slcrephint_unusedprefetch`

- ❖ svt_chi_common_transaction::slcrephint_replacement
- ❖ svt_chi_common_transaction::slcrephint_reserved

HTML page:

`$DESIGNWARE_HOME/vip/svt/amba_svt/latest/doc/class_ref/chi_e_svt_uvm_class_reference/html/transaction/class_svt_chi_common_transaction.html`

11.2.16 Replicated Channels in a Single Interface

This section describes the CHI VIP support for the protocol feature ‘Replicated channels in a single interface’.

The VIP components that support this feature are:

- ❖ RN VIP in Active and Passive modes
- ❖ CHI System Monitor

Replicated channels in a single interface for the following channel types are supported:

- ❖ TXREQ
- ❖ TXDAT
- ❖ TXRSP
- ❖ RXDAT
- ❖ RXRSP

11.2.16.1 Limitations

- ❖ This feature is not supported by the SN VIP.
- ❖ Functional coverage support is not available.
- ❖ Protocol analyzer with interface signal grouping is not supported when this new feature is enabled.
- ❖ Performance analyzer is not supported when this new feature is enabled.
- ❖ VC-ATB, bind IF, parameterized IF are not supported when replicated channels are enabled.
- ❖ This feature is supported only with VCS Simulator.

11.2.16.2 Solution Description

Replicated channels in a single interface is an efficient method to increase the available interface bandwidth by selectively replicating channels that require greater bandwidth.

There are no restrictions on which channels are replicated. Typically, the replication of a channel is based on the expected bandwidth required on that channel.

The characteristics of the replicated channel interface are:

- ❖ All replicated DAT sub-channels corresponding to a single DAT channel must be of the same width.
- ❖ The complete interface must use:
 - ◆ Same Node ID.
 - ◆ Single transaction ID pool.
- ❖ No cross-channel relationship:

- ◆ Request on TXREQ0 can give a response on either RXRSP0 or RXRSP1.
- ◆ All existing cross-channel dependencies remain.
- ◆ Multiple response messages for a single request can come on any sub-channel. For example, DBIDResp for a write transaction is received on RXRSP0, whereas the corresponding comp can be received on RXRSP1.
- ❖ Like non-replicated channels, replicated channels do not provide any in-channel ordering guarantees.
- ❖ All link crediting is done on a sub-channel basis:
 - ◆ Cannot use the credit for TXREQ0 to send the flit on TXREQ1.
- ❖ Protocol credits are for the combined TXREQ channel.
- ❖ There is no support for powering down a sub-channel individually.
- ❖ The two parts of a DVM snoop can come on any sub-channel. Each part can be on a different sub-channels.
- ❖ The number of sub-channels on two connected interfaces must match.
- ❖ Credits are required to be provided by the receiver on all sub-channels.
- ❖ There must be only one set of SActive, LinkActive, and SysCo signals, and optional broadcast control pins.
- ❖ Interface property CCF_Wrap_Order is not permitted to be set to True when the interface includes replicated DAT channels.

11.2.16.3 User Interface Macros

This compile time macro needs to be defined to enable this feature:

- ❖ ``SVT_CHI_REPLICATED_CHANNELS_ENABLE`

This macro is permitted to be defined only when `SVT_CHI_ISSUE_E_ENABLE` macro is defined.

These macros are added to allow users to specify the maximum number of replicated channels at RN/SN for each channel type:

- ❖ ``SVT_CHI_RN_MAX_TXREQ_CHANNELS`
- ❖ ``SVT_CHI_RN_MAX_TXDAT_CHANNELS`
- ❖ ``SVT_CHI_RN_MAX_RXDAT_CHANNELS`
- ❖ ``SVT_CHI_RN_MAX_TXRSP_CHANNELS`
- ❖ ``SVT_CHI_RN_MAX_RXRSP_CHANNELS`
- ❖ ``SVT_CHI_RN_MAX_RXSNP_CHANNELS`
- ❖ ``SVT_CHI_SN_MAX_RXREQ_CHANNELS`
- ❖ ``SVT_CHI_SN_MAX_TXDAT_CHANNELS`
- ❖ ``SVT_CHI_SN_MAX_RXDAT_CHANNELS`
- ❖ ``SVT_CHI_SN_MAX_TXRSP_CHANNELS`

The `SVT_CHI_RN/SN_MAX_*_CHANNELS` macros are set to 1 by default. You can redefine the macros only when the compile time macro `SVT_CHI_REPLICATED_CHANNELS_ENABLE` is defined.

The following macros are defined internally in the VIP:

- ❖ ``SVT_CHI_MAX_REQ_CHANNELS`
Will be set to either ``SVT_CHI_RN_MAX_TXREQ_CHANNELS` or ``SVT_CHI_SN_MAX_RXREQ_CHANNELS`, whichever is greater.
- ❖ ``SVT_CHI_MAX_TXDAT_CHANNELS`
Will be set to either ``SVT_CHI_RN_MAX_TXDAT_CHANNELS` or ``SVT_CHI_SN_MAX_TXDAT_CHANNELS`, whichever is greater.
- ❖ ``SVT_CHI_MAX_RXDAT_CHANNELS`
Will be set to either ``SVT_CHI_RN_MAX_RXDAT_CHANNELS` or ``SVT_CHI_SN_MAX_RXDAT_CHANNELS`, whichever is greater.
- ❖ ``SVT_CHI_MAX_TXRSP_CHANNELS`
Will be set to either ``SVT_CHI_RN_MAX_TXRSP_CHANNELS` or ``SVT_CHI_SN_MAX_TXRSP_CHANNELS`, whichever is greater.
- ❖ ``SVT_CHI_MAX_RXRSP_CHANNELS`
Will be set to either ``SVT_CHI_RN_MAX_RXRSP_CHANNELS` or ``SVT_CHI_SN_MAX_RXRSP_CHANNELS`, whichever is greater.

11.2.16.4 Signal Interface

Some of the existing channel related interface signals in the VIP RN and SN interfaces (`svt_chi_rn_if` and `svt_chi_sn_if`) would be updated to packed arrays that are sized to the corresponding `SVT_CHI_RN/SN_MAX_*_CHANNELS` macros, by default.

For example: In the RN interface, logic `TXREQFLITV` would be updated by default to 'logic [(`SVT_CHI_RN_MAX_TXREQ_CHANNELS-1):0] TXREQFLITV' ; 'logic TXREQFLITPEND' will be updated by default to 'logic [[(`SVT_CHI_RN_MAX_TXREQ_CHANNELS-1):0] TXREQFLITPEND' and so on.

These interface signals are updated on the above lines across all channel types regardless of whether the replicated channels feature is enabled or not:

`*FLITPEND, *FLITV, *FLITLCRDV`

When the `SVT_CHI_REPLICATED_CHANNELS_ENABLE` macro is not defined, the size of the above signal arrays is 1 and therefore existing accesses/assignments such as `chi_if.rn_if[0].TXREQFLITV` would work as expected without the need to pass any array index.

The FLIT interface signals in the VIP RN and SN interfaces (`svt_chi_rn_if` and `svt_chi_sn_if`) would be updated to packed arrays that are sized to the corresponding `SVT_CHI_RN/SN_MAX_*_CHANNELS` macros, when the compile time `SVT_CHI_REPLICATED_CHANNELS_ENABLE` macro is defined. When the compile time macro is not defined, the `*FLIT` signal declaration would remain unchanged.

Example:

In the RN interface, when `SVT_CHI_REPLICATED_CHANNELS_ENABLE` macro is not defined, the request flit signal is declared as 'logic [`SVT_CHI_MAX_REQ_FLIT_WIDTH-1:0] TXREQFLIT'

and

when `SVT_CHI_REPLICATED_CHANNELS_ENABLE` is defined, it is declared as 'logic [[(`SVT_CHI_RN_MAX_TXREQ_CHANNELS-1):0] [`SVT_CHI_MAX_REQ_FLIT_WIDTH-1:0] TXREQFLIT'

11.2.16.5 Configuration Attributes

These are the node configurations:

- ❖ `svt_chi_node_configuration::num_req_channels:`
 - ◆ Specifies the number of Request channels that are used by the node
 - ◆ The value should be less than or equal to `SVT_CHI_RN_MAX_TXREQ_CHANNELS` in case of RNs and `SVT_CHI_SN_MAX_RXREQ_CHANNELS` in case of SNs.
- ❖ `svt_chi_node_configuration::num_tx_data_channels:`
 - ◆ Specifies the number of TX Data channels that are used by the node
 - ◆ The value should be less than or equal to `SVT_CHI_RN_MAX_TXDAT_CHANNELS` in case of RNs and `SVT_CHI_SN_MAX_TXDAT_CHANNELS` in case of SNs.
 - ◆ In case this configuration is set to a value greater than 1, TX CCF wrap order must be set to `FALSE`
- ❖ `svt_chi_node_configuration::num_rx_data_channels`
 - ◆ Specifies the number of RX Data channels that are used by the node
 - ◆ The value should be less than or equal to `SVT_CHI_RN_MAX_RXDAT_CHANNELS` in case of RNs and `SVT_CHI_SN_MAX_RXDAT_CHANNELS` in case of SNs.
 - ◆ In case this configuration is set to a value greater than 1, RX CCF Wrap order must be set to `FALSE`
- ❖ `svt_chi_node_configuration::num_tx_rsp_channels`
 - ◆ Specifies the number of TX Response channels that are used by the node.
 - ◆ The value should be less than or equal to `SVT_CHI_RN_MAX_TXRSP_CHANNELS` in case of RNs and `SVT_CHI_SN_MAX_TXRSP_CHANNELS` in case of SNs.
- ❖ `svt_chi_node_configuration::num_rx_rsp_channels`
 - ◆ Specifies the number of RX Response channels that are used by the node.
 - ◆ Only applicable for RNs.
 - ◆ The value should be less than or equal to `SVT_CHI_RN_MAX_RXRSP_CHANNELS`.
- ❖ `svt_chi_node_configuration::num_snp_channels`
 - ◆ Specifies the number of RX Snoop channels that are used by the node.
 - ◆ Only applicable for RNs.
 - ◆ The value should be less than or equal to `SVT_CHI_RN_MAX_RXSNP_CHANNELS`.

These existing node configuration attributes will be updated to allow programming of per channel information:

- ❖ `rx_dat_vc_flit_buffer_size`
- ❖ `rx_req_vc_flit_buffer_size`
- ❖ `rx_rsp_vc_flit_buffer_size`
- ❖ `rx_snp_vc_flit_buffer_size`
- ❖ `rcvd_dat_flit_to_lcrd_max_delay`
- ❖ `rcvd_dat_flit_to_lcrd_min_delay`
- ❖ `rcvd_req_flit_to_lcrd_max_delay`

- ❖ rcvd_req_flit_to_lcrd_min_delay
- ❖ rcvd_rsp_flit_to_lcrd_max_delay
- ❖ rcvd_rsp_flit_to_lcrd_min_delay
- ❖ rcvd_snp_flit_to_lcrd_max_delay
- ❖ rcvd_snp_flit_to_lcrd_min_delay
- ❖ num_xmitted_rxreq_vc_lcredits_in_rxdeactivate_state
- ❖ num_xmitted_rxrsp_vc_lcredits_in_rxdeactivate_state
- ❖ num_xmitted_rxdnt_vc_lcredits_in_rxdeactivate_state
- ❖ num_xmitted_rxsnp_vc_lcredits_in_rxdeactivate_state

Example:

- ❖ Current declaration:
 - ◆ rand int unsigned rx_dat_vc_flit_buffer_size =
`SVT_CHI_REASONABLE_FLIT_BUFFER_SIZE;
- ❖ New declaration:
 - ◆ bit [`SVT_CHI_MAX_RXDAT_CHANNELS - 1 : 0] [31:0] rx_dat_vc_flit_buffer_size ;
 - ◆ Note that when `SVT_CHI_MAX_RXDAT_CHANNELS is 1, the already existing testbench configuration setting needs no update as the attribute can be accessed as
<rn_cfg[0]>.rx_dat_vc_flit_buffer_size in this case.

11.2.16.6 Transaction Attributes

These controls are added in the transaction class (svt_chi_transaction) and are applicable for transactions:

Table 11-2 Transaction attributes

Transaction	Description
request_channel_transmission_policy_enum svt_chi_transaction::request_channel_transmission_policy	<ul style="list-style-type: none"> • Specifies how the channel ID must be determined for Request flit corresponding to this transaction. • Only applicable for active RN VIP, when SVT_CHI_REPLICATED_CHANNELS_ENABLE macro is defined. • Can take the following values: <ul style="list-style-type: none"> - RANDOM_REQ_CHANNEL : The request flit would be sent on a random channel that is available at the point of scheduling the flits. The field req_channel_id would be automatically populated to indicate which channel the request was sent on by the VIP. - DIRECTED_REQ_CHANNEL : The request flit would be sent out on the user-specified replicated request channel. You must specify the channel on which the request flit must be transmitted by programming req_channel_id. • Currently, only the default value of RANDOM_REQ_CHANNEL is supported.

Transaction	Description
<code>data_channel_transmission_policy_enum</code> <code>svt_chi_transaction::tx_data_channel_transmission_policy</code>	<ul style="list-style-type: none"> Specifies how the channel ID must be determined for Data flits corresponding to this transaction Can take the following values: <ul style="list-style-type: none"> <code>RANDOM_DATA_CHANNELS</code> : Data flits will be sent on a random channel that is available at the point of scheduling <code>DIRECTED_DATA_CHANNELS</code> : Data flits will be sent out on the channel specified by <code>tx_data_channel_id[]</code> Currently, only the default value of <code>RANDOM_DATA_CHANNELS</code> is supported.
<code>response_channel_transmission_policy_enum</code> <code>svt_chi_transaction::tx_response_channel_transmission_policy</code>	<ul style="list-style-type: none"> Specifies how the channel ID must be determined for Response flits corresponding to this transaction Can take the following values: <ul style="list-style-type: none"> <code>RANDOM_RESP_CHANNELS</code> : Response flits will be sent on a random channel that is available at the point of scheduling. <code>DIRECTED_RESP_CHANNELS</code> : Response flits will be sent out on user specified channels. User can specify on which channel each response flit must be transmitted by programming <code>tx_rsp_channel_id[]</code> Currently, only the default value of <code>RANDOM_RESP_CHANNELS</code> is supported
<code>Int svt_chi_transaction::req_channel_id</code>	<ul style="list-style-type: none"> Specifies the REQ channel index on which the request flit is transmitted. In case of active RN, when <code>request_channel_transmission_policy</code> is set to <code>DIRECTED_REQ_CHANNEL</code>, this field is expected to be programmed by the user. When <code>request_channel_transmission_policy</code> is set to <code>RANDOM_REQ_CHANNEL</code>, this field is populated automatically by the active RN VIP
<code>Int svt_chi_transaction::tx_data_channel_id[]</code>	<ul style="list-style-type: none"> Specifies the TX Data channel indices on which the TX DATA flits were sent. Size will be equal to the number of data flits transmitted for the transaction. In case of active components, when <code>tx_data_channel_transmission_policy</code> is set to <code>DIRECTED</code>, the entries in the array are expected to be programmed by the user. When <code>tx_data_channel_transmission_policy</code> is set to <code>RANDOM_DATA_CHANNELS</code>, the entries in the queue are automatically populated by the VIP

Transaction	Description
<code>Int svt_chi_transaction::tx_rsp_channel_id[]</code>	<ul style="list-style-type: none"> Specifies the TX Response channel indices on which the TX RESPONSE flits were sent. Size will be equal to the number of response flits transmitted for the transaction. In case of active components, when <code>tx_response_channel_transmission_policy</code> is set to <code>DIRECTED_RESP_CHANNELS</code>, the entries in the array are expected to be programmed by the user. When <code>tx_response_channel_transmission_policy</code> is set to <code>RANDOM_RESP_CHANNELS</code>, the entries in the array are automatically populated by the VIP
<code>Int svt_chi_transaction::rx_data_channel_id[]</code>	<ul style="list-style-type: none"> Specifies the RX Data channel indices on which the RX Data flits were received. Size will be equal to the number of data flits received for the transaction. The entries in this array will be populated automatically by the VIP.
<code>Int svt_chi_transaction::rx_rsp_channel_id[]</code>	<ul style="list-style-type: none"> Specifies the RX Response channel indices on which the RX Response flits were received. Size will be equal to the number of response flits received for the transaction. Only applicable in case of RN and ICN Full subordinate VIP. The entries in this array will be populated automatically by the VIP
Following members are added to the Snoop transaction class (<code>svt_chi_snoop_transaction</code>): <code>data_channel_transmission_policy_enum</code> <code>svt_chi_snoop_transaction::tx_data_channel_transmission_policy</code>	<ul style="list-style-type: none"> Specifies how the channel ID must be determined for Snoop Data flits corresponding to this Snoop transaction. Can take the following values: <ul style="list-style-type: none"> <code>RANDOM_DATA_CHANNELS</code> : Snoop data response flits will be sent on a random channel that is available at the point of scheduling <code>DIRECTED_DATA_CHANNELS</code> : Snoop data response flits will be sent out on the channel specified by <code>tx_data_channel_id[]</code> Applicable only in case of active RN. Currently, only the default value of <code>RANDOM_DATA_CHANNELS</code> is supported

Transaction	Description
<code>response_channel_transmission_policy_enum</code> <code>svt_chi_snoop_transaction::tx_response_channel_transmission_policy</code>	<ul style="list-style-type: none"> Specifies how the channel ID must be determined for Snoop response flits corresponding to this Snoop transaction. Can take the following values: <ul style="list-style-type: none"> <code>RANDOM_RESP_CHANNELS</code> : Snoop data response flits will be sent on a random channel that is available at the point of scheduling <code>DIRECTED_RESP_CHANNELS</code> : Snoop data response flits will be sent out on the channel specified by <code>tx_rsp_channel_id[]</code> Applicable only in case of active RN. Currently, only the default value of <code>RANDOM_RESP_CHANNELS</code> is supported.
<code>snoop_channel_transmission_policy_enum</code> <code>svt_chi_snoop_transaction::snoop_channel_transmission_policy</code>	<ul style="list-style-type: none"> Specifies how the channel ID must be determined for Snoop flits corresponding to this Snoop transaction Only applicable for ICN Full subordinate VIP. Can take the following values: <ul style="list-style-type: none"> <code>RANDOM_SNP_CHANNELS</code>: Snoop flit will be sent on a random channel that is available at the point of scheduling <code>DIRECTED_SNP_CHANNELS</code>: Snoop flit will be sent out on the channels. You can specify the channel on which each Snoop flit must be transmitted by programming <code>snp_channel_id[]</code>
<code>data_channel_transmission_policy_enum</code> <code>svt_chi_snoop_transaction::fwded_data_channel_transmission_policy</code>	<ul style="list-style-type: none"> Specifies how the channel ID must be determined for Fwded CompData flits corresponding to a Fwd Type Snoop transaction Can take the following values: <ul style="list-style-type: none"> <code>RANDOM_DATA_CHANNELS</code> : Forward CompData flits will be sent on a random channel that is available at the point of scheduling. <code>DIRECTED_DATA_CHANNELS</code> : Forward CompData flits will be sent out on user specified channels. User can specify on which channel each data flit must be transmitted by programming <code>fwded_data_channel_id[]</code> Applicable only for Forward type snoops in case of active RN. Currently, only the default value of <code>RANDOM_DATA_CHANNELS</code> is supported

Transaction	Description
<code>response_channel_transmission_policy_enum</code> <code>svt_chi_snoop_transaction::data_pull_response_channel_transmission_policy</code>	<ul style="list-style-type: none"> Can take the following values: <ul style="list-style-type: none"> <code>RANDOM_RESP_CHANNELS</code> : TX Response flits corresponding to DataPull request (CompAck in case of RN, RespSepData in case of ICN Full subordinate) will be sent on a random channel that is available at the point of scheduling. <code>DIRECTED_RESP_CHANNELS</code> : DataPull response flits will be sent out on user specified channels. You can specify on which channel each response flit must be transmitted by programming <code>data_pull_tx_response_channel_id</code> Currently, only the default value of <code>RANDOM_RESP_CHANNELS</code> is supported.
<code>data_channel_transmission_policy_enum</code> <code>svt_chi_snoop_transaction::pulled_read_data_channel_transmission_policy</code>	<ul style="list-style-type: none"> Specifies how the channel ID must be determined for DataPull Read data flits corresponding to a Stash snoop transaction. Can take the following values: <ul style="list-style-type: none"> <code>RANDOM_DATA_CHANNELS</code> : DataPull read data flits corresponding to DataPull request will be sent on a random channels that are available at the point of scheduling. <code>DIRECTED_DATA_CHANNELS</code> : DataPull read data flits will be sent out on user specified channels. User can specify on which channel each Read data flit must be transmitted by programming <code>pulled_read_data_channel_id[]</code> Applicable only in case of Stash snoops involving DataPull. This field is not applicable for active RN. Currently, only the default value of <code>RANDOM_DATA_CHANNELS</code> is supported.
<code>Int</code> <code>svt_chi_snoop_transaction::tx_data_channel_id[]</code>	<ul style="list-style-type: none"> Specifies the TX Data channel indices on which the Snoop data response (SnpRespData*) flits were sent. Size will be equal to the number of Snoop data response flits transmitted for the transaction. In case of active RN, when <code>tx_data_channel_transmission_policy</code> is set to <code>DIRECTED_DATA_CHANNELS</code>, the entries in the array are expected to be programmed by the user. In all other cases, this field will automatically be populated by the VIP.

Transaction	Description
Int <code>svt_chi_snoop_transaction::tx_rsp_channel_id[]</code>	<ul style="list-style-type: none"> Specifies the TX Response channel indices on which the Snoop Response flits were sent. Size will be equal to the number of snoop response flits transmitted for the transaction. In case of active RN, when <code>tx_response_channel_transmission_policy</code> is set to <code>DIRECTED_RESP_CHANNELS</code>, the entries in the array are expected to be programmed by the user. In all other cases, this field will automatically be populated by the VIP.
Int <code>svt_chi_snoop_transaction::snoop_channel_id[]</code>	<ul style="list-style-type: none"> Specifies the snoop channel indices on which the snoop request flits were sent. Size will be equal to the number of snoop request flits transmitted for the snoop transaction. In case of RN, this field will automatically be populated by the VIP and is read-only for the user.
Int <code>svt_chi_snoop_transaction::fwded_data_channel_id[]</code>	<ul style="list-style-type: none"> Specifies the TX Data channel indices on which the Fwded CompData flits were sent. Size will be equal to the number of Fwded CompData flits transmitted for the Snoop transaction. Only applicable in case of Forward type snoops. In case of active RN, when <code>fwded_data_channel_transmission_policy</code> is set to <code>DIRECTED_DATA_CHANNELS</code>, the entries in the array are expected to be programmed by the user. In all other cases, this field will automatically be populated by the VIP.
Int <code>svt_chi_snoop_transaction::fwded_data_channel_id[]</code>	<ul style="list-style-type: none"> Specifies the TX Data channel indices on which the Fwded CompData flits were sent. Size will be equal to the number of Fwded CompData flits transmitted for the Snoop transaction. Only applicable in case of Forward type snoops. In case of active RN, when <code>fwded_data_channel_transmission_policy</code> is set to <code>DIRECTED_DATA_CHANNELS</code>, the entries in the array are expected to be programmed by the user. In all other cases, this field will automatically be populated by the VIP.

Transaction	Description
Int <code>svt_chi_snoop_transaction::pulled_read_data_channel_id[]</code>	<ul style="list-style-type: none"> Specifies the TX Data channel indices on which the DataPull Read data flits were sent. Size will be equal to the number of DataPull Read data flits transmitted for the Snoop transaction. Only applicable in case of Stash type snoops. In case of active RN, when <code>pulled_read_data_channel_transmission_policy</code> is set to <code>DIRECTED_DATA_CHANNELS</code>, the entries in the array are expected to be programmed by the user. In all other cases, this field will be automatically populated by the VIP.
Int <code>svt_chi_snoop_transaction::data_pull_tx_response_channel_id</code>	<ul style="list-style-type: none"> Specifies the TX Response channel indices on which the DataPull Response flits were sent (CompAck in case of RN and RespSepData in case of ICN). Only applicable in case of Stash snoops involving DataPull. When <code>data_pull_response_channel_transmission_policy</code> is set to <code>DIRECTED_RESP_CHANNELS</code>, the entries in the array are expected to be programmed by the user. In all other cases, this field will automatically be populated by the VIP.
Int <code>svt_chi_snoop_transaction::data_pull_rx_response_channel_id</code>	<ul style="list-style-type: none"> Specifies the RX Response channel indices on which the DataPull Response flits were received (RespSepData in case of RN and CompAck in case of ICN). Only applicable in case of Stash snoops involving DataPull. This field will automatically be populated by the VIP.
Following member is added to the flit class: Int <code>svt_chi_flit::channel_id</code>	<ul style="list-style-type: none"> Represents the channel index on which the flit is sent or received. The channel type can be inferred from the flit type. This is a read-only field and will be populated internally by the VIP driver or monitor.

11.2.16.7 Behavior of VIP Components

ACTIVE RN AGENT

The active RN VIP supports the transmission and reception of flits on the replicated channels for the following channel types:

- ❖ TXREQ,
- ❖ TXRSP,
- ❖ RXRSP,
- ❖ TXDAT, and
- ❖ RXDAT

In case of flits to be transmitted on the TX channels, the controls are provided in the transaction class to specify on which channels the flits must be sent.

Example: The fields `request_channel_transmission_policy` must be programmed by the user to indicate if the VIP must drive the request flit randomly on any of the available request channels or if the flit must be driven on the request channel specified by the user. In case the user indicates that the request flit must be driven on a user-specified request channel, they must program the transaction field `req_channel_id` appropriately. In case the user indicates that the VIP must drive the flit randomly on one of the available request channels, at the point of transmission of the flit, the VIP RN driver dispatches the request flit on one of the available request channels and programs the `req_channel_id` field in the transaction to indicate the index of the request replicated channel on which the flit is dispatched.

In case of flits received on the RX channels, the VIP programs the corresponding `*_channel_id` field in the associated transaction handle to indicate the index of the replicated channel on which the flit was received.

The field `channel_id` in the flit class indicates the replicated channel index on which the flit was transmitted/received and is populated automatically by the VIP RN agent.

PASSIVE RN AGENT

The passive RN VIP is updated to support the reception of flits on the replicated channels for the following channel types:

- ❖ TXREQ,
- ❖ TXRSP,
- ❖ RXRSP,
- ❖ TXDAT, and
- ❖ RXDAT

Whenever any flit is observed on any of the channels, the VIP programs the corresponding `*_channel_id` field in the associated transaction handle to indicate the index of the replicated channel on which the flit was received.

The field `channel_id` in the flit class indicates the replicated channel index on which the flit was transmitted/received and is populated automatically by the VIP RN agent.

Protocol Checks:

The existing link layer checks which are applicable for each replicated channel (like signal valid checks, reset checks, link credit checks, etc.) are enhanced to perform the check for each channel.

11.2.16.8 Link Layer Custom Features

Custom Link Layer features are supported with replicated channels feature being enabled:

- ❖ All the `svt_chi_node_configuration::flitpend_assertion_policy` configurations
- ❖ Auto link deactivation feature `svt_chi_node_configuration::tx_link_deactivation_time`
- ❖ L-credit transmission in RX DEACT state
`svt_chi_node_configuration::num_xmitted_rx<*>_vc_lcredits_in_rxdeactivate_state`
- ❖ Protocol Flits transmission in TX DEACT state
`svt_chi_node_configuration::allow_protocol_flits_transmission_in_tx_lasm_deactivate_state`

- ❖ Suspend and Resume l-credits service requests. If suspend l-credit on <REQ/RSP/DAT > channel service request is received, then l-credits are suspended on all the replicated <REQ/RSP/DAT > channels.

11.2.16.9 Debug Features

- ❖ Verdi Protocol Analyzer and FSDB are supported.
- ❖ Required log file messaging is added.

11.2.16.10 Coverage

- ❖ Coverage is not supported. However, the applicable coverage that is already supported with earlier versions of CHI specification will be automatically supported.
- ❖ Link layer coverage is not supported when replicated channels feature is enabled.

11.2.17 Miscellaneous Updates

11.2.17.1 Transaction Ordering Guarantees

This specification does not require any ordering guarantees between two transactions from the same source that have different ordering requirement. A requester, which changes the ordering requirements of a transaction to a stronger ordering requirement, is required to be consistent in changing ordering requirement of RO to EO on all its transactions.

11.2.17.1.1 Configuration Attributes

There are no new configurations attributes added for this update. This is an existing node configuration which is relevant to this feature:

- ❖ svt_chi_node_configuration::num_req_order_streams

HTML page:

`$DESIGNWARE_HOME/vip/svt/amba_svt/latest/doc/class_ref/chi_e_svt_uvm_class_reference/html/configuration/class_svt_chi_node_configuration.html`

11.2.17.1.2 Transaction Attributes

No new attributes are added. This is an existing attribute which is Relevant to this feature

- ❖ svt_chi_transaction:: req_order_stream_id

HTML page:

`$DESIGNWARE_HOME/vip/svt/amba_svt/latest/doc/class_ref/chi_e_svt_uvm_class_reference/html/transaction/class_svt_chi_transaction.html`

11.2.17.2 Limitations

Other miscellaneous updates of CHI-E specifications are not supported.

11.3 CHI E Protocol Checks

VIP class reference HTML page can be referred for the list of protocol checks. The page link is given below.

[\\$DESIGNWARE_HOME/vip/svt/amba_svt/latest/doc/class_ref/chi_e_svt_uvm_class_reference/html/protocolChecks.html](#)

11.4 CHI E Functional Coverage Report

The details of CHI Issue E specific cover groups are available in CHI-E VIP Class reference HTML document. Link to the HTML page is given below. CHI-E specific cover groups can be viewed by selecting CHI Issue E specific Groups and Subgroups using the drop-down menu.

HTML page:

- ❖ *[\\$DESIGNWARE_HOME/vip/svt/amba_svt/latest/doc/class_ref/chi_e_svt_uvm_class_reference/html/level1_covergroups.html](#)*
- ❖ [AMBA CHI E class reference](#)

11.5 CHI E Unsupported Features

These features are not supported from CHI Issue E specification:

- ❖ Snoop Forward as a Hint
- ❖ Multiple interfaces

A

Unsupported Features and Limitations

A.1 Unsupported Features

The following features are not supported.

CHI VIP:

- ❖ Cache Tracking in Passive RN-F agent
- ❖ System monitor Master/Slave correlation based checks when L3/SLC is enabled within Interconnect DUT
- ❖ Reconfiguring the configuration objects

CHI-B VIP:

- ❖ Trace files
- ❖ TLM GP
- ❖ Interconnect does not support DMT feature
- ❖ Interconnect does not support DCT and RetToSrc
- ❖ SN agents do not support Atomic transactions
- ❖ Interconnect does not support Atomic transactions
- ❖ Active RN agents cannot perform Atomic operations locally

CHI-C VIP:

- ❖ CHI-C features are not yet supported by CHI Interconnect VIP component.
- ❖ Functional coverage is not yet updated for CHI-C specific protocol features.

CHI-D VIP:

- ❖ Icache Invalidation
- ❖ Interface Parity
- ❖ Deep Persistent Cache Maintenance
- ❖ Dvm Early Comp

CHI-E VIP:

- ❖ Snoop Forward as a Hint
- ❖ Multiple interfaces

- ❖ Replicated Channels on a Single Interface

A.2 Known Issues and Limitations

Currently, CHI-B VIP does not support the setup that contains legacy RNs and CHI Issue B compliant RNs connected to a CHI Issue B Compliant Interconnect.

In the case of `CleanSharedPersist`:

- ❖ Currently HN->SN flow is not supported.
- ❖ Interconnect does not support this transaction to SN
- ❖ SN does not support `CleanSharedPersist`.
- ❖ `CleanSharedPersist` targeting HN-I is not supported by RN, interconnect VIP and system monitor.
- ❖ An error is flagged for the above unsupported items.

Snoop and coherent transaction association:

The coherent and snoop transaction association only takes the valid combinations specified in the Table 4-3, Section 4.4 of the CHI-B specification into consideration, even though the specification permits the Interconnect to send various other types of Snoop requests for certain coherent transactions.

An exception to this is for `ReadShared` transactions, along with the snoop requests specified in Table 4-3, the VIP considers `SnpNotSharedDirtyFwd` to also be a valid Snoop request type.



Note

For more information, see the *Known Issues and Limitations* section in the AMBA SVT VIP Release Notes. AMBA SVT VIP Release Notes is present at:
`$DESIGNWARE_HOME/vip/svt/amba_svt/latest/doc/PDFs/amba_svt_rel_notes.pdf`

B

Reporting Problems

B.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.

B.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.

