Verification Continuum™

VC Verification IP
MIPI I3C
UVM User Guide

Version Q-2020.06, June 2020

SYNOPSYS®

Copyright Notice and Proprietary Information

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

About This Guide

This guide contains installation, setup, and usage material for VC VIP for I3C on the Universal Verification Methodology (UVM). This guide is for design or verification engineers who want to verify I3C operations using a UVM testbench written in SystemVerilog. Readers are assumed to be familiar with I3C, Object-Oriented Programming (OOP), SystemVerilog, and UVM techniques.

Guide Organization

The chapters of this user guide are organized as follows:

- Chapter 1, "Introduction", introduces the I3C VIP and its features.
- Chapter 2, "Installation and Setup", describes system requirements and provides instructions on how to install, configure, and begin using the I3C VIP.
- Chapter 3, "General Concepts", introduces the I3C VIP within a UVM environment and describes the data objects and components that comprise the VIP.
- Chapter 4, "Verification Topologies", describes the topologies to verify Master, Slave and interconnect DUT.
- Chapter 5, "Using I3C Verification IP", shows how to install and run a getting started example.
- Chapter 8, "Troubleshooting", provides some useful information that can help you to troubleshoot common issues that you may encounter while using the I3C VIP.
- Appendix A, "Reporting Problems", outlines the process for working through and reporting I3C VIP issues.

Web Resources

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

Customer Support

To register a problem, perform any of the following tasks:

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 Product version for which you want to register the problem.

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:

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

1 Introduction

The Synopsys I3C verification IP supports the verification of SoC designs that include interfaces implementing I3C specification. This document describes the use of the VIP in testbenches that comply with the SystemVerilog UVM. This approach leverages advanced verification technologies and tools that provide the following features:

- 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
- Object-oriented interface that allows OOP techniques

Refer the Synopsys I3C VIP class reference HTML documentation, which is installed at the following location:

```
$DESIGNWARE_HOME/vip/svt/i3c_svt/latest/doc/i3c_svt_uvm_class_reference/html/index.html
```

This chapter consists of the following sections:

1.1 Product Overview

I3C UVM VIP is a suite of UVM-based verification components that are compatible for use with SystemVerilog-compliant testbenches. The I3C VIP suite simulates I3C transactions through active agents, as defined by I3C specifications (compliant with version 1.0).

1.2 Language and Simulator Support

In this current release, the I3C VIP suite is qualified with the following languages and methodology:

- Languages
  - SystemVerilog
- Methodology
  - Qualified with UVM 2.1.2

Table 1 Simulator Platform Support

Simulator	VCSSIM K-2015.09-SP2	VCSSIM L-2016.06-SP2
Synopsys VCS-PC (UUM) Verilog **L-2016.06-SP1**	None	Redhat Enterprise Linux 5.0/32 Redhat Enterprise Linux 5.0/64 Redhat Enterprise Linux 6.0/32 Redhat Enterprise Linux 6.0/64 Suse 11.0/32 Suse 11.0/64
Synopsys VCS-PIP (UUM) Verilog **L-2016.06-SP2**	None	Redhat Enterprise Linux 5.0/32 Redhat Enterprise Linux 5.0/64 Redhat Enterprise Linux 6.0/32 Redhat Enterprise Linux 6.0/64 Suse 11.0/32 Suse 11.0/64
Synopsys VCS (UUM) Verilog **K-2015.09-SP2**	Redhat Enterprise Linux 5.0/32 Redhat Enterprise Linux 5.0/64 Redhat Enterprise Linux 6.0/32 Redhat Enterprise Linux 6.0/64 Solaris sparc 10/32 Solaris sparc 10/64 Suse 11.0/32 Suse 11.0/64	None
Synopsys VCS (UUM) Verilog **L-2016.06-SP2**	None	Redhat Enterprise Linux 5.0/32 Redhat Enterprise Linux 5.0/64 Redhat Enterprise Linux 6.0/32 Redhat Enterprise Linux 6.0/64 Suse 11.0/32 Suse 11.0/64
Synopsys VCS-PC Verilog **L-2016.06-SP2**	None	Redhat Enterprise Linux 5.0/32 Redhat Enterprise Linux 5.0/64 Redhat Enterprise Linux 6.0/32 Redhat Enterprise Linux 6.0/64 Suse 11.0/32 Suse 11.0/64

Simulator	VCSSIM K-2015.09-SP2	VCSSIM L-2016.06-SP2
Synopsys VCS Verilog K-2015.09-SP2	Redhat Enterprise Linux 5.0/32 Redhat Enterprise Linux 5.0/64 Redhat Enterprise Linux 6.0/32 Redhat Enterprise Linux 6.0/64 Solaris sparc 10/32 Solaris sparc 10/64 Suse 11.0/32 Suse 11.0/64	None
Synopsys VCS Verilog L-2016.06-SP2	None	Redhat Enterprise Linux 5.0/32 Redhat Enterprise Linux 5.0/64 Redhat Enterprise Linux 6.0/32 Redhat Enterprise Linux 6.0/64 Suse 11.0/32 Suse 11.0/64

1.3 Features Supported

This section consists of the following sub-sections:

1.3.1 Protocol Features

The I3C VIP currently supports the following protocol functions:

- Compliant with version 0.7 r0.4
- Legacy I2C support
- Multi-Master and Multi-Slave configurations
- SDR mode
- IBI
- Arbitration
- Repeated start or stop

1.3.2 Verification Features

The I3C VIP currently supports the following verification functions:

- Configurability for the master agent and the slave agent
- Built-in protocol checks
- Sequence collection
- The I3C VIP configuration:

- As Legacy I2C Slave
- As I3C Master
- As I3C Slave

1.3.3 Methodology Features

The I3C VIP currently supports the following methodology functions:

- The VIP is organized as the I3C System environment, which includes the set of Master agents and Slave agents.
- Analysis ports for connecting Master or Slave agents to subscribers, such as Scoreboard and coverage collectors.
- Callbacks for the Master agent and the Slave agent.

2
Installation and Setup

This chapter leads you through the installing and setting up the I3C VIP. After completing this checklist, the provided example testbench will be operational and the I3C VIP will be ready to use.

This chapter consists of the following major steps:

2.1 Verifying Hardware Requirements

The I3C VIP requires the following configuration for Solaris or Linux workstation:

- 400 MB available disk space for installation
- 16 GB Virtual Memory recommended
- FTP anonymous access to ftp.synopsys.com (optional)

2.2 Verifying Software Requirements

The I3C VIP is qualified for use with the certain versions of platforms and simulators. This section lists the software required by the I3C VIP and consists of the following sub-sections:

2.2.1 Platform/OS and Simulator Software

Platform/OS and VCS: You need versions of your platform/OS and simulator that have been qualified for use. To see which platform/OS and simulator versions are qualified for use with the AHB OVM VIP, check the support matrix manual.

2.2.2 SCL Software

- The SCL software provides the licensing function for the AHB OVM VIP. Acquiring the SCL software is covered here in the installation instructions in "Licensing Information" on page 17.

2.2.3 Other Third-Party Software

- **Adobe Acrobat**: Synopsys I3C VIP documents are available in Acrobat PDF files. You can get Adobe Acrobat Reader for free from http://www.adobe.com.
- **HTML browser:** Synopsys I3C 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

Perform the following steps to prepare for installation:

a. Set `DESIGNWARE_HOME` to the absolute path where the Synopsys I2C VIP is to be installed:

```
setenv DESIGNWARE_HOME absolute_path_to_designware_home
```

b. Ensure that your environment and PATH variables are set 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 the SCL software or the `port@host` reference to this file.

```
% setenv SNPSLMD_LICENSE_FILE $LM_LICENSE_FILE
<my_Synopsys_license_file|port@host>
```

- `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.

```
% setenv DW_LICENSE_FILE <my_VIP_license_file|port@host>
```

2.4 Downloading and Installing

You can download software from the Download center using either HTTPS or FTP, or with a command-line FTP session. If you do not know your Synopsys SolvNet password or you do not remember it, go to http://solvnet.synopsys.com.

You require the passive mode of FTP. The passive command toggles between the passive and active mode. If your FTP utility does not support the 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

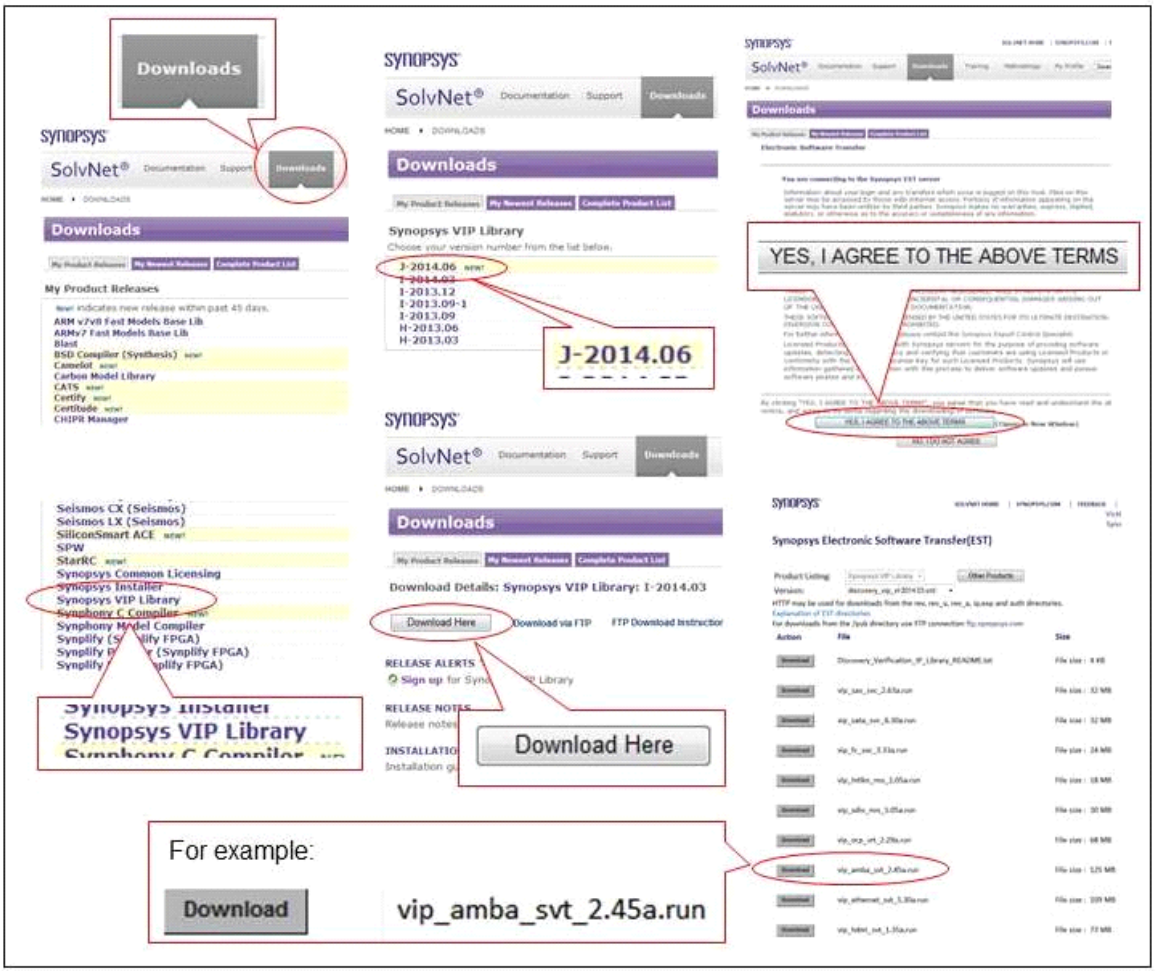
Attention The Electronic Software Transfer (EST) system only displays products that your site is entitled to download. If the product you are looking for is not available, contact est-ext@synopsys.com.

This section consists of the following sub-sections:

2.4.1 Downloading From EST (Download Center)

a. Point your web browser to http://solvnet.synopsys.com.
b. Enter your Synopsys SolvNet Username and Password.
c. Click the `Sign In` button.
d. Make the following selections on SolvNet to download the `.run` file of the VIP (See Figure 2-1).
   i. `Downloads` tab
   ii. VC VIP Library product releases
   iii. <O-2018.09>
   iv. `Download Here` button
   v. `Yes, I Agree to the Above Terms` button
   vi. Download `.run` file for the VIP

Figure 2-1 SolvNet 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.

2.4.2 Downloading Using FTP With a Web Browser

Follow Step a to Step e of Section 2.4.1 and then perform the following steps:

a. Click the **Download via FTP** link instead of the **Download Here** button.

b. Click the **Click Here To Download** button.

c. Select the file(s) that you want to download.

d. Follow browser prompts to select a destination location.

2.5 Setting Up a Testbench Design Directory

A design directory is where the I3C VIP is set up for use in a testbench. The dw_vip_setup utility is provided as design directory is required for using VIP. The dw_vip_setup utility allows you to create the design directory (design_dir), which contains the VIP components, support files (include files), and examples (if any). Add a specific version of the I3C VIP from DESIGNWARE_HOME to a design directory.

For a complete description of dw_vip_setup, see"The dw_vip_setup Utility" on page 22.

The following models are provided with the I3C VIP:

- `mipi_i3c_master_agent_svt`
- `mipi_i3c_slave_agent_svt`

Use the following command to create a design directory and add a model to be used in a testbench:

```
%$DESIGNWARE_HOME/bin/dw_vip_setup -path design_dir -a(dd) <model1> -svtb
```

For example, `%$DESIGNWARE_HOME/bin/dw_vip_setup -path design_dir -a mipi_i3c_master_agent_svt -svtb`

or

```
%$DESIGNWARE_HOME/bin/dw_vip_setup -path design_dir -a(dd) <model1> <model2> <model3> -svtb
```

For example, `%$DESIGNWARE_HOME/bin/dw_vip_setup -path design_dir -a mipi_i3c_master_agent_svt mipi_i3c_slave_agent_svt -svtb`

or

```
%$DESIGNWARE_HOME/bin/dw_vip_setup -path design_dir -a(dd) -model_list <input_file_containing_models_one_per_line> -svtb
```

For example, `%$DESIGNWARE_HOME/bin/dw_vip_setup -path design_dir -a -model_list <filelist> -svtb`

cat file list:

```
mipi_i3c_master_agent_svt
mipi_i3c_slave_agent_svt
```

After running the above command, the model files are installed at the following location:

`<design_dir>/include and`

`<design_dir>/src`

Note
You need to specify the pointer to these installed directories on Simulator analyze or compile-options.

2.6 Licensing Information

The I3C VIP uses the Synopsys Common Licensing (SCL) software to control its usage. You can find general SCL information from the following link,

http://www.synopsys.com/keys

The I3C VIP uses a licensing mechanism that is enabled by the following license features:

- VIP-I3C-SVT

Only one license is consumed per simulation session, irrespective of how many VIP products are instantiated in the design.

The licensing key must reside in the files that are indicated by specific environment variables. For information about setting these licensing environment variables, see "Environment Variable and Path Settings" on page 19.

This section consists of the following sub-sections:

- "Controlling License Usage" on page 18
- "License Polling" on page 18
- "Simulation License Suspension" on page 18

2.6.1 Controlling License Usage

You can control which license is used, using the DW_LICENSE_OVERRIDE environment variable, as follows:

- To use only the VIP-I3C-SVT license, set DW_LICENSE_OVERRIDE to VIP-I3C-SVT If DW_LICENSE_OVERRIDE is set to a value and the corresponding feature is not available, a license error message is issued.

2.6.2 License Polling

If you request a license and none are available, License Polling allows your request to exist until a license is available instead of exiting immediately. To control License Polling, use the DW_WAIT_LICENSE environment variable in the following way:

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 Simulation License Suspension

All the Verification IP products support License Suspension. The simulators that support License Suspension allows the model to check-in its license token while the simulator is suspended and then checkout the license token when the simulation is resumed.

Note
This capability is simulator-specific; All simulators do not support license check-in during License Suspension.

2.7 Environment Variable and Path Settings

The following are environment variables and path settings required by the I3C VIP verification models:

- `DESIGNWARE_HOME`: The absolute path to where the VIP is installed.
- `DW_LICENSE_FILE` - The absolute path to file that contains the license keys for the VIP product software or the port@host reference to this file.
- `SNPSLMD_LICENSE_FILE`: The absolute path to file(s) that contains the license keys for Synopsys software (VIP and/or other Synopsys Software tools) or the port@host reference to this file.

Note
For faster license checkout of Synopsys VIP software please ensure to place the desired license files at the front of the list of arguments to `SNPSLMD_LICENSE_FILE`.

- LM_LICENSE_FILE: The absolute path to a file that contains the license keys for both Synopsys software and/or your third-party tools.

Note
The Synopsys VIP License can be set in either of the 3 license variables mentioned above with the order of precedence for checking the variables being:

- DW_LICENSE_FILE -> SNPSLMD_LICENSE_FILE -> LM_LICENSE_FILE, but also note If DW_LICENSE_FILE environment variable is enabled, VIP will ignore SNPSLMD_LICENSE_FILE and LM_LICENSE_FILE settings.

Hence to get the most efficient Synopsys VIP license checkout performance, set the DW_LICENSE_FILE with only the License servers which contain Synopsys VIP licenses. Also, include the absolute path to the third party executable in your PATH variable.

2.8 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
```

2.9 Integrating an I3C Verification IP into Your Testbench

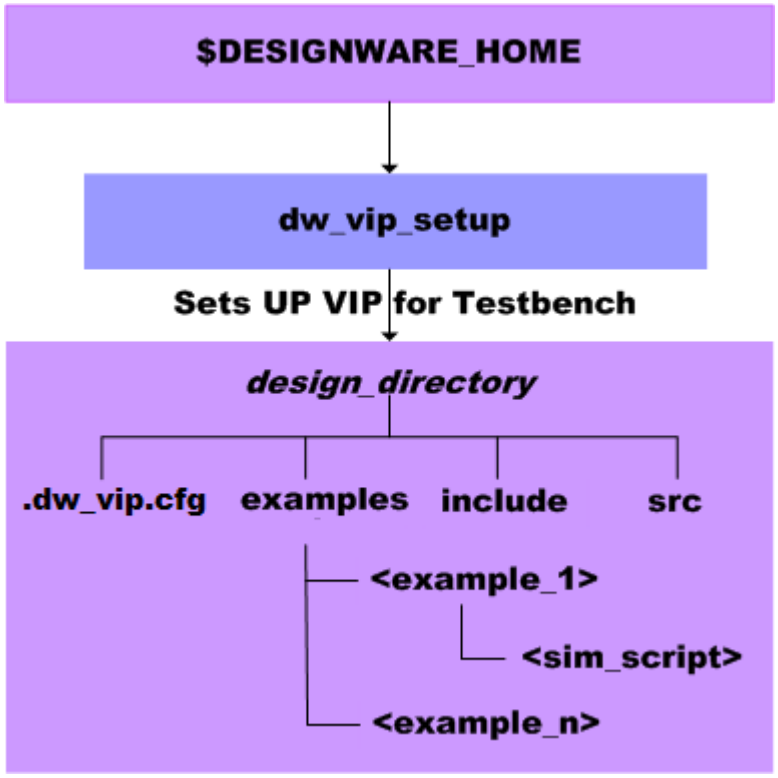
After installing VIP, use the following procedures to set up VIP for use in testbenches:

- "Creating a Testbench Design Directory" on page 19
- "The dw_vip_setup Utility" on page 22

2.9.1 Creating a Testbench Design Directory

Figure 2-2 shows this process and the contents of a design directory.

Figure 2-2Design Directory Created by dw_vip_setup



A design directory contains the following sub-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 the files required for model, suite, and system testbenches.

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

src — The 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 the VIP models used in the testbench. The `dw_vip_setup` utility reads this file to rebuild or recreate a design setup.

Note Do not modify this file because `dw_vip_setup` depends on the original content.

2.9.2 Running the Example with +incdir+

In the current setup, you install the VIP under `DESIGNWARE_HOME` followed by creation of a design

directory which contains the versioned VIP files. With every newer version of the already installed VIP requires the design directory to be updated. This results in:

- Consumption of additional disk space
- Increased complexity to apply patches

The new alternative approach of directly pulling in all the files from DESIGNWARE_HOME eliminates the need for design directory creation. VIP version control is now in the command line invocation.

The following code snippet shows how to run the basic example from a script:

```
cd <testbench_dir>/examples/sverilog/mipi_i3c_svt/tb_mipi_i3c_svt_uvm_basic_sys/
// To run the example using the generated run script with +incdir+
./run_mipi_i3c_svt_uvm_basic_sys -verbose -incdir <testname> vcsvlog
```

For example, the following compile log snippet shows the paths and defines set by the new flow to use VIP files right out of DESIGNWARE_HOME instead of design_dir.

```
vcs -l ./logs/compile.log -q -Mdir=./output/csrc
+define+DESIGNWARE_INCDIR=<DESIGNWARE_HOME> \
+define+SVT_LOADER_UTIL_ENABLE_DWHOME_INCDIRS
+incdir+<DESIGNWARE_HOME>/vip/svt/mipi_i3c_svt/<vip_version>/sverilog/include \
-ntb_opts uvm -full64 -sverilog +define+UVM_DISABLE_AUTO_ITEM_RECORDING
+define+UVM_PACKER_MAX_BYTES=1500000 \
+define+UVM_NO_DEPRECATED -timescale=100ps/100ps \
+lint=none +define+SVT_UVM_TECHNOLOGY +define+SYNOPSYS_SV
+incdir+<testbench_dir>/examples/sverilog/ethernet_svt/tb_mipi_i3c_svt_uvm_basic_sys/.
                        \
+incdir+<testbench_dir>/examples/sverilog/ethernet_svt/tb_mipi_i3c_svt_uvm_basic_sys/..
                        /
../env \
+incdir+<testbench_dir>/examples/sverilog/ethernet_svt/tb_mipi_i3c_svt_uvm_basic_sys/..
                        /
env \
+incdir+<testbench_dir>/examples/sverilog/ethernet_svt/tb_mipi_i3c_svt_uvm_basic_sys/
env \
+incdir+<testbench_dir>/examples/sverilog/ethernet_svt/tb_mipi_i3c_svt_uvm_basic_sys/
dut \
+incdir+<testbench_dir>/examples/sverilog/ethernet_svt/tb_mipi_i3c_svt_uvm_basic_sys/
hdl_interconnect \
+incdir+<testbench_dir>/examples/sverilog/ethernet_svt/tb_mipi_i3c_svt_uvm_basic_sys/
lib \
+incdir+<testbench_dir>/examples/sverilog/ethernet_svt/tb_mipi_i3c_svt_uvm_basic_sys/
tests \
-o ./output/simvcssvlog -f top_files -f hdl_files
```

Note
For VIPs with dependency, include the +incdir+ for each dependent VIP.

2.9.3 Getting Help on Example Run/make Scripts

1. Invoke the run script with no switches, as in:

```
usage: run_i3c_svt_uvm_basic_sys, run_i3c_svt_uvm_passive_sys
[-32][-incdir][-verbose] [-debug_opts] [-waves] [-clean] [-nobuild] [-buildonly]
[-norun] [-pa] <test name> <simulator>
where   <test name> is one of: the tests included in tests folder of the test
bench
<simulator> is one of: vcsvlog mtivlog ncvlog
-32          forces 32-bit mode on 64-bit machines
-incdir      use DESIGNWARE_HOME include files instead of design directory
-verbose     enable verbose mode during compilation
-debug_opts  enable debug mode for VIP technologies that support this option
-waves      [fsdb|verdi|dump] enables waves dump and optionally opens viewer (VCS
only)
-seed        run simulation with specified seed value
-clean       clean simulator generated files
-nobuild     skip simulator compilation
-buildonly   exit after simulator build
-norun       only echo commands (do not execute)
-pa          invoke Verdi after execution
```

2. Invoke the make file with help switch as in:

```
gmake help
Usage: gmake USE_SIMULATOR=<simulator> [VERBOSE=1] [DEBUG=1] [FORCE_32BIT=1]
[WAVES=fsdb|verdi|dump] [NOBUILD=1] [PA=1] [<test name> ...]
Valid simulators are: vcsvlog mtivlog ncvlog
Valid scenarios are: tests included in tests folder of the test bench
```

Note
You must have PA installed if you use the -pa or PA=1 switches.

2.9.4 The dw_vip_setup Utility

The `dw_vip_setup` utility provides the following features:

- Adds, removes, or updates 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
- Supports Protocol Analyzer (PA)

- Supports the FSDB wave format

2.9.4.1 Setting Environment Variables

Before running `dw_vip_setup`, the `DESIGNWARE_HOME` environment must point to the location where the VIP is installed.

2.9.4.2 The dw_vip_setup Command

From the command prompt, invoke `dw_vip_setup`. The `dw_vip_setup` command checks command-line argument syntax and makes sure that the requested input files exist. The general form of the command is as follows:

```
% 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 the `dw_vip_setup` operation.

Table 2-1 lists the switches and their applicable sub-switches.

Table 2-1 Setup Program Switch Descriptions

Switch	Description
-a[dd] (*model* [**-v**[ersion] *version*]) …	Adds the specified model or models to the specified design directory or current working directory. If you do not specify a version, the latest version is assumed. The model names are as follows: • i3c_master_agent_svt • i3c_slave_agent_svt The -add switch makes dw_vip_setup to build suite libraries from the same suite as the specified models, and to copy the other necessary files from $DESIGNWARE_HOME.
-r[emove] *model*	Removes all versions of the specified model or models from the design. The dw_vip_setup program does not attempt to remove any include files used solely by the specified model or models. The model names are as follows: • i3c_master_agent_svt • i3c_slave_agent_svt

Table 2-1 Setup Program Switch Descriptions (Continued)

Switch	Description
-u[pdate] (*model* [**-v**[ersion] *version*]) …	Updates to the specified model version for the specified model or models. The dw_vip_setup script updates to the latest models when you do not specify a version. The model names are as follows: • i3c_master_agent_svt • i3c_slave_agent_svt The -update switch causes dw_vip_setup to build suite libraries from the same suite as the specified models, and to copy the other necessary files from $DESIGNWARE_HOME.
-e[xample] {*scenario* | *model*/*scenario*} [**-v**[ersion] *version*]	The dw_vip_setup script configures a testbench example for a single model or a system testbench for a group of models. The program creates a simulator run program for all the supported simulators. If you specify a scenario (or system) example testbench, the models needed for the testbench are included automatically and do not need to be specified in the command. **Note:** Use the -info switch to list all the available system examples.
-ntb	Not supported.
-svtb	Use this switch to set up models and example testbenches for SystemVerilog UVM. The resulting design directory is streamlined and can only be used in SystemVerilog simulations.
-c[lean] {*scenario* | *model*/*scenario*}	Cleans the specified scenario/testbench in either the design directory (as specified by the -path switch) or the current working directory. This switch deletes all files in the specified directory, then restores all Synopsys-created files to their original contents.
-i/nfo design | home[:<product>[:<version>[:<methodology>]]]	Generate an informational report on a design directory or VIP installation. `design`: If the '`-info design`' switch is specified, the tool displays product and version content within the specified design directory to standard output. This output can be captured and used as a modellist file for input to this tool to create another design directory with the same content. `home`: If the '`-info home`' switch is specified, the tool displays product, version, and example content within the VIP installation to standard output. Optional filter fields can also be specified such as <product>, <version>, and <methodology> delimited by colons (:). An error will be reported if a nonexistent or invalid filter field is specified. Valid methodology names include: OVM, RVM, UVM, VMM and VLOG.
-h[elp]	Returns a list of valid dw_vip_setup switches and the correct syntax.

Table 2-1 Setup Program Switch Descriptions (Continued)

Switch	Description
model	The i3c VIP models are as follows: • i3c_master_agent_svt • i3c_slave_agent_svt The *model* argument defines the model or models that dw_vip_setup acts upon. This argument is not needed with the -info or -help switches. All switches that require the *model* argument may also use a model list. You may specify a version for each listed *model*, using the -version option. If omitted, dw_vip_setup uses the latest version. The -update switch ignores *model* version information.
-m[odel_list] *filename*	Specifies a file name, which contains a list of suite names to be added, updated, or removed from the design directory. This switch is valid during the following switch operations; for example, -add, -update, or -remove. The -m/odel_list switch displays one model name per line and each model includes a version selector. The default version is the latest. This switch is optional, but the filename argument is required whenever mentioned. Lines in the file starting with the pound symbol (#) are ignored. *model_name* [**-v** *version*] –or– **# Comments**
-s/uite_list <filename>	Specifies a file name, which contains a list of suite names to be added, updated, or removed from the design directory. This switch is valid during the following switch operations; for example, -add, -update, or -remove. The -s/uite_list switch displays one suite name per line and each suite includes a version selector. The default version is the latest. This switch is optional, but the filename argument is required whenever mentioned. Lines in the file starting with the pound symbol (#) are ignored.
-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.
-pa	Enables the run scripts and Makefiles generated by dw_vip_setup to support PA. If this switch is enabled, and the testbench example produces XML files, PA will be launched and the XML files will be read at the end of the example execution. For run scripts, specify `-pa`. For Makefiles, specify `-pa = 1`.
-waves	Enables the run scripts and Makefiles generated by dw_vip_setup to support the `fsdb` waves option . To support this capability, the testbench example must generate an FSDB file when compiled with the WAVES Verilog macro set to `fsdb`, that is, `+define+WAVES=\"fsdb\"`. If a .fsdb file is generated by the example, the Verdi nWave viewer will be launched. For run scripts, specify `-waves fsdb`. For Makefiles, specify `WAVES=fsdb`.

Table 2-1 Setup Program Switch Descriptions (Continued)

Switch	Description
-doc	Creates a doc directory in the specified design directory which is populated with symbolic links to the `DESIGNWARE_HOME` installation for documents related to the given model or example being added or updated.
-methodology <name>	When specified with -doc, only documents associated with the specified methodology name are added to the design directory. Valid methodology names include: OVM, RVM, UVM, VMM, and VLOG.
-copy	When specified with -doc, documents are copied into the design directory, not linked.
-simulator <vendor>	When used with the `-example` switch, only simulator flows associated with the specified vendor are supported with the generated run script and Makefile. **Note**: Currently the vendors VCS, MTI, and NCV are supported.

The `dw_vip_setup` utility treats all lines beginning with "#" as comments.

3 General Concepts

This chapter introduces the I3C VIP within a UVM environment and describes the data objects and components that comprise the VIP. This chapter assumes that you are familiar with SystemVerilog.

This chapter consists of the following sections:

3.1 Introduction to UVM

UVM is an object-oriented approach. It provides a blueprint for building testbenches using the constrained random verification. In addition, the resulting structure supports Directed testing. This chapter describes the data objects that support higher structures which comprise the I3C VIP.

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

- For the IEEE SystemVerilog standard, refer the following:
  - IEEE Standard for SystemVerilog - Unified Hardware Design, Specification, and Verification Language
- For essential reference guides describing UVM as it is represented in SystemVerilog, see http://www.accellera.org/.

3.2 I3C VIP Components

This section describes the following I3C VIP components:

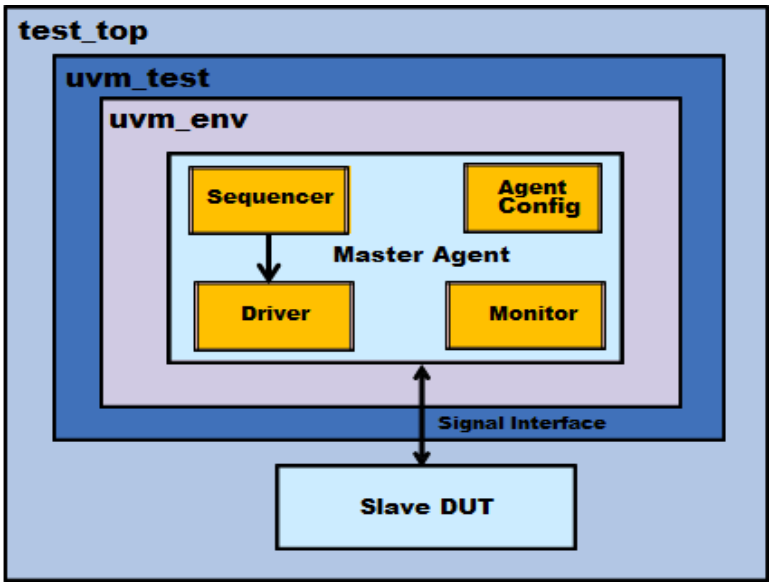
3.2.1 Master Agent

Master Agent encapsulates Master sequencer, Master driver, and Monitor. The agent can be configured to operate either in the active mode or in the passive mode. You can run user-defined sequences on Master sequencer. Master agent is configured using the agent configuration, which is available in the System configuration.

Within Master agent, Master Driver gets transactions from Master Sequencer. Master Driver then drives the I3C transactions on the I3C port. After an I3C transaction on the bus is complete, the completed sequence item is provided to the analysis port of Monitor for use by the testbench.

Figures 3-1 shows the usage with standalone master agent.

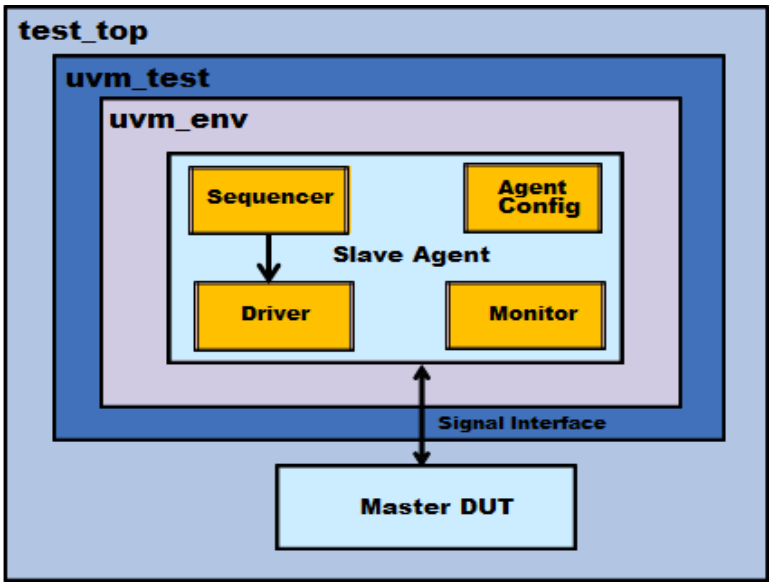
Figure 3-1Usage With Standalone Master Agent



3.2.2 Slave Agent

Slave Agent encapsulates Slave Sequencer, Slave Driver, and Port Monitor. Slave Agent is configured using the agent configuration, which is available in the System configuration. The response from Slave Agent can be configured by executing slave transactions on Slave Agent. For the details on the types of responses that can be configured by slave transactions, see the Transaction Objects section.

Figure 3-2Usage With Standalone Slave Agent



3.2.3 System Environment

The I3C system environment encapsulates master agents, slave agents, the system configuration, and the system sequencer. The system environment can be easily configured to have up to 3 slave agents and 3 master agents. This section discusses the following sub-sections:

- "System Configuration" on page 29
- "System Sequencer" on page 30

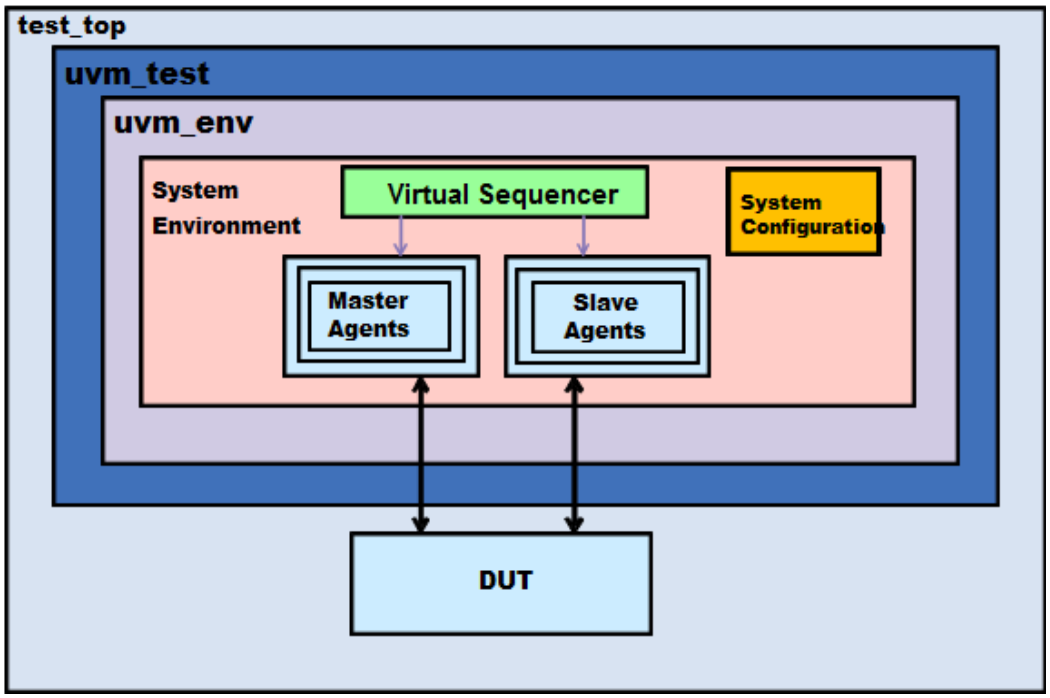
For more details about master agents and slave agents, see the "Master Agent" on page 28 and "Slave Agent" on page 28 respectively.

3.2.3.1 System Configuration

The number of configured master and slave agents is based on the system configuration. In the build phase, the system environment builds master agents and slave agents. After master and slave agents are built, they are configured by the system environment using the agent configuration information available in the system configuration.

Figures 3-3 shows the usage with the system environment.

Figure 3-3Usage With System Environment



3.2.3.2 System Sequencer

I3C System sequencer is a virtual sequencer with references to each Master and Slave sequencers in the system. The sequencer is created in the build phase of the system environment. The System configuration is provided to System Sequencer. System Sequencer is used to synchronize between the sequencers in Master and Slave agents.

3.3 I3C VIP User Interface

The following sections give an overview of the user interface into the I3C VIP:

- "Configuration Objects" on page 30
- "Transaction Objects" on page 31
- "Analysis Ports" on page 32
- "Interfaces and Modports" on page 33

3.3.1 Configuration Objects

Configuration data objects convey the system-level and port-level testbench configuration. The configuration of agents is done in the build() phase of an environment or a testcase. The configuration data objects contain built-in constraints, which come into effect when the configuration objects are randomized. The I3C VIP defines the following configuration classes:

System Configuration (svt_mipi_i3c_system_configuration)

The System configuration class contains the configuration information, which is applicable across the entire system. The system-level configuration parameters can be specified through this class. The system

configuration needs to be provided to the system environment from the environment or the testcase. The system configuration mainly specifies the following:

- The number of master and slave agents in the system environment
- Configurations for master and slave agents
- Virtual top-level I3C interface

Agent Configuration (svt_mipi_i3c_agent_configuration)

The agent configuration class contains the configuration information, which is applicable to individual I2C master or slave agents in the system environment. Some of the important information provided by the agent configuration class is as follows:

- Active or Passive mode of the Master/Slave port agent
- Enable or disable protocol checks
- The agent configuration contains the virtual interface for the port
- Timing parameter values
- Agent id
- Speed mode
- Enable or disable Monitor log

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

For details on individual members of configuration classes, refer to the I3C VIP class reference HTML documentation:

```
$DESIGNWARE_HOME/vip/svt/mipi_i3c_svt/latest/doc/mipi_i3c_svt_uvm_class_reference/html/index.html
```

3.3.2 Transaction Objects

Transaction objects, which extends from the `uvm_sequence_item` base class, define a unit of the I3C protocol information that is passed across the bus. The attributes of transaction objects are public and are accessed directly for setting and getting values. Most transaction attributes can be randomized. Transaction objects represent the desired activity to be simulated on the bus, or the actual bus activity that was monitored. I3C transaction data objects store data content and the protocol execution information for I3C transactions in terms of the timing details of transactions.

These data objects extend from the `uvm_sequence_item` base class and implement all methods specified by UVM for that class.

I3C transaction data objects are used to perform the following:

- Generate random and directed stimulus
- Report observed transactions
- 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. It provides the following two set of constraints:

The valid_ranges constraints limit generated values to those acceptable to drivers. These constraints ensure basic VIP operation and should never be disabled.

The reasonable_* constraints, which can be disabled individually or as a block, limit the simulation by doing the following:

- Enforcing the protocol. These constraints are typically enabled unless errors are injected in the simulation.
- Setting simulation boundaries. Disabling these constraints may slow the simulation and introduce system memory issues.

The VIP supports extending transaction data classes for customizing randomization constraints. This allows you to disable some reasonable_* constraints and replace them with constraints appropriate to your system. Individual reasonable_* constraints map to independent fields, each of which can be disabled. The class provides the `reasonable_constraint_mode()` method to enable or disable the blocks of reasonable_* constraints. The I3C VIP defines the following transaction classes:

- I3C Base transaction (`svt_i2c_transaction`)

  This is the base transaction type, which contains all the physical attributes of a transaction, such as cmd_type (write/read), slave address, and data, and so on.

- I3C Master transaction (`svt_mipi_i3c_master_transaction`)

  The master transaction class extends from the I2C master transaction class, namely, svt_i2c_master_transaction and has properties to specify the following:

  - If Master has to dump the current transaction or retry if it loses arbitration.
  - Master has to arbitrate for the current transaction. Master waits till a START condition is generated by other master on the bus.
  - Time interval between the completion of a transaction to the start of next transaction.
  - Number of times Master will try to complete current transaction in case of not getting an not getting an ACK from slave or when it loses arbitration.
  - If Master has to abort the current transaction or retry in case of NACK from Slave.
  - If Master has to generate the repeated START (Sr) or a STOP (P)

- I3C Slave transaction (`svt_mipi_i3c_slave_transaction`)

  The slave transaction class extends from the I2C slave transaction lass, namely `svt_i2c_slave_transaction`.

For details on the individual members of transaction classes, refer to the I3C VIP class reference HTML documentation:

`$DESIGNWARE_HOME/vip/svt/mipi_i3c_svt/doc/mipi_i3c_svt_uvm_class_reference/html/index.html`

3.3.3 Analysis Ports

In active as well as passive mode of operation of the master or slave agent, you can use the analysis port for connecting to the scoreboard, or for any other purpose, where a transaction object, that is, `svt_i2c_transaction` for the completed transaction is required.

The monitor in the agent provides an analysis port. At the end of a transaction, the monitor within the agent provides the completed `svt_mipi_i3c_master_transaction` and `svt_mipi_i3c_slave_transaction` object from its analysis port.

The analysis ports for transmit and receive ports are as follows:

- `svt_i2c_master_monitor::tx_xact_observed_port`
- `svt_i2c_master_monitor::rx_xact_observed_port`
- `svt_i2c_slave_monitor::tx_xact_observed_port`
- `svt_i2c_slave_monitor::rx_xact_observed_port`

3.3.4 Interfaces and Modports

SystemVerilog models signal connections using interfaces and modports. Interfaces define the set of signals, which make up a port connection. Modports define the collection of signals for a given port, the direction of the signals, and the clock with respect to which these signals are driven and sampled.

The I3C VIP provides the SystemVerilog interface, which can be used to connect the VIP to the DUT.

Synopsys defines the top-level interface, `svt_mipi_i3c_if`. The top-level interface is contained in the system configuration class. The top-level interface is specified to the system configuration class using the `svt_mipi_i3c_system_configuration::set_if` method. This interface is also passed to Master and Slave agents.

The port interface, `svt_mipi_i3c_if`, contains the following modports, which you should use to connect the VIP to the DUT:

- `svt_mipi_i3c_I3C_monitor_modport`: This modport is used by monitor component inside the master and slave agents.

3.4 Sequence Collection

The I3C VIP provides a collection of I3C master and slave sequences. These sequences can be registered with the master and slave sequencers within master and slave agents respectively to generate different types of I3C scenarios.

For a list of all master and slave sequences, refer to the following I3C VIP class reference HTML documentation:

`$DESIGNWARE_HOME/vip/svt/mipi_i3c_svt/latest/doc/mipi_i3c_svt_uvm_class_reference/html/sequencepages.html`

3.5 Protocol Checks

VIP provides set of protocol checks as per I3C Specification V0.7 r1. Detailed list of protocol checks has been mentioned in class reference guide present at the following location:

`$DESIGNWARE_HOME/vip/svt/mipi_i3c_svt/latest/doc/mipi_i3c_svt_uvm_class_reference/html/protocolChecks.html`

4 Verification Topologies

The chapter consists of the following chapters that show you from a high-level how the I3C VIP can be used to test Master, Slave, or Interconnect DUT:

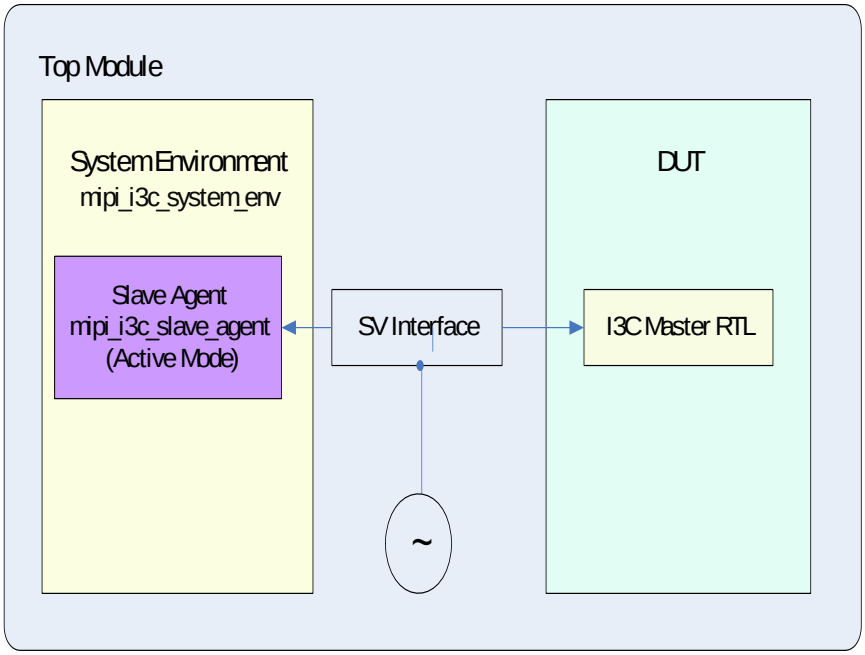
4.1 Master DUT and Slave VIP

Scenario: The VIP is required to verify the I3C Master DUT.

Testbench Setup: Configure the I3C System configuration to have one slave agent in an active mode. The active slave agent responds to the transactions generated by Master DUT. The Slave agent also performs passive functions, such as Protocol checking, Coverage generation, and transaction logging.

Figures 4-1 shows Master DUT and Slave VIP: Usage with the system environment.

Figure 4-1Master DUT and Slave VIP - Usage With the System Environment



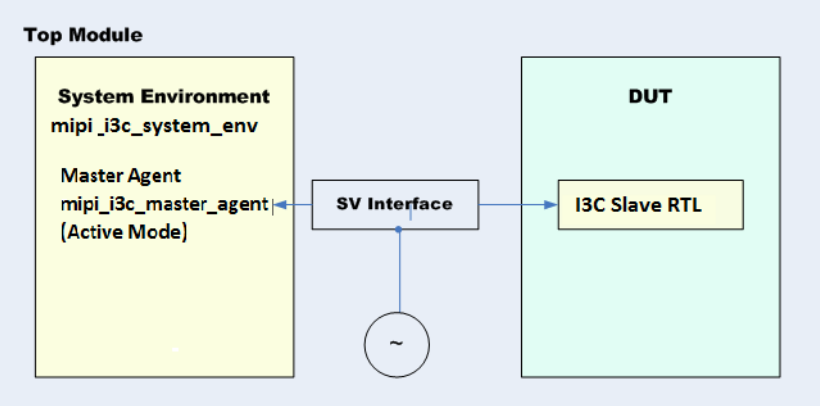
4.2 Slave DUT and Master VIP

Scenario: The VIP is required to verify the I3C Slave DUT.

Testbench Setup: Configure the I3C System configuration to have one master agent in an active mode.

Figures 4-2 shows Slave DUT and Master VIP: Usage with the system environment.

Figure 4-2Slave DUT and Master VIP - Usage With the System Environment



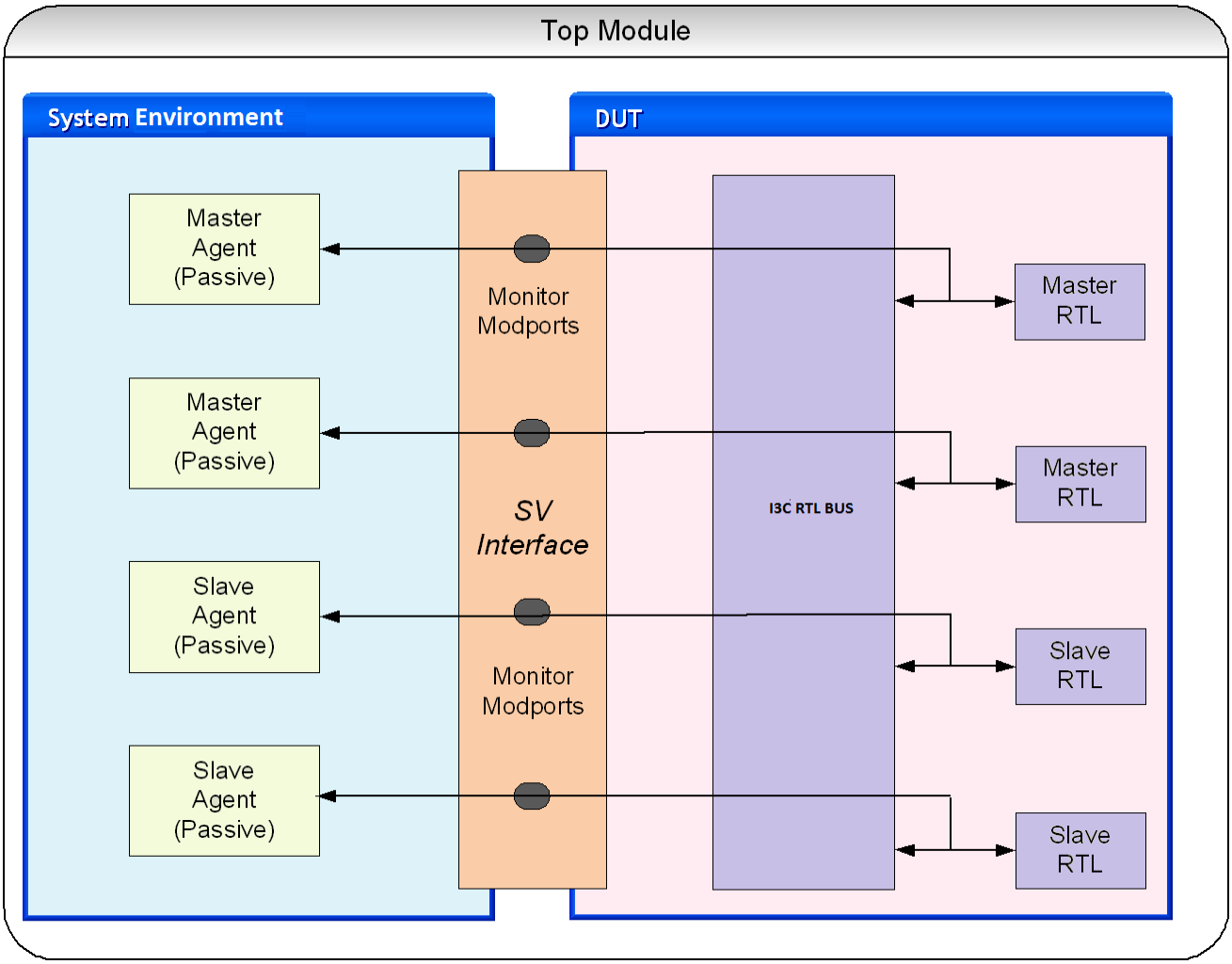
4.3 System DUT and Passive VIP

Scenario: The DUT is an I3C system with multiple I3C masters, slaves, and interconnect. The VIP is required to monitor the DUT.

Testbench Setup: Assuming that the I3C System has M masters and S slaves, configure the I3C System environment to have M master agents and S slave agents, in the passive mode. The passive master and slave agents perform passive functions, such as Protocol checking, coverage generation, and transaction logging.

Figures 4-3 show System DUT with Passive VIP.

Figure 4-3System DUT With Passive


Top Module
System Environment
DUT
Master
Agent
(Passive)
Master
Agent
(Passive)
Slave
Agent
(Passive)
Slave
Agent
(Passive)
Monitor
Modports
SV
Interface
Monitor
Modports
I3C RTL BUS
Master
RTL
Master
RTL
Slave
RTL
Slave
RTL

5
Using I3C Verification IP

This section describes SystemVerilog UVM example testbenches that show the general usage for various applications. Tables 5-1 lists the summary of the examples.

Table 5-1SystemVerilog Example Summary

Example Name	Level	Description
tb_i3c_svt_uvm_basic_sys	Basic	The example consists of the following: • A top-level testbench in SystemVerilog • A dummy DUT in the testbench, which has two I3C interfaces • A UVM verification environment having an I3C system environment configured with single master and slave agent • Two tests illustrating the directed and random transaction generation from master and slave agents A quickstart for this example is available at the following location: `$DESIGNWARE_HOME/vip/svt/i3c_svt/latest/examples/sverilog/tb_i3c_svt_uvm_basic_sys/doc/tb_i3c_svt_uvm_basic_sys/index_basic.html`

6 Usage Notes

- All the CCCs are supported
- Scoreboard features

  Comparison logic available for the following CCCs in scoreboard:
  - ENTDAA
  - SETDASA
  - DEFSLVS
  - BROADCAST SETMWL
  - BROADCAST SETMRL
  - BROADCAST RSTDAA
  - BROADCAST ENTAS0
  - BROADCAST ENTAS1
  - BROADCAST ENTAS2
  - BROADCAST ENTAS3
  - DIRECT RSTDAA
  - DIRECT ENTAS0
  - DIRECT ENTAS1
  - DIRECT ENTAS2
  - DIRECT ENTAS3
  - GETPID
  - GETBCR
  - GETDCR
- Error Exceptions

Note
For DUT to VIP type of test setup, its mandatory to instantiate a passive VIP component having same configurations as passed to DUT.

Exceptions for master transactions and their descriptions are as follows :

Table 6-1 Exceptions for Master Transaction

Exception Macros	Description
PARITY_ERROR	/**<Covers SDR S1,S2,S3*/
HDR_PARITY_ERROR	/**<Used to corrupt parity in DDR packets*/
HDR_SYMBOL_ERROR	/**<Used to corrupt particular ternary word hence corrupting corresponding symbols*/
CRC_ERROR	/**<Used to corrupt CRC*/
PREAMBLE_ERROR	/**<Used to corrupt preamble in DDR packet*/
CCC_CORRUPT_ERROR	/**<Covers SDR M0*/
BRDCAST_ADDR_ERROR	/**<Covers SDR S4(in comb with daa-sr_brdcast_addr),S0*/
NACK_BROADCAST_ADDR_ERROR	/**<Covers SDR M2*/
RD_WR_TXN_CMD_BIT_ERROR	/**<Covers SDR S6,M1*/
GETACCMST_ERROR	/**<Used to address/parity returned by secondary master in GETACCMST ccc*/
CCC_TXN_CMD_BIT_ERROR	/**<Covers SDR S5*/
I3C_CHAR_REG_ERROR	/**<Used to corrupt char registers*/
I3C_ENTDAA_B2B_NACK_ERROR	/**<Used to insert error in ENTDAA by acking dynamic addr >2 times*/
I3C_SDR_HDR_RNW_ERROR	/**<Used to insert error by changing the RnW bit to read*/
I3C_ENTDAA_CCC_END_ERROR	/**<Used to insert error in ENTDAA by not ending it with STOP*/
I3C_GETACC_END_ERROR	/**< Used to insert error in GETACC by not ending it with STOP or SR, while writing sequence don't consider sr_or_p_gen
I3C_DEFSLVS_FRAME_ERROR	/**<Used to corrupt DEFSLVS ccc related address (static/dynamic) , length etc.
I3C_ILLEGAL_FORMAT_CCC_ERROR	/**<Used to insert M0 error. Sec Master will return number of bytes equal to variable "bytes_corrupt_value" for a GET CCC.*/
I3C_SEC_MST_NO_EXIT_PAT_ DURING_RD	/**<Used to corrupt the start of EXIT pattern during HDR-TSP/TSP Read in sec mst.*/
I3C_ILLEGAL_EXIT_PAT_DURING_ SDR	/**<Used to insert EXIT pattern illegally by Current Master during SDR mode [For information, refer Appendix section A.25]*/
I3C_INITIATE_EXIT_PAT_FOR_ UNKNOWN_ADDR	/**<Used to insert initial pattern for an unknown address in HDR-TSP/TSL by sec-mst. For reference example see, testbench: tb_i3c_svt_uvm_basic_sys test: ts.i3c_slv_insert_initial_pattern_for_unknown_addr_test.sv*/

Table 6-1 Exceptions for Master Transaction

Exception Macros	Description
I3C_ILLEGAL_EXIT_PAT_DURING_SDR	Used to insert EXIT pattern illegally by Current Master during SDR mode [SDR- WR, CCC]
MASTER_MISSING_START_ERROR	Used to send the traffic without any Start.
I3C_SETBRGTGT_FRAME_ERROR	Used to corrupt SETBRG ccc count value.
MASTER_DONT_SEND_01_ADDR_IN_P2P_MODE	Used to send address (other than 'h01) dynamic address during Setdasa-P2P mode.
MASTER_MISS_STOP_AFTER_EXIT_PAT	Used to send the exit pattern without stop in HDR-DDR mode. After exit pattern any garbage transaction will follow.
I3C_DDR_INCAPABLE_SEC_MST_ACK_RD_CMD_WORD	Used to ACK (with preamble[0]=1'b0) the command word, even if BCR[5] or hdrcap_modes[0] are disabled.
HDR_DDR_UNEXPECTED_EXIT_RESTART_PAT	Used to send the EXIT/RESTART pattern in HDR-DDR WRITE after preamble where it is not expected.
HDR_DDR_EXPECTED_EXIT_RESTART_PAT_MISS	Used to skip expected EXIT/RESTART pattern in HDR-DDR after either the slave NACK or CRC finished.
I3C_RSTACT_DEF_BYTE_MISMTCH_RD	Used by sec-masters to corrupt the defining byte of GET-RSTACT. Use variable "corrupted_value" to provide corrupted def-byte.
GETCAP_FORMAT2_ILLEGAL_ACK	Used to send ACK response from sec-masters even when they do not support GETCAPS Format-2.
GETCAP_SEND_INVALID_DATA	Used to send corrupted data from sec-masters during GETCAP Format-1 and Format-2 (with defining bytes, SLVCAPS, TESTPAT, MSTCAPS, VSCAPS).
GETSTATUS_FORMAT2_ILLEGAL_ACK	Used to send ACK response from sec-masters even when they do not support GETSTATUS Format-2.
GETMXDS_FORMAT2_ILLEGAL_ACK	Used to send ACK response from sec-masters even when they do not support GETMXDS Format-2.
I3C_RSTACT_ILLEGAL_ACK_FOR_NON_VS_CAP_DEV	Used to send ACK response for defining byte 'h04 and 'h84 for RSTACT CCC from sec-masters even when they are not-VS capable device.
DEFGRPA_DYN_ADDR_LSB_ERROR	Used to corrupt LSB of dynamic address received in DEFGRPA CCC.
DEFGRPA_GRP_ADDR_LSB_ERROR	Used to corrupt LSB of group sent received in DEFGRPA CCC.
DYN_ADDR_NOT_ALLOCATED_WITH_GRP_ADDR	Used to corrupt dynamic address recieved in DEFGRPA CCC
SETGRPA_LSB_ERROR_CB	Used to corrpt LSB of Group address to be allocated through SETGRPA CCC
SEC_MST_SETGRPA_ACK_ANY_GRP_ADDR	Used to corrupt response from NACK to ACK for out of range group address allocations or of the device doesn't support group addressing

Exceptions for slave transactions and their descriptions are as follows :

Table 6-2 Exceptions for Slave Transaction

Exception Macros	Description
NACK_BROADCAST_ADDR_ERROR	Used to send NACK on broadcast header.
CRC_ERROR	Used to corrupt the CRC driven by the slave during DDR read.
HDR_PARITY_ERROR	Used to corrupt PARITY driven by slave during DDR read.
HDR_SYMBOL_ERROR	Used to corrupt particular ternary word(during TSP/TSL) hence corrupting corresponding symbol.
CCC_CORRUPT_ERROR	Used to insert M0 error. The slave will return number of bytes equal to variable *no_of_byte_m0_error* for a GET CCC.
I3C_CHAR_REG_ERROR	Used to corrupt BCR/DCR/PID returned by slave during GETBCR/GETGCR/GETPID.
I3C_SETDASA_ACK_RESP_ERROR	Used to corrupt NACK to ACK for static address during SETDASA for already allocated address.
I3C_SLV_NO_EXIT_PAT_DURING_RD	Used to corrupt the start of EXIT pattern during HDR-TSP/TSP Read in slave.
I3C_INITIATE_EXIT_PAT_FOR _UNKNOWN_ADDR	Used to insert initial pattern for an unknown address in HDR-TSP/TSL by slave. For reference example see, testbench: tb_i3c_svt_uvm_basic_sys test: ts.i3c_slv_insert_initial_pattern_for_unknown_addr_test.sv
I3C_DDR_INCAPABLE_SLV_ACK_RD_ CMD_WORD	Used to ACK (with preamble[0]=1'b0) the command word, even if BCR[5] or hdrcap_modes[0] are disabled.
I3C_SLV_RSTACT_DEF_BYTE_MISMT CH_RD	Used by slaves to corrupt the defining byte of GET-RSTACT. Use variable ‘corrupted_value’ to provide corrupted def-byte.
I3C_ALREADY_DYN_ADDR_SLV_ACK_ ENTDAA_7E_RD	Used to ACK the 7E-> Read in ENTDAA even when dynamic address is assigned to it. Afterwards slave will send PID/BCR/DCR as well.
I3C_BRIDGE_SLV_ACK_ENTDAA_7E_ RD	Used to ACK the 7E-> Read in ENTDAA by slave configured as bridged slave.
I3C_SLV_SEND_HJ_REQ_AFTER_DIS EC_RECVD	Used to send the erroneous Hot-Join request even after the DISEC (with DISHJ) is sent to that slave.
I3C_SLV_SEND_INVALID_HJ_REQ	Used to send the invalid Hot-Join request from slave.
I3C_SLV_VIOLATE_IDLE_AVAL_TIME_ DURING_HJ_REQ	Used to violate the idle time or aval time from slave during HJ request. Use variable violate_idle_or_aval to corrupt idle/aval time.
GETCAP_FORMAT2_ILLEGAL_ACK	Used to send ACK response from slaves even when they do not support GETCAPS Format-2.

Table 6-2 Exceptions for Slave Transaction

Exception Macros	Description
GETCAP_SEND_INVALID_DATA	Used to send corrupted data from sec-masters during GETCAP Format-1 and Format-2 (with defining bytes, SLVCAPS, TESTPAT, MSTCAPS, VSCAPS).
GETSTATUS_FORMAT2_ILLEGAL_ACK	Used to send ACK response from slaves even when they do not support GETSTATUS Format-2.
GETSTATUS_FORMAT2_PURE_SLV_ILLEGAL_ACK	Used to send ACK response from pure slave componenets when GETSTATUS Format-2 with defining byte 'h91 arrives.
GETMXDS_FORMAT2_ILLEGAL_ACK	Used to send ACK response from slaves even when they do not support GETMXDS Format-2.
GETMXDS_FORMAT2_PURE_SLV_ILLEGAL_ACK	Used to send ACK response from pure slave componenets when GETMXDS Format-2 with defining byte 'h91 arrives.
I3C_RSTACT_ILLEGAL_ACK_FOR_NON_VS_CAP_DEV	Used to send ACK response for defining byte 'h04 and 'h84 for RSTACT CCC from slaves even when they are not-VS capable device.
SLV_SETGRPA_ACK_ANY_GRP_ADDR	Used to corrupt response from NACK to ACK for out of range group address allocations or of the device doesn't support group addressing

7 VIP Tools

7.1 Using Native Protocol Analyzer for Debugging

7.1.1 Introduction

This feature enables you to invoke Protocol Analyzer from Verdi GUI. You can synchronize the Verdi wave window, smart log and the source code with the Protocol Analyzer transaction view.

Protocol Analyzer can be enabled in an interactive and post-processing mode. The new features available in Native Protocol Analyzer includes layer based grouping of the transactions, Quick filter, Call stack, horizontal zoom and reverse debug with the interactive support.

7.1.2 Prerequisites

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.`

You can dump the transaction database either by setting the pa_format_type configuration variable or by passing the runtime switch as shown below:

Configuration Variable Setting:

```
/** Configure Master and Slave configurations */
cfg.master_cfg[0].enable_xml_gen_i3c = 1'b1; // Enable XML generation (PA)
for Master
cfg.slave_cfg[0].enable_xml_gen_i3c = 1'b1; // Enable XML generation
(PA) for Slave
```

```
cfg.master_cfg[1].enable_xml_gen_i3c = 1'b1; // Enable XML generation (PA)
for Master
cfg.slave_cfg[1].enable_xml_gen_i3c = 1'b1; // Enable XML generation

(PA) for Slave
where master_cfg and slave_cfg are instances of class ,
svt_mipi_i3c_agent_configuration and cfg is handle of
cust_svt_mipi_i3c_system_configuration which is extended from class
cust_svt_mipi_i3c_system_configuration
cfg.master_cfg[0].pa_format_type_i3c =
svt_xml_writer::format_type_enum'(`SVT_WRITER_FORMAT_FSDB);
cfg.slave_cfg[0].pa_format_type_i3c =
svt_xml_writer::format_type_enum'(`SVT_WRITER_FORMAT_FSDB);
cfg.master_cfg[1].pa_format_type_i3c =
svt_xml_writer::format_type_enum'(`SVT_WRITER_FORMAT_FSDB);
cfg.slave_cfg[1].pa_format_type_i3c =
svt_xml_writer::format_type_enum'(`SVT_WRITER_FORMAT_FSDB);
```

7.1.3 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 shown above through Verdi. The Protocol Analyzer transaction view gets updated during the simulation.

7.1.4 Documentation

The documentation for Protocol Analyzer is available at the following path:

`$VERDI_HOME/doc/Verdi_Transaction_and_Protocol_Debug.pdf.`

7.1.5 Limitations

Interactive support is available only for VCS.

8 Troubleshooting

This chapter provides some useful information that can help you to troubleshoot common issues that you may encounter while using the I3C VIP.

The chapter consists of the following section:

8.1 Enabling Traffic logs

You can enable or disable traffic logs using the `enable_traffic_log` variable, which is defined in the `svt_mipi_i3c_agent_configuration` class. The default value of the variable is 1, which enables traffic logs, by default.

To disable traffic logs, set the value of the related trace variable to 1 in the derived class, which is used in your VIP agent. When the environment is simulated, traffic log files are generated according to the configuration.

The following code setting illustrate how you can enable the traffic log assuming the `master_cfg` handle is of the cust_svt_mipi_i3c_agent_configuration class:

```
master_cfg.enable_traffic_log = 1;
```

To disable traffic logs, set the value of the `enable_traffic_log` variable to 0 in the `cust_svt_mipi_i3c_agent_configuration` class, which is used in your VIP agent. When the environment is simulated, traffic log files are generated according to the configuration.

8.2 Setting Verbosity Levels

You can set the verbosity levels of the VIP debug either in the testbench or as an option during runtime. This section consists of the following sub-sections:

8.2.1 Setting Verbosity in the Testbench

To set the verbosity level in the testbench, use the UVM-specified log levels in the code. The components extend from the `uvm_report_object` method. You can use the `uvm_report_object` method to change the verbosity for the agent.

Consider the following example:

```
vip_agent.set_report_verbosity_level(<level>);
```

Here, the following verbosity levels define all the possible levels:

- UVM_NONE: Prints the report always. The setting of the verbosity level cannot disable it.
- UVM_LOW: Prints the report if the configured verbosity is set to UVM_LOW or above.
- UVM_MEDIUM: Prints the report if the configured verbosity is set to UVM_MEDIUM or above.
- UVM_HIGH: Prints the report if the configured verbosity is set to UVM_HIGH or above.
- UVM_FULL: Prints the report if the configured verbosity is set to UVM_FULL or above.

8.2.2 Setting Verbosity During Runtime

To set the verbosity level during runtime, you can use one of the following methods:

- "Method 1: To Enable the Specified Severity in the VIP, DUT, and Testbench" on page 88
- "Method 2: To Enable the Specified Severity to the Specific Sub-Classes of VIP" on page 88

8.2.2.1 Method 1: To Enable the Specified Severity in the VIP, DUT, and Testbench

The example for VCS is as follows:

```
vcs <other runtime options> +UVM_VERBOSITY=UVM_MEDIUM
```

8.2.2.2 Method 2: To Enable the Specified Severity to the Specific Sub-Classes of VIP

The example for this method is as follows:

```
+uvm_set_verbosity=component_name,id,verbosity,phase_name,optional_time
```

A
Reporting Problems

A.1 Introduction

This chapter outlines the process for working through and reporting VIP 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. This section outlines the process for working through and reporting problems. It shows how to use Debug Automation to enable all the debug capabilities of any VIP. In addition, the VIP provides a case submittal tool to help you pack and send all pertinent debug information to Synopsys Support.

A.2 Debug Automation

Every Synopsys model contains a feature called "debug automation". It is enabled through *svt_debug_opts* plusarg. The Debug Automation feature allows you to enable all relevant debug information. The following are critical features of debug automation:

- Enabled by the use of a command line run-time plusarg.
- Can be enabled on individual VIP instances or multiple instances using regular expressions.
- Enables debug or verbose message verbosity:
  - The timing window for message verbosity modification can be controlled by supplying `start_time` and `end_time`.
- Enables at one time any, or all, standard debug features of the VIP:
  - Transaction Trace File generation
  - Transaction Reporting enabled in the transcript
  - PA database generation enabled
  - Debug Port enabled
  - Optionally, generates a file name *svt_model_out.fsdb* when Verdi libraries are available

When the Debug feature is enabled, then all VIP instances that are enabled for debug will have their messages routed to a file named *svt_debug.transcript*.

A.3 Enabling and Specifying Debug Automation Features

Debug Automation is enabled through the use of a run-time plusarg named *+svt_debug_opts. T*his plusarg accepts an optional string-based specification to control various aspects Debug Automation. If this

command control specification is not supplied, then the feature will default to being enabled on all VIP instances with the default options listed as follows:

Note the following about the plusarg:

- The command control string is a comma separated string that is split into the multiple fields.
- All fields are optional and can be supplied in any order.

The command control string uses the following format (white space is disallowed):

```
inst:<inst>,type:<string>,feature:<string>,start_time:<longint>,end_time:<longint>,verbosity:<string>
```

The following table explains each control string:

Table A-1 Control Strings for Debug Automation plusarg

Field	Description
inst	Identifies the VIP instance to apply the debug automation features. Regular expressions can be used to identify multiple VIP instances. If this value is not supplied, and if a type value is not supplied, then the debug automation feature will be enabled on all VIP instances.
type	Identifies a class type to apply the debug automation features. When this value is supplied then debug automation will be enabled for all instances of this class type.
feature	Identifies a sub-feature that can be defined by VIP designers to identify smaller grouping of functionality that is specific to that title. The definition and implementation of this field is left to VIP designers, and by default it has no effect on the debug automation feature. (Specific to VIP titles)
start_time	Identifies when the debug verbosity settings will be applied. The time must be supplied in terms of the timescale that the VIP is compiled. If this value is not supplied, then the verbosity settings will be applied at time zero.
end_time	Identifies when the debug verbosity settings will be removed. The time must be supplied in terms of the timescale that the VIP is compiled. If this value is not supplied, then the debug verbosity remains in effect until the end of the simulation.
verbosity	Message verbosity setting that is applied at the `start_time`. Two values are accepted in all methodologies: DEBUG and VERBOSE. UVM and OVM users can also supply the verbosity that is native to their respective methodologies (UVM_HIGH/UVM_FULL and OVM_HIGH/OVM_FULL). If this value is not supplied then the verbosity defaults to DEBUG/UVM_HIGH/OVM_HIGH. When this feature is enabled, then all VIP instances that are enabled for debug will have their messages routed to a file named `svt_debug.transcript`.

Examples:

Enable on all VIP instances with default options:

```
+svt_debug_opts
```

Enable on all instances:

- containing the string "endpoint" with a verbosity of UVM_HIGH
- starting at time zero (default) until the end of the simulation (default):

```
+svt_debug_opts=inst:/.*endpoint.*/,verbosity:UVM_HIGH
```

Enable on all instances:

- starting at time 1000 until time 1500:

```
+svt_debug_opts=start_time:1000,end_time:1500,verbosity:VERBOSE
```

Enable debug feature on all instances using default options:

- By setting the macro SVT_DEBUG_OPTS to 1 in the command line, the debug feature is enabled on all instances using default options. The macro will enable the XMLs and Trace files.

```
gmake <testname> SVT_DEBUG_OPTS=1 PA=FSDB
```

Note

The SVT_DEBUG_OPTS option is available through the installed VIP examples, but if required, in customer environments, then a similar feature should be added to their environment.
The PA=FSDB option is available in public examples and is required to enable Verdi libraries, and that when this option is used, then the Debug Opts file will record VIP activity to a file named `svt_model_log.fsdb`.
In addition, the SVT Automated Debug feature will enable waveform generation to an FSDB file, if the Verdi libraries are available. When enabled this feature, it should cause the simulator to dump waveform information only for the VIP interfaces.

When this feature is enabled then all VIP instances that have been enabled for debug will have their messages routed to a file named `svt_debug.transcript`.

A.4 Debug Automation Outputs

The Automated Debug feature generates a *svt_debug.out* file. It records important information about the debug feature itself, and data about the environment that the VIPs are operating in. This file records the following information:

- The compiled timeunit for the SVT package
- The compiled timeunit for each SVT VIP package
- Version information for the SVT library
- Version information for each SVT VIP
- Every SVT VIP instance, and whether the VIP instance has been enabled for debug
- For every SVT VIP enabled for debug, a list of configuration properties that have been modified to enable debug will be listed
- A list of all methodology phases will be recorded, along with the start time for each phase

The following are the output files generated:

- *svt_debug.out*: It records important information about the debug feature itself, and data about the environment that the VIPs are operating. One file is optionally created when this feature is enabled, depending on if the Verdi libraries are available.
- *svt_debug.transcript*: Log files generated by the simulation run.
- *svt_model_log.fsdb*: Contains PA FSDB information (if the VIP supports this), and which contains other recorded activity. The additional information records signal activity associated with the VIP interface, TLM input (through SIPP ports), other TLM output activity, configurations applied to the VIP, and all callback activity (recorded by before and after callback execution).

A.5 FSDB File Generation

To enable FSDB writing capabilities, the simulator compile-time options and environment must be updated to enable this. The steps to enable this are specific to the simulator being used (the {LINUX/LINUX64} label

needs to be replaced based on the platform being used). The ability to write to an FSDB file requires that the user supplies the Verdi dumper libraries when they compile their testbench. If these are not supplied then the VIP will not be enabled to generate the *svt_model_log.fsdb* file.

A.5.1 VCS

The following must be added to the compile-time command:

```
-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.`

A.5.2 Questa

The following must be added to the compile-time command:

```
+define+SVT_FSDB_ENABLE -pli novas_fli.so
```

A.5.3 Incisive

The following must be added to the compile-time command:

```
+define+SVT_FSDB_ENABLE -access +r
```

A.6 Initial Customer Information

Follow these steps when you call the Synopsys Support Center:

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.

Enable the Debug Opts feature. For more information, see the "Debug Automation" on page 89.

A.7 Sending Debug Information to Synopsys

To help you debug testing issues, follow the given instructions to pack all pertinent debug information into one file which you can send to Synopsys (or to other users in your company):

1. Create a description of the issue under investigation. Include the simulation time and bus cycle of the failure, as well as any error or warning messages that are part of the failure.
2. Create a description of your verification environment. Assemble information about your simulation environment, making sure to include:
   - OS type and version
   - Testbench language (SystemVerilog or Verilog)
   - Simulator and version
   - DUT languages (Verilog)
3. Use the VIP case submittal tool to pack a file with the appropriate debug information. It has the following usage syntax:

   ```
   $DESIGNWARE_HOME/bin/snps_vip_debug [-directory <path>]
   ```

The tool will generate a “<username>.<uniqid>.svd” file in the current directory. The following files are packed into a single file:

- FSDB
- HISTL
- MISC
- SLID
- SVTO
- SVTX
- TRACE
- VCD
- VPD
- XML

If any one of the above files are present, then the files will be saved in the "<username>.<uniqid>.svd" in the current directory. The simulation transcript file will not be part of this and it will be saved separately.

The -directory switch can be specified to select an alternate source directory.

4. You will be prompted by the case submittal tool with the option to include additional files within the SVD file. The simulation transcript files cannot be automatically identified and it must be provided during this step.
5. The case submittal tool will display options on how to send the file to Synopsys.

A.8 Limitations

Enabling DEBUG or VERBOSE verbosity is an expensive operation, both in terms of runtime and disk space utilization. The following steps can be used to minimize this cost:

- Only enable the VIP instance necessary for debug. By default, the `+svt_debug_opts` command enables Debug Opts on all instances, but the '`inst`' argument can be used to select a specific instance.
- Use the `start_time` and `end_time` arguments to limit the verbosity changes to the specific time window that needs to be debugged.

A.9 Hot Join Feature

- On receiving hot join request from the hot join device, irrespective of whether it is NACKed or ACKed, the hot join device behaves like any other I3C device without dynamic address. It will sample all the B-CCCs.
- If the Hot Join request is ACKed by the main master, then the hot join device can send the hot join request only after it drops off from the bus.
- If the hot join request is ACKed, it does not imply that ENTDAA follows immediately. ACK is only an indication that master has agreed to do a ENTDAA. It may wait for prolonged period of time before initiating ENTDAA.
- The hot join request is NACKed by the main master:

- If the device is not allocated a dynamic address by the main master, then the hot join device re-initiates the hot join request after Taval. If no START is received within Taval time, then the slave generates a START for sending hot join request.
- After the hot join request is NACKed by the main master, the master can choose to send a broadcast DISEC CCC with DISHJ set. This disables any further hot join requests by the hot join device.
  - The main master can choose to do an ENTDAA. In such a case, the hot join device must participate in the DAA procedure. Since the DAA has been done, the device does not re-initiate hot join after ENEC.
  - If DAA is not done, then the hot join device continues to sample CCCs, but initiates a hot join after the broadcast of ENEC with ENHJ bit set is sent out on the bus.

- If HJ is NACKed and before HJ is retried by the device when S0 or S1 error is detected, then the device will not retry hot join. Once recovery is done using exit pattern, hot join is retried after Taval.
- Old hot join behavior has been included in the `SVT_MIPI_I3C_HJ_OLD_BEHAVIOR` macro.
  As per older implementation, only ENTDAA, B-ENEC,B-DISEC were allowed after HJ ACK. But now the behavior after HJ ACK is sequence controlled.
  Older legacy implementation allowed hot join devices to drop off after RSTDAA, in newer approach drop-off is controlled by trans class variables.

Note

If ACK/NACK has been driven for HJ request and ownership has to be transferred, then these steps are mandatory:

a. Address allocation of Hot-join device.

b. DEFSLVS post address allocation.

For more details on SVDOC description of the following configuration and transaction class variables, see the following:

Master transaction class: arbitrate, `hj_device_drop_off`

Slave transaction class: `slv_arbitrate, hj_device_drop_off`

Configuration class: `hj_drop_off_address`

For more information on working of hot join feature, see `tb_i3c_svt_uvm_basic_sys/tests/ts.entdaa_hj_entdaa_drop_off_hj_nack_hj_entdaa_test.sv.`

A.10 Disabling I2C Slave for TSP/TSL, DDR traffic

A Hook has been added in VIP to support rejection for the TSP/TSL, DDR traffic.

The following steps must be performed to disable TSP/TSL, DDR traffic.

1. Declare environment handle as:
```
i3c_basic_env env;
bit status;
```
2. In the body() of the sequence ,
```
get the handle status = uvm_config_db#(i3c_basic_env)::get(m_sequencer,
get_type_name(), "env", env);
`uvm_info("body", $sformatf("%0s to get env handle form test",  status ? "Able" :
"Unable"),UVM_LOW);
```

3. Before you fire DDR traffic, that is ENTHDR0 command, disable the slave with syntax as follows:
```
env.i3c_system_env.slave[0].driver.common.if_slv.disable_start_stop_detect = 1;
//DISABLE legacy slave.
```
4. Once the HDR traffic is complete, if you want to resume to SDR transactions, then set the `env.i3c_system_env.slave[0].driver.common.if_slv.disable_start_stop_detect` variable to 0. For example,
```
env.i3c_system_env.slave[0].driver.common.if_slv.disable_start_stop_detect = 0;
// enable legacy slave
// FIRE SDR transaction
`uvm_create_on(tr1,p_sequencer.master_sequencer[0]) if (!tr1.randomize() with {
tr1.cmd              == svt_mipi_i3c_master_transaction::READ;
tr1.hdr_cc           == 7'h50;
tr1.arbitrate        == 0;
tr1.sr_or_p_gen   == 0;
})
`uvm_error("Randomization Failure","i3c_hdr_ddr_setdasa_wr_rd_sequence")
else begin
tr1.addr = 'h34;
tr1.set_ccc = 0;
end
`uvm_send(tr1)
```
5. Corresponding to the sequence, in the test case you must pass the environment handle in the sequence as follows:
```
uvm_config_db#(i3c_basic_env)::set(this,"env.i3c_system_env.sequencer.main_mst_
hdr_random_sceanrio_for_abort_sequence", "env", env);
```

A.11 WRITE ABORT Feature

The WRITE ABORT feature can be exercised as follows:

```
// STARTING WITH HDR TRAFFIC , so rejecting DDR traffic from i2c slave.
   env.i3c_system_env.slave[0].driver.common.if_slv.disable_start_stop_detect = 1;

   `uvm_create_on(tr,p_sequencer.master_sequencer[0])
   tr.reasonable_sr_or_p_gen.constraint_mode(0);
   if (!tr.randomize() with {
                              tr.cmd              ==
svt_mipi_i3c_master_transaction::WRITE;
                              tr.arbitrate        == 0;
                              tr.sr_or_p_gen      == 0;
                             })
                             `uvm_error("Randomization
Failure","main_mst_hdr_wr_abort_with_exit_pattern_sequence")
   else begin
     tr.addr = 'h7e;
     tr.ccc = `SVT_MIPI_I3C_ENTHDR0;
   end
```

```
   `uvm_send(tr)

`uvm_info("body", "Entered HDR mode::performing WRITE..", UVM_DEBUG)
   `uvm_create_on(tr,p_sequencer.master_sequencer[0])
   if (!tr.randomize() with {
                              tr.cmd              ==
svt_mipi_i3c_master_transaction::WRITE;
                              tr.hdr_cc           == 7'h10;
                              tr.data.size()      == 5;
                              tr.arbitrate        == 0;
                              tr.broadcast_header == 0;
                             })
                              `uvm_error("Randomization
Failure","main_mst_hdr_wr_abort_with_exit_pattern_sequence")
   else begin
     tr.addr = 'h35;
     tr.set_ccc = 0;
     tr.sr_or_p_gen = 0;
   end
`uvm_info("body", " Slave setting data size for READ Response HDR_DDR mode..",
UVM_DEBUG)
   `uvm_create_on(slv_tr,p_sequencer.slave_sequencer[1])
   if (!slv_tr.randomize() with {
                               slv_tr.set_ibi       == 0;
                              })
                              `uvm_error("Randomization
Failure","main_mst_hdr_wr_abort_with_exit_pattern_sequence")
   else
      slv_tr.nack_data = 2;
fork
   `uvm_send(tr)
   `uvm_send(slv_tr)
join

//*************************************************************************
// Scenario 2
//*************************************************************************
  `uvm_create_on(tr,p_sequencer.master_sequencer[0])
   tr.reasonable_sr_or_p_gen.constraint_mode(0);
   if (!tr.randomize() with {
                              tr.cmd              ==
svt_mipi_i3c_master_transaction::WRITE;
```

```
                                tr.arbitrate        == 0;
                                tr.sr_or_p_gen      == 0;
                               })
                               `uvm_error("Randomization
Failure","main_mst_hdr_wr_abort_with_exit_pattern_sequence")
   else begin
     tr.addr = 'h7e;
     tr.ccc = `SVT_MIPI_I3C_ENTHDR0;
   end
   `uvm_send(tr)

   `uvm_create_on(tr,p_sequencer.master_sequencer[0])
   if (!tr.randomize() with {
                                tr.cmd              ==
svt_mipi_i3c_master_transaction::WRITE;
                                tr.hdr_cc           == 7'h10;
                                tr.data.size()      == 4;
                                tr.arbitrate        == 0;
                                tr.broadcast_header == 0;
                               })
                               `uvm_error("Randomization
Failure","main_mst_hdr_wr_abort_with_exit_pattern_sequence")
   else begin
     tr.addr = 'h35;
     tr.set_ccc = 0;
     tr.sr_or_p_gen = 0;
end
   `uvm_info("body", " Slave setting data size for READ Response HDR_DDR mode..",
UVM_DEBUG)
   `uvm_create_on(slv_tr,p_sequencer.slave_sequencer[1])
   if (!slv_tr.randomize() with {
                                slv_tr.set_ibi       == 0;
                               })
                               `uvm_error("Randomization
Failure","main_mst_hdr_wr_abort_with_exit_pattern_sequence")
   else
      slv_tr.nack_data = 2;

fork
   `uvm_send(tr)
   `uvm_send(slv_tr)
join
```

In first scenario, the slave will abort after second word, whereas in second scenario there is no abort since the data size set in master is four (implying two words to be sent) and slave nack data set is two (implying slave is configured to abort after two words).

A.12 Master and Slave Coverage

The following table shows the covergroups, coverpoints and the corresponding bins:

Table A-2 Coverage

Cover Group	Cover Points	Bins	Present in	Description
transaction_type	cmd	write_cmd	only master	This coverpoint captures coverage of commands sent by I3C Master
		read_cmd	only master	

Table A-2 (Continued)Coverage

Cover Group	Cover Points	Bins	Present in	Description
	broadcast_header	with_brdcast_header	only master	This bin hits only when we see pattern of type 7E+W+Sr+pvt command as one unit.
		without_brdcast_header	only master	
	abort_cmd	abort_by_master	only master	This coverpoint counts after which the data is NACKed.
	ccc	ccc_setdasa	Master/Slave both	This coverpoint captures both Broadcast and directed CCC sent by I3C Master.
		ccc_setdasa	Master/Slave both	
		ccc_brdcast_enec	Master/Slave both	
		ccc_brdcast_disec	Master/Slave both	
		ccc_brdcast_entas0	Master/Slave both	
		ccc_brdcast_rstdaa	Master/Slave both	
		ccc_brdcast_setmwl	Master/Slave both	
		ccc_brdcast_setmrl	Master/Slave both	
		ccc_defslvs	Master/Slave both	
		ccc_direct_enec	Master/Slave both	
		ccc_direct_disec	Master/Slave both	
		ccc_direct_entas0	Master/Slave both	
		ccc_direct_rstdaa	Master/Slave both	
		ccc_setnewda	Master/Slave both	
		ccc_direct_setmwl	Master/Slave both	
		ccc_direct_setmrl	Master/Slave both	
		ccc_getmwl	Master/Slave both	
		ccc_getmrl	Master/Slave both	
		ccc_getpid	Master/Slave both	
		ccc_getbcr	Master/Slave both	

Table A-2 (Continued)Coverage

Cover Group	Cover Points	Bins	Present in	Description
		ccc_getdcr	Master/Slave both	
		ccc_getstatus	Master/Slave both	
		ccc_direct_setxtime	Master/Slave both	
		ccc_getxtime	Master/Slave both	
		ccc_gethdrcap	Master/Slave both	
		ccc_direct_setbrgtgt	Master/Slave both	
		ccc_getaccmst	Master/Slave both	
		ccc_brdcast_entas1	Master/Slave both	
		ccc_brdcast_entas2	Master/Slave both	
		ccc_brdcast_entas3	Master/Slave both	
		ccc_direct_entas1	Master/Slave both	
		ccc_direct_entas2	Master/Slave both	
		ccc_direct_entas3	Master/Slave both	
		ccc_brdcast_setxtime	Master/Slave both	
		ccc_getmxds	Master/Slave both	
		ccc_enthdr0	only master	
		ccc_enthdr1	Master/Slave both	
		ccc_enthdr2	Master/Slave both	
		ccc_defslvs	Master/Slave both	
		ccc_direct_endxfer	Master/Slave both	
		ccc_brdcst_endxfer	Master/Slave both	
		ccc_enttm	Master/Slave both	
		ccc_broadcast_vendor	Master/Slave both	
		ccc_directed_vendor	Master/Slave both	

Table A-2 (Continued)Coverage

Cover Group	Cover Points	Bins	Present in	Description
		ccc_direct_rstact	Master/Slave both	
		ccc_brdcast_rstact	Master/Slave both	
		ccc_direct_setgrpa	Master/Slave both	
		ccc_defgrpa	Master/Slave both	
		ccc_direct_rstgrpa	Master/Slave both	
		ccc_brdcast_rstgrpa	Master/Slave both	
	prev_cmd_cross_cmd	cross prev_cmd, cmd	Master/Slave both	This captures the cross combinations for previous command and current command.
	prev_cmd_cross_ccc	cross prev_cmd, ccc	Master/Slave both	This coverpoint captures the crosses between previous command(write/read) and current ccc.
	prev_ccc_cross_cmd	cross prev_ccc, cmd	Master/Slave both	This coverpoint captures the crosses between previous ccc and current command(write/read).
	prev_ccc_cross_ccc	cross prev_ccc, ccc	Master/Slave both	This coverpoint captures the crosses between previous ccc and current ccc.
	prev_ccc_cross_hdr_mode_cross_hdr_wr_rd	cross prev_ccc, hdr_mode, hdr_wr_rd	Master/Slave both	This coverpoint captures cross-bins for previous_ccc and hdr_mode(ddr/tsp/tsl) and hdr_wr_rd
	directed_ccc_cross_retry_if_nack	cross directed_ccc, retry_if_nack	Master/Slave both	This coverpoint captures cross-bins between directed_ccc and retry_if_nack.
	prev_cmd_cross_hj_detected_by_main_mst	cross prev_cmd, hj_detected_by_main_mst	Master only	This captures the cross-bins between prev_cmd and hj_detected_by_main_mst.
	prev_ccc_cross_hj_detected_by_main_mst	cross prev_ccc, hj_detected_by_main_mst	Master only	This captures the cross-bins between prev_ccc and hj_detected_by_main_mst.

Table A-2 (Continued)Coverage

Cover Group	Cover Points	Bins	Present in	Description
	prev_cmd_ibi_payload_ACK_detect_by_main_mst	cross prev_cmd, payload_ACK_detect_by_main_mst	Master only	This coverpoint captures cross-bins between prev_cmd & payload_ACK_detect_by_main_mst.
	prev_ccc_cross_ibi_wo_payload_detect_by_main_mst	cross prev_ccc, ibi_wo_payload_detect_by_main_mst	Master only	This coverpoint captures the cross-bins between prev_ccc & ibi_wo_payload_detect_by_main_mst.
	prev_cmd_cross_ibi_wo_payload_detect_by_main_mst	cross prev_cmd, ibi_wo_payload_detect_by_main_mst	Master only	This coverpoint captures the cross-bins between prev_cmd & ibi_wo_payload_detect_by_main_mst.
	prev_ccc_ibi_payload_ACK_detect_by_main_mst	cross prev_ccc, payload_ACK_detect_by_main_mst	Master only	This coverpoint captures cross-bins between prev_ccc & payload_ACK_detect_by_main_mst
	prev_cmd_cross_mr_detected_by_main_mst	cross prev_cmd, mr_detected_by_main_mst	Master only	This coverpoint captures cross-bins between prev_cmd & mr_detected_by_main_mst.
	prev_ccc_cross_mr_detected_by_main_mst	cross prev_ccc, mr_detected_by_main_mst	Master only	This coverpoint captures cross-bins between prev_ccc & mr_detected_by_main_mst.
	exit_pattern	exit_pattern_detected	Master/Slave both	This cover-point detected the exit_pattern in HDR mode.

Table A-2 (Continued)Coverage

Cover Group	Cover Points	Bins	Present in	Description
	start_detected	no_start_detected (ignored)	Master/Slave both	This coverpoint captures coverage of start_detected
		start_detected	Master/Slave both	
	rep_start_detected	no_rep_start_detected (ignored)	Master/Slave both	This coverpoint captures coverage of rep_start_detected
		rep_start_detected	Master/Slave both	
	pkt_end_with_sr_detected	pkt_end_with_sr	Master/Slave both	In case of Master: This coverpoint is used to hit the bin when repeated start arrives after following I3C packets: 1. If SR comes after broadcast ccc. 2. If SR comes after IBI. 3. If SR comes after MR. 4. If SR comes after HJ. In case of slave: This coverpoint is used to hit the bin when repeated start arrives after following I3C packets: 1. If SR comes after broadcast ccc. 2. If SR comes after IBI. 3. If SR comes after HJ.
		no_pkt_end_with_sr (igonred)	Master/Slave both	
	stop_detected	no_stop_detected (ignored)	Master/Slave both	This coverpoint captures coverage of stop_detected
		stop_detected	Master/Slave both	
	hdr_mode	hdr_ddr	only master	This coverpoint captures hdr_mode for i3c master
		hdr_tsp	only master	
		hdr_tsl	only master	
	data_type	data_type_I2C	Master/Slave both	This coverpoint captures coverage of data_type [I2C/SR/HDR]
		data_type_SDR	Master/Slave both	
		data_type_HDR	Master/Slave both	
	data_size	data_size_min_range	Master/Slave both	This coverpoint captures the number of data bytes transmitted
		data_size_mid_range	Master/Slave both	

Table A-2 (Continued)Coverage

Cover Group	Cover Points	Bins	Present in	Description
		data_size_max_range	Master/Slave both	
	nack_resp_to_addr	nack_for_slave_address	only master	This coverpoint captures the NACK response to the Slave Address
	nack_addr	nack_for_slave_address	only slave	This coverpoint captures the NACK response to the Slave Address
	nack_resp_count	nack_addr_count_1	only master	This coverpoint captures the number of times the Slave address will be NACKed
		nack_addr_count_greater_than_1	only master	
	nack_data	nack_data_min_range	Master/Slave both	This coverpoint captures the number of bytes after which master aborts SDR read
		nack_data_mid_range	Master/Slave both	
		nack_data_max_range	Master/Slave both	
	retry_if_nack	retry_if_nack	only master	This coverpoint captures whether the Master retry or not on receiving NACK on the same address after a repeated start
	num_of_retry	num_of_retry_1	only master	This coverpoint captures the number of retries made by the MAster when it receives NACK on the same address after a repeated start
		num_of_retry_2	only master	
		num_of_retry_3	only master	
		num_of_retry_4	only master	
		num_of_retry_greater_than_4	only master	
	directed_ccc	ccc_setdasa	Master/Slave both	coverpoint added to hit cross of directed cccs with nack_resp_to_addr, cross of nack_resp_to_addr with broadcast cccs is not required since there is no nack of slave address in broadcast cccs.

Table A-2 (Continued)Coverage

Cover Group	Cover Points	Bins	Present in	Description
		ccc_entdaa	Master/Slave both	
		ccc_direct_enec	Master/Slave both	
		ccc_direct_disec	Master/Slave both	
		ccc_direct_entas0	Master/Slave both	
		ccc_direct_rstdaa	Master/Slave both	
		ccc_setnewda	Master/Slave both	
		ccc_direct_setmwl	Master/Slave both	
		ccc_getmwl	Master/Slave both	
		ccc_getmrl	Master/Slave both	
		ccc_getpid	Master/Slave both	
		ccc_getbcr	Master/Slave both	
		ccc_getstatus	Master/Slave both	
		ccc_direct_setxtime	Master/Slave both	
		ccc_getxtime	Master/Slave both	
		ccc_gethdrcap	Master/Slave both	
		ccc_direct_setbrgtgt	Master/Slave both	
		ccc_getaccmst	Master/Slave both	
		ccc_direct_entas1	Master/Slave both	
		ccc_direct_entas2	Master/Slave both	
		ccc_direct_entas3	Master/Slave both	
		ccc_getmxds	Master/Slave both	
		ccc_direct_endxfer	Master/Slave both	
		ccc_directed_vendor	Master/Slave both	
		ccc_enthdr0	Master/Slave both	
		ccc_enthdr1	Master/Slave both	
		ccc_enthdr2	Master/Slave both	
		ccc_direct_rstact	Master/Slave both	

Table A-2 (Continued)Coverage

Cover Group	Cover Points	Bins	Present in	Description
		ccc_direct_setgrpa	Master/Slave both	
		ccc_direct_rstgrpa	Master/Slave both	
	prev_ccc	prev_ccc_getxtime	Master/Slave both	
		prev_ccc_direct_setbrgtgt	Master/Slave both	
		prev_ccc_getaccmst	Master/Slave both	
		prev_ccc_brdcast_entas1	Master/Slave both	
		prev_ccc_brdcast_entas2	Master/Slave both	
		prev_ccc_brdcast_entas3	Master/Slave both	
		prev_ccc_direct_entas1	Master/Slave both	
		prev_ccc_direct_entas2	Master/Slave both	
		prev_ccc_direct_entas3	Master/Slave both	
		prev_ccc_getmxds	Master/Slave both	
		prev_ccc_brdcast_setxtime	Master/Slave both	
		prev_ccc_direct_setxtime	Master/Slave both	
		prev_ccc_gethdrcap	Master/Slave both	
		prev_ccc_getcaps	Master/Slave both	
		prev_ccc_enthdr0	Master/Slave both	
		prev_ccc_enthdr1	Master/Slave both	
		prev_ccc_enthdr2	Master/Slave both	
		prev_ccc_direct_endxfer	Master/Slave both	
		prev_ccc_brdcst_endxfer	Master/Slave both	

Table A-2 (Continued)Coverage

Cover Group	Cover Points	Bins	Present in	Description
		prev_ccc_brdcast_rstact	Master/Slave both	
		prev_ccc_direct_rstact	Master/Slave both	
		prev_ccc_defslvs	Master/Slave both	
		prev_ccc_enttm	Master/Slave both	
		prev_ccc_broadcast_vendor	Master/Slave both	
		prev_ccc_directed_vendor	Master/Slave both	
		prev_ccc_setdasa	Master/Slave both	
		prev_ccc_entdaa	Master/Slave both	
		prev_ccc_brdcast_enec	Master/Slave both	
		prev_ccc_brdcast_disec	Master/Slave both	
		prev_ccc_brdcast_entas0	Master/Slave both	
		prev_ccc_brdcast_rstdaa	Master/Slave both	
		prev_ccc_brdcast_setmwl	Master/Slave both	
		prev_ccc_brdcast_setmrl	Master/Slave both	
		prev_ccc_direct_enec	Master/Slave both	
		prev_ccc_direct_disec	Master/Slave both	
		prev_ccc_direct_entas0	Master/Slave both	
		prev_ccc_direct_rstdaa	Master/Slave both	
		prev_ccc_setnewda	Master/Slave both	

Table A-2 (Continued)Coverage

Cover Group	Cover Points	Bins	Present in	Description
		prev_ccc_direct_setmwl	Master/Slave both	
		prev_ccc_direct_setmrl	Master/Slave both	
		prev_ccc_getmwl	Master/Slave both	
		prev_ccc_getmrl	Master/Slave both	
		prev_ccc_getpid	Master/Slave both	
		prev_ccc_getbcr	Master/Slave both	
		prev_ccc_getdcr	Master/Slave both	
		prev_ccc_getstatus	Master/Slave both	
		prev_ccc_direct_setgrpa	Master/Slave both	
		prev_ccc_defgrpa	Master/Slave both	
		prev_ccc_direct_rstgrpa	Master/Slave both	
		prev_ccc_brdcast_rstgrpa	Master/Slave both	

Table A-2 (Continued)Coverage

Cover Group	Cover Points	Bins	Present in	Description
	read_write	write_cmd	only slave	This coverpoint captures coverage of read_write
		read_cmd	only slave	
	broadcast_ccc	ccc_brdcast_enec	Master/Slave both	coverpoint added to hit the broadcast_ccc (excluding HDR cccs and ENTDAA ccc.)and it's cross with "pkt_end_with_sr_detected".
		ccc_brdcast_disec	Master/Slave both	
		ccc_brdcast_entas0	Master/Slave both	
		ccc_brdcast_entas1	Master/Slave both	
		ccc_brdcast_entas2	Master/Slave both	
		ccc_brdcast_entas3	Master/Slave both	
		ccc_brdcast_rstdaa	Master/Slave both	
		ccc_defslvs	Master/Slave both	
		ccc_brdcast_setmwl	Master/Slave both	
		ccc_brdcast_setmrl	Master/Slave both	
		ccc_brdcast_setxtime	Master/Slave both	
		ccc_enttm	Master/Slave both	
		ccc_brdcast_endxfer	Master/Slave both	
		ccc_setaasa	Master/Slave both	
		ccc_entdaa	Master/Slave both	
		ccc_broadcast_vendor	Master/Slave both	
		ccc_brdcast_rstact	Master/Slave both	
		ccc_defgrpa	Master/Slave both	
		ccc_brdcast_rstgrpa	Master/Slave both	

Table A-2 (Continued)Coverage

Cover Group	Cover Points	Bins	Present in	Description
	directed_ccc_ending_with_sr_7e	ccc_setdasa	Master/Slave both	coverpoint is used to detect if the directed ccc is ending with Sr -> 7e.
		ccc_direct_enec	Master/Slave both	
		ccc_direct_disec	Master/Slave both	
		ccc_direct_entas0	Master/Slave both	
		ccc_direct_entas1	Master/Slave both	
		ccc_direct_entas2	Master/Slave both	
		ccc_direct_entas3	Master/Slave both	
		ccc_direct_rstdaa	Master/Slave both	
		ccc_setnewda	Master/Slave both	
		ccc_direct_setmwl	Master/Slave both	
		ccc_direct_setmrl	Master/Slave both	
		ccc_getmwl	Master/Slave both	
		ccc_getmrl	Master/Slave both	
		ccc_getpid	Master/Slave both	
		ccc_getbcr	Master/Slave both	
		ccc_getdcr	Master/Slave both	
		ccc_getstatus	Master/Slave both	
		ccc_direct_setxtime	Master/Slave both	
		ccc_getxtime	Master/Slave both	
		ccc_direct_setbrgtgt	Master/Slave both	
		ccc_getaccmst	Master/Slave both	
		ccc_getmxds	Master/Slave both	
		ccc_direct_endxfer	Master/Slave both	
		ccc_directed_vendor	Master/Slave both	
		ccc_direct_rstact	Master/Slave both	

Table A-2 (Continued)Coverage

Cover Group	Cover Points	Bins	Present in	Description
		ccc_direct_setgrpa	Master/Slave both	
		ccc_direct_rstgrpa	Master/Slave both	

Table A-2 (Continued)Coverage

Cover Group	Cover Points	Bins	Present in	Description
Transaction type	direct_vendor_ccc	direct_vendor_ccc_E0	Master/Slave both	This coverbin gets hit for direct vendor ccc 'hE0
		direct_vendor_ccc_E1	Master/Slave both	This coverbin gets hit for direct vendor ccc 'hE1
		direct_vendor_ccc_E2	Master/Slave both	This coverbin gets hit for direct vendor ccc 'hE2
		direct_vendor_ccc_E3	Master/Slave both	This coverbin gets hit for direct vendor ccc 'hE3
		direct_vendor_ccc_E4	Master/Slave both	This coverbin gets hit for direct vendor ccc 'hE4
		direct_vendor_ccc_E5	Master/Slave both	This coverbin gets hit for direct vendor ccc 'hE5
		direct_vendor_ccc_E6	Master/Slave both	This coverbin gets hit for direct vendor ccc 'hE6
		direct_vendor_ccc_E7	Master/Slave both	This coverbin gets hit for direct vendor ccc 'hE7
		direct_vendor_ccc_E8	Master/Slave both	This coverbin gets hit for direct vendor ccc 'hE8
		direct_vendor_ccc_E9	Master/Slave both	This coverbin gets hit for direct vendor ccc 'hE9
		direct_vendor_ccc_EA	Master/Slave both	This coverbin gets hit for direct vendor ccc 'hEA
		direct_vendor_ccc_EB	Master/Slave both	This coverbin gets hit for direct vendor ccc 'hEB
		direct_vendor_ccc_EC	Master/Slave both	This coverbin gets hit for direct vendor ccc 'hEB
		direct_vendor_ccc_ED	Master/Slave both	This coverbin gets hit for direct vendor ccc 'hEB
		direct_vendor_ccc_EE	Master/Slave both	This coverbin gets hit for direct vendor ccc 'hEB
		direct_vendor_ccc_EF	Master/Slave both	This coverbin gets hit for direct vendor ccc
		direct_vendor_ccc_F0	Master/Slave both	This coverbin gets hit for direct vendor ccc 'Hf0
		direct_vendor_ccc_F1	Master/Slave both	This coverbin gets hit for direct vendor ccc 'Hf1

Table A-2 (Continued)Coverage

Cover Group	Cover Points	Bins	Present in	Description
		direct_vendor_ccc_F2	Master/Slave both	This coverbin gets hit for direct vendor ccc 'Hf2
		direct_vendor_ccc_F3	Master/Slave both	This coverbin gets hit for direct vendor ccc 'Hf3
		direct_vendor_ccc_F4	Master/Slave both	This coverbin gets hit for direct vendor ccc 'Hf4
		direct_vendor_ccc_F5	Master/Slave both	This coverbin gets hit for direct vendor ccc 'Hf5
		direct_vendor_ccc_F6	Master/Slave both	This coverbin gets hit for direct vendor ccc 'Hf6
		direct_vendor_ccc_F7	Master/Slave both	This coverbin gets hit for direct vendor ccc 'Hf7
		direct_vendor_ccc_F8	Master/Slave both	This coverbin gets hit for direct vendor ccc 'Hf8
		direct_vendor_ccc_F9	Master/Slave both	This coverbin gets hit for direct vendor ccc 'Hf9
		direct_vendor_ccc_FA	Master/Slave both	This coverbin gets hit for direct vendor ccc 'Hfa
		direct_vendor_ccc_FB	Master/Slave both	This coverbin gets hit for direct vendor ccc 'hFB
		direct_vendor_ccc_FC	Master/Slave both	This coverbin gets hit for direct vendor ccc 'Hfc
		direct_vendor_ccc_FD	Master/Slave both	This coverbin gets hit for direct vendor ccc 'Hfd
		direct_vendor_ccc_FE	Master/Slave both	This coverbin gets hit for direct vendor ccc 'Hfe
	broadcast_vendor_ccc	brdcast_vendor_ccc_61	Master/Slave both	This coverbin gets hit for broadcast vendor ccc 'h61
		brdcast_vendor_ccc_62	Master/Slave both	This coverbin gets hit for broadcast vendor ccc 'h62
		brdcast_vendor_ccc_63	Master/Slave both	This coverbin gets hit for broadcast vendor ccc 'h63
		brdcast_vendor_ccc_64	Master/Slave both	This coverbin gets hit for broadcast vendor ccc 'h64
		brdcast_vendor_ccc_65	Master/Slave both	This coverbin gets hit for broadcast vendor ccc 'h65

Table A-2 (Continued)Coverage

Cover Group	Cover Points	Bins	Present in	Description
		brdcast_vendor_ccc_66	Master/Slave both	This coverbin gets hit for broadcast vendor ccc 'h66
		brdcast_vendor_ccc_67	Master/Slave both	This coverbin gets hit for broadcast vendor ccc 'h67
		brdcast_vendor_ccc_68	Master/Slave both	This coverbin gets hit for broadcast vendor ccc 'h68
		brdcast_vendor_ccc_69	Master/Slave both	This coverbin gets hit for broadcast vendor ccc 'h69
		brdcast_vendor_ccc_6A	Master/Slave both	This coverbin gets hit for broadcast vendor ccc 'H6a
		brdcast_vendor_ccc_6B	Master/Slave both	This coverbin gets hit for broadcast vendor ccc 'H6b
		brdcast_vendor_ccc_6C	Master/Slave both	This coverbin gets hit for broadcast vendor ccc 'H6c
		brdcast_vendor_ccc_6D	Master/Slave both	This coverbin gets hit for broadcast vendor ccc 'H6d
		brdcast_vendor_ccc_6E	Master/Slave both	This coverbin gets hit for broadcast vendor ccc 'H6e
		brdcast_vendor_ccc_6F	Master/Slave both	This coverbin gets hit for broadcast vendor ccc 'H6f
		brdcast_vendor_ccc_70	Master/Slave both	This coverbin gets hit for broadcast vendor ccc 'H70
		brdcast_vendor_ccc_71	Master/Slave both	This coverbin gets hit for broadcast vendor ccc 'H71
		brdcast_vendor_ccc_72	Master/Slave both	This coverbin gets hit for broadcast vendor ccc 'H72
		brdcast_vendor_ccc_73	Master/Slave both	This coverbin gets hit for broadcast vendor ccc 'H73
		brdcast_vendor_ccc_74	Master/Slave both	This coverbin gets hit for broadcast vendor ccc 'H74
		brdcast_vendor_ccc_75	Master/Slave both	This coverbin gets hit for broadcast vendor ccc 'H75
		brdcast_vendor_ccc_76	Master/Slave both	This coverbin gets hit for broadcast vendor ccc 'H76
		brdcast_vendor_ccc_77	Master/Slave both	This coverbin gets hit for broadcast vendor ccc 'H77

Table A-2 (Continued)Coverage

Cover Group	Cover Points	Bins	Present in	Description
		brdcast_vendor_ccc_78	Master/Slave both	This coverbin gets hit for broadcast vendor ccc 'H78
		brdcast_vendor_ccc_79	Master/Slave both	This coverbin gets hit for broadcast vendor ccc 'H79
		brdcast_vendor_ccc_7A	Master/Slave both	This coverbin gets hit for broadcast vendor ccc 'H7A
		brdcast_vendor_ccc_7B	Master/Slave both	This coverbin gets hit for broadcast vendor ccc 'H7B
		brdcast_vendor_ccc_7C	Master/Slave both	This coverbin gets hit for broadcast vendor ccc 'H7C
		brdcast_vendor_ccc_7D	Master/Slave both	This coverbin gets hit for broadcast vendor ccc 'H7D
		brdcast_vendor_ccc_7E	Master/Slave both	This coverbin gets hit for broadcast vendor ccc 'H7E
		brdcast_vendor_ccc_7F	Master/Slave both	This coverbin gets hit for broadcast vendor ccc 'H7F

Table A-2 (Continued)Coverage

Cover Group	Cover Points	Bins	Present in	Description
	rep_start_with_7e_detected	rep_start_with_7e	Master/Slave both	This coverpoint is used to hit the starting header (Sr -> 7E/W) when any "ccc" arrives. Also this coverpoint is crossed with "ccc".
		no_rep_start_with_7e	Master/Slave both	
	ibi_wo_payload_trig_by_sec_mst	ibi_wo_payload_ACK_trig_by_sec_mast	Master only	This coverpoint is used to hit if IBI without payload with ACK/NACK is sent by any active sec-master.
		ibi_wo_payload_NACK_trig_by_sec_mast	Master only	
	ibi_payload_ACK_trig_by_sec_mst	ibi_payload_ACK_trig_by_sec_mast	Master only	This coverpoint is used to hit if IBI with payload and ACK is sent by any active sec-master.
	ibi_wo_payload_detect_by_main_mst	ibi_wo_payload_ACK_detect_by_main_mast	Master only	This coverpoint is used to hit if IBI without payload with ACK/NACK is detected by active main-master.
		ibi_wo_payload_NACK_detect_by_main_mast	Master only	
	ibi_payload_ACK_detect_by_main_mst	ibi_payload_ACK_detect_by_main_mast	Master only	This coverpoint is used to hit if IBI with payload and ACK is detected by active main-master.
	mr_trig_by_sec_mst	mr_with_ACK_trig_by_sec_mast	Master only	This coverpoint is used to hit if mastership request with ACK/NACK is triggered by any active sec-master
		mr_with_NACK_trig_by_sec_mast	Master only	
	mr_detected_by_main_mst	mr_with_ACK_detected_by_main_mast	Master only	This coverpoint is used to hit if mastership request with ACK/NACK is detected by main-master
		mr_with_NACK_detected_by_main_mast	Master only	
	hj_trig_by_sec_mst	hj_with_ACK_trig_by_main_mast	Master only	This coverpoint is used to hit if Hot-Join request with ACK/NACK is triggered by any active sec-master
		hj_with_NACK_trig_by_sec_mast	Master only	

Table A-2 (Continued)Coverage

Cover Group	Cover Points	Bins	Present in	Description
	hj_detected_by_main_mst	hj_with_ACK_detected_by_main_mast	Master only	This coverpoint is used to hit if Hot-Join request with ACK/NACK is detected by main-master
		hj_with_NACK_detected_by_main_mast	Master only	
	ibi_wo_payload_trig_by_slv	ibi_wo_payload_ACK_trig_by_slv	Slave only	This coverpoint is used to hit if IBI without payload with ACK/NACK is sent by any active slave.
		ibi_wo_payload_NACK_trig_by_slv	Slave only	
	ibi_payload_ACK_trig_by_slv	ibi_payload_ACK_trig_by_slv	Slave only	This coverpoint is used to hit if IBI with payload and ACK is sent by any active slave.
	hj_trig_by_slv	hj_with_ACK_trig_by_slv	Slave only	This coverpoint is used to hit if Hot-Join request with ACK/NACK is triggered by any active slave
		hj_with_NACK_trig_by_slv	Slave only	
	s0_error_detected	s0_error_with_3E	Master/Slave both	This coverpoint is used to hit when S0-type error occurs.
		s0_error_with_5E	Master/Slave both	
		s0_error_with_6E	Master/Slave both	
		s0_error_with_76	Master/Slave both	
		s0_error_with_7A	Master/Slave both	
		s0_error_with_7C	Master/Slave both	
		s0_error_with_7F	Master/Slave both	
		s0_error_with_7E_read	Master/Slave both	
	s1_error_detected	s1_error_detected	Master/Slave both	This coverpoint is used to hit when S1-type error occurs.
		s1_error_not_detected	Master/Slave both	
	s2_error_detected	s2_error_detected	Master/Slave both	This coverpoint is used to hit when S2-type error occurs.
		s2_error_not_detected	Master/Slave both	

Table A-2 (Continued)Coverage

Cover Group	Cover Points	Bins	Present in	Description
	s3_error_detected	s3_error_detected	Master/Slave both	This coverpoint is used to hit when S3-type error occurs.
		s3_error_not_detected	Master/Slave both	
	s4_error_detected	s4_error_detected	Master/Slave both	This coverpoint is used to hit when S4-type error occurs.
		s4_error_not_detected	Master/Slave both	
	s5_error_detected	s5_error_detected	Master/Slave both	This coverpoint is used to hit when S5-type error occurs.
		s5_error_not_detected	Master/Slave both	
	m0_error_detected	m0_error_detected	Master/Slave both	This coverpoint is used to hit when M0-type error occurs.
		m0_error_not_detected	Master/Slave both	
	m2_error_detected	m2_error_detected	Master/Slave both	This coverpoint is used to hit when M2-type error occurs.
		m2_error_not_detected	Master/Slave both	
	hdr_wr_or_rd	hdr_write	Master/Slave both	Coverpoint to hit the WRITE/READ cmd during HDR command.
		hdr_read	Master/Slave both	
	hdr_start_or_end_with_restart_or_exit_pattern	hdr_pkt_start_with_ccc_end_with_restart_p	Master/Slave both	Coverpoint captures 2 things: 1. HDR WR/RD packet starting with CCC (ENTHDR0, ENTHDR1, ENTHDR2) or restart pattern. 2. HDR WR/RD packet ending with restart pattern or Exit pattern. Note: 1. This coverpoint will not hit in the packet in which HDR ccc is coming. 2. This coverpoint is crossed with "hdr_mode" cross "hdr_wr_or_rd"
		hdr_pkt_start_with_restart_p_end_with_restart_p	Master/Slave both	
		hdr_pkt_start_with_ccc_end_with_exit_p	Master/Slave both	
		hdr_pkt_start_with_restart_p_end_with_exit_p	Master/Slave both	

Table A-2 (Continued)Coverage

Cover Group	Cover Points	Bins	Present in	Description
	hdr_cmd_function_val	cmd_function_val_min_range	Master/Slave both	Coverpoint to hit the command function in HDR mode and cross this with "hdr_wr_or_rd"
		cmd_function_val_mid_range	Master/Slave both	
		cmd_function_val_max_range	Master/Slave both	
	ccc_cross_start_detected	cross ccc, start_detected	Master/Slave both	
	ccc_cross_stop_detected	cross ccc, stop_detected	Master/Slave both	
	ccc_cross_rep_start_detected	cross ccc, rep_start_detected	Master/Slave both	
	ccc_cross_nack_resp_to_addr	directed_ccc, nack_resp_to_addr	only master	
	cmd_cross_data_size	cross cmd, data_size	only master	
	cmd_cross_broadcast_header	cross cmd , broadcast_header	only master	
	cmd_cross_abort_cmd	cross cmd, abort_cmd	only master	
	nack_data_cross_data_type	cross nack_data, data_type	Master/Slave both	
	hdr_mode_cross_hdr_wr_or_rd_cross_hdr_start_or_end_with_restart_or_exit_pattern	cross hdr_mode, hdr_wr_or_rd, hdr_start_or_end_with_restart_or_exit_pattern	Master/Slave both	
	hdr_cmd_function_val_cross_hdr_wr_or_rd	cross hdr_wr_or_rd, hdr_cmd_function_val	Master/Slave both	
	ccc_cross_rep_start_with_7e_detected	cross ccc, rep_start_with_7e_detected	Master/Slave both	

Table A-2 (Continued)Coverage

Cover Group	Cover Points	Bins	Present in	Description
	directed_ccc_cross_stop_detected	cross directed_ccc, stop_detected	Master/Slave both	
	brdcast_ccc_cross_end_with_sr	cross broadcast_ccc, pkt_end_with_sr_detected illegal_bins entdaa_end_with_SR_7e_write	Master/Slave both	
	brdcast_ccc_cross_stop_detected	cross broadcast_ccc, stop_detected	Master/Slave both	
	ibi_wo_payload_trig_by_sec_mst_cross_stop_detected	cross ibi_wo_payload_trig_by_sec_mst, stop_detected	Master/Slave both	
	ibi_wo_payload_trig_by_sec_mst_cross_rep_start_detected	cross ibi_wo_payload_trig_by_sec_mst, pkt_end_with_sr_detected	Master/Slave both	
	ibi_payload_ACK_trig_by_sec_mst_cross_stop_detected	cross ibi_payload_ACK_trig_by_sec_mst, stop_detected	Master/Slave both	
	ibi_payload_ACK_trig_by_sec_mst_cross_rep_start_detected	cross ibi_payload_ACK_trig_by_sec_mst, pkt_end_with_sr_detected	Master/Slave both	
	ibi_wo_payload_detect_by_main_mst_cross_stop_detected	cross ibi_wo_payload_detect_by_main_mst, stop_detected	Master/Slave both	
	ibi_wo_payload_detect_by_main_mst_cross_rep_start_detected	cross ibi_wo_payload_detect_by_main_mst, pkt_end_with_sr_detected	Master/Slave both	

Table A-2 (Continued)Coverage

Cover Group	Cover Points	Bins	Present in	Description
	ibi_payload_ACK_detect_by_main_mst_cross_stop_detected	cross ibi_payload_ACK_detect_by_main_mst, stop_detected	Master/Slave both	
	ibi_payload_ACK_detect_by_main_mst_cross_rep_start_detected	cross ibi_payload_ACK_detect_by_main_mst, pkt_end_with_sr_detected	Master/Slave both	
	mr_trig_by_sec_mst_cross_stop_detected	cross mr_trig_by_sec_mst, stop_detected	only master	
	mr_trig_by_sec_mst_cross_rep_start_detected	cross mr_trig_by_sec_mst, pkt_end_with_sr_detected	only master	
	mr_detected_by_main_mst_cross_stop_detected	cross mr_detected_by_main_mst, stop_detected	only master	
	mr_detected_by_main_mst_cross_rep_start_detected	cross mr_detected_by_main_mst, pkt_end_with_sr_detected	only master	
	hj_trig_by_sec_mst_cross_stop_detected	cross hj_trig_by_sec_mst, stop_detected	Master/Slave both	
	hj_trig_by_sec_mst_cross_rep_start_detected	cross hj_trig_by_sec_mst, pkt_end_with_sr_detected	Master/Slave both	
	hj_detected_by_main_mst_cross_stop_detected	cross hj_detected_by_main_mst, stop_detected	Master/Slave both	

Table A-2 (Continued)Coverage

Cover Group	Cover Points	Bins	Present in	Description
	hj_detected_by_main_mst_cross_rep_start_detected	cross hj_detected_by_main_mst, pkt_end_with_sr_detected	Master/Slave both	
ccc_format	ccc_format1	setdasa_ccc_format	master/Slave both	Coverage group to capture the specific command format where all the format bytes and events like ACK, start and stop are tracked
		direct_rstdaa_format	master/Slave both	
		direct_entas_0_format	master/Slave both	
		direct_entas_1_format	master/Slave both	
		direct_entas_2_format	master/Slave both	
		direct_entas_3_format	master/Slave both	
		broadcast_rstdaa_format	master/Slave both	
		broadcast_entas_0_format	master/Slave both	
		broadcast_entas_1_format	master/Slave both	
		broadcast_entas_2_format	master/Slave both	
		broadcast_entas_3_format	master/Slave both	
		direct_enec_format	master/Slave both	
		direct_disec_format	master/Slave both	
		get_bcr_format	master/Slave both	
		get_dcr_format	master/Slave both	
		get_hdrcap_format	master/Slave both	

Table A-2 (Continued)Coverage

Cover Group	Cover Points	Bins	Present in	Description
		broadcast_enec_format	master/Slave both	
		broadcast_disec_format	master/Slave both	
		defslvs_format	master/Slave both	
		direct_setmwl	master/Slave both	
		direct_setmrl	master/Slave both	
		direct_getmwl	master/Slave both	
		direct_getmrl	master/Slave both	
		broadcast_setmwl	master/Slave both	
		broadcast_setmrl	master/Slave both	
		setnewda_ccc_format	master/Slave both	
		direct_getaccmst	master/Slave both	
		direct_getmxds	master/Slave both	
		direct_getxtime	master/Slave both	
		direct_getpid	master/Slave both	
		direct_getstatus	master/Slave both	
		entdaa_ccc_format	master/Slave both	
		setbrgtgt_ccc_format	master/Slave both	
		direct_endxfer_ccc_format	master/Slave both	
		brdcst_endxfer_ccc_format	master/Slave both	
		enttm_ccc_format	master/Slave both	
		vendor_specific_directed_ccc_format	master/Slave both	
		broadcast_rstact	master/Slave both	
		direct_rstact	master/Slave both	

Table A-2 (Continued)Coverage

Cover Group	Cover Points	Bins	Present in	Description
		direct_setmwl_getmwl_setmrl_getmrl_format	only master	
		brdcst_setmwl_setmrl	only master	
		random_direct_ccc_format	only master	
		random_ccc_0	only master	
		random_ccc_1	only master	
		random_ccc_2	only master	
		random_ccc_3	only master	
		random_ccc_4	only master	
		random_ccc_5	only master	
command_pattern	cmd_sequence	cmd_setdasa_entdaa_sequence	only master	Coverage for hitting particular command pattern
cov_for_reset_flow	rst_after_ibi_detect_by_main_mst	rst_ibi_wo_payload_ACK_with_Sr_or_P_detect_by_main_mst	master only	
		rst_ibi_wo_payload_NACK_with_Sr_or_P_detect_by_main_mst	master only	
		rst_ibi_payload_ACK_with_Sr_or_P_detect_by_main_mst	master only	
		rst_ibi_wo_payload_ACK_detect_by_main_mst	master only	
		rst_ibi_wo_payload_NACK_detect_by_main_mst	master only	
		rst_ibi_payload_ACK_with_detect_by_main_mst	master only	

Table A-2 (Continued)Coverage

Cover Group	Cover Points	Bins	Present in	Description
	rst_after_mr_detected_by_main_mst	rst_mr_with_ACK_SR_or_P_detected_by_main_mast	master only	
		rst_mr_with_NACK_SR_or_P_detected_by_main_mast	master only	
		rst_mr_with_ACK_detected_by_main_mast	master only	
		rst_mr_with_NACK_detected_by_main_mast	master only	
	rst_hj_detected_by_main_mst	rst_hj_with_ACK_SR_or_P_detected_by_main_mast	master only	
		rst_hj_with_NACK_SR_or_P_detected_by_main_mast	master only	
		rst_hj_with_ACK_detected_by_main_mast	master only	
		rst_hj_with_NACK_detected_by_main_mast	master only	
	rst_with_addr_or_data_phase	rst_with_pvt_addr	master/slave both	
		rst_with_pvt_addr_rd	master/slave both	
		rst_with_pvt_addr_rd_ack	master/slave both	
		rst_with_pvt_addr_rd_nack	master/slave both	
		rst_with_pvt_addr_wr	master/slave both	
		rst_with_pvt_addr_wr_ack	master/slave both	

Table A-2 (Continued)Coverage

Cover Group	Cover Points	Bins	Present in	Description
		rst_with_pvt_addr_wr_nack	master/slave both	
		rst_with_pvt_addr_rd_data	master/slave both	
		rst_with_pvt_addr_wr_data	master/slave both	
		rst_with_7E	master/slave both	
		rst_with_7E_RD	master/slave both	
		rst_with_7E_WR	master/slave both	
		rst_with_7E_WR_NACK	master/slave both	
		rst_with_7E_WR_ACK	master/slave both	
		rst_with_7E_RD_NACK	master/slave both	
		rst_with_7E_RD_ACK	master/slave both	
	brdcast_ccc_flow_with_rst	rst_with_brdcast_enec_opc	master/slave both	
		rst_with_brdcast_enec_t_bit	master/slave both	
		rst_with_brdcast_enec_t_bit_def_byte	master/slave both	
		rst_with_brdcast_disec_opc	master/slave both	
		rst_with_brdcast_disec_t_bit	master/slave both	
		rst_with_brdcast_disec_t_bit_def_byte	master/slave both	
		rst_with_brdcast_entas0_opc	master/slave both	
		rst_with_brdcast_entas0_t_bit	master/slave both	

Table A-2 (Continued)Coverage

Cover Group	Cover Points	Bins	Present in	Description
		rst_with_brdcast_entas1_opc	master/slave both	
		rst_with_brdcast_entas1_t_bit	master/slave both	
		rst_with_brdcast_entas2_opc	master/slave both	
		rst_with_brdcast_entas2_t_bit	master/slave both	
		rst_with_brdcast_entas3_opc	master/slave both	
		rst_with_brdcast_entas3_t_bit	master/slave both	
		rst_with_brdcast_rstdaa_opc	master/slave both	
		rst_with_brdcast_rstdaa_t_bit	master/slave both	
		rst_with_brdcast_entdaa_opc	master/slave both	
		rst_with_brdcast_entdaa_t_bit	master/slave both	
		rst_with_brdcast_entdaa_t_bit_data	master/slave both	
		rst_with_brdcast_defslvs_opc	master/slave both	
		rst_with_brdcast_defslvs_t_bit	master/slave both	
		rst_with_brdcast_defslvs_t_bit_data	master/slave both	
		rst_with_brdcast_setmwl_opc	master/slave both	
		rst_with_brdcast_setmwl_t_bit	master/slave both	
		rst_with_brdcast_setmwl_t_bit_def_byte	master/slave both	
		rst_with_brdcast_setmrl_opc	master/slave both	

Table A-2 (Continued)Coverage

Cover Group	Cover Points	Bins	Present in	Description
		rst_with_brdcast_setmrl_t_bit	master/slave both	
		rst_with_brdcast_setmrl_t_bit_def_byte	master/slave both	
		rst_with_brdcast_enttm_opc	master/slave both	
		rst_with_brdcast_enttm_t_bit	master/slave both	
		rst_with_brdcast_enttm_t_bit_def_byte	master/slave both	
		rst_with_brdcast_endxfer_opc	master/slave both	
		rst_with_brdcast_endxfer_t_bit	master/slave both	
		rst_with_brdcast_endxfer_t_bit_data	master/slave both	
		rst_with_brdcast_enthdr0_opc	master/slave both	
		rst_with_brdcast_enthdr0_t_bit	master/slave both	
		rst_with_brdcast_enthdr0_t_bit_data	master/slave both	
		rst_with_brdcast_enthdr1_opc	master/slave both	
		rst_with_brdcast_enthdr1_t_bit	master/slave both	
		rst_with_brdcast_enthdr1_t_bit_data	master/slave both	
		rst_with_brdcast_enthdr2_opc	master/slave both	
		rst_with_brdcast_enthdr2_t_bit	master/slave both	
		rst_with_brdcast_enthdr2_t_bit_data	master/slave both	

Table A-2 (Continued)Coverage

Cover Group	Cover Points	Bins	Present in	Description
		rst_with_brdcast_setxtime_opc	master/slave both	
		rst_with_brdcast_setxtime_t_bit	master/slave both	
		rst_with_brdcast_setxtime_t_bit_data	master/slave both	
		rst_with_brdcast_setaasa_opc	master/slave both	
		rst_with_brdcast_setaasa_t_bit	master/slave both	
		rst_with_brdcast_rstact_opc	master/slave both	
		rst_with_brdcast_rstact_t_bit	master/slave both	
		rst_with_brdcast_rstact_t_bit_def_byte	master/slave both	
		rst_with_brdcast_vendor_opc_61	master/slave both	
		rst_with_brdcast_vendor_opc_61_t_bit	master/slave both	
		rst_with_brdcast_vendor_opc_61_t_bit_data	master/slave both	
		rst_with_brdcast_vendor_opc_62	master/slave both	
		rst_with_brdcast_vendor_opc_62_t_bit	master/slave both	
		rst_with_brdcast_vendor_opc_62_t_bit_data	master/slave both	
		rst_with_brdcast_vendor_opc_63	master/slave both	
		rst_with_brdcast_vendor_opc_63_t_bit	master/slave both	

Table A-2 (Continued)Coverage

Cover Group	Cover Points	Bins	Present in	Description
		rst_with_brdcast_vendor_opc_63_t_bit_data	master/slave both	
		rst_with_brdcast_vendor_opc_64	master/slave both	
		rst_with_brdcast_vendor_opc_64_t_bit	master/slave both	
		rst_with_brdcast_vendor_opc_64_t_bit_data	master/slave both	
		rst_with_brdcast_vendor_opc_65	master/slave both	
		rst_with_brdcast_vendor_opc_65_t_bit	master/slave both	
		rst_with_brdcast_vendor_opc_65_t_bit_data	master/slave both	
		rst_with_brdcast_vendor_opc_66	master/slave both	
		rst_with_brdcast_vendor_opc_66_t_bit	master/slave both	
		rst_with_brdcast_vendor_opc_66_t_bit_data	master/slave both	
		rst_with_brdcast_vendor_opc_67	master/slave both	
		rst_with_brdcast_vendor_opc_67_t_bit	master/slave both	
		rst_with_brdcast_vendor_opc_67_t_bit_data	master/slave both	
		rst_with_brdcast_vendor_opc_68	master/slave both	

Table A-2 (Continued)Coverage

Cover Group	Cover Points	Bins	Present in	Description
		rst_with_brdcast_vendor_opc_68_t_bit	master/slave both	
		rst_with_brdcast_vendor_opc_68_t_bit_data	master/slave both	
		rst_with_brdcast_vendor_opc_69	master/slave both	
		rst_with_brdcast_vendor_opc_69_t_bit	master/slave both	
		rst_with_brdcast_vendor_opc_69_t_bit_data	master/slave both	
		rst_with_brdcast_vendor_opc_6A	master/slave both	
		rst_with_brdcast_vendor_opc_6A_t_bit	master/slave both	
		rst_with_brdcast_vendor_opc_6A_t_bit_data	master/slave both	
		rst_with_brdcast_vendor_opc_6B	master/slave both	
		rst_with_brdcast_vendor_opc_6B_t_bit	master/slave both	
		rst_with_brdcast_vendor_opc_6B_t_bit_data	master/slave both	
		rst_with_brdcast_vendor_opc_6C	master/slave both	
		rst_with_brdcast_vendor_opc_6C_t_bit	master/slave both	
		rst_with_brdcast_vendor_opc_6C_t_bit_data	master/slave both	

Table A-2 (Continued)Coverage

Cover Group	Cover Points	Bins	Present in	Description
		rst_with_brdcast_vendor_opc_6D	master/slave both	
		rst_with_brdcast_vendor_opc_6D_t_bit	master/slave both	
		rst_with_brdcast_vendor_opc_6D_t_bit_data	master/slave both	
		rst_with_brdcast_vendor_opc_6E	master/slave both	
		rst_with_brdcast_vendor_opc_6E_t_bit	master/slave both	
		rst_with_brdcast_vendor_opc_6E_t_bit_data	master/slave both	
		rst_with_brdcast_vendor_opc_6F	master/slave both	
		rst_with_brdcast_vendor_opc_6F_t_bit	master/slave both	
		rst_with_brdcast_vendor_opc_6F_t_bit_data	master/slave both	
		rst_with_brdcast_vendor_opc_70	master/slave both	
		rst_with_brdcast_vendor_opc_70_t_bit	master/slave both	
		rst_with_brdcast_vendor_opc_70_t_bit_data	master/slave both	
		rst_with_brdcast_vendor_opc_71	master/slave both	
		rst_with_brdcast_vendor_opc_71_t_bit	master/slave both	

Table A-2 (Continued)Coverage

Cover Group	Cover Points	Bins	Present in	Description
		rst_with_brdcast_vendor_opc_71_t_bit_data	master/slave both	
		rst_with_brdcast_vendor_opc_72	master/slave both	
		rst_with_brdcast_vendor_opc_72_t_bit	master/slave both	
		rst_with_brdcast_vendor_opc_72_t_bit_data	master/slave both	
		rst_with_brdcast_vendor_opc_73	master/slave both	
		rst_with_brdcast_vendor_opc_73_t_bit	master/slave both	
		rst_with_brdcast_vendor_opc_73_t_bit_data	master/slave both	
		rst_with_brdcast_vendor_opc_74	master/slave both	
		rst_with_brdcast_vendor_opc_74_t_bit	master/slave both	
		rst_with_brdcast_vendor_opc_74_t_bit_data	master/slave both	
		rst_with_brdcast_vendor_opc_75	master/slave both	
		rst_with_brdcast_vendor_opc_75_t_bit	master/slave both	
		rst_with_brdcast_vendor_opc_75_t_bit_data	master/slave both	
		rst_with_brdcast_vendor_opc_76	master/slave both	

Table A-2 (Continued)Coverage

Cover Group	Cover Points	Bins	Present in	Description
		rst_with_brdcast_vendor_opc_76_t_bit	master/slave both	
		rst_with_brdcast_vendor_opc_76_t_bit_data	master/slave both	
		rst_with_brdcast_vendor_opc_77	master/slave both	
		rst_with_brdcast_vendor_opc_77_t_bit	master/slave both	
		rst_with_brdcast_vendor_opc_77_t_bit_data	master/slave both	
		rst_with_brdcast_vendor_opc_78	master/slave both	
		rst_with_brdcast_vendor_opc_78_t_bit	master/slave both	
		rst_with_brdcast_vendor_opc_78_t_bit_data	master/slave both	
		rst_with_brdcast_vendor_opc_79_t_bit	master/slave both	
		rst_with_brdcast_vendor_opc_79	master/slave both	
		rst_with_brdcast_vendor_opc_79_t_bit_data	master/slave both	
		rst_with_brdcast_vendor_opc_7A	master/slave both	
		rst_with_brdcast_vendor_opc_7A_t_bit	master/slave both	
		rst_with_brdcast_vendor_opc_7A_t_bit_data	master/slave both	

Table A-2 (Continued)Coverage

Cover Group	Cover Points	Bins	Present in	Description
		rst_with_brdcast_vendor_opc_7B	master/slave both	
		rst_with_brdcast_vendor_opc_7B_t_bit	master/slave both	
		rst_with_brdcast_vendor_opc_7B_t_bit_data	master/slave both	
		rst_with_brdcast_vendor_opc_7C	master/slave both	
		rst_with_brdcast_vendor_opc_7C_t_bit	master/slave both	
		rst_with_brdcast_vendor_opc_7C_t_bit_data	master/slave both	
		rst_with_brdcast_vendor_opc_7D	master/slave both	
		rst_with_brdcast_vendor_opc_7D_t_bit	master/slave both	
		rst_with_brdcast_vendor_opc_7D_t_bit_data	master/slave both	
		rst_with_brdcast_vendor_opc_7E	master/slave both	
		rst_with_brdcast_vendor_opc_7E_t_bit	master/slave both	
		rst_with_brdcast_vendor_opc_7E_t_bit_data	master/slave both	
		rst_with_brdcast_vendor_opc_7F	master/slave both	
		rst_with_brdcast_vendor_opc_7F_t_bit	master/slave both	

Table A-2 (Continued)Coverage

Cover Group	Cover Points	Bins	Present in	Description
		rst_with_brdcast_vendor_opc_7F_t_bit_data	master/slave both	
		rst_with_brdcast_defgrpa_opc	master/slave both	
		rst_with_brdcast_defgrpa_t_bit	master/slave both	
		rst_with_brdcast_defgrpa_t_bit_data	master/slave both	
		rst_with_brdcast_rstgrpa_opc	master/slave both	
		rst_with_brdcast_rstgrpa_t_bit	master/slave both	
	rst_after_brdcast_ccc_end_with_stop	rst_after_ccc_entdaa_then_P	master/slave both	
		rst_after_ccc_enthdr0_then_P	master/slave both	
		rst_after_ccc_enthdr1_then_P	master/slave both	
		rst_after_ccc_enthdr2_then_P	master/slave both	
		rst_after_ccc_brdcast_enec_then_P	master/slave both	
		rst_after_ccc_brdcast_disec_then_P	master/slave both	
		rst_after_ccc_brdcast_entas0_then_P	master/slave both	
		rst_after_ccc_brdcast_entas1_then_P	master/slave both	
		rst_after_ccc_brdcast_entas2_then_P	master/slave both	
		rst_after_ccc_brdcast_entas3_then_P	master/slave both	

Table A-2 (Continued)Coverage

Cover Group	Cover Points	Bins	Present in	Description
		rst_after_ccc_brdcast_rstdaa_then_P	master/slave both	
		rst_after_ccc_defslvs_then_P	master/slave both	
		rst_after_ccc_brdcast_setmwl_then_P	master/slave both	
		rst_after_ccc_brdcast_setmrl_then_P	master/slave both	
		rst_after_ccc_enttm_then_P	master/slave both	
		rst_after_ccc_setaasa_then_P	master/slave both	
		rst_after_ccc_brdcast_setxtime_then_P	master/slave both	
		rst_after_ccc_brdcast_endxfer_then_P	master/slave both	
		rst_after_ccc_brdcast_rstact_then_P	master/slave both	
		rst_after_brdcast_vendor_ccc_61_then_P	master/slave both	
		rst_after_brdcast_vendor_ccc_62_then_P	master/slave both	
		rst_after_brdcast_vendor_ccc_63_then_P	master/slave both	
		rst_after_brdcast_vendor_ccc_64_then_P	master/slave both	
		rst_after_brdcast_vendor_ccc_65_then_P	master/slave both	
		rst_after_brdcast_vendor_ccc_66_then_P	master/slave both	

Table A-2 (Continued)Coverage

Cover Group	Cover Points	Bins	Present in	Description
		rst_after_brdcast_vendor_ccc_67_then_P	master/slave both	
		rst_after_brdcast_vendor_ccc_68_then_P	master/slave both	
		rst_after_brdcast_vendor_ccc_69_then_P	master/slave both	
		rst_after_brdcast_vendor_ccc_6A_then_P	master/slave both	
		rst_after_brdcast_vendor_ccc_6B_then_P	master/slave both	
		rst_after_brdcast_vendor_ccc_6C_then_P	master/slave both	
		rst_after_brdcast_vendor_ccc_6D_then_P	master/slave both	
		rst_after_brdcast_vendor_ccc_6E_then_P	master/slave both	
		rst_after_brdcast_vendor_ccc_6F_then_P	master/slave both	
		rst_after_brdcast_vendor_ccc_70_then_P	master/slave both	
		rst_after_brdcast_vendor_ccc_71_then_P	master/slave both	
		rst_after_brdcast_vendor_ccc_72_then_P	master/slave both	
		rst_after_brdcast_vendor_ccc_73_then_P	master/slave both	

Table A-2 (Continued)Coverage

Cover Group	Cover Points	Bins	Present in	Description
		rst_after_brdcast_vendor_ccc_74_then_P	master/slave both	
		rst_after_brdcast_vendor_ccc_75_then_P	master/slave both	
		rst_after_brdcast_vendor_ccc_76_then_P	master/slave both	
		rst_after_brdcast_vendor_ccc_77_then_P	master/slave both	
		rst_after_brdcast_vendor_ccc_78_then_P	master/slave both	
		rst_after_brdcast_vendor_ccc_79_then_P	master/slave both	
		rst_after_brdcast_vendor_ccc_7A_then_P	master/slave both	
		rst_after_brdcast_vendor_ccc_7B_then_P	master/slave both	
		rst_after_brdcast_vendor_ccc_7C_then_P	master/slave both	
		rst_after_brdcast_vendor_ccc_7D_then_P	master/slave both	
		rst_after_brdcast_vendor_ccc_7E_then_P	master/slave both	
		rst_after_brdcast_vendor_ccc_7F_then_P	master/slave both	
		rst_after_ccc_defgrpa_then_P	master/slave both	

Table A-2 (Continued)Coverage

Cover Group	Cover Points	Bins	Present in	Description
		rst_after_ccc_brdcast_rstgrpa_then_P	master/slave both	
	rst_after_brdcast_ccc_end_with_rep_start	rst_after_ccc_brdcast_enec_then_Sr	master/slave both	
		rst_after_ccc_brdcast_disec_then_Sr	master/slave both	
		rst_after_ccc_brdcast_entas0_then_Sr	master/slave both	
		rst_after_ccc_brdcast_entas1_then_Sr	master/slave both	
		rst_after_ccc_brdcast_entas2_then_Sr	master/slave both	
		rst_after_ccc_brdcast_entas3_then_Sr	master/slave both	
		rst_after_ccc_brdcast_rstdaa_then_Sr	master/slave both	
		rst_after_ccc_defslvs_then_Sr	master/slave both	
		rst_after_ccc_brdcast_setmwl_then_Sr	master/slave both	
		rst_after_ccc_brdcast_setmrl_then_Sr	master/slave both	
		rst_after_ccc_enttm_then_Sr	master/slave both	
		rst_after_ccc_setaasa_then_Sr	master/slave both	
		rst_after_ccc_brdcast_setxtime_then_Sr	master/slave both	

Table A-2 (Continued)Coverage

Cover Group	Cover Points	Bins	Present in	Description
		rst_after_ccc_brdcast_endxfer_then_Sr	master/slave both	
		rst_after_ccc_brdcast_rstact_then_Sr	master/slave both	
		rst_after_brdcast_vendor_ccc_61_then_Sr	master/slave both	
		rst_after_brdcast_vendor_ccc_62_then_Sr	master/slave both	
		rst_after_brdcast_vendor_ccc_63_then_Sr	master/slave both	
		rst_after_brdcast_vendor_ccc_64_then_Sr	master/slave both	
		rst_after_brdcast_vendor_ccc_65_then_Sr	master/slave both	
		rst_after_brdcast_vendor_ccc_66_then_Sr	master/slave both	
		rst_after_brdcast_vendor_ccc_67_then_Sr	master/slave both	
		rst_after_brdcast_vendor_ccc_68_then_Sr	master/slave both	
		rst_after_brdcast_vendor_ccc_69_then_Sr	master/slave both	
		rst_after_brdcast_vendor_ccc_6A_then_Sr	master/slave both	
		rst_after_brdcast_vendor_ccc_6B_then_Sr	master/slave both	

Table A-2 (Continued)Coverage

Cover Group	Cover Points	Bins	Present in	Description
		rst_after_brdcast_vendor_ccc_6C_then_Sr	master/slave both	
		rst_after_brdcast_vendor_ccc_6D_then_Sr	master/slave both	
		rst_after_brdcast_vendor_ccc_6E_then_Sr	master/slave both	
		rst_after_brdcast_vendor_ccc_6F_then_Sr	master/slave both	
		rst_after_brdcast_vendor_ccc_70_then_Sr	master/slave both	
		rst_after_brdcast_vendor_ccc_71_then_Sr	master/slave both	
		rst_after_brdcast_vendor_ccc_72_then_Sr	master/slave both	
		rst_after_brdcast_vendor_ccc_73_then_Sr	master/slave both	
		rst_after_brdcast_vendor_ccc_74_then_Sr	master/slave both	
		rst_after_brdcast_vendor_ccc_75_then_Sr	master/slave both	
		rst_after_brdcast_vendor_ccc_76_then_Sr	master/slave both	
		rst_after_brdcast_vendor_ccc_77_then_Sr	master/slave both	
		rst_after_brdcast_vendor_ccc_78_then_Sr	master/slave both	

Table A-2 (Continued)Coverage

Cover Group	Cover Points	Bins	Present in	Description
		rst_after_brdcast_vendor_ccc_79_then_Sr	master/slave both	
		rst_after_brdcast_vendor_ccc_7A_then_Sr	master/slave both	
		rst_after_brdcast_vendor_ccc_7B_then_Sr	master/slave both	
		rst_after_brdcast_vendor_ccc_7C_then_Sr	master/slave both	
		rst_after_brdcast_vendor_ccc_7D_then_Sr	master/slave both	
		rst_after_brdcast_vendor_ccc_7E_then_Sr	master/slave both	
		rst_after_brdcast_vendor_ccc_7F_then_Sr	master/slave both	
		rst_after_ccc_defgrpa_then_Sr	master/slave both	
		rst_after_ccc_brdcast_rstgrpa_then_Sr	master/slave both	
	direct_ccc_flow_with_rst	rst_with_direct_enec_opc	master/slave both	
		rst_with_direct_enec_t_bit	master/slave both	
		rst_with_direct_enec_t_bit_Sr	master/slave both	
		rst_with_direct_enec_t_bit_Sr_addr	master/slave both	
		rst_with_direct_enec_t_bit_Sr_addr_ack	master/slave both	

Table A-2 (Continued)Coverage

Cover Group	Cover Points	Bins	Present in	Description
		rst_with_direct_enec_t_bit_Sr_addr_nack	master/slave both	
		rst_with_direct_enec_t_bit_Sr_addr_data	master/slave both	
		rst_with_direct_disec_opc	master/slave both	
		rst_with_direct_disec_t_bit	master/slave both	
		rst_with_direct_disec_t_bit_Sr	master/slave both	
		rst_with_direct_disec_t_bit_Sr_addr	master/slave both	
		rst_with_direct_disec_t_bit_Sr_addr_ack	master/slave both	
		rst_with_direct_disec_t_bit_Sr_addr_nack	master/slave both	
		rst_with_direct_disec_t_bit_Sr_addr_data	master/slave both	
		rst_with_direct_entas0_opc	master/slave both	
		rst_with_direct_entas0_t_bit	master/slave both	
		rst_with_direct_entas0_t_bit_Sr	master/slave both	
		rst_with_direct_entas0_t_bit_Sr_addr	master/slave both	
		rst_with_direct_entas0_t_bit_Sr_addr_ack	master/slave both	
		rst_with_direct_entas0_t_bit_Sr_addr_nack	master/slave both	

Table A-2 (Continued)Coverage

Cover Group	Cover Points	Bins	Present in	Description
		rst_with_direct_entas1_opc	master/slave both	
		rst_with_direct_entas1_t_bit	master/slave both	
		rst_with_direct_entas1_t_bit_Sr	master/slave both	
		rst_with_direct_entas1_t_bit_Sr_addr	master/slave both	
		rst_with_direct_entas1_t_bit_Sr_addr_ack	master/slave both	
		rst_with_direct_entas1_t_bit_Sr_addr_nack	master/slave both	
		rst_with_direct_entas2_opc	master/slave both	
		rst_with_direct_entas2_t_bit	master/slave both	
		rst_with_direct_entas2_t_bit_Sr	master/slave both	
		rst_with_direct_entas2_t_bit_Sr_addr	master/slave both	
		rst_with_direct_entas2_t_bit_Sr_addr_ack	master/slave both	
		rst_with_direct_entas2_t_bit_Sr_addr_nack	master/slave both	
		rst_with_direct_entas3_opc	master/slave both	
		rst_with_direct_entas3_t_bit	master/slave both	
		rst_with_direct_entas3_t_bit_Sr	master/slave both	
		rst_with_direct_entas3_t_bit_Sr_addr	master/slave both	

Table A-2 (Continued)Coverage

Cover Group	Cover Points	Bins	Present in	Description
		rst_with_direct_entas3_t_bit_Sr_addr_ack	master/slave both	
		rst_with_direct_entas3_t_bit_Sr_addr_nack	master/slave both	
		rst_with_direct_rstdaa_opc	master/slave both	
		rst_with_direct_rstdaa_t_bit	master/slave both	
		rst_with_direct_rstdaa_t_bit_Sr	master/slave both	
		rst_with_direct_rstdaa_t_bit_Sr_addr	master/slave both	
		rst_with_direct_rstdaa_t_bit_Sr_addr_ack	master/slave both	
		rst_with_direct_rstdaa_t_bit_Sr_addr_nack	master/slave both	
		rst_with_direct_setdasa_opc	master/slave both	
		rst_with_direct_setdasa_t_bit	master/slave both	
		rst_with_direct_setdasa_t_bit_Sr	master/slave both	
		rst_with_direct_setdasa_t_bit_Sr_addr	master/slave both	
		rst_with_direct_setdasa_t_bit_Sr_addr_ack	master/slave both	
		rst_with_direct_setdasa_t_bit_Sr_addr_nack	master/slave both	
		rst_with_direct_setdasa_t_bit_Sr_addr_data	master/slave both	

Table A-2 (Continued)Coverage

Cover Group	Cover Points	Bins	Present in	Description
		rst_with_direct_set newda_opc	master/slave both	
		rst_with_direct_set newda_t_bit	master/slave both	
		rst_with_direct_set newda_t_bit_Sr	master/slave both	
		rst_with_direct_set newda_t_bit_Sr_a ddr	master/slave both	
		rst_with_direct_set newda_t_bit_Sr_a ddr_ack	master/slave both	
		rst_with_direct_set newda_t_bit_Sr_a ddr_nack	master/slave both	
		rst_with_direct_set newda_t_bit_Sr_a ddr_data	master/slave both	
		rst_with_direct_set mwl_opc	master/slave both	
		rst_with_direct_set mwl_t_bit	master/slave both	
		rst_with_direct_set mwl_t_bit_Sr	master/slave both	
		rst_with_direct_set mwl_t_bit_Sr_addr	master/slave both	
		rst_with_direct_set mwl_t_bit_Sr_addr _ack	master/slave both	
		rst_with_direct_set mwl_t_bit_Sr_addr _nack	master/slave both	
		rst_with_direct_set mwl_t_bit_Sr_addr _data	master/slave both	
		rst_with_direct_set mrl_opc	master/slave both	

Table A-2 (Continued)Coverage

Cover Group	Cover Points	Bins	Present in	Description
		rst_with_direct_set mrl_t_bit	master/slave both	
		rst_with_direct_set mrl_t_bit_Sr	master/slave both	
		rst_with_direct_set mrl_t_bit_Sr_addr	master/slave both	
		rst_with_direct_set mrl_t_bit_Sr_addr _ack	master/slave both	
		rst_with_direct_set mrl_t_bit_Sr_addr _nack	master/slave both	
		rst_with_direct_set mrl_t_bit_Sr_addr _data	master/slave both	
		rst_with_direct_get mwl_opc	master/slave both	
		rst_with_direct_get mwl_t_bit	master/slave both	
		rst_with_direct_get mwl_t_bit_Sr	master/slave both	
		rst_with_direct_get mwl_t_bit_Sr_addr	master/slave both	
		rst_with_direct_get mwl_t_bit_Sr_addr _ack	master/slave both	
		rst_with_direct_get mwl_t_bit_Sr_addr _nack	master/slave both	
		rst_with_direct_get mwl_t_bit_Sr_addr _data	master/slave both	
		rst_with_direct_get mrl_opc	master/slave both	
		rst_with_direct_get mrl_t_bit	master/slave both	
		rst_with_direct_get mrl_t_bit_Sr	master/slave both	

Table A-2 (Continued)Coverage

Cover Group	Cover Points	Bins	Present in	Description
		rst_with_direct_getmrl_t_bit_Sr_addr	master/slave both	
		rst_with_direct_getmrl_t_bit_Sr_addr_ack	master/slave both	
		rst_with_direct_getmrl_t_bit_Sr_addr_nack	master/slave both	
		rst_with_direct_getmrl_t_bit_Sr_addr_data	master/slave both	
		rst_with_direct_getpid_opc	master/slave both	
		rst_with_direct_getpid_t_bit	master/slave both	
		rst_with_direct_getpid_t_bit_Sr	master/slave both	
		rst_with_direct_getpid_t_bit_Sr_addr	master/slave both	
		rst_with_direct_getpid_t_bit_Sr_addr_ack	master/slave both	
		rst_with_direct_getpid_t_bit_Sr_addr_nack	master/slave both	
		rst_with_direct_getpid_t_bit_Sr_addr_data	master/slave both	
		rst_with_direct_getbcr_opc	master/slave both	
		rst_with_direct_getbcr_t_bit	master/slave both	
		rst_with_direct_getbcr_t_bit_Sr	master/slave both	
		rst_with_direct_getbcr_t_bit_Sr_addr	master/slave both	

Table A-2 (Continued)Coverage

Cover Group	Cover Points	Bins	Present in	Description
		rst_with_direct_get bcr_t_bit_Sr_addr _ack	master/slave both	
		rst_with_direct_get bcr_t_bit_Sr_addr _nack	master/slave both	
		rst_with_direct_get bcr_t_bit_Sr_addr _data	master/slave both	
		rst_with_direct_get dcr_opc	master/slave both	
		rst_with_direct_get dcr_t_bit	master/slave both	
		rst_with_direct_get dcr_t_bit_Sr	master/slave both	
		rst_with_direct_get dcr_t_bit_Sr_addr	master/slave both	
		rst_with_direct_get dcr_t_bit_Sr_addr _ack	master/slave both	
		rst_with_direct_get dcr_t_bit_Sr_addr _nack	master/slave both	
		rst_with_direct_get dcr_t_bit_Sr_addr _data	master/slave both	
		rst_with_direct_get status_opc	master/slave both	
		rst_with_direct_get status_t_bit	master/slave both	
		rst_with_direct_get status_t_bit_Sr	master/slave both	
		rst_with_direct_get status_t_bit_Sr_ad dr	master/slave both	
		rst_with_direct_get status_t_bit_Sr_ad dr_ack	master/slave both	

Table A-2 (Continued)Coverage

Cover Group	Cover Points	Bins	Present in	Description
		rst_with_direct_getstatus_t_bit_Sr_addr_nack	master/slave both	
		rst_with_direct_getstatus_t_bit_Sr_addr_data	master/slave both	
		rst_with_direct_getaccmst_opc	master/slave both	
		rst_with_direct_getaccmst_t_bit	master/slave both	
		rst_with_direct_getaccmst_t_bit_Sr	master/slave both	
		rst_with_direct_getaccmst_t_bit_Sr_addr	master/slave both	
		rst_with_direct_getaccmst_t_bit_Sr_addr_ack	master/slave both	
		rst_with_direct_getaccmst_t_bit_Sr_addr_nack	master/slave both	
		rst_with_direct_getaccmst_t_bit_Sr_addr_data	master/slave both	
		rst_with_direct_endxfer_opc	master/slave both	
		rst_with_direct_endxfer_t_bit	master/slave both	
		rst_with_direct_endxfer_t_bit_def_byte	master/slave both	
		rst_with_direct_endxfer_t_bit_def_byte_Sr	master/slave both	
		rst_with_direct_endxfer_t_bit_def_byte_addr_wr	master/slave both	

Table A-2 (Continued)Coverage

Cover Group	Cover Points	Bins	Present in	Description
		rst_with_direct_endxfer_t_bit_def_byte_addr_rd	master/slave both	
		rst_with_direct_endxfer_t_bit_def_byte_addr_wr_ack	master/slave both	
		rst_with_direct_endxfer_t_bit_def_byte_addr_wr_nack	master/slave both	
		rst_with_direct_endxfer_t_bit_def_byte_addr_rd_ack	master/slave both	
		rst_with_direct_endxfer_t_bit_def_byte_addr_rd_nack	master/slave both	
		rst_with_direct_endxfer_t_bit_def_byte_addr_data	master/slave both	
		rst_with_setbrgtgt_opc	master/slave both	
		rst_with_setbrgtgt_t_bit	master/slave both	
		rst_with_setbrgtgt_t_bit_Sr	master/slave both	
		rst_with_setbrgtgt_t_bit_Sr_addr	master/slave both	
		rst_with_setbrgtgt_t_bit_Sr_addr_ack	master/slave both	
		rst_with_setbrgtgt_t_bit_Sr_addr_nack	master/slave both	
		rst_with_setbrgtgt_t_bit_Sr_addr_data	master/slave both	
		rst_with_getmxds_opc	master/slave both	
		rst_with_getmxds_t_bit	master/slave both	

Table A-2 (Continued)Coverage

Cover Group	Cover Points	Bins	Present in	Description
		rst_with_getmxds_t_bit_Sr	master/slave both	
		rst_with_getmxds_t_bit_Sr_addr	master/slave both	
		rst_with_getmxds_t_bit_Sr_addr_ack	master/slave both	
		rst_with_getmxds_t_bit_Sr_addr_nack	master/slave both	
		rst_with_getmxds_t_bit_Sr_addr_data	master/slave both	
		rst_with_gethdrcaps_opc	master/slave both	
		rst_with_gethdrcaps_t_bit	master/slave both	
		rst_with_gethdrcaps_t_bit_Sr	master/slave both	
		rst_with_gethdrcaps_t_bit_Sr_addr	master/slave both	
		rst_with_gethdrcaps_t_bit_Sr_addr_ack	master/slave both	
		rst_with_gethdrcaps_t_bit_Sr_addr_nack	master/slave both	
		rst_with_gethdrcaps_t_bit_Sr_addr_data	master/slave both	
		rst_with_setxtime_opc	master/slave both	
		rst_with_setxtime_t_bit	master/slave both	
		rst_with_setxtime_t_bit_Sr	master/slave both	
		rst_with_setxtime_t_bit_Sr_addr	master/slave both	

Table A-2 (Continued)Coverage

Cover Group	Cover Points	Bins	Present in	Description
		rst_with_setxtime_t_bit_Sr_addr_ack	master/slave both	
		rst_with_setxtime_t_bit_Sr_addr_nack	master/slave both	
		rst_with_setxtime_t_bit_Sr_addr_data	master/slave both	
		rst_with_getxtime_opc	master/slave both	
		rst_with_getxtime_t_bit	master/slave both	
		rst_with_getxtime_t_bit_Sr	master/slave both	
		rst_with_getxtime_t_bit_Sr_addr	master/slave both	
		rst_with_getxtime_t_bit_Sr_addr_ack	master/slave both	
		rst_with_getxtime_t_bit_Sr_addr_nack	master/slave both	
		rst_with_getxtime_t_bit_Sr_addr_data	master/slave both	
		rst_with_direct_rstact_opc	master/slave both	
		rst_with_direct_rstact_t_bit	master/slave both	
		rst_with_direct_rstact_t_bit_def_byte	master/slave both	
		rst_with_direct_rstact_t_bit_def_byte_Sr	master/slave both	
		rst_with_direct_rstact_t_bit_def_byte_Sr_addr	master/slave both	

Table A-2 (Continued)Coverage

Cover Group	Cover Points	Bins	Present in	Description
		rst_with_direct_rstact_t_bit_def_byte_Sr_addr_ack	master/slave both	
		rst_with_direct_rstact_t_bit_def_byte_Sr_addr_nack	master/slave both	
		rst_with_direct_rstact_t_bit_def_byte_Sr_addr_data	master/slave both	
		dir_vendor_E0_opc	master/slave both	
		dir_vendor_E0_t_bit	master/slave both	
		dir_vendor_E0_def_byte	master/slave both	
		dir_vendor_E0_def_byte_Sr	master/slave both	
		dir_vendor_E0_def_byte_Sr_addr	master/slave both	
		dir_vendor_E0_def_byte_Sr_addr_ack	master/slave both	
		dir_vendor_E0_def_byte_Sr_addr_nack	master/slave both	
		dir_vendor_E0_def_byte_Sr_addr_data	master/slave both	
		dir_vendor_E1_opc	master/slave both	
		dir_vendor_E1_t_bit	master/slave both	
		dir_vendor_E1_def_byte	master/slave both	
		dir_vendor_E1_def_byte_Sr	master/slave both	
		dir_vendor_E1_def_byte_Sr_addr	master/slave both	

Table A-2 (Continued)Coverage

Cover Group	Cover Points	Bins	Present in	Description
		dir_vendor_E1_def_byte_Sr_addr_ack	master/slave both	
		dir_vendor_E1_def_byte_Sr_addr_nack	master/slave both	
		dir_vendor_E1_def_byte_Sr_addr_data	master/slave both	
		dir_vendor_E2_opc	master/slave both	
		dir_vendor_E2_t_bit	master/slave both	
		dir_vendor_E2_def_byte	master/slave both	
		dir_vendor_E2_def_byte_Sr	master/slave both	
		dir_vendor_E2_def_byte_Sr_addr	master/slave both	
		dir_vendor_E2_def_byte_Sr_addr_ack	master/slave both	
		dir_vendor_E2_def_byte_Sr_addr_nack	master/slave both	
		dir_vendor_E2_def_byte_Sr_addr_data	master/slave both	
		dir_vendor_E3_opc	master/slave both	
		dir_vendor_E3_t_bit	master/slave both	
		dir_vendor_E3_def_byte	master/slave both	
		dir_vendor_E3_def_byte_Sr	master/slave both	
		dir_vendor_E3_def_byte_Sr_addr	master/slave both	

Table A-2 (Continued)Coverage

Cover Group	Cover Points	Bins	Present in	Description
		dir_vendor_E3_def_byte_Sr_addr_ack	master/slave both	
		dir_vendor_E3_def_byte_Sr_addr_nack	master/slave both	
		dir_vendor_E3_def_byte_Sr_addr_data	master/slave both	
		dir_vendor_E4_opc	master/slave both	
		dir_vendor_E4_t_bit	master/slave both	
		dir_vendor_E4_def_byte	master/slave both	
		dir_vendor_E4_def_byte_Sr	master/slave both	
		dir_vendor_E4_def_byte_Sr_addr	master/slave both	
		dir_vendor_E4_def_byte_Sr_addr_ack	master/slave both	
		dir_vendor_E4_def_byte_Sr_addr_nack	master/slave both	
		dir_vendor_E4_def_byte_Sr_addr_data	master/slave both	
		dir_vendor_E5_opc	master/slave both	
		dir_vendor_E5_t_bit	master/slave both	
		dir_vendor_E5_def_byte	master/slave both	
		dir_vendor_E5_def_byte_Sr	master/slave both	
		dir_vendor_E5_def_byte_Sr_addr	master/slave both	

Table A-2 (Continued)Coverage

Cover Group	Cover Points	Bins	Present in	Description
		dir_vendor_E5_def_byte_Sr_addr_ack	master/slave both	
		dir_vendor_E5_def_byte_Sr_addr_nack	master/slave both	
		dir_vendor_E5_def_byte_Sr_addr_data	master/slave both	
		dir_vendor_E6_opc	master/slave both	
		dir_vendor_E6_t_bit	master/slave both	
		dir_vendor_E6_def_byte	master/slave both	
		dir_vendor_E6_def_byte_Sr	master/slave both	
		dir_vendor_E6_def_byte_Sr_addr	master/slave both	
		dir_vendor_E6_def_byte_Sr_addr_ack	master/slave both	
		dir_vendor_E6_def_byte_Sr_addr_nack	master/slave both	
		dir_vendor_E6_def_byte_Sr_addr_data	master/slave both	
		dir_vendor_E7_opc	master/slave both	
		dir_vendor_E7_t_bit	master/slave both	
		dir_vendor_E7_def_byte	master/slave both	
		dir_vendor_E7_def_byte_Sr	master/slave both	
		dir_vendor_E7_def_byte_Sr_addr	master/slave both	

Table A-2 (Continued)Coverage

Cover Group	Cover Points	Bins	Present in	Description
		dir_vendor_E7_def_byte_Sr_addr_ack	master/slave both	
		dir_vendor_E7_def_byte_Sr_addr_nack	master/slave both	
		dir_vendor_E7_def_byte_Sr_addr_data	master/slave both	
		dir_vendor_E8_opc	master/slave both	
		dir_vendor_E8_t_bit	master/slave both	
		dir_vendor_E8_def_byte	master/slave both	
		dir_vendor_E8_def_byte_Sr	master/slave both	
		dir_vendor_E8_def_byte_Sr_addr	master/slave both	
		dir_vendor_E8_def_byte_Sr_addr_ack	master/slave both	
		dir_vendor_E8_def_byte_Sr_addr_nack	master/slave both	
		dir_vendor_E8_def_byte_Sr_addr_data	master/slave both	
		dir_vendor_E9_opc	master/slave both	
		dir_vendor_E9_t_bit	master/slave both	
		dir_vendor_E9_def_byte	master/slave both	
		dir_vendor_E9_def_byte_Sr	master/slave both	
		dir_vendor_E9_def_byte_Sr_addr	master/slave both	

Table A-2 (Continued)Coverage

Cover Group	Cover Points	Bins	Present in	Description
		dir_vendor_E9_def_byte_Sr_addr_ack	master/slave both	
		dir_vendor_E9_def_byte_Sr_addr_nack	master/slave both	
		dir_vendor_E9_def_byte_Sr_addr_data	master/slave both	
		dir_vendor_EA_opc	master/slave both	
		dir_vendor_EA_t_bit	master/slave both	
		dir_vendor_EA_def_byte	master/slave both	
		dir_vendor_EA_def_byte_Sr	master/slave both	
		dir_vendor_EA_def_byte_Sr_addr	master/slave both	
		dir_vendor_EA_def_byte_Sr_addr_ack	master/slave both	
		dir_vendor_EA_def_byte_Sr_addr_nack	master/slave both	
		dir_vendor_EA_def_byte_Sr_addr_data	master/slave both	
		dir_vendor_EB_opc	master/slave both	
		dir_vendor_EB_t_bit	master/slave both	
		dir_vendor_EB_def_byte	master/slave both	
		dir_vendor_EB_def_byte_Sr	master/slave both	
		dir_vendor_EB_def_byte_Sr_addr	master/slave both	

Table A-2 (Continued)Coverage

Cover Group	Cover Points	Bins	Present in	Description
		dir_vendor_EB_def_byte_Sr_addr_ack	master/slave both	
		dir_vendor_EB_def_byte_Sr_addr_nack	master/slave both	
		dir_vendor_EB_def_byte_Sr_addr_data	master/slave both	
		dir_vendor_EC_opc	master/slave both	
		dir_vendor_EC_t_bit	master/slave both	
		dir_vendor_EC_def_byte	master/slave both	
		dir_vendor_EC_def_byte_Sr	master/slave both	
		dir_vendor_EC_def_byte_Sr_addr	master/slave both	
		dir_vendor_EC_def_byte_Sr_addr_ack	master/slave both	
		dir_vendor_EC_def_byte_Sr_addr_nack	master/slave both	
		dir_vendor_EC_def_byte_Sr_addr_data	master/slave both	
		dir_vendor_ED_opc	master/slave both	
		dir_vendor_ED_t_bit	master/slave both	
		dir_vendor_ED_def_byte	master/slave both	
		dir_vendor_ED_def_byte_Sr	master/slave both	
		dir_vendor_ED_def_byte_Sr_addr	master/slave both	

Table A-2 (Continued)Coverage

Cover Group	Cover Points	Bins	Present in	Description
		dir_vendor_ED_def_byte_Sr_addr_ack	master/slave both	
		dir_vendor_ED_def_byte_Sr_addr_nack	master/slave both	
		dir_vendor_ED_def_byte_Sr_addr_data	master/slave both	
		dir_vendor_EE_opc	master/slave both	
		dir_vendor_EE_t_bit	master/slave both	
		dir_vendor_EE_def_byte	master/slave both	
		dir_vendor_EE_def_byte_Sr	master/slave both	
		dir_vendor_EE_def_byte_Sr_addr	master/slave both	
		dir_vendor_EE_def_byte_Sr_addr_ack	master/slave both	
		dir_vendor_EE_def_byte_Sr_addr_nack	master/slave both	
		dir_vendor_EE_def_byte_Sr_addr_data	master/slave both	
		dir_vendor_EF_opc	master/slave both	
		dir_vendor_EF_t_bit	master/slave both	
		dir_vendor_EF_def_byte	master/slave both	
		dir_vendor_EF_def_byte_Sr	master/slave both	
		dir_vendor_EF_def_byte_Sr_addr	master/slave both	

Table A-2 (Continued)Coverage

Cover Group	Cover Points	Bins	Present in	Description
		dir_vendor_EF_def_byte_Sr_addr_ack	master/slave both	
		dir_vendor_EF_def_byte_Sr_addr_nack	master/slave both	
		dir_vendor_EF_def_byte_Sr_addr_data	master/slave both	
		dir_vendor_F0_opc	master/slave both	
		dir_vendor_F0_t_bit	master/slave both	
		dir_vendor_F0_def_byte	master/slave both	
		dir_vendor_F0_def_byte_Sr	master/slave both	
		dir_vendor_F0_def_byte_Sr_addr	master/slave both	
		dir_vendor_F0_def_byte_Sr_addr_ack	master/slave both	
		dir_vendor_F0_def_byte_Sr_addr_nack	master/slave both	
		dir_vendor_F0_def_byte_Sr_addr_data	master/slave both	
		dir_vendor_F1_opc	master/slave both	
		dir_vendor_F1_t_bit	master/slave both	
		dir_vendor_F1_def_byte	master/slave both	
		dir_vendor_F1_def_byte_Sr	master/slave both	
		dir_vendor_F1_def_byte_Sr_addr	master/slave both	

Table A-2 (Continued)Coverage

Cover Group	Cover Points	Bins	Present in	Description
		dir_vendor_F1_def_byte_Sr_addr_ack	master/slave both	
		dir_vendor_F1_def_byte_Sr_addr_nack	master/slave both	
		dir_vendor_F1_def_byte_Sr_addr_data	master/slave both	
		dir_vendor_F2_opc	master/slave both	
		dir_vendor_F2_t_bit	master/slave both	
		dir_vendor_F2_def_byte	master/slave both	
		dir_vendor_F2_def_byte_Sr	master/slave both	
		dir_vendor_F2_def_byte_Sr_addr	master/slave both	
		dir_vendor_F2_def_byte_Sr_addr_ack	master/slave both	
		dir_vendor_F2_def_byte_Sr_addr_nack	master/slave both	
		dir_vendor_F2_def_byte_Sr_addr_data	master/slave both	
		dir_vendor_F3_opc	master/slave both	
		dir_vendor_F3_t_bit	master/slave both	
		dir_vendor_F3_def_byte	master/slave both	
		dir_vendor_F3_def_byte_Sr	master/slave both	
		dir_vendor_F3_def_byte_Sr_addr	master/slave both	

Table A-2 (Continued)Coverage

Cover Group	Cover Points	Bins	Present in	Description
		dir_vendor_F3_def_byte_Sr_addr_ack	master/slave both	
		dir_vendor_F3_def_byte_Sr_addr_nack	master/slave both	
		dir_vendor_F3_def_byte_Sr_addr_data	master/slave both	
		dir_vendor_F4_opc	master/slave both	
		dir_vendor_F4_t_bit	master/slave both	
		dir_vendor_F4_def_byte	master/slave both	
		dir_vendor_F4_def_byte_Sr	master/slave both	
		dir_vendor_F4_def_byte_Sr_addr	master/slave both	
		dir_vendor_F4_def_byte_Sr_addr_ack	master/slave both	
		dir_vendor_F4_def_byte_Sr_addr_nack	master/slave both	
		dir_vendor_F4_def_byte_Sr_addr_data	master/slave both	
		dir_vendor_F5_opc	master/slave both	
		dir_vendor_F5_t_bit	master/slave both	
		dir_vendor_F5_def_byte	master/slave both	
		dir_vendor_F5_def_byte_Sr	master/slave both	
		dir_vendor_F5_def_byte_Sr_addr	master/slave both	

Table A-2 (Continued)Coverage

Cover Group	Cover Points	Bins	Present in	Description
		dir_vendor_F5_def_byte_Sr_addr_ack	master/slave both	
		dir_vendor_F5_def_byte_Sr_addr_nack	master/slave both	
		dir_vendor_F5_def_byte_Sr_addr_data	master/slave both	
		dir_vendor_F6_opc	master/slave both	
		dir_vendor_F6_t_bit	master/slave both	
		dir_vendor_F6_def_byte	master/slave both	
		dir_vendor_F6_def_byte_Sr	master/slave both	
		dir_vendor_F6_def_byte_Sr_addr	master/slave both	
		dir_vendor_F6_def_byte_Sr_addr_ack	master/slave both	
		dir_vendor_F6_def_byte_Sr_addr_nack	master/slave both	
		dir_vendor_F6_def_byte_Sr_addr_data	master/slave both	
		dir_vendor_F7_opc	master/slave both	
		dir_vendor_F7_t_bit	master/slave both	
		dir_vendor_F7_def_byte	master/slave both	
		dir_vendor_F7_def_byte_Sr	master/slave both	
		dir_vendor_F7_def_byte_Sr_addr	master/slave both	

Table A-2 (Continued)Coverage

Cover Group	Cover Points	Bins	Present in	Description
		dir_vendor_F7_def_byte_Sr_addr_ack	master/slave both	
		dir_vendor_F7_def_byte_Sr_addr_nack	master/slave both	
		dir_vendor_F7_def_byte_Sr_addr_data	master/slave both	
		dir_vendor_F8_opc	master/slave both	
		dir_vendor_F8_t_bit	master/slave both	
		dir_vendor_F8_def_byte	master/slave both	
		dir_vendor_F8_def_byte_Sr	master/slave both	
		dir_vendor_F8_def_byte_Sr_addr	master/slave both	
		dir_vendor_F8_def_byte_Sr_addr_ack	master/slave both	
		dir_vendor_F8_def_byte_Sr_addr_nack	master/slave both	
		dir_vendor_F8_def_byte_Sr_addr_data	master/slave both	
		dir_vendor_F9_opc	master/slave both	
		dir_vendor_F9_t_bit	master/slave both	
		dir_vendor_F9_def_byte	master/slave both	
		dir_vendor_F9_def_byte_Sr	master/slave both	
		dir_vendor_F9_def_byte_Sr_addr	master/slave both	

Table A-2 (Continued)Coverage

Cover Group	Cover Points	Bins	Present in	Description
		dir_vendor_F9_def_byte_Sr_addr_ack	master/slave both	
		dir_vendor_F9_def_byte_Sr_addr_nack	master/slave both	
		dir_vendor_F9_def_byte_Sr_addr_data	master/slave both	
		dir_vendor_FA_opc	master/slave both	
		dir_vendor_FA_t_bit	master/slave both	
		dir_vendor_FA_def_byte	master/slave both	
		dir_vendor_FA_def_byte_Sr	master/slave both	
		dir_vendor_FA_def_byte_Sr_addr	master/slave both	
		dir_vendor_FA_def_byte_Sr_addr_ack	master/slave both	
		dir_vendor_FA_def_byte_Sr_addr_nack	master/slave both	
		dir_vendor_FA_def_byte_Sr_addr_data	master/slave both	
		dir_vendor_FB_opc	master/slave both	
		dir_vendor_FB_t_bit	master/slave both	
		dir_vendor_FB_def_byte	master/slave both	
		dir_vendor_FB_def_byte_Sr	master/slave both	
		dir_vendor_FB_def_byte_Sr_addr	master/slave both	

Table A-2 (Continued)Coverage

Cover Group	Cover Points	Bins	Present in	Description
		dir_vendor_FB_def_byte_Sr_addr_ack	master/slave both	
		dir_vendor_FB_def_byte_Sr_addr_nack	master/slave both	
		dir_vendor_FB_def_byte_Sr_addr_data	master/slave both	
		dir_vendor_FC_opc	master/slave both	
		dir_vendor_FC_t_bit	master/slave both	
		dir_vendor_FC_def_byte	master/slave both	
		dir_vendor_FC_def_byte_Sr	master/slave both	
		dir_vendor_FC_def_byte_Sr_addr	master/slave both	
		dir_vendor_FC_def_byte_Sr_addr_ack	master/slave both	
		dir_vendor_FC_def_byte_Sr_addr_nack	master/slave both	
		dir_vendor_FC_def_byte_Sr_addr_data	master/slave both	
		dir_vendor_FD_opc	master/slave both	
		dir_vendor_FD_t_bit	master/slave both	
		dir_vendor_FD_def_byte	master/slave both	
		dir_vendor_FD_def_byte_Sr	master/slave both	
		dir_vendor_FD_def_byte_Sr_addr	master/slave both	

Table A-2 (Continued)Coverage

Cover Group	Cover Points	Bins	Present in	Description
		dir_vendor_FD_def_byte_Sr_addr_ack	master/slave both	
		dir_vendor_FD_def_byte_Sr_addr_nack	master/slave both	
		dir_vendor_FD_def_byte_Sr_addr_data	master/slave both	
		dir_vendor_FE_opc	master/slave both	
		dir_vendor_FE_t_bit	master/slave both	
		dir_vendor_FE_def_byte	master/slave both	
		dir_vendor_FE_def_byte_Sr	master/slave both	
		dir_vendor_FE_def_byte_Sr_addr	master/slave both	
		dir_vendor_FE_def_byte_Sr_addr_ack	master/slave both	
		dir_vendor_FE_def_byte_Sr_addr_nack	master/slave both	
		dir_vendor_FE_def_byte_Sr_addr_data	master/slave both	
		rst_with_direct_set grpa_opc	master/slave both	
		rst_with_direct_set grpa_t_bit	master/slave both	
		rst_with_direct_set grpa_t_bit_Sr	master/slave both	
		rst_with_direct_set grpa_t_bit_Sr_addr	master/slave both	

Table A-2 (Continued)Coverage

Cover Group	Cover Points	Bins	Present in	Description
		rst_with_direct_setgrpa_t_bit_Sr_addr_ack	master/slave both	
		rst_with_direct_setgrpa_t_bit_Sr_addr_nack	master/slave both	
		rst_with_direct_setgrpa_t_bit_Sr_addr_data	master/slave both	
		rst_with_direct_rstgrpa_opc	master/slave both	
		rst_with_direct_rstgrpa_t_bit	master/slave both	
		rst_with_direct_rstgrpa_t_bit_Sr	master/slave both	
		rst_with_direct_rstgrpa_t_bit_Sr_addr	master/slave both	
		rst_with_direct_rstgrpa_t_bit_Sr_addr_ack	master/slave both	
		rst_with_direct_rstgrpa_t_bit_Sr_addr_nack	master/slave both	
	rst_after_direct_ccc_end_with_rep_start_7e	rst_after_ccc_setdasa_end_Sr_7e	master/slave both	
		rst_after_ccc_direct_enec_end_Sr_7e	master/slave both	
		rst_after_ccc_direct_disec_end_Sr_7e	master/slave both	
		rst_after_ccc_direct_entas0_end_Sr_7e	master/slave both	
		rst_after_ccc_direct_rstdaa_end_Sr_7e	master/slave both	

Table A-2 (Continued)Coverage

Cover Group	Cover Points	Bins	Present in	Description
		rst_after_ccc_setnewda_end_Sr_7e	master/slave both	
		rst_after_ccc_direct_setmwl_end_Sr_7e	master/slave both	
		rst_after_ccc_direct_setmrl_end_Sr_7e	master/slave both	
		rst_after_ccc_getmwl_end_Sr_7e	master/slave both	
		rst_after_ccc_getmrl_end_Sr_7e	master/slave both	
		rst_after_ccc_getpid_end_Sr_7e	master/slave both	
		rst_after_ccc_getbcr_end_Sr_7e	master/slave both	
		rst_after_ccc_getdcr_end_Sr_7e	master/slave both	
		rst_after_ccc_getstatus_end_Sr_7e	master/slave both	
		rst_after_ccc_direct_setbrgtgt_end_Sr_7e	master/slave both	
		rst_after_ccc_getaccmst_end_Sr_7e	master/slave both	
		rst_after_ccc_direct_entas1_end_Sr_7e	master/slave both	
		rst_after_ccc_direct_entas2_end_Sr_7e	master/slave both	
		rst_after_ccc_direct_entas3_end_Sr_7e	master/slave both	
		rst_after_ccc_getmxds_end_Sr_7e	master/slave both	
		rst_after_ccc_getxtime_end_Sr_7e	master/slave both	

Table A-2 (Continued)Coverage

Cover Group	Cover Points	Bins	Present in	Description
		rst_after_ccc_direct_setxtime_end_Sr_7e	master/slave both	
		rst_after_ccc_gethdrcap_end_Sr_7e	master/slave both	
		rst_after_ccc_direct_endxfer_end_Sr_7e	master/slave both	
		rst_after_ccc_direct_rstact_end_Sr_7e	master/slave both	
		rst_after_direct_vendor_ccc_E0_end_Sr_7e	master/slave both	
		rst_after_direct_vendor_ccc_E1_end_Sr_7e	master/slave both	
		rst_after_direct_vendor_ccc_E2_end_Sr_7e	master/slave both	
		rst_after_direct_vendor_ccc_E3_end_Sr_7e	master/slave both	
		rst_after_direct_vendor_ccc_E4_end_Sr_7e	master/slave both	
		rst_after_direct_vendor_ccc_E5_end_Sr_7e	master/slave both	
		rst_after_direct_vendor_ccc_E6_end_Sr_7e	master/slave both	
		rst_after_direct_vendor_ccc_E7_end_Sr_7e	master/slave both	
		rst_after_direct_vendor_ccc_E8_end_Sr_7e	master/slave both	

Table A-2 (Continued)Coverage

Cover Group	Cover Points	Bins	Present in	Description
		rst_after_direct_vendor_ccc_E9_end_Sr_7e	master/slave both	
		rst_after_direct_vendor_ccc_EA_end_Sr_7e	master/slave both	
		rst_after_direct_vendor_ccc_EB_end_Sr_7e	master/slave both	
		rst_after_direct_vendor_ccc_EC_end_Sr_7e	master/slave both	
		rst_after_direct_vendor_ccc_ED_end_Sr_7e	master/slave both	
		rst_after_direct_vendor_ccc_EE_end_Sr_7e	master/slave both	
		rst_after_direct_vendor_ccc_EF_end_Sr_7e	master/slave both	
		rst_after_direct_vendor_ccc_F0_end_Sr_7e	master/slave both	
		rst_after_direct_vendor_ccc_F1_end_Sr_7e	master/slave both	
		rst_after_direct_vendor_ccc_F2_end_Sr_7e	master/slave both	
		rst_after_direct_vendor_ccc_F3_end_Sr_7e	master/slave both	
		rst_after_direct_vendor_ccc_F4_end_Sr_7e	master/slave both	
		rst_after_direct_vendor_ccc_F5_end_Sr_7e	master/slave both	

Table A-2 (Continued)Coverage

Cover Group	Cover Points	Bins	Present in	Description
		rst_after_direct_vendor_ccc_F6_end_Sr_7e	master/slave both	
		rst_after_direct_vendor_ccc_F7_end_Sr_7e	master/slave both	
		rst_after_direct_vendor_ccc_F8_end_Sr_7e	master/slave both	
		rst_after_direct_vendor_ccc_F9_end_Sr_7e	master/slave both	
		rst_after_direct_vendor_ccc_FA_end_Sr_7e	master/slave both	
		rst_after_direct_vendor_ccc_FB_end_Sr_7e	master/slave both	
		rst_after_direct_vendor_ccc_FC_end_Sr_7e	master/slave both	
		rst_after_direct_vendor_ccc_FD_end_Sr_7e	master/slave both	
		rst_after_direct_vendor_ccc_FE_end_Sr_7e	master/slave both	
		rst_after_ccc_setgrpa_end_Sr_7e	master/slave both	
		rst_after_ccc_direct_rstgrpa_end_Sr_7e	master/slave both	
	rst_after_direct_ccc_end_with_stop	rst_after_ccc_setdasa_stop	master/slave both	
		rst_after_ccc_direct_enec_stop	master/slave both	
		rst_after_ccc_direct_disec_stop	master/slave both	

Table A-2 (Continued)Coverage

Cover Group	Cover Points	Bins	Present in	Description
		rst_after_ccc_direct_entas0_stop	master/slave both	
		rst_after_ccc_direct_rstdaa_stop	master/slave both	
		rst_after_ccc_setnewda_stop	master/slave both	
		rst_after_ccc_direct_setmwl_stop	master/slave both	
		rst_after_ccc_direct_setmrl_stop	master/slave both	
		rst_after_ccc_getmwl_stop	master/slave both	
		rst_after_ccc_getmrl_stop	master/slave both	
		rst_after_ccc_getpid_stop	master/slave both	
		rst_after_ccc_getbcr_stop	master/slave both	
		rst_after_ccc_getdcr_stop	master/slave both	
		rst_after_ccc_getstatus_stop	master/slave both	
		rst_after_ccc_direct_setbrgtgt_stop	master/slave both	
		rst_after_ccc_getaccmst_stop	master/slave both	
		rst_after_ccc_direct_entas1_stop	master/slave both	
		rst_after_ccc_direct_entas2_stop	master/slave both	
		rst_after_ccc_direct_entas3_stop	master/slave both	
		rst_after_ccc_getmxds_stop	master/slave both	
		rst_after_ccc_gettime_stop	master/slave both	

Table A-2 (Continued)Coverage

Cover Group	Cover Points	Bins	Present in	Description
		rst_after_ccc_direct_setxtime_stop	master/slave both	
		rst_after_ccc_gethdrcap_stop	master/slave both	
		rst_after_ccc_direct_endxfer_stop	master/slave both	
		rst_after_ccc_direct_rstact_stop	master/slave both	
		rst_after_direct_vendor_ccc_E0_stop	master/slave both	
		rst_after_direct_vendor_ccc_E1_stop	master/slave both	
		rst_after_direct_vendor_ccc_E2_stop	master/slave both	
		rst_after_direct_vendor_ccc_E3_stop	master/slave both	
		rst_after_direct_vendor_ccc_E4_stop	master/slave both	
		rst_after_direct_vendor_ccc_E5_stop	master/slave both	
		rst_after_direct_vendor_ccc_E6_stop	master/slave both	
		rst_after_direct_vendor_ccc_E7_stop	master/slave both	
		rst_after_direct_vendor_ccc_E8_stop	master/slave both	
		rst_after_direct_vendor_ccc_E9_stop	master/slave both	
		rst_after_direct_vendor_ccc_EA_stop	master/slave both	
		rst_after_direct_vendor_ccc_EB_stop	master/slave both	
		rst_after_direct_vendor_ccc_EC_stop	master/slave both	

Table A-2 (Continued)Coverage

Cover Group	Cover Points	Bins	Present in	Description
		rst_after_direct_vendor_ccc_ED_stop	master/slave both	
		rst_after_direct_vendor_ccc_EE_stop	master/slave both	
		rst_after_direct_vendor_ccc_EF_stop	master/slave both	
		rst_after_direct_vendor_ccc_F0_stop	master/slave both	
		rst_after_direct_vendor_ccc_F1_stop	master/slave both	
		rst_after_direct_vendor_ccc_F2_stop	master/slave both	
		rst_after_direct_vendor_ccc_F3_stop	master/slave both	
		rst_after_direct_vendor_ccc_F4_stop	master/slave both	
		rst_after_direct_vendor_ccc_F5_stop	master/slave both	
		rst_after_direct_vendor_ccc_F6_stop	master/slave both	
		rst_after_direct_vendor_ccc_F7_stop	master/slave both	
		rst_after_direct_vendor_ccc_F8_stop	master/slave both	
		rst_after_direct_vendor_ccc_F9_stop	master/slave both	
		rst_after_direct_vendor_ccc_FA_stop	master/slave both	
		rst_after_direct_vendor_ccc_FB_stop	master/slave both	
		rst_after_direct_vendor_ccc_FC_stop	master/slave both	
		rst_after_direct_vendor_ccc_FD_stop	master/slave both	
		rst_after_direct_vendor_ccc_FE_stop	master/slave both	

Table A-2 (Continued)Coverage

Cover Group	Cover Points	Bins	Present in	Description
		rst_after_ccc_setgrpa_stop	master/slave both	
		rst_after_ccc_direct_rstgrpa_stop	master/slave both	

A.13 Automatic Driving of GETACCMST After MR ACK

The VIP Master schedules a GETACCMST after MR has been ACKED, by default. Therefore, after MR has been ACKED the VIP allows transaction to be scheduled after Repeated START only.

For example,

MR->ACK (by VIP)->Sr->txn1->Sr->txn2…..->Sr->GETACCMST (driven by VIP automatically)

In order to prevent GETACCMST from being scheduled automatically following configuration variable has to be set to 0:

`automatic_getacc_after_MR_ack` : accessible as
`cfg.master_cfg[0].automatic_getacc_after_MR_ack`

Once this variable is set to zero, the control comes back to sequence after MR has been ACKED or NACKED and user can schedule any transaction. If nothing is scheduled the

Master inserts a STOP after MR ACK/NACK.

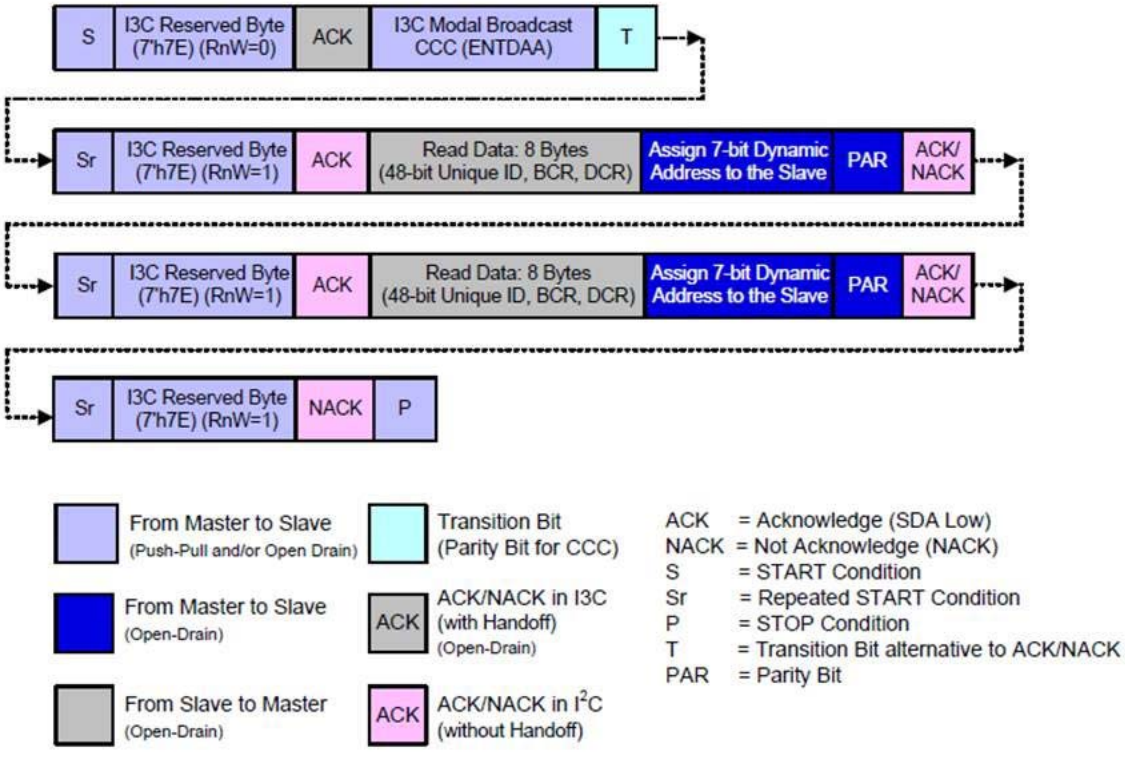
In case Main master is DUT and secondary master is a VIP instance, `automatic_getacc_after_MR_ack` has to be set to 0 for VIP instance as the entire scheduling will be controlled by DUT.

To generate traffic of type MR->ACK/NACK (by main master VIP)->Sr->txn1(driven by main master VIP): txn1 has to be arbitrated with MR. The main master will eventually lose arbitration, drive a *Sr* and send the transaction (txn1) due to which arbitration was lost after MR ->ACK/NACK.

A.14 ACK for DAA During ENTDAA

ENTDAA Implementation in VIP:

Figure A-1 Dynamic Address Assignment Transaction



ACK of 7E+R and Master assigned dynamic address has to be without hand-off. By default, VIP Slave/Secondary masters do a full cycle ACK i.e. drive SDA low for entire C9 duration whereas VIP main Master drives SDA low on posedge of C9 itself once it sees an ACK. This has been done to ensure compatibility with slave IPs doing a half cycle ACK for Master assigned dynamic address.

- `entdaa_in_od` - On setting this configuration, the entire ENTDAA is driven in open drain. Also under this configuration both Master and Slave devices honor full cycle ACK (without hand-off).

Bugzilla for reference: https://bugzilla.mipi.org/show_bug.cgi?id=6034

A.15 IBI ACK Handling in VIP

Default Implementation:

For a Payload capable IBI Slave, the VIP drives SDA to 0 on C9 posedge after it sees an ACK. The Master will however continue to drive the SDA line for entire C9 pulse and will release the SDA line after Negedge of C9.

For Non payload capable IBI ,Master will do a full cycle ACK and release its SDA line after negedge of C9.

Configuration Dependent Behavior:

- `master_ibi_handoff`: On setting this variable to 1, both the Master and Slave honor hand-off type ACK for payload capable IBI.

A.16 I3C Analysis Ports

The following are two types of analysis ports in both the master and slave VIPs:

a. tx_xact_observed_port

b. rx_xact_observed_port

A.16.1 tx_xact_observed_port

The `tx_xact_observed_port` is an analysis port which is driven by configuring the sequencer of the VIP with its transaction class object. Basically, the port provides all transaction class attributes from VIP. The timings of provision of information on the port varies in the case of master and slave VIPs.

- Master VIP: The Master VIP populates the information on the port once the current transaction is completed. In other words, for master population at `tx_xact_observed_port` and `rx_xact_observed_port` happens at same time.
- Slave VIP: The behavior of slave VIP's tx port data reporting has been flagged under a system configuration variable named `tx_port_slv_modify`. Once this configuration variable is set, packet reporting is affected as follows:

Packet Type	cfg.tx_port_slv_modify =0	cfg.tx_port_slv_modify =1
IBI	Transaction packet is always reported on TX port of VIP irrespective of the fact whether arbitration is won/lost by slave requesting IBI. The packet is reported on TX port as soon as it is sent from the sequence	Transaction packet is reported on TX port of VIP only if the slave wins the arbitration and IBI is actually driven on the bus. The packet is reported when a Sr/P is detected after IBI has completed
Private Read	Transaction is reported only when a sequence for directed data is fired on VIP slave. The transaction is reported as soon as it is sent from the sequence.	Transaction is reported on TX when a transaction is fired on slave VIP as well as when slave drives random data in absence of sequence from slave. Transaction is reported when SR/P of detected after Rd transfer is over.
General Reporting	As soon as sequence is fired on slave sequencer, transaction is reported on TX port. Whether slave responds to this packet or not is immaterial. TX port will always be populated instantly	Once a transaction is fired on slave VIP, it gets reported on TX port only when slave VIP participates in bus traffic. Once it gets to participate , packet is reported on TX port once Sr/P is detected.
Reporting for GET-CCC data (GET-CCC data is never configurable from the sequence apart from GETSTATUS)	Only intent possible is NACK for Slave dynamic address driven GET CCC. Packet sent from the sequence is reported on TX port as soon as it is sent from sequence. This is irrespective of the fact whether slave actually participated in CCC and NACKEd its address.	Only intent possible is NACK for Slave dynamic address driven GET CCC. Data driven by slave during GET CCC is NOT REPORTED on TX port. Packet sent from the sequence is reported on TX if slave gets to participate in GET CCC.

Special Scenarios:

1. If slave is configured with nack_addr =1 and nack_addr_count>1, then the packet is reported at TX port when Sr/P is received after slave NACKs for the first time. For subsequent NACKs packets are not reported at TX port.
2. Master driving READ on VIP slave S1 and Slave S1 initiating IBI. In such a case deadlock occurs. But because the Slave VIP never lost arbitration, its TX port is populated. Although the Slave's intention of putting IBI was never met, still TX port is populated as it did not lose arbitration.
3. Master driving Write on VIP slave S1 and Slave S1 is configured for IBI and NACKing its address. Traffic on bus: START -> <S1's dynamic address> +W (S1 was driving RD and lost arbitration at this point) -> (NACK from S1 as it was configured for NACK addr) -> Sr/P. In such a case, VIP slave's intention of NACKing its address is being met but still TX port is not populated because Slave VIP has lost the arbitration.

A.16.2 rx_xact_observed_port

For both master and slave VIP, packet is reported at this port once the current transaction ends.

If an active VIP component requests IBI/MR and wins arbitration. This means if it is able to drive its dynamic address and command bit (1 in case of IBI and 0 in case of MR), then the packet reported at the `rx_xact_observed_port` of that component would have `set_ibi` set to '1'.

The packet that corresponds to IBI/MR can be distinguished using CMD field. It has:

WRITE: for packet corresponding to MR.

READ: for packet corresponding to IBI.

A.16.3 Comparing Slave IBI Payload Data

A system level configuration value, `scbrd_check_for_slv_data` needs to be set for comparing slave IBI payload data. For example:

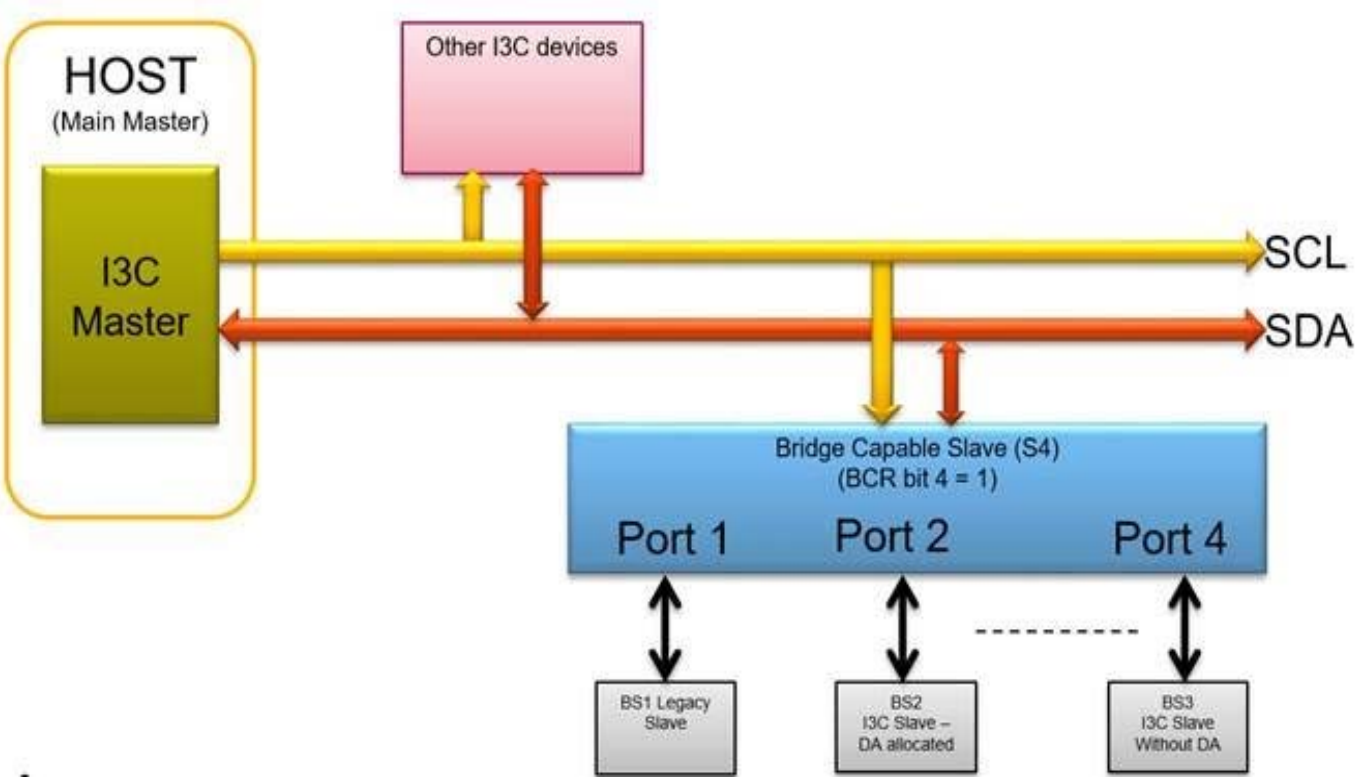
```
`uvm_create_on(slv_tr,p_sequencer.slave_sequencer[1])
    if (!slv_tr.randomize() with
                {
                  slv_tr.set_ibi       == 1;
                  slv_tr.nack_addr     == 0;
                  slv_tr.data.size     == 5;
                  foreach(slv_tr.data[i]) slv_tr.data[i] == i;
                })
                `uvm_error("Randomization Failure","i3c_main_mst_slv_ibi_sequence")
    else
       slv_tr.slv_arbitrate = 0;


      `uvm_send(slv_tr)
```

The above sequence sent to slave (for payload capability BCR[2:1] = 2'b11) results in IBI with a payload of `8'h00,8'h01,8'h02,8'h03,8'h04`. The same payload data can be compared with the corresponding packet received at `rx_xact_observed_port` which also has `set_ibi` set to 1 and the data field having the same payload as was configured from sequence.

A.17 SETBRGTGT CCC USE MODEL

The following illustration shows the SETBRGTGT use model:



These are the two types of slaves present on I3C bus:

- Bridged Slave (BS)
- Bridged Capable Slave or Bridge Device (S4)

1. At any instance, the bridge device might have an allocated DA and can have three types of devices connected to itself, such as legacy I2C slave, I3C slave with allocated DA, and I3C slave without allocated DA. It is also possible to have another bridge device (or other types of slaves, such as SPI and so on) connected to the S4.
   a. All devices in the system (including S4 and BS* devices) needs to have a unique BCR-DCR-PID triplet. Though the BS* devices do not participate during ENTDAA, these are needed for ultimate DEFSLVS support for sec-masters.
2. All communication between I3C master and S4 bridge are recommended to be SDR (either broadcast or using DA of S4/BS2). By doing that, BS1 has a hidden slave, and the main master is never aware of it at any instance during simulation. It shall be the responsibility of S4 to communicate with BS1/BS3 using ongoing SDR traffic between I3C master and S4.
   a. Once the DA of legacy BS and I3C BS devices are deallocated, the VIP still allows use of static address of BS1/BS3, if required. This does not restrict any of the other discussed functionalities of VIP.

3. All functional attributes of S4 & BS* devices can be chosen by user to be kept identical for proposed use case. However, you can choose to keep different attributes among them as well. There is a SINGLE rule for MRL/MWL among such devices which specifies that the attempted length must be lesser (or equal) than both the corresponding BS devices and associated S4 device.
4. If S4 device gets de-allocated for address (currently, through D-RSTDAA), it is possible for I3C master to access BS2 device through SDR messaging on bus.
5. The IDX0-1 is a build-time configurable item, unique for each device. I3C master (VIP) is allowed to drive user-defined DA values and IDX0-1 values during `SETBRGTGT`.
   a. The IDX0-1 values can be decided upon by you at `SETBRGTGT`, however they must be among the configured values at build-time.
   b. Across bus RESET, the associativity of BS devices with S4 are reset. The associativity stands to be re-established using `SETBRGTGT`, that allows any combination of BS-S4 associativity across bus RESET. This indicates that any check of associativity is between two contiguous bus RESET and not across them.
6. The addresses (DA) of BS shall be unique across
   a. Two or more Bridge devices
   b. With respect to other I3C devices.
7. The `SETBRG CCC` only acts as medium to allocate the first DA for a BS device. Subsequently, that BS device is accessible through regular CCC flow for any updates in DA values.

Note
Only I3C slave can behave as Bridge Devices.

The newly added rules for `SETBRGTGT CCC` are the following:

1. `setbrg_ccc_on_nonbrdg_dev`
2. `pvt_rd_len_exceed_brdg_error`
3. `pvt_wr_len_exceed_brdg_error`
4. `non_bs_dev_setbrg_error`
5. `idx01_config_msmtch_error`
6. `idx01_brdge_assoc_error`
7. `brdge_dev_as_bridged_slave_error`
8. `brdged_slv_triplet_match_error`
9. `alrdy_alloc_da_trplt_match_error`

Refer the `main_mst_setbrgtgt_test` in TESTBENCH `tb_i3c_svt_uvm_basic_sys` for usage details.

A.18 VENDOR SPECIFIC CCC

1. Defined range:

   0x61-0x7F : Applicable range for vendor defined broadcast CCC

   0xE0-0xFE : Applicable range for vendor defined direct CCC

   CCCs marked as "Reserved by MIPI" will not be applicable for vendor defined CCC support.

Figure 35 CCC Broadcast General Frame Format

2. For Broadcast/Directed CCC:

   Trans-class field (`data_bytes_in_vendor_ccc`) is added to indicate if data bytes are to be driven by Master VIP after CCC opcode and before Repeated Start.

Figure 36 CCC Direct General Frame Format

3. For directed SET CCC: Trans class field(`data_bytes_in_vendor_ccc`) has been added to indicate if data bytes are to be driven by Master VIP after slave address.
4. Number of data bytes returned by Slave VIP during directed GET vendor specific CCC is sequence controlled. Therefore, master error type M0 is not applicable for such CCCs for Master VIP.
5. Directed Vendor specific CCC will not qualify for S5 error from slave VIP. Driving of CMD bit after dynamic address is sequence controlled.
6. Any CCC specific rules is not added for vendor specific CCCs.

   For possible use case refer to:

   ```
   tb_i3c_svt_uvm_basic_sys/tests/ts.svt_mipi_i3c_setdasa_vendor_ccc_brdcst_virtual_test.sv
   tb_i3c_svt_uvm_basic_sys/tests/ts.svt_mipi_i3c_setdasa_vendor_ccc_direct_virtual_test.sv
   ```

   Libraries added for vendor specific CCC:

   a. `svt_mipi_i3c_vendor_ccc_opcode_sequence`: This library is used to send vendor specific CCC (broadcast or direct) with or without defining bytes.

b. `svt_mipi_i3c_vendor_ccc_data_bytes_sequence`: This library is used to send data bytes for direct vendor CCC.

Library added for Slave data :

`svt_mipi_i3c_slave_user_defined_data_sequence` : This library is used to send user defined read data bytes from slave.

Library added for Slave ibi with payload :

`svt_mipi_i3c_slave_user_defined_data_sequence` : This library is used to send payload with ibi from slave from the same packet. For this library to be used, system configuration `scbrd_check_for_slv_data` needs to be set from the test.

A.19 RSTACT CCC and Slave Reset feature

A new master transaction class variable `drive_reset_pattern` is added to drive Slave RESET pattern from Main Master VIP. RSTACT CCC support is added in all VIP components.

1. The default value of RSTACT defining byte stored in Slave and Secondary Masters is 8'h01. VIP supports driving of defining bytes 8'h00:8'h04, 8'h40, 8'h7F (both SET/GET) and 8'h81:8'h84, 8'hC0:8'hFF (only GET) based on this table.

Table 46 RSTACT Defining Byte Values

Defining Byte Value	Description	NACK Means Default Of	Notes	SET / GET
0x00	**No Reset on Slave Reset Pattern**	–	**SET:** No data (Defining Byte is the value) **GET:** Returns currently set Defining Byte value (i.e., previously received in Direct SET) or 0x00	SET / GET
0x01	**Reset the I3C Peripheral Only** (Default)	–		SET / GET
0x02	**Reset the Whole Slave**	–		SET / GET
0x03	**Debug Network Adaptor Reset** (The Slave shall not reset the I3C Peripheral)	Debug for I3C not supported		SET / GET
0x04 (Direct CCC only)	**Virtual Slave Detect** • SET: The Slave shall not prepare for reset, but shall set a flag in its shared Peripheral logic that can be read by all other linked Virtual Slaves. This flag can be cleared with Defining Byte 0x00 (Direct or Broadcast) • GET: The Slave shall return 0x01 if the flag is set, i.e., if a previous SET operation was directed to any linked Virtual Slave in this Device (including itself), without an intervening use of Defining Byte 0x00. The Slave shall return 0x00 if no SET was received, or if the flag was cleared by the RSTACT CCC with Defining Byte 0x00 (No Reset on the Slave Reset Pattern)	Slave is not VS-capable		SET / GET
0x05 to 0x3F	Reserved by MIPI	–		–
0x40 to 0x7F	Reserved for Vendors and external standards	–		–
0x80	Reserved by MIPI	–	**SET:** Ignored (not used) **GET:** Returns the time, based upon the Defining Byte	–
0x81	**Return Time to Reset Peripheral**	1 ms		GET
0x82	**Return Time to Reset Whole Slave**	1 s		GET
0x83	**Return Time for Debug Network Adaptor Reset**	100 ms		GET
0x84	**Return VS Indication** Return 0x01 if this Slave is a Virtual Slave and supports Defining Byte 0x04 (Virtual Slave Detect). Otherwise return 0x00.	Slave is not VS-capable		GET
0x85 to 0xBF	Reserved for timing for MIPI reserved values	–		GET
0xC0 to 0xFF	Reserved for timing for Vendor and external standards reserved values	–		GET

2. When defining byte 8'h00-8'h02 is driven with READ command in directed RSTACT, then slave always returns the Defining byte value stored in it.

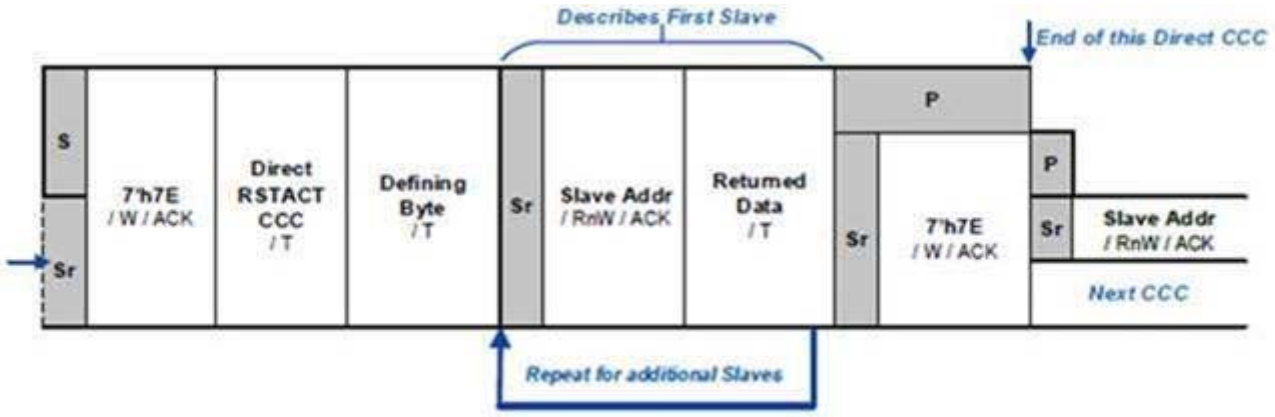


Figure 76 RSTACT Format 3: Direct Read

3. Since the slave is supposed to cancel the RSTACT CCC on seeing the START (not repeated START), the defining byte value configured using RSTACT CCC (direct/broadcast) is stored only till START. At every START all the slaves reset the defining byte value to 8'h01.
4. Once reset pattern is detected by the ACTIVE VIP slave, it will:
   a. Trigger an event to indicate that RESET pattern has been detected.
   b. Indicate the defining byte stored in it.
   c. Indicate whether the RSTACT CCC was directed to it or not. This bit type variable will always be set in case of B-RSTACT and only if dynamic address of slave is driven during Directed RSTACT.

      All the variable/events used for points 4a,4b and 4c above will be accessible to the user from testbench.

Configuration Value:

1. VIP components does not take any action automatically on receiving slave RESET pattern. At run time, VIP components will need to be informed about the action to be taken after receiving RESET pattern.
2. A new system configuration function `recycle_addresses_after_reset_pattern` is provided. After RESET pattern has been driven, the build time configured value of dynamic addresses of the components that need to be RESET needs to be populated in a dynamic array and this array needs to be passed to this system configuration function. The VIP then automatically clears the given dynamic addresses.
3. VIP method `reconfigure_via_task` (<cfg handle with dynamic array populated>) then needs to be called for all VIP agents (active/passive).
4. Configuration variables corresponding to time to reset peripheral and time to reset whole slave respectively are also be added. These variables are:

   `a. time_to_reset_peripheral`

b. `time_to_reset_whole_slave`

Active VIP slave returns these values in response to defining bytes 0x81 and 0x82 during directed RSTACT.

Note

If S0 or S1 error is inserted during RSTACT CCC, then RESET pattern has to be followed by exit pattern, if all devices were not reset using `reconfigure_via_task`, and broadcast CCCs have to be used after RESET pattern.

No exit pattern is required if all the I3C devices except the Main master were reset using `reconfigure_via_task`.

For test example, refer:

Test: ts.`mipi_i3c_svt_rstact_ccc_and_slave_reset_pattern_test.sv`

TB: `tb_i3c_svt_uvm_basic_sys`.

A.20 I3C Basic V1.0 Feature Set

A MACRO based approach is used in the VIP to exclude the advanced features from VIP. Only the basic functionality from v1.0 is supported from VIP.

Following are the things which are restricted or allowed by the VIP:

1. Master VIP:
   a. All restricted CCCs are still allowed under the mentioned switched if configured by user.
   b. HDR traffic is restricted to ENTHDR* opcode followed by an EXIT pattern from VIP, sent automatically.
2. Slave VIP:
   a. The VIP chooses to ignore sampling/driving of data post opcode, for all restricted CCCs, under the switch.
   b. The VIP continues to detect START-STOP post ENTHDR* opcode. This could lead to hang like scenario if user deliberately fires HDR traffic under the switch, recommended to NOT initiate such traffic. The VIP continues to enabled for detecting EXIT pattern to come out of HDR mode.
3. Checker: The checker restricts features outside I3C-Basic V1.0 under the switch, errors to follow if such traffic attempted (CCCs, HDR traffic).
4. Coverage: The feature coverage outside i3C Basic v1.0 ceased to be created under the switch.

All of this functionality is put under a macro `IS_I3C_BASIC_SYSTEM`.

A.21 Driving of forced EXIT pattern from Master VIP and Detection from Slave-VIP

New capability added in MASTER-VIP that if an HDR-TSP/TSL Read is driven, than master can choose to do the following:

1. It can simply choose to complete the EXIT pattern which was initiated by slave. (default behavior)
2. It can force its EXIT pattern after waiting for 2*tsymbol time knowing that the slave has release SDA-SCL after driving all relevant symbols. (new behavior)

All this decision making is done with help of newly added trans-class variable `master_force_exit_pattern_during_ternary_tx`.

This trans-class variable is added to determine if exit pattern is forced or not.

Possible values of `master_force_exit_pattern_during_ternary_tx`:

1'b0	1'b1
No change in behavior of VIP. The Master will not drive force exit pattern in TSP/TL read. It completes the EXIT pattern which is initiated by Slave/sec-mst.	The Master VIP acts differently from the normal behavior and rather than completing EXIT pattern initiated by slave/sec-mst, it drives its own EXIT pattern after waiting for 2-symbol time, when the slave/sec-mst stop driving the bus.

Note

If the above trans-class variable is set, then MASTER-VIP would always force exit pattern. Even if sr_or_p_gen =1, it does not drive RESTART pattern, but it drives EXIT pattern.

Detection of forced Exit pattern from slave/sec-mst:

Similarly, new capability has been added to SLAVE/SEC-MASTER VIP that if an HDR-TSP/TSL Read transfer is going on, then the slave will also be able to detect the forced EXIT pattern which was driven by Master rather than completing the remaining EXIT pattern.

The detection of forced EXIT pattern is done when Master transmits command and address. The slave will always look for bus idle for 2*tsymbol time after frame is finished. If bus is idle for 2*tsymbol time and slave detects complete EXIT pattern after that, then it will recover from it.

Possible scenarios:

Scenario 1: When Master is sending HDR-TSP/TSL mode with WR command without any parity or symbol error.

If a valid address has been driven by Master, then that Slave samples the data and detects the EXIT/RESTART pattern on frame boundary.

Scenario 2: When Master is sending HDR-TSP/TSL mode with WR command with either parity or symbol error.

Since it is a WR command, the slave does not drive anything and wait for EXIT/RESTART pattern at the frame boundary for recovery.

Scenario 3: When Master VIP is sending HDR-TSP/TSL mode with RD command without any parity or symbol error without setting transaction class property `master_force_exit_pattern_during_ternary_tx` (default behavior).

The Master VIP works normally and does not drive force EXIT pattern. It waits for the slave to drive its initial pattern and then complete the EXIT pattern.

Scenario 4: Master VIP is sending HDR-TSP/TSL mode with RD command with setting transaction class property `master_force_exit_pattern_during_ternary_tx` =1 (irrespective of parity/symbol in data from slave).

The Master VIP does not continue completing the rest of EXIT pattern from its side in continuity to EXIT pattern initiation of the Slave. The Master VIP always sends forced EXIT pattern subsequently in entirety after waiting for twice of symbol time.

The Slave VIP in corresponding will check if any forced EXIT pattern is driven. If yes, then it will recover from it.

Scenario 5: When Master VIP is sending HDR-TSP/TSL mode with RD command with parity error

If a parity error is detected, the Slave just NACK it by sending its initial pattern. This is an active VIP fix done from P-2019.03-2 release onwards against STAR- 9001485942. Initially the VIP used to respond with data for such scenario, which was incorrect.

Scenario 6: When Master VIP is sending HDR-TSP/TSL mode with RD command with symbol error in address (such that the address does not matches with any component on bus)?

The Slave VIP goes into undefined state or hangs if any symbol error is detected. This is expected because the slave VIPs on bus does not automatically respond either ACK/NACK or send data for such mismatching address.

A.22 SETDASA-P2P

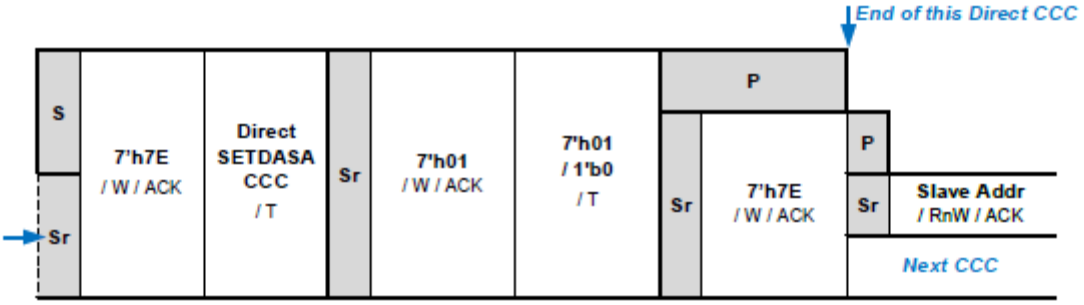


Figure 51 SETDASA Format 2: Point-to-Point

An I3C Master shall not use this special form of the SETDASA CCC unless the I3C Bus is known to have precisely one Master and either:

1. One active I3C Slave Device that is capable of supporting this special usage of the SETDASA CCC, (D-1 device type) or
2. One or more I3C Slave Devices that act strictly as receivers, i.e., that do not use In-Band Interrupt and that will not be sent Read requests. This arrangement permits a single Master Device to, in effect, Broadcast to the Slave Devices. (D-2 device type)

Apart from the devices mentioned above, other devices can co-exist on the bus with D-1 and D-2.

D1:

- Only single instance of this type of device can be present on the bus.
- No device of type D2 can co-exist with D1.
- This type of device can support Private WR/RD, HJ/IBI/MR, CCC.

D2:

- Multiple instance of D2 can exist on the bus.
- No device of type D1 can co-exist on the bus with device type D2.
- This type of devices will not support IBI/MR/HJ/Private RD/GET CCC.

VIP devices are distinguishable from devices supporting P2P mode using an agent configuration value:
bit [1:0] i3c_p2p_bus_type = 2'b00;

2'b00: P2P support is disabled.
2'b01: P2P support has been enabled and only single Main master and one VIP Slave (or sec master exists on bus). This value should be set for only ONE non-MAIN MASTER VIP instance. (D1)
2'b10: P2P support has been enabled and single Main master and multiple VIP slave (or sec master exist on bus). All the devices for which these variables have been set will act strictly as receivers. They will not support IBI/MR/HJ/Read commands. (D2)
2'b11: Invalid value.

In addition to above agent cfg a system level `cfg is_p2p_device_present` has to be enabled.

If any device with P2P support is present on the bus, this system level cfg must be set.

SETDASA/RSTDAA:

SETDASA needs to be fired for only once for static address 'h01. If this address is ACKed, the only possible dynamic address that can follow is 'h01 itself. This dynamic address will be allocated for all the devices supporting P2P mode.

Similarly, in directed RSTDAA if 'h01 address ACKed, Master will assume that all the devices supporting P2P will reset their dynamic address assignment.

All the P2P devices will either have their dynamic address allocated or in reset state.

SETAASA:

P2P devices will support SETAASA CCC.

ENTDAA:

Devices of type D1 can be allocated dynamic address using ENTDAA.

Devices of type D2 can't participate in ENTDAA because they act strictly as receivers. Basically, D2 devices will always NACK the 7e/Read.

SETNEWDA:

Since dynamic address 'h01 is reserved for P2P mode, therefore dynamic address of P2P devices can't be changed to any other value other than 'h01. Their dynamic address can only be reset.

Thus, both D1 and D2 type of devices will always NACK their address('h01) during SETNEWDA.

HDR WR/RD:

Device type D1 will support DDR/TSP/TSL write and read operations.

Device type D2 will support DDR/TSP/TSL write commands. Such devices will NACK DDR read commands.

TSP/TSL read commands should not be directed to such devices.

IBI/MR/HJ:

Device type D1 will support both IBI/MR/HJ.

Device type D2 will always drop the IBI/MR/HJ request and the request will never be seen on the bus.

GET-CCC's and READ command:

Device type D1 will support both Read command and all the GET ccc's.

Device type D2 will just NACK at their address during read command and GET ccc's.

The checker does not flash errors currently for the following scenarios:

a. If any ACK comes for scenario mentioned in (i, ii, iii) above.
b. SETNEWDA not allowed on the devices which support any type of P2P mode.
c. The ACK response for address in SETNEWDA not allowed.

These checks will be added in the upcoming releases.

Note

VIP instance configured as Main Master at build time can never be of D1 or D2 type.

Note

The configuration parameters is_p2p_device_present and i3c_p2p_bus_type are not allowed to change dynamically i.e., they are not allowed inside `reconfigure_via_task`.

Note

SETDASA-P2P is not supported for devices VIP configured as bridged slaves.

The following scenarios with configuration parameters are invalid and would lead to fatal:

a. If the value of system cfg is_p2p_device_present = 0, then the non-zero value of agent cfg i3c_p2p_bus_type of any component is not allowed.
b. If the value of system cfg is_p2p_device_present is non-zero i.e., SETDASA-P2P mode is ON, then at least for one component on bus, the value of agent cfg i3c_p2p_bus_type should be non-zero.
c. If the value of system cfg is_p2p_device_present = 1, then the non-zero value of agent cfg i3c_p2p_bus_type is not allowed for the component configured as main-master or legacy I2C slave.
d. Device type D-1 and D-2 are not allowed to present on the bus together.
e. The configured address for P2P device must always be 'h01.
f. The configured address for non P2P device must never be 'h01.
g. Device type D-2 not allowed to be configured for Hot join device.

A.23 Driving strengths of SDA

1. HDR TSP-TSL traffic: During this mode, the symbol identification is done using SCL-SDA transition in pair. That condition allows a possibility for SDA to change its strength (for example, release/start-driving bus by changing strength, say STRONG-1 <-> WEAK-1 etc). The VIP during TSP/TSL keep sampling any change in SDA-SCL pair for anew symbol. The issue is, that the strength change also gets sampled as an SDA transition, indicating a new symbol, which is not correct. So it implies that the mentioned macro should not be used in TSP/TSL modes, when a strength change from master is possible. Runtime strength change on SDA during TSP/TSL is not supported under the switch.
2. HDR-DDR traffic: During HDR-DDR, traffic is not affected with SDA strength change because in DDR traffic value of SDA matters at clock edge.
3. SDR-traffic: During SDR traffic, except for S/SR/P, all SDA transitions are bound to happen in SCL-low duration. While using above switch during SDR, the master never looks to drive a strength toggle among STRONG-1 <-> WEAK-1, during SCL high duration. The switch in question can be used in SDR without any issues.

   Conclusion: The switch `SVT_I2C_SLV_SDA_STRENGTH` can be used for SDR and HDR-DDR traffic, but not HDR-TSP/TSL traffic.

A.24 Disabling I2C SVT VIP

1. By default, when the run time command argument - `svt_i2c_disable_vip` is not used, the I2C VIPs remain enabled. In general, you must use the VIP normally. In the scenario, where the AGENT_ID parameters in Verilog top and num_of_masters/num_of_slaves in sys_cfg are not used in the recommended manner, you will see a fatal error.

   If you want the VIP disabled, when not using command arg - svt_i2c_disable_vip, you must ensure to:

   a. Set the Verilog wrapper instance parameter `SVT_I2C_ENABLE_WRAPPER_CONNECTION_CHECK` to 0 in the Verilog top.

   b. Do not create the `uvm_agents` in the environment file.
2. When you choose to exercise the argument - `svt_i2c_disable_vip`, the following are the possibilities:

   a. When `svt_i2c_disable_vip`=0, the VIPs remain enabled. In this case, if the AGENT_ID parameters in Verilog top and num_of_masters/num_of_slaves in sys_cfg are used in the recommended manner, the simulation proceeds without issues. If you deviate from the recommended usage of the above params/sys-cfg-vars, you will see the fatal error mentioned in #1.

   b. When `svt_i2c_disable_vip`=1, the VIPs get disabled. In such a scenario, the user environment must ensure not to create the uvm_agents for I2C VIP. A public example is already available for that (tb_i2c_svt_uvm_disable_vip).

A.25 Inserting Exit Pattern Between SDR Transactions

A callback mechanism is provided in Master VIP to insert EXIT PATTERN during I3C-SDR mode, where I3C Master would deliberately insert exit pattern in between the transactions.

Callback used: `I3C_ILLEGAL_EXIT_PAT_DURING_SDR` (for all callbacks present see Table 6-1).

These are the variables used to insert exit pattern at a position:

1. `insert_exit_pat_in_byte`

   This variable is used to insert the Exit pattern at a particular byte of a `FRAME` during SDR- WR and CCC driving.

   For complete use-model, refer this testcase:
   ts.i3c_main_mst_insert_exit_pat_callback_at_various_location_test.sv in testbench:
   tb_i3c_svt_uvm_basic_sys

   The possible values must be greater than or equal to 1.

   1: Insert Exit pattern in the 1st byte of a frame (in the address phase). This condition is special, and the callback may not be driven based on the scenario. For more information, see the *Rules to be followed while inserting callback* sub-section.

   2: Insert Exit pattern in the 2nd byte of a frame (Write during SDR-WR and CCC opcode during CCC)

   3: Insert exit pattern in the 3rd byte of the frame and so on.

Note

a) The Directed CCC is divided into 2 packets (or more based on multiple addresses) i.e., 1st packet for CCC opcode, 2nd packet for directed-slave address and any data if required. The variable `insert_exit_pat_in_byte` is configured differently for all packets. For example, if Directed-DISEC is sent, and we want to insert EXIT pattern in slave address phase (which is sent in 2nd packet), then `insert_exit_pat_in_byte` is configured as 1 for 2nd packet (for actual usage, one can also look into the testcase.
b) The variable `insert_exit_pat_in_byte` can only be 1 if the transaction is fired after repeated Start and it would not take any effect after Start. Since the first byte after the Start qualifies for arbitration and won by any other slave device. This means that the Current Master would lose the control of SDA line and the EXIT PATTERN is not inserted in that byte.

2. `insert_exit_pat_at_pos`

   This variable is used to insert the Exit pattern at a position or bit of a byte

   The possible values can be from 1 to 7.

   1: After 1st bit of any particular byte is transmitted on the bus, exit pattern will be driven.

   2: After 2st bit of any particular byte is transmitted on the bus, exit pattern will be driven.

   and so on.

3. `insert_exit_pat_with_stop`

   This variable decides if the EXIT Pattern must be followed by STOP or not.

   0: No Stop after EXIT pattern will arrive and the transaction will continue normally.

   1: Stop will follow and the transaction abruptly terminates in between.

Rules to be followed while inserting callback

The complete exit pattern needs to be driven from Current Master, and therefore the control of SDA must be with the current master at that instance.

During the Pvt. Write command, CCC's where only master is driving (ex, SETDASA, SETAASA etc.), one can drive exit pattern at any place except the arbitration header (more info below).

During Pvt. Read command, Direct GET CCC's (or the CCC's where component other than current master is driving at some point), IBI, Mastership, Hot-Join and during arbitration the control goes with the other devices. Hence, the current Master would not be able to drive exit pattern using callback. However, during direct-GET ccc, the control of the SDA for driving the CCC part, slave address part is with the current master, meaning the master can drive the exit pattern in these places.

During arbitration header(after start), where arbitration between the current master and other devices are possible, the current master might lose arbitration. To avoid this possibility, post Start (during arbitration header- possibility of arbitration), the master would never drive exit pattern even if it is configured. In such cases, the VIP gives a warning that exit pattern is driven at arbitration header and you must change the byte position.

To see the use case, refer:

TESTCASE: ts.i3c_main_mst_insert_exit_pat_callback_at_various_location_test.sv

TESTBENCH: tb_i3c_svt_uvm_basic_sys

A.26 Ending ENTDAA Without Assigning DA to all Devices

For Dynamic address assignment, ENTDAA ccc is used. In this ccc, the Current Master assigns dynamic address based on the triplet {PID, BCR, DCR}.

These are two variations in which current master can end this ccc:

1. When current master sees a NACK for 7E/READ.
2. When Current master receives ACK, it can simply choose to issue a stop.

In the first case, you must issue an ENTDAA and the VIP automatically assigns address to all devices present on the bus.

In the second case, a trans-class variable `entdaa_device_count` is provided. This variable must be used with ENTDAA ccc packet.

Consider that there are 'n' components present on the bus, except Main master. The possible values of `entdaa_device_count` are:

a. `entdaa_device_count` > n: The Current Master completes all the DAA for all devices and exits with 7E/READ -> NACK -> Stop.

b. 0 < `entdaa_device_count` < n: The current master completes DAA for number of devices configured in `entdaa_device_count` and exits with stop. The address is not assigned to the remaining devices.

c. `entdaa_device_count` = 0: The current master cannot assign address to any device on the bus. Therefore, if `entdaa_device_count` is configured as 0, it assigns address to 1 component and then stops.

d. `entdaa_device_count` <0: This scenario is same as the first case.

For example, refer

Testcase: i3c_entdaa_device_count_test

Testbench: tb_i3c_svt_uvm_basic_sys

A.27 Driving Traffic in FM/FM+ for Devices with Spike Filter Enabled

According to protocol section 5.1.2.1, the slave devices (capable of acting as I2C slaves) when operating on I3C BUS, may have an I2C FM/FM+ spike filter, which it shall disable once it sees a

Start, 7'h7E on bus. This means that in such cases the Master must be capable of driving the Start, 7'h7E in FM/FM+ mode also.

This means that, if slave device has spike filter enabled, after Start, 7'h7E arrives on bus, the filter must be disabled. The Master has to make sure that the driving of Start, 7'h7E should be in FM/FM+ mode, if the slaves with spike filter are present. Later, it shall move the traffic in default mode (can be either FM/FM+, OD/PP depending on the traffic).

A system configuration (`en_i2c_mode_before_7e`) is provided in the VIP to handle such scenarios where the slaves are expecting the Start, 7'h7E in FM/FM+ mode. Enabling this configuration means that the Master VIP will be able to send the Start, 7'h7E in FM/FM+ mode and the Sec-masters and slave VIPs will be able to detect them.

This table summarizes the possibilities with configuration parameters (`set_od_pp_bus_speed` and `en_i2c_mode_before_7e`) and the VIP behavior:

Set_od_pp_bus_speed	en_i2c_mode_before_7e	Remarks
0	0	Default behavior, the VIP moves all I2C and I3C traffic in FM/FM+ only.
1	0	Default behavior, since the config `en_i2c_mode_before_7e` is disabled the VIP moves all I2C traffic in FM/FM+ and I3C traffic in OD/PP mode.
0	1	Default behavior, since all the traffic will move in FM/FM+ mode only, enabling `en_i2c_mode_before_7e` does not matter.
1	1	The speed mode till Start, 7'h7E would be in FM/FM+ mode and rest would be according to I3C specification (for details see below).

When both the system-configuration (`en_i2c_mode_before_7e`) and agent-configuration (`set_od_pp_bus_speed`) are enabled, I3C VIP behaves the following way:

1. The Master-VIP sends all the traffic in FM/FM+ mode until Start, 7'h7E (this includes all the I2C and I3C transfers).
2. The Master-VIP changes the speed from FM/FM+ to OD/PP at 7th SCL rising pulse (here complete 7'h7E is transmitted on the bus).
3. After detection of Start, 7'h7E, all of the VIP components (accept legacy I2C components) will transfer their speed mode from FM/FM+ to OD/PP automatically and disable this feature.
4. The components enable this feature again after a dynamic reset. This means that all the above-mentioned points are applicable after dynamic reset too.
5. This system config `en_i2c_mode_before_7e` cannot be changed during run-time. This indicates that it is not reconfigurable.
6. If the bus is found pure by the current master, then the transaction always moves in OD/PP mode irrespective of the configuration `en_i2c_mode_before_7e` is enabled or not.

Note
Definition of pure bus assumed by VIP: If there are no legacy I2C devices and no I3C devices with static address present no the bus, then the bus is defined as pure bus.

Enabling this feature (both above mentioned configurations are enabled) with Hot-Join devices present:

If any hot-join device is present on the bus, which means that if any VIP slave is configured as hot-join, then hot-join request by any VIP slave ('h02) is according to OD/PP speed mode irrespective of config `en_i2c_mode_before_7e` enabled or not.

This means, if HJ request arrives before Start, 7'h7E on the bus, the timings are mismatched. Since Master VIP drives the SCL in FM/FM+ mode, while the HJ request by slave follows OD/PP timings. This is problematic and multiple timing errors are reported. It is recommended to send any hot-join request after Start, 7'h7E is sent on the bus. This ensures correct timings.

A.28 Updates in CCCs according to latest MIPI I3C specifications

These are the CCCs which have been enhanced based on the latest MIPI I3C specifications:

1. GETCAPS Format-1 support with variable data size:

   To enable the functionality of GETCAPS ccc, the system configuration parameter `enable_getcaps_ccc` must be set. If this configuration is disabled, then all the components Main-Master, Sec-masters, Slaves would interpret CCC opcode as GETHDRCAP ccc and works accordingly.

Note
This configuration is added to support both GETCAPS and GETHDRCAP in VIP. You can choose to use whichever you want based on your requirements.

Slave and Sec-Master components are equipped to send variable number of data bytes during Format-1 of GETCAPS ccc. You can configure the agent configuration (getcaps_num_of_bytes) of slave/sec-master components to send the number of data bytes.

Possible values of getcaps_num_of_bytes:

- 0: Slave component will only send GETCAP1_byte. (default value)
- 1: Slave component will send GETCAP1_byte and GETCAP2_byte.
- 2: Slave component will send GETCAP1_byte, GETCAP2_byte and GETCAP3_byte.
- 3: Slave component will send GETCAP1_byte, GETCAP2_byte, GETCAP3_byte and GETCAP4_byte.

The GETCAP1_byte, GETCAP2_byte, GETCAP3_byte and GETCAP4_byte can also be configured using the agent configurations (hdrcap_modes, getcaps2_byte, getcaps3_byte, getcaps4_byte) respectively.

2. GETCAPS Format-2 support with defining bytes:

   All components including Main-Master, Sec-masters, Slaves are equipped to handle GETCAPS Format-2 support.

   To drive defining byte from Current Master, trans-class variable `drive_def_byte_after_ccc` must be enabled in Master.

   To drive the value of defining byte, trans-class variable data must be used.

Slave/Sec-master components requires prerequisite that getcaps3_byte[3] should be enabled otherwise slaves will NACK the Format-2 GETCAPS automatically.

These are the agent configurations required during Format-2:

a. getcaps_num_of_bytes, hdrcap_modes, getcaps2_byte, getcaps3_byte, getcaps4_byte are required during defining byte SLVCAPS (this is similar to Format-1).

b. mstcaps_num_of_bytes, mstcaps1_byte, mstcaps2_byte are required during defining byte MSTCAPS.

c. vscaps_num_of_bytes, vscaps1_byte, vscaps2_byte are required during defining byte VSCAPS.

d. Variable number of data bytes could be sent from slave/sec-master component during DBGCAPS, Vendor defining bytes using the trans-class variable `data`.

Note
For more details on configuration related variables, refer SVDOC.

Note
The current support for GETCAPS bytes in the I3C VIP are realized only for getcaps2_byte[5:4] (group addressing capabilities), getcaps3_byte[3] (defining byte support for getcaps) and getcaps3_byte[4] (defining byte support for getstatus). The other getcapx_byte, mstcapx_byte and vscapx_byte are not yet supported. Currently the mentioned bytes are supported in the sense that slaves would return these bytes in GETCAPS CCC.

3. GETSTATUS Format-2 support with defining bytes:

   All components including Main-Master, Sec-masters, Slaves are equipped to handle GETSTATUS Format-2 support. To drive defining byte from Current Master, the trans-class variable `drive_def_byte_after_ccc` must be enabled in Master. To drive the value of defining byte, trans-class variable `data` must be used. The Slave components would respond with the data which will be configured in the trans-class variable `data`. The slaves can send any amount of data for vendor specific defining bytes.

4. GETMXDS Format-3 support with defining bytes:

   All components including Main-Master, Sec-masters, Slaves are equipped to handle GETMXDS Format-2 support. To drive defining byte from Current Master, the trans-class variable `drive_def_byte_after_ccc` must be enabled in Master.

   To drive the value of defining byte, the trans-class variable `data` must be used.

   The Slave components would respond with the data according to defining byte value.

   a. For defining byte `WRRDTURN`, the slaves would respond with data similar to Format-1 and 2 (Same configuration parameters are required. For more information on Format-1 and Format-2, see `testcase ts.getmxds_resp_with_turnaround_test.sv` in testbench `tb_i3c_svt_uvm_basic_sys`).

   b. For defining byte `MSTHDLY`, the slave components which are master capable would respond with 1 byte of data which can be configured with the trans-class variable `data`.

   Slaves can send any amount of data for vendor specific defining bytes.

A.29 Group Addressing

Four CCC's support added in Master and Slave VIP for group addressing:

1. SETGRPA

2. DEFGRPA
3. Direct RSTGRPA
4. Broadcast RSTGRPA

   SETGRPA CCC can only be run on a device which has its dynamic address allocated.

   Slaves/Sec-Masters can have 0, 1, 2 or multiple group addresses based on the number of group addresses allowed in GETCAPS byte2[5:4], which can be set using agent configuration `getcaps2_byte`. The master can write to group addresses but cannot send READ commands to group addresses.

   SETNEWDA and SETBRGTGT cannot be directed to a Group address.

   For DEFGRPA, two trans-class variables are added:

   a. group_addr_for_defgrpa: 7 bit variable to hold group address value.

   b. group_descr: 8 bit variable to hold group descriptor value.

   Group addresses would be deleted for a device if the dynamic address allocation is removed. For example after RSTDAA.

   For more details, refer to public example `ts.group_addressing_test.sv` present in `tb_i3c_svt_uvm_basic_sys.`

A.30 CCC Modality in HDR Mode, Use Model and Driving

Special framings are used for transmitting Common Command Codes in HDR Modes. CCCs that are supported in any HDR Modes may be sent by a Current Master, as long as it has entered the HDR Mode using the Enter HDR Mode CCC. Both Broadcast and Direct CCCs are supported in I3C VIP, with both framing and driving details specific to each HDR Mode mentioned below. Currently, CCCs in HDR DDR mode are supported. CCCs in HDR TSP/TSL mode would be supported in future releases.

As specified in I3C spec v1.1, flow control elements are introduced with new functionalities w.r.t WRITE or READ command operations for both HDR-DDR and HDR-TSP/TSL modes.

I3C VIP also offers HDR-DDR flow control elements in Master, Slave agents. The Flow Control settings can be changed during runtime by using ENDXFER CCC with defining byte 0XF7. Below is the snapshot of structure of ENDXFER CCC with 0XF7 defining byte.

ENDXFER CCC Additional Data Byte for Sub-Command 0xF7 (HDR-DDR Protocol)

Bits	Description	Values
[7:6]	CRC Word Indicator	**2'b11**: No CRC Word follows Early Termination request **2'b01**: CRC5 Word follows Early Termination request **Other**: Reserved for future definition by MIPI Alliance Sensor WG (eventually other CRC types such as CRC16 or CRC32)
[5]	Enable WRITE Early Termination Request	**1'b0**: Enable **1'b1**: Disable
[4]	Enable ACK/NACK Capability for WRITE Command	**1'b0**: Enable **1'b1**: Disable
[3:0]	Reserved for future use.	–

These are some of the basic rules followed by I3C VIP before driving or sampling CCC structure in HDR-DDR mode:

1. Considerations for Bit [4] value in 0XF7 defining byte during ENDXFER CCC. Following values are possible with their meanings:

Bit[4]=0 (ACK/NACK Capability for WRITE Command enabled)	Bit[4]=1 (ACK/NACK Capability for WRITE Command disabled)
This means that ACK/NACK capability for WR cmd present with slave devices.	No ACK/NACK capability present with slave.
The preamble 2'b10 or 2'b11 after WR cmd word will now be driven by the slave. Current Master will sample the value and understand it as either ACK(2'b10) or NACK (2'b11).	Only possible preamble value for WR cmd is 2'b10, which will be driven by Current Master only. The slave will have to sample the data afterwards.
HDR-DDR CCC structure can only be driven in this mode.	The CCC will be considered as normal data with no meaning to it.

2. If multiple slaves are present on the bus, ACK/NACK capability using ENDXFER CCC can be enabled for one or multiple devices.
3. If HDR-DDR CCC structure is driven from VIP Current Master ('h7e/W + CCC opcode) and at least one slave is found to have WR cmd ACK/NACK capability enabled, the VIP Current Master would not drive the preamble after WR cmd word. It samples the preamble value as ACK/NACK and the VIP slave component(s) whose ACK/NACK capabilities are enabled would always ACK (in the form of preamble 2'b10) the 7E/W. Later, all VIP slave components with ACK/NACK capability

enabled would sample the HDR CCC normally while ACK/NACK incapable slave(s) do not understand CCC flow.

4. If mixed slaves are present on the bus (ACK/NACK capability enabled for some components and disabled for others), and the addressed slave during directed Write HDR-DDR CCC is the one whose ACK/NACK capability is disabled, then it will be erroneous condition. This is because VIP Current Master awaits for preamble value (to find if it is ACK or NACK) and the addressed slave would be waiting for preamble value 2'b10. In this condition, because neither Master nor slave has driven SDA line, preamble value received would be 2'b11 and Master would consider this as NACK, it will drive Exit or Restart pattern afterwards.
5. If flow control element for all slave components are disabled and VIP Current Master is configured to drive CCC modality in DDR mode, then after CMD word (7E/W) VIP Current Master sends preamble 2'b10 (since no device is WR cmd ACK/NACK capable).

   In this scenario any HDR CCC does not make any sense. VIP Slave components are unable to understand Broadcast CCCs. The Directed Write of the preamble during 2nd packet would also be driven by Current Master, but VIP slave components would be unable to process any CCC request, for Directed Read CCC VIP slave component(s) will ACK/NACK accordingly and drive data as if normal READ cmd was being fired, because Slave component does not understand CCC flow.
6. Recommended procedure for using CCC structure in any HDR mode is via enabling the ACK/NACK capability of all slave components via either Broadcast ENDXFER or Direct Write ENDXFER to all slaves.
7. All the above points are valid of group address also.

Note

I3C VIP offers a system configuration property `ddr_flow_control_enabled` to set the default behavior of Flow Control Elements of all slave and sec-master VIP components present on bus during initialization.

The possible Values of this configuration property are:

`ddr_flow_control_enabled` = 1'b0: The flow control elements are disabled, means Bit[4] = 1'b1 (ACK/NACK capability disabled), Bit[5] = 1'b1 (WRITE Early Termination Request disabled), Bit[7:6] = 2'b11 (No CRC Word follows Early Termination request).

`ddr_flow_control_enabled` = 1'b1: The flow control elements are enabled, means Bit[4] = 1'b0 (ACK/NACK capability enabled), Bit[5] = 1'b0 (WRITE Early Termination Request enabled), Bit[7:6] = 2'b10 (CRC Word follows Early Termination request).

These CCCs are supported in HDR-DDR mode in I3C VIP:

S.No	CCC Name	Broadcast or Direct
1	ENEC	Both
2	DISEC	Both
3	ENTAS0 ENTAS1 ENTAS2 ENTAS3	Both
4	DEFSLVS	Broadcast

S.No	CCC Name	Broadcast or Direct
5	SETMWL	Both
6	SETMRL	Both
7	GETMWL	Direct
8	GETMRL	Direct
9	ENDXFER	Both
10	SETXTIME	Both
11	GETXTIME	Direct
12	RSTACT	Both
13	SETGRPA	Direct
14	DEFGRPA	Broadcast
15	RSTGRPA	Both
16	GETBCR	Direct
17	GETDCR	Direct
18	GETSTATUS	Direct
19	SETBRGTGT	Direct
20	GETMXDS	Direct
21	GETCAPS	Direct
22	['h61:'h7F]	Broadcast vendor
23	['hE0:'hFE]	Direct vendor

Driving CCC from VIP Current Master: Driving CCCs from Master is similar to driving CCCs in SDR mode, with few additional variables:

1. `set_ccc_in_hdr`: This variable is used analogous to set_ccc (used in SDR mode). When 7E/W+CCC opcode is to be sent, you must set this variable.
2. `drive_end_procedure_with_generic_txn`: This variable is used to send 7E/W without CCC opcode, to provide end procedure followed by another HDR-DDR Generic Transaction (+Optional HDR EXIT).
3. `tr.data: "data[0]" field of master_transaction` object is used to provide the CCC opcode value, data[1], data[2],… data[n] to provide additional CCC defining byte (if any), or data bytes corresponding to CCC (if any).

Example Sequence:

//PART 1: To drive ENTHDR0

```
`uvm_create_on(tr,p_sequencer.master_sequencer[0])
tr.reasonable_sr_or_p_gen.constraint_mode(0);
```

```
   if (!tr.randomize() with {
                                tr.cmd              ==
svt_mipi_i3c_master_transaction::WRITE;
                                tr.arbitrate        == 0;
                                tr.sr_or_p_gen      == 0;
                               })
                               `uvm_error("Randomization Failure","hdr_basic_sequence")
   else begin
     tr.addr = 'h7e;
     tr.ccc = `SVT_MIPI_I3C_ENTHDR0;
   end
   `uvm_send(tr)
```

//PART 2: This packet should contain CCC opcode (either Broadcast or Direct)

//Providing example of Broadcast CCC in this packet

```
`uvm_create_on(tr,p_sequencer.master_sequencer[0])
    tr.reasonable_sr_or_p_gen.constraint_mode(0);
    if (!tr.randomize() with {
                                tr.cmd              ==
svt_mipi_i3c_master_transaction::WRITE;
                                tr.arbitrate        == 0;
                                tr.sr_or_p_gen      == 0;
                                tr.data.size        == 2;
//always provide the CCC info in data[0] field
                                tr.data[0]          == `SVT_MIPI_I3C_BRDCAST_ENDXFER;
//provide the defining byte value here.
  tr.data[1] == 'hF7;
//provide the data value afterwards.
  tr.data[2] == 'hFF;
                               })
                               `uvm_error("Randomization Failure","hdr_basic_sequence")
    else begin
      tr.addr = 'h7e; //Since CCC needs to be transmitted, addr should be 7E/W
      tr.set_ccc_in_hdr = 1; //always enable this field if CCC in HDR mode needs to be
transmitted.
      tr.set_ccc = 0;
    end
    `uvm_send(tr)
```

//Providing example of Direct CCC in this packet

//The VIP Current Master will automatically drive Restart pattern after direct CCC.

```
`uvm_create_on(tr,p_sequencer.master_sequencer[0])
    tr.reasonable_sr_or_p_gen.constraint_mode(0);
    if (!tr.randomize() with {
                                tr.cmd              ==
svt_mipi_i3c_master_transaction::WRITE;
                                tr.arbitrate        == 0;
                                tr.sr_or_p_gen      == 0;
                                tr.data.size        == 2;
//always provide the CCC info in data[0] field
                                tr.data[0]          == `SVT_MIPI_I3C_DIRECT_ENDXFER;
//provide the defining byte value here.
  tr.data[1] == 'hF7;
                               })
                               `uvm_error("Randomization Failure","hdr_basic_sequence")
    else begin
      tr.addr = 'h7e; //Since CCC needs to be transmitted, addr should be 7E/W
      tr.set_ccc_in_hdr = 1; //always enable this field if CCC in HDR mode needs to be
transmitted.
      tr.set_ccc = 0;
    end
    `uvm_send(tr)
```

//PART 3: Since Direct CCC is transmitted before, this packet needs to be for addressed slave

```
`uvm_create_on(tr,p_sequencer.master_sequencer[0])
    tr.reasonable_sr_or_p_gen.constraint_mode(0);
    if (!tr.randomize() with {
                                tr.cmd              ==
svt_mipi_i3c_master_transaction::WRITE;
                                tr.arbitrate        == 0;
                                tr.sr_or_p_gen      == 0;
                                tr.data.size        == 1;
// provide the data info in data[0] field, and corresponding data info in data[1, … n]
                                tr.data[0]          == 'hFF;
                               })
                               `uvm_error("Randomization Failure","hdr_basic_sequence")
    else begin
      tr.addr = slave[1]_addr_value; //Set the value of addressed slave here
     tr.set_ccc_in_hdr = 0; //disable this field if slave addr packet of CCC in HDR mode
needs to be transmitted.
      tr.set_ccc = 0;
    end
    `uvm_send(tr)
```

Note

The example provided above is the basic structure to configure the VIP sequencer to send HDR-DDR CCC. For detailed information on driving particular CCCs in DDR mode, refer `ts.i3c_hdr_ddr_ccc_*_test.sv` in testbench `tb_i3c_svt_uvm_basic_sys` (here * refers to the CCC name that needs to be referred.)