Verification Continuum™

VC Verification IP
SPMI
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 System Power Management Interface (SPMI). Also, it is for the design or verification engineers who want to verify SPMI operation using an UVM testbench written in SystemVerilog. Readers are assumed to be familiar with SPMI, Object Oriented Programming (OOP), SystemVerilog, and Universal Verification Methodology (UVM) techniques.

Guide Organization

The chapters of this guide are organized as follows:

- Chapter 1, "Introduction", introduces the SPMI VIP and its features.
- Chapter 2, "Download and Installation", describes system requirements and provides instructions on how to install, configure, and begin using the SPMI VIP.
- Chapter 3, "SPMI VIP and Example Architecture", describes SPMI with an example architecture.
- Chapter 4, "Verification Topologies", describes the verification topologies.

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

This chapter gives a basic introduction, overview, and features of the SPMI UVM VIP.

The Introduction chapter discusses the following topics:

- Product Overview
- User Prerequisites
- References
- Product Features

1.1 Product Overview

The System Power Management Interface (SPMI) is a two-wire serial interface that connects the integrated Power Controller (PC) of a System-on-Chip (SoC) processor system with one or more Power Management Integrated Circuits (PMIC) voltage regulation systems. SPMI enables systems to dynamically adjust the supply and substrate bias voltages of the voltage domains inside the SoC using a single SPMI bus. Within the PC, the SPMI-related functions are referred to as the "Master". Within a PMIC, the SPMI related functions are referred to as the "Slave". There may be up to four Master devices and up to sixteen Slaves on a single SPMI bus. Multiple SPMI Masters and Slaves can reside on a single IC, on several separate ICs or any combination of the two. Slave device that can initiate Sequences on a SPMI bus is called a Request Capable Slave (RCS) device. A Slave device that can not initiate Sequences is called a Non-Request Capable Slave (NRCS) device. This Specification defines the operating states, the command set, the physical interface, and the protocol for data communication between SPMI devices on a SPMI bus to insure the compatibility of command and data transfers. The SPMI Command Sequence set includes Slave and Master addressing, control of the Slave operating state, register read from and register write to Master and Slave devices, as well as commands supporting the use of MIPI Device Descriptor Block data read and bus management.

1.2 User Prerequisites

Familiarity with:

- SPMI
- Object Oriented Programming
- SystemVerilog
- The current version of UVM Methodology

1.3 References

For more information on SPMI, see the following documents:

- MIPI SPMI UVM User Guide available at:

  $DESIGNWARE_HOME/vip/svt/mipi_spmi_svt/latest/doc/mipi_spmi_svt_uvm_user_guide.pdf

- MIPI SPMI UVM class reference HTML available at:

  $DESIGNWARE_HOME/vip/svt/mipi_spmi_svt/latest/doc/mipi_spmi_svt_uvm_class_reference/html/index.html

- Reference to Protocol: MIPI Alliance Standard for System Power Management Interface 2.0

1.4 Product Features

SPMI is a suite of UVM-based verification components that are compatible for use with System Verilog compliant test benches. The MIPI SPMI VIP suite simulates SPMI Master and SPMI Slave transactions, through agents, as defined by the SPMI specification.

The SPMI Master and SPMI Slave agents support functionality normally associated with UVM components, including the creation of transactions, checking, and reporting protocol correctness, transaction logging, and functional coverage.

2
Download and Installation

The Download and Installation chapter discusses the following topics:

- Downloading From SolvNet and Installing
- Licensing
- Environment Variables
- The dw_vip_setup Administrative Tool

Note If you encounter any problem in installing the SPMI VIP, contact Customer Support.

2.1 Downloading From SolvNet and Installing

You can download the 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 the following web page:

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

This section consists of the following subsections:

- Downloading From Electronic Software Transfer (EST) System (Download Center)
- Downloading Using FTP With a Web Browser

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

2.1.1 Downloading From Electronic Software Transfer (EST) System (Download Center)

a. Point your web browser to http://solvnetplus.synopsys.com.

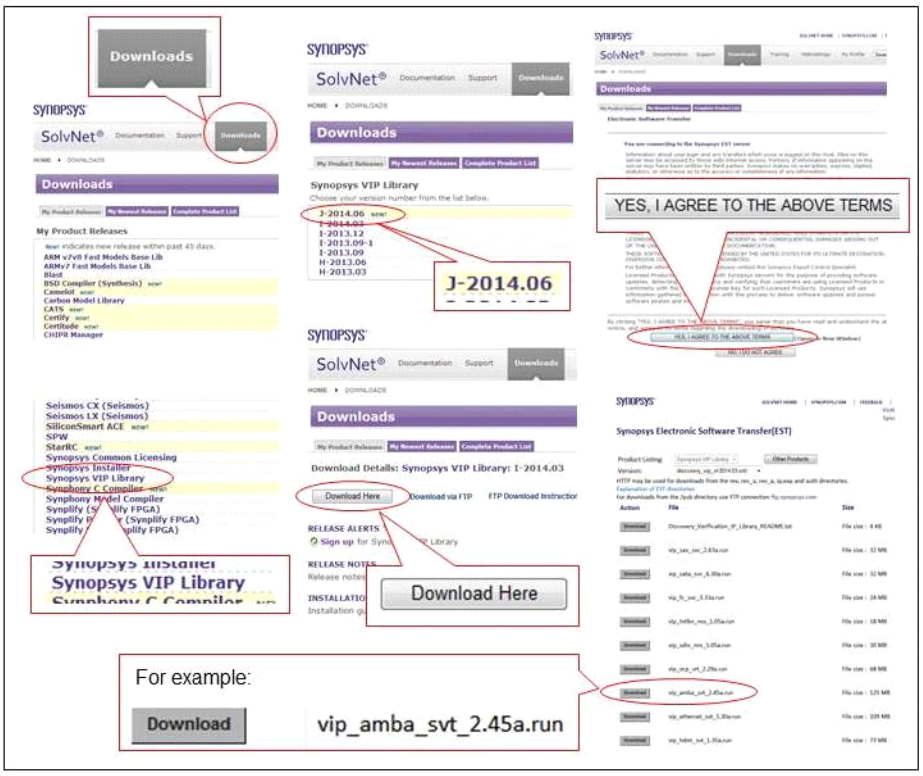
b. Enter your Synopsys SolvNet Username and Password.

c. Click Sign In button.

d. Make the following selections on SolvNet to download the `.run` file of the VIP.

   i. Downloads tab
   ii. Synopsys VIP Library product releases
   iii. O-2018.09-2
   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
```

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

Note
You may download multiple files simultaneously.

2.1.2 Downloading Using FTP With a Web Browser

Follow Step a to Step e of Section 2.3.1 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.

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

2.2 Licensing

The SPMI VIP uses the SCL software to control its usage. You can find general SCL information from the following link:

http://www.synopsys.com/keys

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

- VIP-SPMI-SVT
- VIP-LIBRARY-SVT
- DesignWare-Regression

Only one license is consumed per simulation, no matter how many VIP models are instantiated in the design.

The licensing key must reside in files that are indicated by specific environment variables. For information about setting these licensing environment variables, see Environment Variables.

2.2.1 License Keys

This section consists of the following subsections:

- License Keys
- Simulation License Suspension
- License Polling

To control which license is used, use the `DW_LICENSE_OVERRIDE` environment variable as follows:

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

  DesignWare-Regression VIP-LIBRARY-SVT
- To use only a VIP-SPMI-SVT license, set `DW_LICENSE_OVERRIDE` to:

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

2.2.2 Simulation License Suspension

All VIP products support license suspension. The simulators that support license suspension allow a model to check-in its license token while a 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 suspension.

2.2.3 License Polling

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

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

2.3 Environment Variables

The SPMI VIP verification models require the following environment variables and path settings:

- `DESIGNWARE_HOME`: The absolute path to where the VIP is installed.
- `SNPSLMD_LICENSE_FILE`: The absolute path to a file that contains the license keys for SCL software or the port@host reference to this file.
- `LM_LICENSE_FILE`: The absolute path to a file that contains the license keys for your third-party tools. Also, include the absolute path to the third party executable in your PATH variable.

2.4 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_spmi_svt/tb_mipi_spmi_svt_uvm_basic_sys/
// To run the example using the generated run script with +incdir+
./run_mipi_spmi_svt_uvm_basic_sys -verbose -incdir base_test 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_INCDIR>/vip/svt/mipi_spmi_svt/<vip_version>/sverilog/include \
-ntb_opts uvm -full64 -sverilog +define+UVM_PACKER_MAX_BYTES=16000
+define+UVM_PACKER_MAX_BYTES=1500000 \
```

```
+define+UVM_DISABLE_AUTO_ITEM_RECORDING -timescale=1ns/1ps +define+SVT_UVM_TECHNOLOGY
+define+SYNOPSYS_SV \
+incdir+<testbench_dir>/examples/sverilog/mipi_spmi_svt/tb_mipi_spmi_svt_uvm_basic_sys/
. \
+incdir+<testbench_dir>/examples/sverilog/mipi_spmi_svt/tb_mipi_spmi_svt_uvm_basic_sys/
../../env \
+incdir+<testbench_dir>/examples/sverilog/mipi_spmi_svt/tb_mipi_spmi_svt_uvm_basic_sys/
../env \
+incdir+<testbench_dir>/examples/sverilog/mipi_spmi_svt/tb_mipi_spmi_svt_uvm_basic_sys/
env \
+incdir+<testbench_dir>/examples/sverilog/mipi_spmi_svt/tb_mipi_spmi_svt_uvm_basic_sys/
dut \
+incdir+<testbench_dir>/examples/sverilog/mipi_spmi_svt/tb_mipi_spmi_svt_uvm_basic_sys/
hdl_interconnect \
+incdir+<testbench_dir>/examples/sverilog/mipi_spmi_svt/tb_mipi_spmi_svt_uvm_basic_sys/
lib \
+incdir+<testbench_dir>/examples/sverilog/mipi_spmi_svt/tb_mipi_spmi_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.

2.5 Getting Help on Example Run/make Scripts

You can get help on the generated make/run scripts in the following ways:

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

```
run_mipi_spmi_svt_uvm_basic_sys
usage: run_mipi_spmi_svt_uvm_basic_sys [-32][-incdir][-verbose] [-debug_opts] [-
waves]
[-clean] [-nobuild] [-buildonly] [-norun] [-pa] <scenario> <simulator>
where   <scenario> is one of: all base_test spmi_master_connect_disconnect_test
spmi_unsupported_address_in_extended_write_read_long_test
spmi_extended_register_write_read_test spmi_master_noise_before_arbitration_test
spmi_unsupported_address_in_register_write_read_test
<simulator> is one of: vcsmxvlog mtivlog vcsvlog vcszsimvlog vcsscvlog ncvlog
vcszebuvlog vcsmxpcvlog vcsvhdl vcsmxpipvlog ncmvlog vcspcvlog
-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|dve|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] [<scenario> ...]
Valid simulators are: vcsvlog vcsmxvlog mtivlog vcsmxpcvlog vcsmxpipvlog ncvlog
vcspcvlog vcsscvlog vcsvhdl ncmvlog
Valid scenarios are: all base_test spmi_master_connect_disconnect_test
spmi_unsupported_address_in_extended_write_read_long_test
spmi_extended_register_write_read_test spmi_master_noise_before_arbitration_test
spmi_unsupported_address_in_register_write_read_test
```

Note
You must have PA installed if you use the -pa or PA=1 switches.
PA is currently not supported in the SPMI VIP.

2.6 The dw_vip_setup Administrative Tool

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

The `dw_vip_setup` utility provides the following features:

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

The `dw_vip_setup` utility is as follows:

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

  To create a design directory and add a model to use it in a testbench, use the following command:

  ```
  $DESIGNWARE_HOME/bin/dw_vip_setup -path <design_directory_path> -e
  mipi_spmi_svt/tb_mipi_spmi_svt_uvm_basic_sys -svtb
  ```

- Supports the FSDB wave format

The SPMI VIP provides the following models:

- mipi_spmi_master_agent_svt
- mipi_spmi_master_svt
- mipi_spmi_master_group_svt
- mipi_spmi_slave_agent_svt
- mipi_spmi_slave_svt

- mipi_spmi_slave_group_svt

2.6.1 Setting Environment Variables

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

2.6.2 The dw_vip_setup Command

Invoke `dw_vip_setup` from the command prompt. The `dw_vip_setup` command checks the syntax of command-line arguments and makes sure that the requested input files exist. The syntax 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 `dw_vip_setup` operation. Table 2-1 lists 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 the current working directory. If you do not specify a version, the latest version is assumed. The model names are as follows: • mipi_spmi_master_agent_svt • mipi_spmi_master_svt • mipi_spmi_master_group_svt • mipi_spmi_slave_agent_svt • mipi_spmi_slave_svt • mipi_spmi_slave_group_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` command does not attempt to remove any include files used solely by the specified model or models. The model names are as follows: • mipi_spmi_master_agent_svt • mipi_spmi_master_svt • mipi_spmi_master_group_svt • mipi_spmi_slave_agent_svt • mipi_spmi_slave_svt • mipi_spmi_slave_group_svt

Table 2-1 Setup Program Switch Descriptions (Continued)

Switch	Description
-u[pdate] (*model* [**-v**[ersion] *version*]) …	Updates to the specified model version for the specified model or models. The dw_vip_setup script updates to the latest models when you do not specify a version. The model names are as follows: • mipi_spmi_master_agent_svt • mipi_spmi_master_svt • mipi_spmi_master_group_svt • mipi_spmi_slave_agent_svt • mipi_spmi_slave_svt • mipi_spmi_slave_group_svt The -update switch makes dw_vip_setup to build suite libraries from the same suite as the specified models, and to copy 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 script creates a simulator-run program for all supported simulators. If you specify a scenario (or system) for example, testbench the models needed for the testbench are included automatically and do not need to be specified in the command. **Note**: Use the -info switch to list all available system examples.
-ntb	Not supported.
-svtb	Use this switch to set up models and example testbenches for SystemVerilog. The resulting design directory is streamlined and you can use it in SystemVerilog simulations.
-c[lean] {*scenario* | *model*/*scenario*}	Cleans the specified scenario or 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>]]]	Generates 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 model list file, as an 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.

Table 2-1 Setup Program Switch Descriptions (Continued)

Switch	Description
-h[elp]	Returns a list of valid dw_vip_setup switches and their correct syntax.
model	The SPMI VIP models are as follows: • mipi_spmi_master_agent_svt • mipi_spmi_master_svt • mipi_spmi_master_group_svt • mipi_spmi_slave_agent_svt • mipi_spmi_slave_svt • mipi_spmi_slave_group_svt The *model* argument defines the model or models that dw_vip_setup acts upon. This argument is not needed with the -info or -help switches. All switches that require the model argument may also use a model list. You may specify a version for each listed model using the -version option. If omitted, dw_vip_setup uses the latest version. The -update switch ignores the 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.
-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.
-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` command treats all lines beginning with “#” as comments.

3 SPMI VIP Example Architecture

This chapter provides an example architecture of SPMI VIP. This chapter contains the following sections:

- "SPMI Master VIP <==> SPMI Slave VIP"

3.1 SPMI Master VIP <==> SPMI Slave VIP

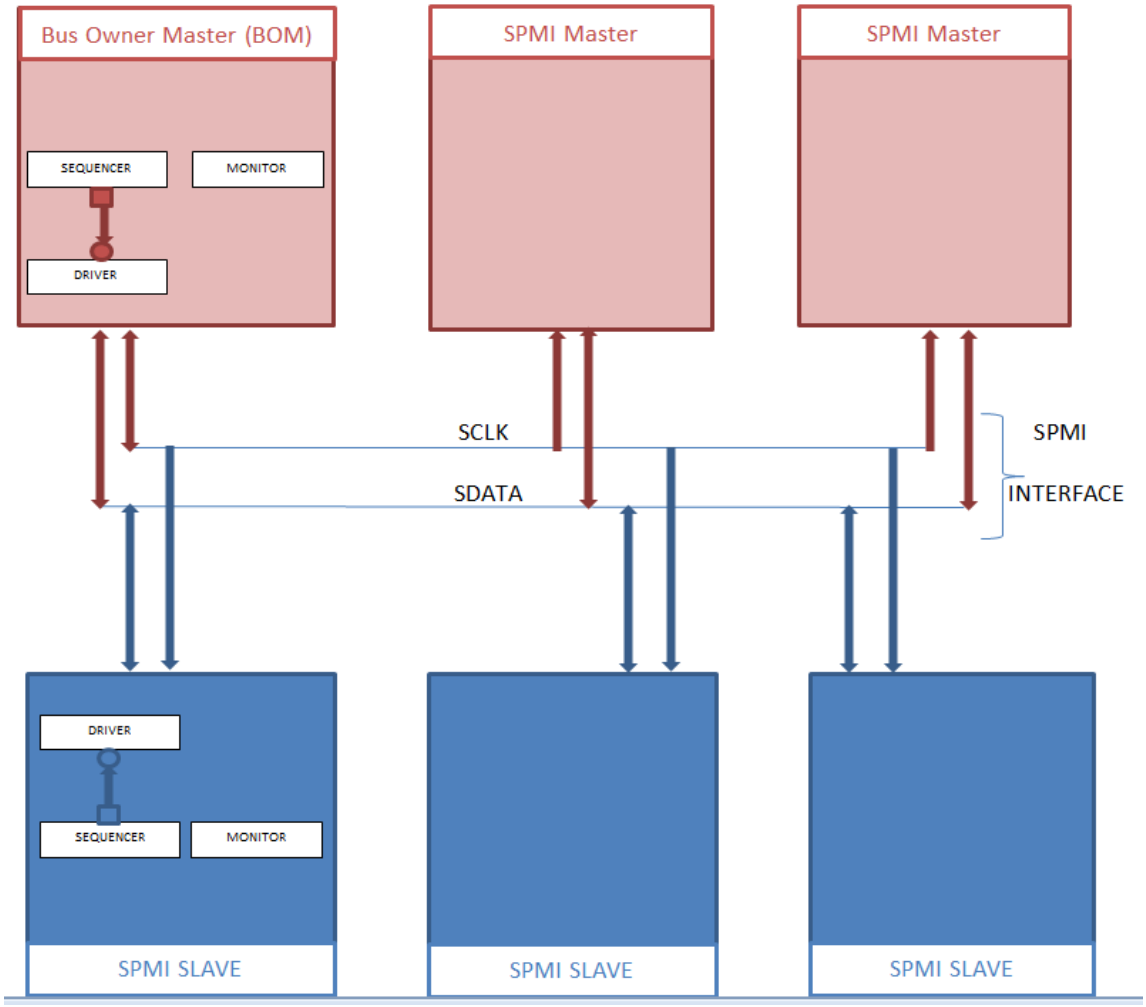
This section contains the following topics:

- "Block Diagram"

3.1.1 Block Diagram

The following diagram shows SPMI slaves and masters connected to each other through SPMI interface that includes SDATA and SCLK signals: SDATA being bidirectional data signal and SCLK driven by Bus Owner Master (BOM).

Figure 3-1 SPMI Master VIP connected to SPMI Slave VIP through SPMI interface



3.1.2 Verification Environment

SPMI supports the following verification environment:

- SPMI driver component in master agents connected to the SPMI interface.
- SPMI driver component in slave agents connected to the SPMI interface .
- SPMI Monitor component in master agents to capture SPMI command sequences from slaves or other masters.
- SPMI Monitor component in slave agents to capture SPMI command sequences from masters or other slaves.
- SPMI scoreboard connected between two masters and a master and salve .
- SPMI scoreboard connected between two slaves and a slave and master.
- SPMI command sequences to drive SPMI traffic on SPMI interface via slave and master drivers.

4 Verification Topologies

The SPMI VIP can be used to test master and slave DUT. This chapter discusses the following topics:

- Testing a Master DUT Using an UVM Slave VIP
- Testing a Slave DUT Using an UVM Master VIP

4.1 Testing a Master DUT Using an UVM Slave VIP

In this scenario, test a SPMI VIP master DUT using an UVM SPMI VIP slave.

- Testbench setup: Configure the SPMI VIP System environment to have one slave agent, in active mode. The active slave agent will respond to the transactions generated by master DUT. The slave agent will also perform passive functions such as protocol checking, coverage generation and transaction logging.
- To implement this topology, set the following properties (For example, instance name of the system configuration is `system_cfg` and that of agent specific configuration is cfg):
  - System configuration settings are as follows:
    - `system_cfg.num_masters` = 0;
    - `system_cfg.num_slaves` = 2;
  - Agent specific configuration settings are as follows:
    - `cfg.system_cfg.slave_cfg[0].slave_id` = 4'b0000;
    - `cfg.system_cfg.slave_cfg[1].slave_id` = 4'b0001;
  - Port configuration settings are as follows:
    - `system_cfg.slave_cfg[0].is_active` = 1;

4.2 Testing a Slave DUT Using an UVM Master VIP

In this scenario, test a SPMI VIP slave DUT using an UVM SPMI VIP master. Configure the SPMI VIP system environment to have one master agent, in active mode. The active master agent will generate SPMI VIP transactions for the slave DUT. The master agent will also perform passive functions such as protocol checking, coverage generation, and transaction logging.

To implement this topology, set the following properties (For example, instance name of the system configuration is `system_cfg` and that of agent specific configuration is cfg):

The system configuration settings are as follows:

- `system_cfg.num_masters = 2;`
- `system_cfg.num_slaves = 0;`
- Agent specific configuration settings are as follows:
  - `cfg.system_cfg.master_cfg[0].master_id = 2'b00;`
  - `cfg.system_cfg.master_cfg[1].master_id = 2'b01;`
- Port configuration settings are as follows:
  - `system_cfg.master_cfg[0].is_active = 1;`

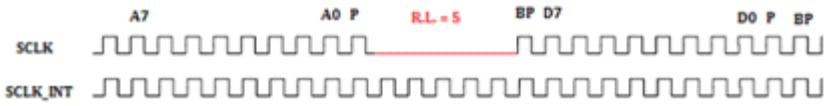
5 SPMI VIP Usage Notes

5.1 Programmable Read Latency

5.1.1 Feature Description

Master VIP should insert programmable number of latency (in terms of master's internal clock) after the address frame parity bit and before bus park (BP) cycle or during the second half of the (BP) cycle when a read command is issued on the bus. This value will be less than the *sequence timeout* value described in 2.b). An illustration is shown as follows:

Figure 5-1 Programmable Read Latency Illustration



Implementation:

The following configuration variable is added to control read latency:

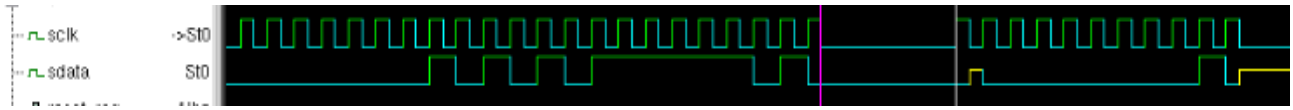
```
/** Specifies delay interms of number of sclk_int clk cycle after address frame  of read_commad issued*/
int unsigned read_latency = 0;
/** Specifies whether the read latency is occuring in address parity or in  BPC between address and data frame */
svt_mipi_spmi_types::read_latency_type_enum read_latency_type = svt_mipi_spmi_types::READ_LATENCY_IN_PARITY;
```

When `read_latency` is set to a value greater than 0, latency is introduced in Address parity or during the second half of the BPC based on the `read_latency_type` parameter.

Example:

Scenario : Slave initiated read sequence (Master_read)) is depicted as follows:

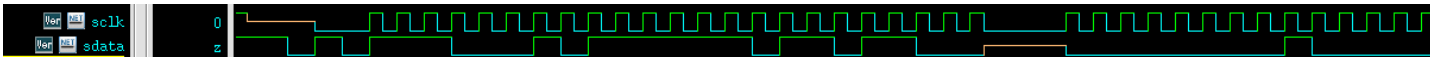
Figure 5-2 Read sequence for read latency during parity



Note

When read latency is enabled and during `read_latency` time period, `spmi_clock_freq_range_check` is disabled as it is expected to fire.

Figure 5-3 Read sequence for read latency during BPC



5.2 Noise Error Injection

Master VIP supports injection of NOISE on sdata signal. Below are the 5 different kinds of noise supported:

- ERR_NOISE_BEFORE_ARBITRATION
- ERR_NOISE_DURING_MASTER_PRIORITY_ARBITRATION
- ERR_NOISE_DURING_MASTER_SECONDARY_ARBITRATION
- ERR_NOISE_DURING_SLAVE_A_BIT_ARBITRATION
- ERR_NOISE_DURING_SLAVE_SR_BIT_ARBITRATION

Noise Injection Mechanism:

A new sequence called "svt_mipi_spmi_master_noise_injection_during_arbitration_sequence" is defined in sequence collection file. The error sequence contains two variables "noise type" (to select type of noise to be injected) , "priority_type".(to select priority level) and "index"(to select master index)

Youy must call this sequence by doing below settings to create different noise scenarios:

- NOISE_DURING_MASTER_PRIORITY_ARBITRATION:

  `uvm_do_with(noise_seq, {noise_type == svt_mipi_spmi_transaction_exception::ERR_NOISE_DURING_MASTER_PRIORITY_ARBITRATION;priority_type == 1;})
- NOISE_DURING_MASTER_SECONDARY_ARBITRATION:

  `uvm_do_with(noise_seq, {noise_type == svt_mipi_spmi_transaction_exception::ERR_NOISE_DURING_MASTER_SECONDARY_ARBITRATION;priority_type == 0;})
- NOISE_DURING_SLAVE_A_BIT_ARBITRATION:

  `uvm_do_with(noise_seq, {noise_type == svt_mipi_spmi_transaction_exception:: ERR_NOISE_DURING_SLAVE_A_BIT_ARBITRATION;priority_type == 1;})
- NOISE_DURING_SLAVE_SR_BIT_ARBITRATION:

  `uvm_do_with(noise_seq, {noise_type == svt_mipi_spmi_transaction_exception:: ERR_NOISE_DURING_SLAVE_SR_BIT_ARBITRATION;priority_type == 0;})

- NOISE_BEFORE_ARBITRATION:

  `uvm_do_with(noise_seq, {noise_type == svt_mipi_spmi_transaction_exception::
  ERR_NOISE_BEFORE_ARBITRATION;})

5.3 Memory Address Configuration

The SPMI Memory Address Configuration is done as follows:

1. In Master or Slave VIP, memory start and end address is configured using below two variables present in spmi_configuration class:

```
  /**
   * Specifies the starting address of register space
   * Applicable only when register_address_pair_q size is 0
   */
  rand bit [15:0] register_address_start = 16'h0;
  /** Specifies the last address of register space
   * Applicable only when register_address_pair_q size is 0
   */
  rand bit [15:0] register_address_end   = 16'hFFFF;
 (or) can use below queue where we can have multiple ranges for valid address pairs.
  /**
   * Specifies a valid register start and end address pair
   * [31:16] Start Address;
   * [15:0]  End Address
   */
  bit [31:0] register_address_pair_q[$];
Entries into the queue can be added into the queue using the below method in the
configuration class.
/**
   * This method is used to populate register start and end address pairs in queue
   * and return the number of address in the queue.
   * instead of just a start and end address for a slave
   * @param start_address bit [15:0] Start Address
   * @param end_address   bit [15:0] End Address
   */
   extern virtual function void add_register_start_end_address_pair(bit [15:0]
start_address, bit [15:0] end_address);
```

2. Register Write/Read limit for each type of write/read command can be set using below macros:

```
/** Define for maximum address limit for Slave REGISTER_WRITE/READ operation   */
`ifndef SVT_MIPI_SLAVE_REGISTER_WRITE_READ_ADDRESS_LIMIT
`define SVT_MIPI_SLAVE_REGISTER_WRITE_READ_ADDRESS_LIMIT  5'h1F
`endif
/** Define for maximum address limit for Slave EXTENDED_REGISTER_WRITE/READ operation
*/
`ifndef SVT_MIPI_SLAVE_EXTENDED_REGISTER_WRITE_READ_ADDRESS_LIMIT
`define SVT_MIPI_SLAVE_EXTENDED_REGISTER_WRITE_READ_ADDRESS_LIMIT  8'hFF
`endif
/** Define for maximum address limit for Slave EXTENDED_REGISTER_WRITE/READ_LONG
operation   */
`ifndef SVT_MIPI_SLAVE_EXTENDED_REGISTER_WRITE_READ_LONG_ADDRESS_LIMIT
`define SVT_MIPI_SLAVE_EXTENDED_REGISTER_WRITE_READ_LONG_ADDRESS_LIMIT  16'hFFFF
`endif
/** Define for maximum address limit for Master WRITE/READ_LONG operation   */
`ifndef SVT_MIPI_MASTER_WRITE_READ_ADDRESS_LIMIT
`define SVT_MIPI_MASTER_WRITE_READ_ADDRESS_LIMIT  8'hFF
```

```
`endif
```

VIP Response:

Any write/read address which exceeds the cfg.register_address_end value (or) will not fall in queue address pairs will lead to `spmi_slave_unsupported_address_for_write_read_operation` check getting fired from VIP. NACK for write command and NRF for read command will be transmitted from VIP.

If write/read address falls within the range of cfg. register_address_start and cfg. register_address_end, but exceeds the address limit value of above mentioned macros, will lead to `spmi_slave_invalid_address_for_write_read_operation` check getting fired from VIP. For Write operation, the last address i.e address_limit of macro will be overwritten and for Read operation, data from the last address i.e address_limit of macro will be read.

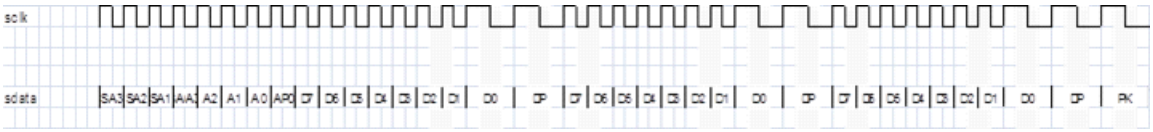
5.4 Clock Stretching

5.4.1 Description

SPMI Clock Stretching (when enabled):

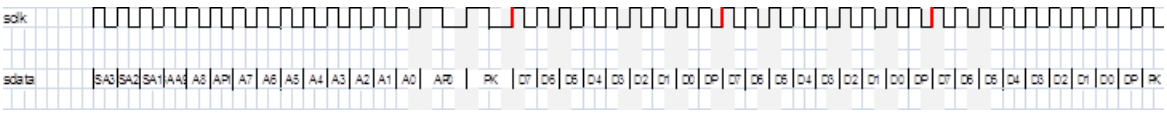
- Write Commands: For write commands initiated by the Master VIP, the clock phase for the last two cycles of each data byte (D0 and Parity) and last three cycles of the last data byte (D0, Parity and BP) should be extended by reducing the SCLK frequency by half.

Figure 5-4 Write - SCLK Freuency



- Read Commands: For read commands initiated by the Master VIP, the clock phase for the last two cycles of the last address byte (Parity and BP) should be extended by reducing the SCLK frequency by half.

Figure 5-5 Read - SCLK Freuency



5.4.2 Implementation:

New configuration variables are added to enable and control this feature:

```
/** User should set this variable to required clock stretched value */
real t_sclk_stretch = `SVT_MIPI_SPMI_T_SCLK_MIN*2;
  /** Specifies whether clock stretching should be enabled during write/read transfer */
  bit t_sclk_stretch_en = 0;
```

For example,

User cfg.t_sclk_stretch_en = 1 , cfg.tc_sclk_stretch=<user_value> to enable clk stratching feature.

Figure 5-6 Master initiated write (Extended register_write long)

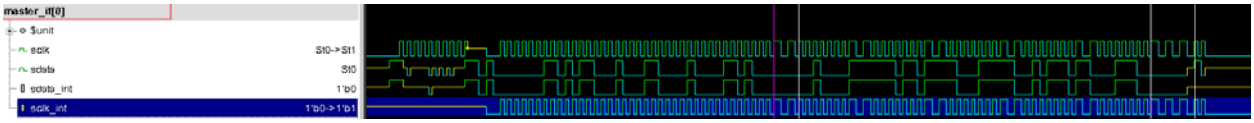
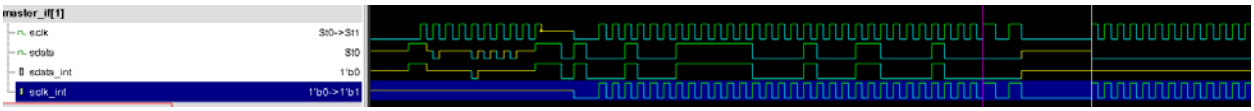


Figure 5-7 Master initiated read (Extended register read)



Note

When clock stretching is enabled and during clock stretching period, `spmi_clock_freq_range_check` is disabled as it is expected to fire.

5.5 Programmable Number of Nacks Generation

When there are multiple back-2-back write commands to the master or slave VIPs and there was no parity error on the SPMI bus in the command sequence received by the VIP, there should be a programmable value (up to 4) for which the VIP should keep giving the NACK indication after which it should give an ACK indication.

New configuration variable is added to program number of NACKs Generation:

```
  /** Specifies number of NAC to be sent for valid write_cmd (User should set this to
greater than zero to enable this feature)*/

  int unsigned nac_for_write_cmd_counter = 0;
```

Above variable is encrypted and will not be seen in cfg.print() . Customer needs to set this variable to have value greater than 0 to exploit NACK generation feature.

5.6 Events for Each Phase (Command, Address, and Data)

VIP will provide the corresponding Events after reception of Command, Address, and Data Phases.

- User should extend Master or Slave callback to get the transaction from "transaction_started" method. Direction should be "RX" to ensure the handle is for the receive path.

Example displaying slave callback.

```
class slave_monitor_tr_events_callback extends svt_mipi_spmi_slave_monitor_callback;

 function new(string name = "slave_monitor_tr_events_callback");

  super.new(name);

 endfunction


 virtual function void transaction_started(svt_mipi_spmi_slave_monitor slave_mon,
svt_mipi_spmi_transaction xact);

  if(xact != null && xact.direction == svt_mipi_spmi_transaction::RX) begin

   slave_tr[xact.cfg.slave_id] = xact;

  end

 endfunction

endclass
```

- There are 3 events in the transaction class which will indicate command, address, and data phases. Address and Data phase variables will be incremented up on receiving each frame. So customers can wait for required events as per their requirement.

```
/** Indicates detection of Command Phase */

 bit EVENT_COMMAND_PHASE;


 /** Indicates detection of AddressPhase */

 bit [1:0] EVENT_ADDRESS_PHASE;


 /** Indicates detection of Data Phase, value will be incremented upon detection of each data
phase */

 int EVENT_DATA_PHASE;
```

The use model is applicable for all active/passive masters/slaves.

5.7 RESERVED Command Support

Reserved Commands: Command - 17h, 18h, 19h, 1Dh, 1Eh and 1Fh.

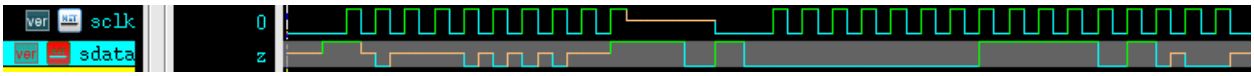
Command Frame Payload	Command	Command Frame		Data Frame	

17, 18, 19h 1D, 1E, 1Fh	RESERVED					0	0	0	1	x	x	x	x	P	BP	Nack	BP

The reserved command is neither targeted to Master nor Slave. Master or Slave VIP will drive the first 4 Bits with the value in `slave_or_master_address` field. It will drive the received command frame payload followed by Parity. It will execute Bus Park Cycle. The received component will drive Nack indicating that it received invalid command followed by Bus Park cycle. BOM will stop the clock and is ready to process next command.

Master will not disconnect from the bus after receiving RESERVED command.

Figure 5-8 Slave initiated reserved command (1Eh)



5.8 Reset Functionality

A reset api method by name `async_reset`(int unsigned reset_type) is added inside `master_agent` and `slave_agent`.

Input argument reset type carries the following values:

```
reset_type = 1(`SPMI_HARD_RESET)
reset_type = 2(`SPMI_PEER_RESET_IND)
```

- When `HARD_RESET` is applied, master/slave resets to initial state and discard current transaction.
- Sends ABORTED indication in transaction sent from monitor.
- `PEER_RESET_IND`: Should be given when `HARD_RESET` is applied on master/slave, then slave/master will discard current transaction and returns to initial state.
- Slave:
  - When `HARD_RESET` is applied to slave, slave register space is cleared.
  - When `PEER_RESET_ind` is applied to slave, slave register space is not cleared.
- When Master undergoes `HARD_RESET`, it is recommended to give `PEER_RESET_ind` to Slave, as Slave will be hung forever waiting for next posedge (Only during parity error in command frame, master can stop sclk and slave can monitor `sclk_inactivity` timer).
- `PEER_RESET_ind` indication helps MASTER/SLAVE to send ABORTED status transaction by Monitor, or else transaction with few errors like parity will be populated in transaction sent from monitor.

5.9 LOOP Back Feature

Loop back feature enables a slave component to re-transmit the received payload back to the initiator with the same or different command.

- Set the following configuration parameter to enable the loop back feature. By default, it is disabled.

```
enable_loop_back_feature = 1;
```

- The following configuration parameter must be set to identify the Source type and ID.

```
enable_data_and_sclk_en_signal_checks = 1;
```

- Loop back feature is applicable for slave to slave transfers and the following commands:
  - Register 0 Write
  - Register Write
  - Extended Register Write
  - Extended Register Write Long
- When Loop back feature is enabled, VIP must be used as a res-ponder. It does not expect any commands from test bench.
- Depending on the payload size, targeted slave VIP will frame a packet automatically (with same or different command) with same data payload back to the initiator.

5.10 Reset Coverage

Description:

One of the important feature to verify DUT is to validate RESET operation in all phases.

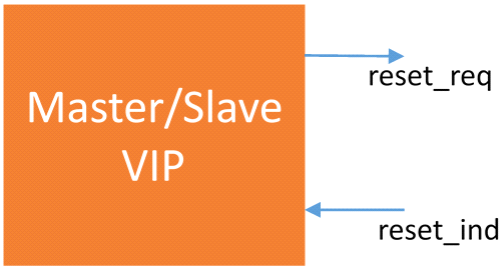
SPMI VIP provides functionality to capture clock event at which, Master/Slave VIP or Master/Slave DUT undergoes asynchronous reset and samples the information in reset covergroups.

A new configuration parameter called `enable_reset_cov` is added and when the parameter is set to one, it creates reset related covergroups.

Reset Signals:

- Two new reset signals called `reset_req` and `reset_ind` are added in SPMI Slave/Master VIP interface.
- When VIP undergoes reset, it will toggle `reset_req`.
- Active/Passive VIP continuously monitors for posedge of `reset_ind` signal and samples appropriate reset coverpoints.

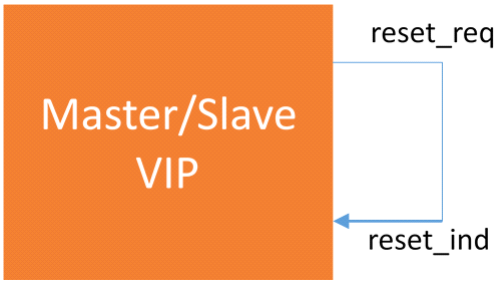
Figure 5-9 Reset Signals



Usage:

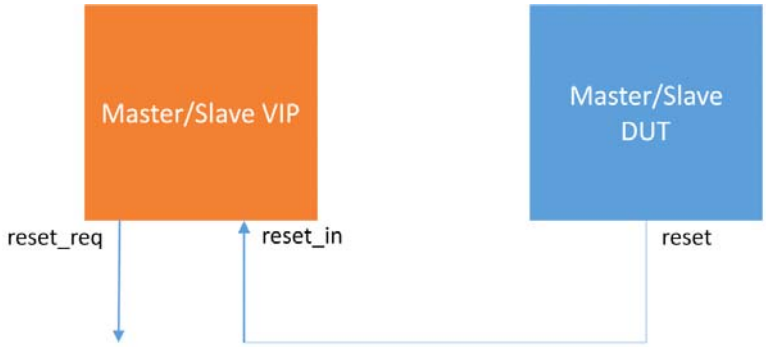
- Case 1 : If any Master/Slave VIP reset coverage is required, then connect individual master/slave `reset_ind` signal to its own `reset_req` signal.

Figure 5-10 Case 1



- Case 2 : If any Master/Slave DUT reset coverage is required, then Master/Slave DUT's reset pin can be connected to any available Master/Slave VIP's `reset_ind` pin. The Master/Slave VIP in the picture gives coverage for reset occurred in Master/Slave DUT.

Figure 5-11 Case 2



- Case 3 : If in a system, there are more Master/Slave DUTs than Master/Slave VIPs or we require reset coverage for all available Master/Slave DUT and active Master/Slave VIP, then connect DUTs reset pin to Master/Slave Passive VIP's `reset_ind` pin. In this setup, passive Master/Slave VIP gives DUT reset coverage.

Figure 5-12 Case 3

5.11 Dynamic Reconfiguration

It is an important feature that provides flexibility to change the configuration parameters on runtime.

Example Code for slave_id / master_id Reconfiguration

The reconfiguration of `slave_id` can be done after a sequence item is fired on a sequencer. The following code depicts the reconfiguration of `slave_id`:

```
begin
//Register_write sequence initiated from Master to Slave
    `svt_xvm_do_on_with(ms_write, p_sequencer.master_sequencer[0], {command_type ==
svt_mipi_spmi_transaction::EXTENDED_REGISTER_WRITE_LONG;

slave_or_master_address == slave_address;

address_frame == 16'h1000;

data_frame.size == 1;

data_frame[0] == 1;

priority_sequence == 1;})
  end
  //Slave_id reconfigured to 7
  sys_cfg.slave_cfg[0].slave_id = 4'b0111;
  system_env.slave[0].reconfigure(sys_cfg.slave_cfg[0]);

  //slave_id_map[0] updated with new slave_id value
  foreach(sys_cfg.master_cfg[i]) begin
    sys_cfg.master_cfg[i].slave_id_map[0] = 4'b0111; //
```

Note

The reconfiguration is done for `slave_id_map[0]` as the `slave_id` of 0th slave has been changed.

```
 system_env.master[i].reconfigure(sys_cfg.master_cfg[i]);
end
foreach(sys_cfg.slave_cfg[i]) begin
  sys_cfg.slave_cfg[i].slave_id_map[0] = 4'b0111; //
```

Note

The reconfiguration is done for `slave_id_map[0]` as the `slave_id` of 0th slave has been changed.

```
 system_env.slave[i].reconfigure(sys_cfg.slave_cfg[i]);
 end
```

Note

A similar example execution must be followed for `master_id` reconfiguration. The `master_id_map` array is present in the configuration object for each master.

5.12 Debug Port Support

- Configuration parameter is used to enable the debug ports.
- Added the following parameters in the interface to support debug ports.
  - `command_type`: Interface parameter `command_type` to indicate arbitration phase and `command_type` in command sequence phase.
  - `cs_info`: Interface parameter `cs_info` to indicate the sub phases in arbitration and command sequence phases like Command Frame, Command frame parity, BPC, address, address parity, data frame, `data_parity` etc.
  - `cs_data`: Interface parameter `cs_data` to indicate the data information of `slave_or_master_address`, `command_frame`, `command_parity`, `address_frame`, `address_parity`, `data_frame`, `data_parity` etc

Figure 5-13 Debug Ports Illustration in Arbitration Phase

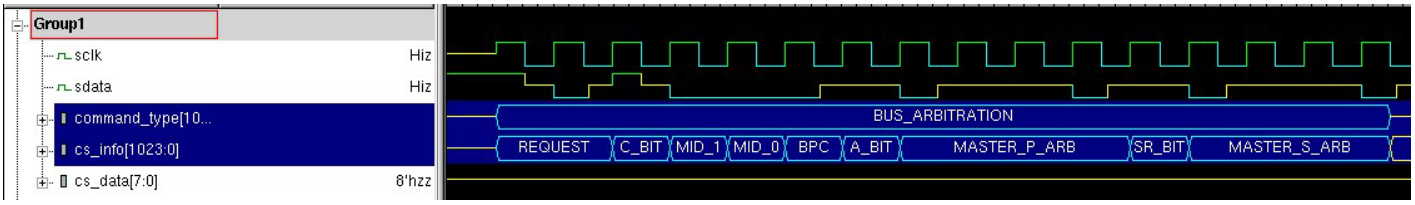


Figure 5-14 Debug Ports Illustration in TX Command Sequence Phase

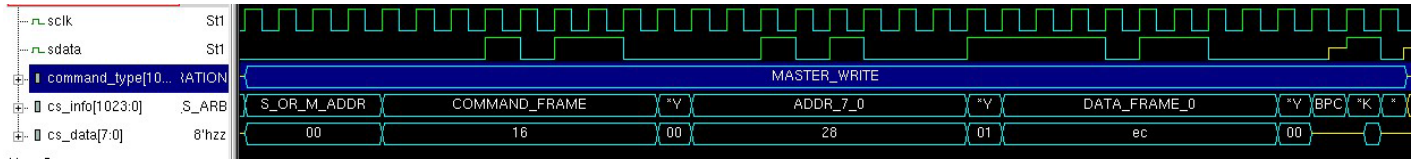
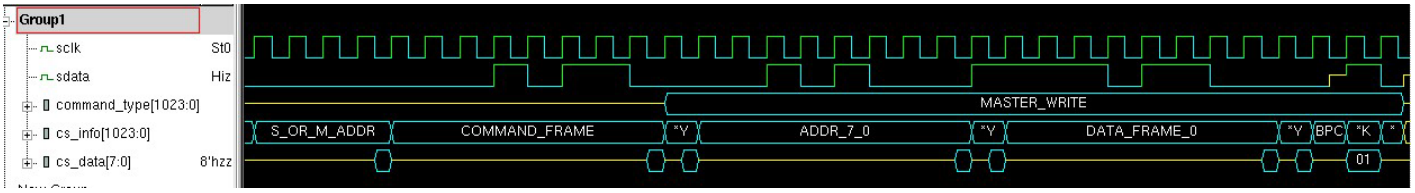


Figure 5-15 Debug Ports Illustration in RX Command Sequence Phase



5.13 Demoting Checks

See *ts.base_test.sv* in Basic example TB area (`tb_mipi_spmi_svt_uvm_basic_sys`).

1. Take instance of `svt_err_catcher`.

```
/** Create report_catcher */
 svt_err_catcher err_catcher;
```

2. Create and add the messages which should be demoted in build phase of test case.

```
virtual function void build_phase(uvm_phase phase);
    super.build_phase(phase);
```

```
    /** Create response catcher class. */
    err_catcher = svt_err_catcher::type_id::create("err_catcher");

    /** Add the report_catcher to Callback */
    uvm_report_cb::add(null, err_catcher);

    err_catcher.add_message_id_to_demote("spmi_slave_invalid_command");
  endfunction : build_phase
```

The different types of demotion for an error message is as follows:

UVM_ERROR /slowfs/vgvips21/chiranje/vip_mipi_spmi_svt_N-2018.03-3/vip/svt/common/N-2017.12/sverilog/src/vcs/svt_err_check_stats.svp(583) @ 830962000: uvm_test_top.m_env.spmi_bus_slave_agent_15 [register_fail:Bus Arbitration:Slave Arbitration:spmi_sdata_assertion_slave_bus_arbitration_check]

Description: Slave which didn't asserted sdata participated in Slave Bus Arbitration, Reference: MIPI Alliance Specification for System Power Management Interface (SPMI) Version 2.0 : 8.4 Slave Arbitration - Arbitration only if first cycle sdata asserted : Slave[15] which didn't assert sdata to initate arbitration has set sdata in slave A-bit/SR bit arbitration

- Demotion based on message ID:

```
err_catcher.add_message_id_to_demote
("spmi_sdata_assertion_slave_bus_arbitration_check");
```

- Demotion based on message text. Some text in the error message:

```
err_catcher.add_message_text_to_demote ("Arbitration only if first cycle sdata
asserted");
```

5.14 Generating NRF From Master and Slave

There are two ways in which Master sends NRF to Slave.

- While sending command from Master/Slave -
  - This can be achieved only through exception as Master/Slave never sends NRF in normal case.
  - It can be either in Command frame, Address frame or Data frame.
    - Command frame - Ensure Slave ID is 0 and command is Extended Register Write for NRF.

```
                    svt_mipi_spmi_transaction req1= new();
                  /* In body of the sequence create instances for exception and
exception_list */
                    svt_mipi_spmi_transaction_exception exc;
                    exc =
svt_mipi_spmi_transaction_exception::type_id::create("exc");
                    svt_mipi_spmi_transaction_exception_list exc_list ;
                    exc_list =
svt_mipi_spmi_transaction_exception_list::type_id::create("exc_list");

                  /* setting exception for error type */
```

```
                        exc.ERR_INVALID_PARITY_BIT_IN_COMMAND_FRAME_wt = 1;
                        exc.spmi_transaction_error =
svt_mipi_spmi_transaction_exception:: ERR_INVALID_PARITY_BIT_IN_COMMAND_FRAME;
                       /* setting number of exceptions in the exception list */
                        exc_list.num_exceptions = 1;
                        exc_list.exceptions = new[exc_list.num_exceptions];
                        exc_list.exceptions[0] = exc;

                      `svt_xvm_create_on(req1,
p_sequencer.master_sequencer[master_index]);
                       req1.reasonable_connect_request.constraint_mode(0);
                       req1.exception_list = new();
assert(req1, p_sequencer.slave_sequencer[3], {command_type ==
svt_mipi_spmi_transaction::EXTENDED_REGISTER_WRITE;

command == 0;

slave_or_master_address == 0;
                                                                              });
                        req1.exception_list = exc_list;
                       `svt_xvm_send(req1);
```

- Address frame - Ensure Address is targeted to location 0 for NRF.

```
                           svt_mipi_spmi_transaction req1=  new();
/* In body of the sequence create instances for exception and exception_list */
     svt_mipi_spmi_transaction_exception exc;
     exc = svt_mipi_spmi_transaction_exception::type_id::create("exc");
     svt_mipi_spmi_transaction_exception_list exc_list ;
     exc_list
=svt_mipi_spmi_transaction_exception_list::type_id::create("exc_list");

/* setting exception for error type */
     exc.ERR_INVALID_PARITY_BIT_IN_ADDRESS_FRAME_wt = 1;
     exc.spmi_transaction_error = svt_mipi_spmi_transaction_exception::
ERR_INVALID_PARITY_BIT_IN_ADDRESS_FRAME;

/* setting byte position , in which driving invalid parity */
     exc.corrupt_address_parity_byte_position = 1;

/* setting number of exceptions in the exception list */
     exc_list.num_exceptions = 1;
     exc_list.exceptions = new[exc_list.num_exceptions];
```

```
     exc_list.exceptions[0] = exc;
     `svt_xvm_create_on(req1, p_sequencer.master_sequencer[master_index]);
     req1.reasonable_connect_request.constraint_mode(0);
     req1.exception_list = new();
   assert(req1.randomize with                    {command_type ==
svt_mipi_spmi_transaction::EXTENDED_REGISTER_WRITE;

address_frame == 0;

                                                                        });
     req1.exception_list = exc_list;
     `svt_xvm_send(req1);
```

- Data frame - Ensure Data byte is 0. Only for the write commands.

```
                          svt_mipi_spmi_transaction req1= new();
/* In body of the sequence create instances for exception and exception_list */
     svt_mipi_spmi_transaction_exception exc =
svt_mipi_spmi_transaction_exception::type_id::create("exc");
     svt_mipi_spmi_transaction_exception_list exc_list
=svt_mipi_spmi_transaction_exception_list::type_id::create("exc_list");

/* setting exception for error type */
      exc.ERR_INVALID_PARITY_BIT_IN_DATA_FRAME_wt = 1;
      exc.spmi_transaction_error = svt_mipi_spmi_transaction_exception::
ERR_INVALID_PARITY_BIT_IN_DATA_FRAME;

/* setting byte position , in which driving invalid parity */
      exc.corrupt_data_parity_byte_position = 1;

/* setting number of exceptions in the exception list */
     exc_list.num_exceptions = 1;
     exc_list.exceptions = new[exc_list.num_exceptions];
     exc_list.exceptions[0] = exc;
     `svt_xvm_create_on(req1, p_sequencer.master_sequencer[master_index]);
     req1.reasonable_connect_request.constraint_mode(0);
     req1.exception_list = new();
   assert(req1.randomize with                    {command_type ==
svt_mipi_spmi_transaction::EXTENDED_REGISTER_WRITE;

data_frame[0]   == 0;

                                                                        });
     req1.exception_list = exc_list;
```

```
`svt_xvm_send(req1);
```

You can run any sequence in the transaction sequence collection but ensure that you are passing invalid parity exception to designated Master/Slave.

- While sending data in response to the command (Master/Slave).
  - Send a Master Read command frame to Master with invalid address. The following parameters decide the valid start and end register addresses. If the received address does not fall into this range, then the Master drives NRF in response to the command in DATA FRAME.
  - Send a Register Read or Extended Register Read or Extended Register Read long command frame to Slave with invalid address. The following parameters decide the valid start and end register addresses. If the received address not fall into this range, then the Slave drives NRF in response to the command in DATA FRAME.
  - The following configuration parameter indicates valid Register Start address.

    ```
    rand bit [15:0] register_address_start = 16'h0;
    ```
  - The following configuration parameter indicates valid Register End address.

    ```
    rand bit [15:0] register_address_end   = 16'hFFFF;
    ```

    Example :

    ```
    register_start_address = 0;
    register_end_address   = 10;
    command can be issued to any address location except in range [0,10] will
    result in NRF.
    ```

5.15 Connecting Master through C-Bit in Single and Multiple Master System

- When only master is present in environment:
  - Ensure that the bus is not initialized, that is, the Master is not connected yet. This can be achieved by setting below configuration parameter to 0.

    ```
    cfg.system_cfg.master_cfg[0].is_connected_initial = 0;
    ```
  - Firstly, initiate request from RCS Slave device and then initiate `connect_request` from Master with some delay that allows the Slave to initiate request. Then, the Master connects through C bit.

```
svt_mipi_spmi_transaction req1,req2;
                    fork
                    begin
                     `svt_xvm_do_on_with(req1, p_sequencer.slave_sequencer[3],
{command_type == svt_mipi_spmi_transaction::REGISTER_WRITE;

slave_or_master_address == 0;

priority_sequence              == 1;

})
                     end
                     begin
                       #1us;
```

```
                        `svt_xvm_create_on(req2, p_sequencer.master_sequencer[3]);
                         req2.reasonable_connect_request.constraint_mode(0);
                       assert(req2.randomize with {connect_request
== 1;
                                                                slave_or_master_address
== 0;
                                                command_type
==svt_mipi_spmi_transaction::REGISTER_WRITE;
                                                                    });
                          `svt_xvm_send(req2);
                        end
                        join
```

- In case of multiple masters are present in environment:
  - Ensure that the Master intended to be connected is not disconnected initially. This can be achieved by setting the following configuration parameter to 0.

    ```
    cfg.system_cfg.master_cfg[0].is_connected_initial = 0;
    ```

  - Initiate connect request from the disconnected Master.
    - Connect to bus and drive command.

      ```
      svt_mipi_spmi_transaction req2;
      `svt_xvm_create_on(req2, p_sequencer.master_sequencer[3]);

      req2.reasonable_connect_request.constraint_mode(0);
                          assert(req2.randomize with {connect_request
      == 1;

      slave_or_master_address == 0;
                                                        command_type
      == svt_mipi_spmi_transaction::REGISTER_WRITE;
                                                                    });
                                    `svt_xvm_send(req2);
      ```

    - Connect to bus and don't drive any command.

      ```
      svt_mipi_spmi_transaction req2;
      `svt_xvm_create_on(req2, p_sequencer.master_sequencer[1]);

      req2.reasonable_connect_request.constraint_mode(0);

      req2.reasonable_is_transmission_disable.constraint_mode(0);
                            assert(req2.randomize with {connect_request
      == 1;
                                     is_transmission_disable == 1;
                                   });
                                  `svt_xvm_send(req2);
      ```

5.16 PullDown Data Signal Support

Added new signal `sdata_pd` in *svt_mipi_spmi_if.svi* file for the pulldown feature. This new signal `sdata_pd` should be connected to the DUT sdata signal instead of VIP sdata.

- `SVT_MIPI_SPMI_PULL_DOWN_SUPPORT` define must be set.
- The strength of the pulldown signal can be specified by setting the define `SVT_MIPI_SPMI_PULLDOWN_SIGNAL_STRENGTH`, whose default value is We0.

5.17 Corrupt Ack/ Nack From Slave

Added new callback `corrupt_ack_nack` on the Slave side to corrupt ack/nack field on the Slave side.

- `corrupt_ack_nack_field` must be set to 1 to corrupt ack/nack field.
- Value in `ack_nack_err_value` field will be driven on the bus.

```
virtual function void corrupt_ack_nack(svt_mipi_spmi_slave slave, input
svt_mipi_spmi_transaction xact, ref bit corrupt_ack_nack_field, ref logic
ack_nack_err_value);

endfunction
```