

Verification Continuum™

VC Verification IP

MIPI M-PHY

UVM User Guide

Version U-2022.12, December 2022



Copyright Notice and Proprietary Information

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

Destination Control Statement

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

Disclaimer

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

Trademarks

Synopsys company and certain 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

Chapter	
Preface	9
About This Guide	9
Web Resources	9
Customer Support	9
Synopsys Statement on Inclusivity and Diversity	9
Chapter 1	
Introduction	11
1.1 Product Overview	11
1.2 Prerequisites	12
1.3 References	12
1.4 Supported M-PHY Features	12
1.4.1 Simulator and Methodology Support	12
1.4.2 Protocol Features	12
1.4.3 Verification Features	14
1.4.4 Debug Features	14
1.4.5 Methodology Features	14
1.4.6 Ease of Use Features	14
1.4.7 Unsupported Features	14
Chapter 2	
Installation and Setup	15
2.1 Verifying the Hardware Requirements	15
2.2 Verifying the Software Requirements	15
2.2.1 Platform or OS and Simulator Software	16
2.2.2 Synopsys Common Licensing (SCL) Software	16
2.2.3 Other Third Party Softwares	16
2.3 Preparing for Installation	16
2.4 Downloading and Installing	17
2.4.1 Downloading From the Electronic Software Transfer (EST) System (Download Center)	17
2.4.2 Downloading Using FTP with a Web Browser	18
2.5 What's Next?	20
2.5.1 Licensing Information	20
2.5.2 Environment Variable and Path Settings	21
2.5.3 Determining Your Model Version	21
2.5.4 Integrating a Synopsys IP into Your Testbench	22
2.6 Installing and Setting Up More than One VIP Protocol Suite	24
2.6.1 Running the Example With +incdir+	24
2.6.2 Getting Help on Example Run/make Scripts	25

2.7	Updating an Existing Model	26
2.7.1	Removing VC VIP Models From a Design Directory	26
2.7.2	Reporting Information About DESIGNWARE_HOME or a Design Directory	26
2.7.3	The dw_vip_setup Utility	27
2.7.4	Setting Environment Variables	27
2.7.5	The dw_vip_setup Command	27
2.8	Include and Import Model Files into Your Testbench	30
2.9	Compile and Run Time Options	31
Chapter 3		
General Concepts		33
3.1	Introduction to UVM	33
3.2	M-PHY VIP in an UVM Environment	33
3.2.1	Base Classes	35
3.2.2	UVM Components	35
3.2.3	M-PHY VIP Objects	36
3.2.4	Interfaces and Modports	41
3.2.5	Constraints	41
3.2.6	Functional Coverage	43
3.3	Covergroup Organization	43
3.3.1	Enable or Disable Functional Coverage	45
Chapter 4		
M-PHY VIP Agent Overview		49
4.1	M-PHY TX or RX Agent UVM Component Stack	49
4.1.1	M-PHY TX Agent	50
4.1.2	M-PHY RX Agent	51
4.1.3	M-PHY Agent Modes	52
4.1.4	M-PHY TX Agent Driver	53
4.1.5	M-PHY RX Agent Receiver	54
4.1.6	M-PHY Agent Monitor	55
4.1.7	M-PHY Agent Sequencers	57
4.2	M-PORT Lane Management Agent UVM Component	57
4.2.1	M-PHY M-Port Lane Management Agent Driver	58
4.2.2	M-PHY M-Port Lane Management Agent Receiver	60
4.2.3	M-PHY M-Port Lane Management Agent Monitor	61
4.3	M-PHY Model Agent UVM Component	63
4.3.1	M-PHY TX-Model Agent	64
4.3.2	M-PHY RX-Model Agent	64
Chapter 5		
Using M-PHY Verification IP		67
5.1	Creating a Test Environment	67
5.1.1	Creating a Test Environment Without Lane Management	68
5.1.2	Creating a Test Environment With Lane Management	69
5.2	Instantiating the VIP	71
5.2.1	Instantiating the VIP Without Lane Management	71
5.2.2	Instantiating the VIP With Lane Management	71
5.3	Configuring the VIP Models	72
5.3.1	Configuring the VIP Models Without Lane Management	72

5.3.2	Configuring the VIP Models With Lane Management	76
5.3.3	Configuring the Passive Serial Monitors (TX Monitor is Connected to TX DUT or RX Monitor is Connected to RX DUT)	83
5.4	Connecting the Interface to DUT	85
5.4.1	Connecting the Interface to DUT Without Lane Management	85
5.4.2	Connecting Lane Management Interface to DUT	85
5.5	Generating Constrained Random Stimulus	86
5.6	Controlling the Test	86
5.7	Lane-to-Lane Skew	86
5.7.1	Skew Insertion	87
5.7.2	Use Case with Mport LM Agent on SERIAL and RMMI Interface	88
5.7.3	Inserting Random Skew	89
5.8	Exception Support	89
5.8.1	Exception Class Hierarchy	90
5.8.2	Exception Architecture	90
5.8.3	Exception Architecture Development Flow	91
5.8.4	M-PHY Exceptions List	93
5.8.5	M-PHY Exceptions Use Cases	97
5.8.6	Lane Management Exceptions Use Cases	100
5.9	Inserting PPM Drift and Clock Recovery	101
5.10	M-PHY Specification Version 2.0 Feature Support	101
5.10.1	Use Case to Enable M-PHY VIP for Specification Version 2.0 Features	103
5.11	M-PHY Specification Version 3.0 Feature Support	104
5.12	M-PHY Specification Version 3.1 Feature Support	104
5.12.1	Use Case to Enable M-PHY VIP for Specification Version 3.1 Features	106
5.13	Line Reset	106
5.13.1	Interface Behavior on Reception of LINERESET CTRL SAP in BURST State	107
5.13.2	Use Case for Line Reset	108
5.14	Lane to Lane Skew in Ctrl Transactions	108
5.14.1	Use Case	109
5.15	M-PHY Specification Version 4.0 and Later Feature Support	109
5.15.1	Use Case for Adapt Feature	110
5.16	Set Clock Period for PWM and HS Mode	111
Chapter 6		
VIP Tools		113
6.1	Using Native Protocol Analyzer for Debugging	113
6.1.1	Introduction	113
6.1.2	Prerequisites	113
6.1.3	Invoking Protocol Analyzer	114
6.1.4	Documentation	114
6.1.5	Limitations	114
Chapter 7		
Verification Topologies		115
7.1	Single-Lane M-PHY DUT (Signal Interface on Both Sides)	115
7.2	Single-Lane M-PHY DUT (Serial Interface)	117
7.3	Single-Lane M-PHY DUT (RMMI Remote Interface)	118
7.4	Multi-Lane M-PHY DUT (Serial or RMMI Remote Interface)	119
7.5	Multi-Lane M-PHY DUT (Serial or RMMI Local Interface)	120

7.6 Multi-Lane M-PHY VIP with MPHY Model	121
7.7 Multi-Lane M-PHY VIP With Test Modes	122
7.7.1 Near-End Loopback Mode	122
7.7.2 Far-End Loopback Mode	124
Chapter 8	
Usage Notes	127
8.1 Verification Environment Initialization	129
8.1.1 Configuration Initialization at the Start of Simulation	129
8.1.2 Configuration Initialization After Reset	130
8.2 Timescale Usage	132
8.3 M-PHY Package Usage	132
8.4 M-PHY Clocking Requirements	134
8.4.1 Serial Mode Clocking Requirements	134
8.4.2 RMMI Mode Clocking Requirements	135
8.5 Enabling the DIF_Z functionality	137
8.6 Configurable RESET Duration	138
8.6.1 Use Case to Assert and De-Assert RESET Through CTRL SAP	138
8.6.2 Use Case to Assert and De-Assert RESET by Increasing the Reset Duration	139
8.6.3 Use Case Without any Overriding	139
8.7 Configurable Prepare and Sync Length Through PREPARE SAP	139
8.7.1 Use Case With Override Feature	140
8.7.2 Use Case Without any Overrides for Prepare Length, Sync Length and Sync Range	140
8.8 Configurable Number of Fillers in a Transaction at Lane Management	140
8.9 Enable or Disable External SYNC at Lane Management	141
8.10 First SYNC Symbol Selection Between D10.5 and D26.5	142
8.11 EVENT_MPHY_HIBERN8_ENTERED and EVENT_MPHY_HIBERN8_EXITED Events Usage	142
8.12 EVENT_MPHY_DATA_STARTED and EVENT_MPHY_DATA_ENDED Events Usage	142
8.13 M-RX Capability and M-TX Configuration Relation	143
8.14 Transfer Data on TX Sublink and RX Sublink	143
8.15 Midstream De-Assertion of RX_PhyDORDY or TX_ProtDORDY	143
8.15.1 MTX as RMMI REMOTE	144
8.15.2 MTX as RMMI LOCAL	144
8.15.3 Effect on Other Signals	144
8.15.4 Use Case to De-Assert RX_PhyDORDY or TX_ProtDORDY Through DATA SAP	144
8.16 PWM Gear0 Requirements	145
8.17 Aligned and Unaligned Transfers	146
8.18 HIBERN8 Time Capability Configuration	147
8.19 Dynamic Change in Active Lanes	147
8.19.1 Use Model	148
8.20 Bypass Mode	151
8.20.1 Bypass Mode Limitations	152
8.20.2 Marker 0 in Bypass mode	152
8.21 Configuration Register Read Value Check	153
8.22 Test Modes	153
8.22.1 Test Mode Patterns	153
8.22.2 Test Mode Configuration	155
8.22.3 Data Pattern Transmission Modes	157
8.23 Line Control Command (LCC) with Media Convertor	158
8.23.1 TX_LCC_Sequencer Configuration Settings	158

8.23.2 Switching from LS_MODE to HS_MODE and Vice Versa	159
8.23.3 Switching from LS_MODE to HS_MODE via HIBERN8 state	159
8.23.4 LCC_READ operation when OMC is not present	159
8.23.5 Points to Note	160
8.24 User-Specific Burst Closure Pattern	160
8.25 Configuration Register Read on Configurable Number of Clocks	160
8.26 Inline Config Feature (Optional)	161
8.27 De-Assertion of TX_SaveState_Status_N Signal (Optional)	161
8.28 T_activate Timer on Exit to HIBERN8	161
8.28.1 Use Case	162
8.29 Control Symbols MK1,MK3,MK4,MK5 & MK6 insertion in a transaction at Lane Management	163
8.29.1 Points to Remember	166
8.30 Unaligned EOB at M-RX PHY	167
8.30.1 Use Case	168
8.31 RX_Burst Assertion at First MK0 in LS-MODE	168
8.31.1 Use Case	169
8.32 User Defined Attributes	169
8.33 Send Multiple External SYNC Symbols in a Transaction at Lane Management	171
8.33.1 Points to Remember	171
8.33.2 Use Model	171
8.34 Configure Number of Clock to Enable rx_hibern8exit_type_I Signal Check (RMMI Interface).	173
8.35 Configure Initial Value of Running Disparity	173
8.35.1 Use Model	173
8.36 Clock Jitter	174
8.36.1 Use Model With no PPM Limit	174
8.36.2 Use Model With PPM Limit	178
8.37 Time Scale Independent Timers	179
8.38 Support of Dynamic Reconfiguration for MPHY	179
8.39 Dynamic Symbol Length Variation	181
8.39.1 Use Model	181
8.40 Setting of Symbol Length	182
8.41 Configurable Extra Bits in PREPARE State	182
8.41.1 Use Case	183
8.42 Configure Number of Clock to Enable invalid_cfgrdyn_signal_val_check Signal Check (RMMI Interface)	183
8.43 Check for RX_LCCRdDet Signal	184
8.43.1 Use case	184
8.44 Dithering Support	184
8.44.1 Use Case for Dithering	185
8.45 Allow LCC_READ in all PWM gears.	185
8.46 Difference Between Prepare State and RX_Burst	185
8.47 PWM Transmit Ratio Usage	186
8.47.1 Observation	188
8.48 Enable External REF Clock (refclk) Support in VIP	188
8.49 Max Lane Skew	189
8.50 TX_DP and TX_DN Lines Driving User Control	189
8.51 User-Defined Payload Pattern	190
8.52 Enabling Debug Port View	190
8.53 Corrupt 8b10b ADAPT Pattern	190
8.54 Control De-Assertion of TX_ADAPT_REQ	191

8.55	Corrupt 650bits of ADAPT Pattern Length	191
8.56	80bit Symbol Length	192
8.57	TX/RX Reset Mode Support	192
8.58	Support for LCC in MPHY5.0	193
8.59	Support for All PWM Gear in MPHY5.0	193
Chapter 9		
Troubleshooting		195
9.1	Using Trace Files for Debugging	195
9.1.1	Data Transaction Trace File	195
9.1.2	Control Transaction Trace File	196
9.2	Enabling Tracing	196
9.3	Setting Verbosity Levels	196
9.3.1	Setting Verbosity in the Testbench	196
9.3.2	Setting Verbosity During Run-Time	197
9.4	Debug Port	197
9.4.1	Enabling Debug Port	198
10.1	Introduction	199
10.2	Debug Automation	199
10.3	Enabling and Specifying Debug Automation Features	199
10.4	Debug Automation Outputs	201
10.5	FSDB File Generation	202
10.5.1	VCS	202
10.5.2	Questa	202
10.5.3	Incisive	202
10.6	Initial Customer Information	202
10.7	Sending Debug Information to Synopsys	202
10.8	Limitations	203

Preface

About This Guide

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

Web Resources

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

Customer Support

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

1. Go to <https://solvnetplus.synopsys.com> and open a case.
Enter the information according to your environment and your issue.
2. Send an e-mail message to support_center@synopsys.com.
Include the Product name, Sub Product name, and Tool Version in your e-mail so it can be routed correctly.
3. Telephone your local support center.
 - ◆ North America:
Call 1-800-245-8005 from 7 AM to 5:30 PM Pacific time, Monday through Friday.
 - ◆ All other countries:
<https://www.synopsys.com/support/global-support-centers.html>

Synopsys Statement on Inclusivity and Diversity

Synopsys is committed to creating an inclusive environment where every employee, customer, and partner feels welcomed. We are reviewing and removing exclusionary language from our products and supporting customer-facing collateral. Our effort also includes internal initiatives to remove biased language from our engineering and working environment, including terms that are embedded in our software and IPs. At the same time, we are working to ensure that our web content and software applications are usable to people of

varying abilities. You may still find examples of non-inclusive language in our software or documentation as our IPs implement industry-standard specifications that are currently under review to remove exclusionary language.

1

Introduction

This chapter provides you the product overview, prerequisites, VIP integration and test work flow, and the list of features supported by M-PHY VIP.

This chapter has the following sections:

- ❖ [Product Overview](#)
- ❖ [Prerequisites](#)
- ❖ [References](#)
- ❖ [Supported M-PHY Features](#)

1.1 Product Overview

M-PHY is a suite of UVM-based verification components that are compatible for use with SystemVerilog-compliant testbenches. The M-PHY VIP suite simulates TX and RX sublink transactions, through agents, as defined by the M-PHY specification.

VIP M-PHY agent can be configured as a Controller or PHY component.

The M-PHY agent supports functionality normally associated with UVM components, including the creation of transactions, checking and reporting the protocol correctness, transaction logging, and functional coverage.

The M-PHY VIP supports verification of SoC designs that include interfaces implementing the *Draft MIPI Alliance Specification for MPHY Version 4.1 - 01 December 2016*. This document describes the use of this VIP in testbenches that comply with the SystemVerilog Universal Verification Methodology (UVM).

This approach leverages advanced verification technologies and tools that provide:

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

1.2 Prerequisites

You must be familiarized with the following:

- ❖ M-PHY specification
- ❖ Object-oriented programming
- ❖ SystemVerilog and
- ❖ Universal Verification Methodology (UVM).

1.3 References

For more information on M-PHY, you can refer to the following specifications:

- ❖ *MIPI Alliance Specification for M-PHY Version 2.0- 04 April 2012*
- ❖ *MIPI Alliance Specification for M-PHY Version 1.40.00 - 30 November 2011*
- ❖ *MIPI Alliance Specification for M-PHY Version 4.0- Version 21 December 2014*
- ❖ *MIPI Alliance Specification for M-PHY Version 3.1 - 17 February 2014*
- ❖ *MIPI Alliance Specification for M-PHY Version 3.0- Version 06 - 26 July 2013*
- ❖ *MIPI Alliance Specification for M-PHY Version 4.1- 01 December 2016*
- ❖ Class Reference for M-PHY is available at:
`$DESIGNWARE_HOME/vip/svt/mphy_svt/latest/doc/class_ref/mphy_svt_uvm_class_reference/html/index.html`
- ❖ The UVM user examples are available at:
`$DESIGNWARE_HOME/vip/svt/mphy_svt/latest/examples/sverilog`

The README file, which resides in the Examples location, provides the information about the examples.

1.4 Supported M-PHY Features

The following sections discuss the various features supported by M-PHY.

1.4.1 Simulator and Methodology Support

For information on the supported simulators and methodology, see *M-PHY Release Notes*.

1.4.2 Protocol Features

The M-PHY supports these protocol features:

- ❖ Protocol Layer Support
 - ◆ Interfaces Supported:
 - ❖ Serial Interface - SYS signaling and PWM signaling
Serializer (M-TX) or Deserializer (M-RX)
8B10B coding (M-TX) or 8B10B decoding (M-RX)
 - ❖ Reference M-PHY MODULE Interface (RMMI) - Controller and PHY
Configurable 10, 20, and 40 bit data interface widths

- ◆ Type I and Type II Module
- ◆ High Speed and Low Speed
 - ◇ All HS Gears - HSG1, HSG2, HSG3, HS SeriesA, HS SeriesB
 - ◇ All PWM Gears - PWMG1 to PWMG7
- ◆ CTRL and DATA SAP
 - ◇ Interface with the protocol adaptor layer
 - ◇ Independent port or channel for CTRL and DATA SAP
 - ◇ All Capability (READ only) attributes and Configuration attributes (READ or WRITE)
- ◆ Dynamic reconfiguration. For example, LS-HS, HS-LS mode change, and gear change through CTRL SAP Sequencer
- ◆ Programmable parameters like PREPARE length, SYNC pattern and length
- ◆ External or internal SYNC
- ◆ Automatic filler insertion
- ◆ Support to start in a known initial state other than DISABLED (for example, STALL state)
- ◆ Configurable timing attributes
- ❖ Lane Management Agent features:
 - ◆ Top layer is lane management layer
 - ◆ PA SAP transaction object comprises of the following attributes:
 - ◇ Payload related:
 - Payload size, Number of data bytes to be transferred
 - Payload [payload size]
 - ◇ Filler insertion:
 - Filler insertion mode (Random or User defined)
 - Filler size (non-zero indicates the filler to be inserted). By default, it's zero
 - Filler insertion positions [Filler size] => Required when user defined
 - ◆ Data Sequencers:
 - ◇ Virtual sequencer for TX sub-link
 - ◇ For RX path, data is collected from the individual lanes and a PA SAP object is created which will be pushed on an output port
 - ◆ Control Sequencers for both TX and RX Sub-links (for service like reset, lane change, rate change, power management and so on)
 - ◆ Data transaction objects:
 - ◇ Lane management layer
 - Lane management transaction objects - (MASTER_MPORT, SLAVE_MPORT)
 - ◇ Physical adaptation layer
 - M-PHY Data transaction objects - (MTX, MRX)
 - M-PHY Control transaction objects - (MTX, MRX)
 - ◆ Configuration objects:

- ✧ System configuration - MTX
- ✧ System configuration - MRX
- ◆ Status objects:
 - ✧ System status - MTX
 - ✧ System status - MRX
- ◆ Exception objects:
 - ✧ MTX exceptions for both Data and Control on Serial and RMMI interface
 - ✧ Lane Management exceptions
- ◆ Optical Media Converter with data width of 10/20 bits
- ◆ Test mode of operation (Loopback with far-end and near-end)

1.4.3 Verification Features

M-PHY supports the following verification features:

- ❖ VIP configurable to operate as Serial or RMMI Remote (PHY) or RMMI Local (Controller)
- ❖ Built-in protocol checks
- ❖ Exceptions Injection
- ❖ Functional Coverage

1.4.4 Debug Features

M-PHY supports the following debug features:

- ❖ Debug Port
- ❖ Protocol Analyzer
- ❖ Trace Information Support

1.4.5 Methodology Features

M-PHY supports the following methodology features:

- ❖ Single agent configurable as M-TX and M-RX per instance
- ❖ Analysis ports for connecting M-TX or M-RX instance to scoreboard, or any other component
- ❖ Callbacks for M-PHY agent M-TX or M-RX and monitor components

1.4.6 Ease of Use Features

M-PHY supports the following ease of use features:

- ❖ Basic level examples to illustrate to use the VIP in SV: UVM test benches

1.4.7 Unsupported Features

The following protocol features are not supported:

- ❖ Optical Media Converter for data width of 40 bits

2

Installation and Setup

This chapter leads you through installing and setting up the M-PHY. Once you complete the checklist mentioned in this section, the provided example testbench will be operational and the M-PHY VIP will be ready to use.

The checklist consists of the following major steps:

- ❖ [Verifying the Hardware Requirements](#)
- ❖ [Verifying the Software Requirements](#)
- ❖ [Preparing for Installation](#)
- ❖ [Downloading and Installing](#)
- ❖ [What's Next?](#)
- ❖ [Installing and Setting Up More than One VIP Protocol Suite](#)
- ❖ [Updating an Existing Model](#)
- ❖ [Include and Import Model Files into Your Testbench](#)
- ❖ [Compile and Run Time Options](#)



Note

If you encounter any problems while installing the M-PHY, contact customer support.

2.1 Verifying the Hardware Requirements

The M-PHY requires the following configuration for a 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 the Software Requirements

The M-PHY VIP is qualified for use with certain versions of platforms and simulators. This section lists the required softwares for M-PHY VIP.

2.2.1 Platform or OS and Simulator Software

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

Support Matrix for M-PHY: [M-PHY Release Notes](#)

2.2.2 Synopsys Common Licensing (SCL) Software

The SCL software provides the licensing function for the M-PHY. Acquiring the SCL software is covered in the installation instructions in [Licensing Information](#).

2.2.3 Other Third Party Softwares

- ❖ **Adobe Acrobat:** M-PHY documents are available in Acrobat PDF files. Adobe Acrobat Reader is available for free from <http://www.adobe.com>.
- ❖ **HTML browser:** M-PHY includes class reference documentation in HTML. The following browser or platform combinations are supported:
 - ◆ Microsoft Internet Explorer 6.0 or later (Windows)
 - ◆ Firefox 1.0 or later (Windows and Linux)
 - ◆ Netscape 7.x (Windows and Linux)

2.3 Preparing for Installation

1. Set `DESIGNWARE_HOME` to the following absolute path where M-PHY is installed:


```
setenv DESIGNWARE_HOME absolute_path_to_designware_home
```
2. 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 Synopsys 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

Attention

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

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

Passive mode FTP is required. The passive command toggles between passive and active mode. If your FTP utility does not support passive mode, use http. For additional information, refer to the following web page:

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

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

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

Figure 2-1 SolvNetPlus Selections for VIP Download

The figure illustrates the steps to download the Synopsys VIP Library from the SolvNetPlus website. The steps are as follows:

- Click the **Downloads** tab in the top navigation bar.
- Under **Synopsys VIP Library**, choose your version number from the list below. The selected version is **J-2014.06**.
- Click **YES, I AGREE TO THE ABOVE TERMS** to accept the license.
- Click **Download Here** to initiate the download.
- The download details for **Synopsys VIP Library: J-2014.03** are shown, including a list of files and their sizes.
- The filename **vip_amba_svt_2.45a.run** is highlighted as the file to be downloaded.

- 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.

2.4.2 Downloading Using FTP with a Web Browser

- Follow the above instructions through the product version selection step.
- Click the **Download via FTP** link instead of the **Download Here** button.

- c. Click the Click Here To Download button.
- d. Select the file(s) that you want to download.
- e. Follow browser prompts to select a destination location.



Note

If you are unable to download the Verification IP using above instructions, refer to [“Customer Support”](#) section to obtain support for download and installation

2.5 What's Next?

The following sections describe the details of the different steps you performed during installation and setup:

- ❖ [Licensing Information](#)
- ❖ [Environment Variable and Path Settings](#)
- ❖ [Determining Your Model Version](#)
- ❖ [Integrating a Synopsys IP into Your Testbench](#)

2.5.1 Licensing Information

The MPHY uses the Synopsys Common Licensing (SCL) software to control its usage. You can find general SCL information at:

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

The MPHY VIP product is enabled by the following features defined in the order listed. After, a required feature or a set of features are successfully checked out, the VIP stops looking for other licenses.

- ❖ VIP-MPHY-SVT



Note

The licensing key must reside in the files that are indicated by specific environment variables. To get information about setting these licensing environment variables, see [Environment Variable and Path Settings](#).

2.5.1.1 If Licensing Fails

By default, simulations exit with an error when a VIP license cannot be secured. Alternatively, the `DW_NOAUTH_CONTINUE` environment variable can be set to allow simulations to continue when one or more VIP models fail to authorize. Unauthorized VIP models essentially become disabled when `DW_NOAUTH_CONTINUE` is set to any value.

```
% setenv DW_NOAUTH_CONTINUE
```

Also, some simulation environments allow license polling, which pauses the simulation until a license is available. License polling is described in the next section. If you encounter problems with licensing, contact Synopsys customer support.

2.5.1.2 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 following `DW_WAIT_LICENSE` environment variable:

- ❖ 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.

2.5.1.3 Simulation License Suspension

All the Verification IP products support license suspension. The simulators that support license suspension, allow the model to check in its license token while the simulator is suspended, and then check out the license token when the simulation is resumed.

**Note**

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

2.5.2 Environment Variable and Path Settings

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

- ❖ `DESIGNWARE_HOME` – The absolute path to where the VIP is installed.
- ❖ `DW_LICENSE_FILE` – The absolute path to a file that contains the license keys for VIP product software or the `port@host` reference to this file.
- ❖ `SNPSLMD_LICENSE_FILE` – The absolute path to a file that contains the license keys for Synopsys Common Licensing (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. In addition, include the absolute path to the third party executable in your `PATH` variable.

**Note**

For faster license checkout of VC 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 VC VIP License can be set in either of the three license variables mentioned above with the order of precedence for checking the variables being:

`DW_LICENSE_FILE -> SNPSLMD_LICENSE_FILE -> LM_LICENSE_FILE`

**Note**

If `DW_LICENSE_FILE` environment variable is enabled, VIP will ignore `SNPSLMD_LICENSE_FILE` and `LM_LICENSE_FILE` settings.

For efficient VC VIP license checkout performance, set the `DW_LICENSE_FILE` with only the License servers which contain VC VIP licenses.

**Note**

Include the absolute path to the third party executable in your `PATH` variable.

2.5.2.1 Simulator-Specific Settings

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

2.5.3 Determining Your Model Version

The version of the MPHY at the time of publication is **O-2018.12-1**. The following steps describe 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 the 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 following setup utility:

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

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

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

2.5.4 Integrating a Synopsys IP into Your Testbench

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

- ❖ [Creating a Testbench Design Directory](#)
- ❖ [The dw_vip_setup Utility](#)

2.5.4.1 Creating a Testbench Design Directory

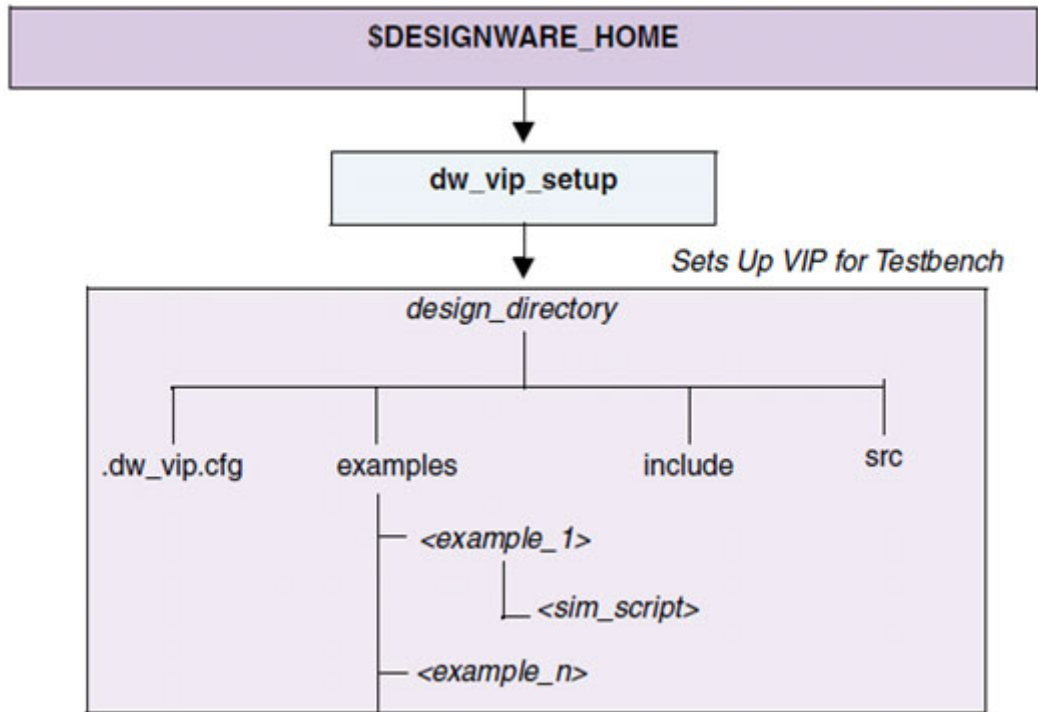
A design directory contains a version of VIP that is set up and ready to use in a testbench. The `dw_vip_setup` utility is used to create design directories. For more information on `dw_vip_setup`, see [The dw_vip_setup Utility](#).

**Note**

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

A design directory gives you the control over the version of VIP in your testbench as it is isolated from the `DESIGNWARE_HOME` installation. You can use `dw_vip_setup` to update the VIP in your design directory. [Figure 2-2](#) shows this process and the contents of a design directory.

Figure 2-2 Design Directory Created by dw_vip_setup



A design directory contains:

examples

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

include

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

src

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

.dw_vip.cfg

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



Note

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



Attention

There must be only one `design_dir` installation per simulation, regardless of the number of Synopsys Verification and Implementation IPs you have in your project. It is recommended not to create this directory in `$DESIGNWARE_HOME`.

2.6 Installing and Setting Up More than One VIP Protocol Suite

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

The examples in this chapter call that directory `design_dir`, but you can use any name. In this example, assume you have the AXI suite set up in the `design_dir` directory. In addition to the AXI VIP, you require the Ethernet and USB VIP suites.

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

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

```
// First install AXI.
%unix> $DESIGNWARE_HOME/bin/dw_vip_setup -path /tmp/design_dir
-add axi_system_env_svt-svlog

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

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

For more information on other model names, see the VIP documentation.



Note

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

2.6.1 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 /examples/sverilog/mphy_svt/tb_mphy_svt_uvm_basic_sys/
// To run the example using the generated run script with +incdir+
./run_mphy_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= \
```



```
+define+SVT_LOADER_UTIL_ENABLE_DWHOME_INCDIRS \
+incdir+/vip/svt/mphy_svt/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+SYNOPSISYS_SV \
+incdir+<testbench_dir>/examples/sverilog/mphy_svt/tb_mphy_svt_uvm_basic_sys/
. \
+incdir+<testbench_dir>/examples/sverilog/mphy_svt/tb_mphy_svt_uvm_basic_sys/
../.. /env \
+incdir+<testbench_dir>/examples/sverilog/mphy_svt/tb_mphy_svt_uvm_basic_sys/
../env \
+incdir+<testbench_dir>/examples/sverilog/mphy_svt/tb_mphy_svt_uvm_basic_sys/
env \
+incdir+<testbench_dir>/examples/sverilog/mphy_svt/tb_mphy_svt_uvm_basic_sys/
dut \
+incdir+<testbench_dir>/examples/sverilog/mphy_svt/tb_mphy_svt_uvm_basic_sys/
hdl_interconnect \
+incdir+<testbench_dir>/examples/sverilog/mphy_svt/tb_mphy_svt_uvm_basic_sys/
lib \
+incdir+<testbench_dir>/examples/sverilog/mphy_svt/tb_mphy_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

2.6.2 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_mphy_model_lm_mport_uvm_sys
usage: run_mphy_model_lm_mport_uvm_sys [-32] [-verbose] [-debug_opts] [-waves] [-clean]
[-nobuild] [-buildonly] [-norun] [-pa] <scenario> <simulator>
```

where <scenario> is one of: test entry under tests directory, sans the ts. prefix and file extension

<simulator> is one of: vcsmxvlog mtivlog vcsvlog vcszsimvlog vcscsvlog 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 viewer (VCS only)	[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 ncivlog

vcspcvlog vcsscivlog vcsvhdl ncmvlog

Valid scenarios are: test entry under tests directory, sans the ts. prefix and file extension



Note

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

2.7 Updating an Existing Model

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

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

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

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

```
%unix> dw_vip_setup -path design_dir -add mphy_agent_svt -v 3.50a model_vmt -v 3.50a
```

2.7.1 Removing VC VIP Models From a Design Directory

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

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

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

2.7.2 Reporting Information About DESIGNWARE_HOME or a Design Directory

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

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

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

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

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

2.7.3 The dw_vip_setup Utility

The dw_vip_setup utility,

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

2.7.4 Setting Environment Variables

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

- ❖ DESIGNWARE_HOME – Points to where the VIP is installed

2.7.5 The dw_vip_setup Command

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

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

or

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

where,

[-p[ath] *directory*] The optional -path argument specifies the path to your design directory. When omitted, dw_vip_setup uses the current working directory.

switch The *switch* argument defines dw_vip_setup operation. [Table 2-1](#) lists the switches and their applicable sub-switches.

Table 2-1 Setup Program Switch Descriptions



Switch	Description
<pre>-a[dd] (model [-v[ersion] version]) ...</pre>	<p>Adds the specified model or models to the specified design directory or current working directory. If you do not specify a version, the latest version is assumed. The model name is <code>mphy_agent_svt</code>.</p> <p>The <code>-add</code> switch causes <code>dw_vip_setup</code> to build suite libraries from the same suite as the specified models, and to copy the other necessary files from <code>\$DESIGNWARE_HOME</code>.</p>
<pre>-r[emove] model</pre>	<p>Removes all versions of the specified model or models from the design. The <code>dw_vip_setup</code> program does not attempt to remove any include files used solely by the specified model or models. The model name is <code>mphy_agent_svt</code>.</p>
<pre>-u[pdate] (model [-v[ersion] version]) ...</pre>	<p>Updates to the specified model version for the specified model or models. The <code>dw_vip_setup</code> script updates to the latest models when you do not specify a version. The model name is <code>mphy_agent_svt</code>.</p> <p>The <code>-update</code> switch causes <code>dw_vip_setup</code> to build suite libraries from the same suite as the specified models, and to copy the other necessary files from <code>\$DESIGNWARE_HOME</code>.</p>
<pre>-e[xample] {scenario model/scenario} [-v[ersion] version]</pre>	<p>The <code>dw_vip_setup</code> 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.</p> <p>If you specify a <i>scenario</i> (or system) example testbench, the models needed for the testbench are included automatically and do not need to be specified in the command.</p> <p> Note Use the <code>-info</code> switch to list all the available system examples.</p>
<pre>-ntb</pre>	<p>Not supported.</p>
<pre>-svtb</pre>	<p>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.</p>
<pre>-c[lean] {scenario model/scenario}</pre>	<p>Cleans the specified scenario or testbench in either the design directory (as specified by the <code>-path</code> 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.</p>

Table 2-1 Setup Program Switch Descriptions (Continued)

Switch	Description
<code>-i[nfo] design home [:<product>[:<version>[:<methodology>]]]</code>	<p>Generates an informational report on a design directory or VIP installation.</p> <p>design: If the <code>-info design</code> 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.</p> <p>home: If the <code>-info home</code> 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 <code><product></code>, <code><version></code> and <code><methodology></code> delimited by colons (:). An error will be reported if a nonexistent or invalid filter field is specified. Valid methodology names include:</p> <p>OVM, RVM, UVM, VMM and VLOG.</p>
<code>-h[elp]</code>	Returns a list of valid <code>dw_vip_setup</code> switches and the correct syntax for each.
<code>model</code>	<p>MPHY VIP model is <code>mphy_agent_svt</code>.</p> <p>The <code>model</code> argument defines the model or models that <code>dw_vip_setup</code> acts upon. This argument is not needed with the <code>-info</code> or <code>-help</code> switches. All switches that require the <code>model</code> argument may also use a model list.</p> <p>You may specify a version for each listed <i>model</i>, using the <code>-version</code> option. If omitted, <code>dw_vip_setup</code> uses the latest version. The <code>-update</code> switch ignores model version information.</p>
<code>-m[odel_list] filename</code>	<p>The <code>-model_list</code> argument causes <code>dw_vip_setup</code> to use a user-specified file to define the list of models that the program acts on. The <i>model_list</i>, like the <i>model</i> argument, can contain model version information. Each line in the file contains:</p> <p><i>model_name</i> [-v <i>version</i>] –or– # Comments</p>
<code>-b/ridge</code>	Updates the specified design directory to reference the current <code>DESIGNWARE_HOME</code> installation. All product versions contained in the design directory must also exist in the current <code>DESIGNWARE_HOME</code> installation.
<code>-pa</code>	<p>Enables the run scripts and Makefiles generated by <code>dw_vip_setup</code> 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.</p> <p>For run scripts, specify <code>-pa</code>.</p> <p>For Makefiles, specify <code>-pa = 1</code>.</p>
<code>-waves</code>	<p>Enables the run scripts and Makefiles generated by <code>dw_vip_setup</code> to support the <code>fsdb waves</code> option. To support this capability, the testbench example must generate an FSDb file when compiled with the WAVES Verilog macro set to <code>fsdb</code>, that is, <code>+define+WAVES=\"fsdb\"</code>. If a <code>.fsdb</code> file is generated by the example, the Verdi nWave viewer will be launched.</p> <p>For run scripts, specify <code>-waves fsdb</code>.</p> <p>For Makefiles, specify <code>WAVES=fsdb</code>.</p>

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 <code>DESIGNWARE_HOME</code> 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.
-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.
-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.
-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` program treats all lines beginning with # as comments.

2.8 Include and Import Model Files into Your Testbench

After you set up the models, you must include and import various files into your top testbench files to use the VIP. Following is a code list of the includes and imports for M-PHY:

```
/* include UVM package before VIP includes, if not included elsewhere*/
`include "svt_mphy.uvm.pkg"

/** Import UVM */
import uvm_pkg::*;
`include "uvm_macros.svh"

/** Import the SVT UVM Package */
```

```
import svt_uvm_pkg::*;

/** Import the MPHY COMPONENT Package */
import svt_mphy_component_uvm_pkg::*;

/** Import the MPHY Package */
import svt_mphy_uvm_pkg::*;
```

Include various VIP directories on the simulator command line.
Add the following switches and directories to all compile scripts:

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

Supported vendors are vcs, mti, and ncv. For example:

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

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

2.9 Compile and Run Time Options

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

```
$DESIGNWARE_HOME/vip/svt/mphy/latest/examples/sverilog/tb_mipi_mphy_svt_uvm_basic_sys/
```

The files containing the options are as follows:

- ❖ sim_build_options (also vcs_build_options)
- ❖ sim_run_options (also vcs_run_options)

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

vcs_build_options

```
Required: +define+UVM_PACKER_MAX_BYTES=1500000
Required: +define+UVM_DISABLE_AUTO_ITEM_RECORDING
Optional: -timescale=1ns/1ps
Required: +define+SVT_mphy_INCLUDE_USER_DEFINES
Required: +define+SYNOPSYS_SV
```



Note

UVM_PACKER_MAX_BYTES define needs to be set to maximum value as required by each VIP title in your testbench. For example, if VIP title 1 needs UVM_PACKER_MAX_BYTES to be set to 8192, and VIP title 2 needs UVM_PACKER_MAX_BYTES to be set to 500000, you need to set UVM_PACKER_MAX_BYTES to 500000.

vcs_run_options

```
Required: +UVM_TESTNAME=$scenario
```



Note

Scenario is the UVM test name you pass to VCS

3

General Concepts

This chapter describes the usage of M-PHY VIP in an UVM environment, and its user interface. This chapter discusses the following topics:

- ❖ [Introduction to UVM](#)
- ❖ [M-PHY VIP in an UVM Environment](#)
- ❖ [Covergroup Organization](#)

3.1 Introduction to UVM

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

For more specific information, you can refer the following:

- ❖ For the IEEE SystemVerilog standard, see:
 - ◆ IEEE Standard for SystemVerilog-Unified Hardware Design, Specification, and Verification Language
- ❖ For an essential reference guide describing UVM as it is represented in SystemVerilog, along with a class reference, see:
 - ◆ *Universal Verification Methodology (UVM) User's Guide* at:
<http://www.accellera.org/>
- ❖ For the description of attributes and properties of the objects mentioned in this chapter, see the Class Reference HTML.

3.2 M-PHY VIP in an UVM Environment

M-PHY VIP suite is a set of SVT based verification components, intended to be used in SystemVerilog UVM compliant testbenches. The VIP suite infrastructure builds upon SystemVerilog and adds to UVM. It is used to create a consistent and common look among VIP suites.

This common infrastructure is referred to as the Synopsys SVT (SystemVerilog Verification IP Technology) toolkit. The M-PHY VIP suite is built using this toolkit. This document describes the components and its usage information in the M-PHY VIP suite.

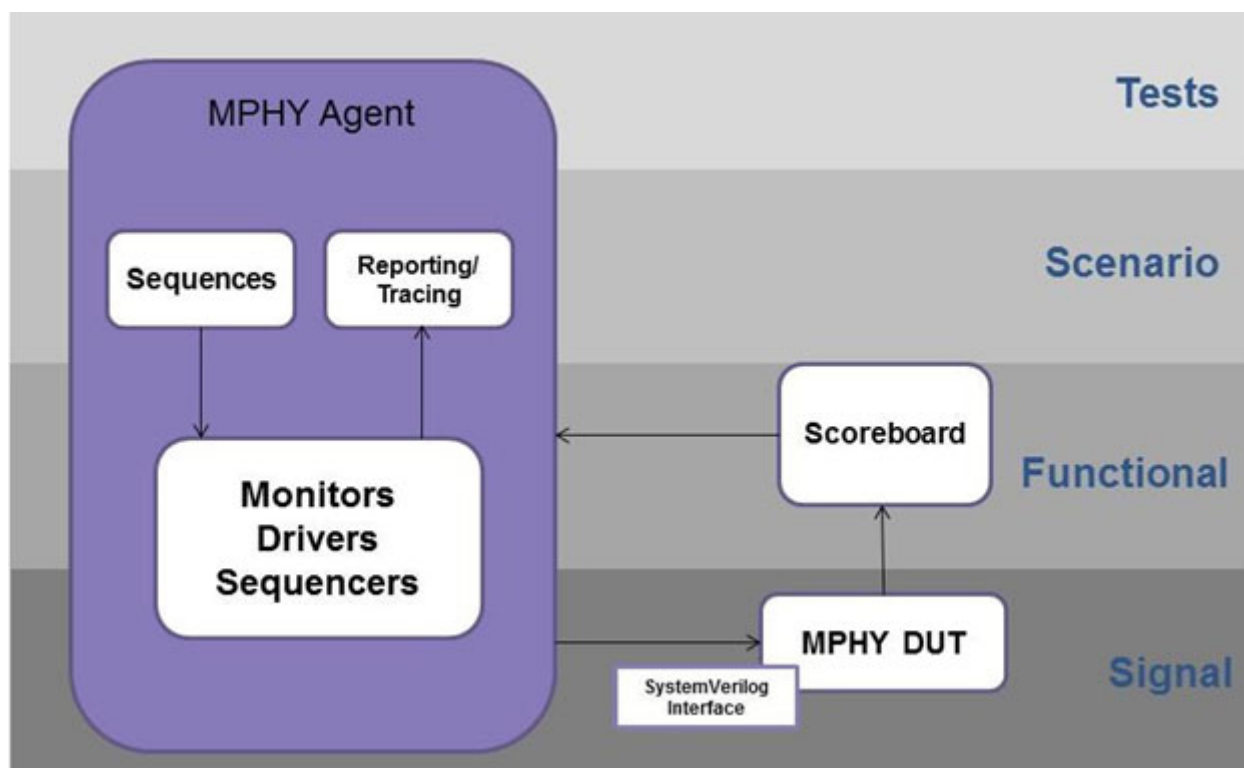
The UVM system-level verification environment is constructed with an UVM Agent which contains the following UVM components:

- ❖ M-PHY Sequencer
- ❖ M-PHY protocol driver
- ❖ M-PHY Monitor

M-PHY VIP contains the following three UVM agents:

- ❖ [svt_mphy_tx_agent](#) for TX Module
- ❖ [svt_mphy_rx_agent](#) for RX Module
- ❖ [svt_mphy_mport_lm_agent](#) for Lane Management Module

Figure 3-1 M-PHY VIP In Layered Testbench Architecture



3.2.1 Base Classes

In an object-oriented programming environment, a set of base classes form the foundation for the entire system. Base classes provide common functionality and structure. SystemVerilog base classes are specifically designed for the UVM approach to verification. The base classes provide common functionality and structure needed for simulation (such as reporting) and support any sort of verification function. The M-PHY VIP classes are extended from these base classes, providing an actual implementation and demonstrating that UVM is not simply a set of guidelines and recommendations. So, instead of writing your own reporting routine, you can reuse the UVM report class. Inheritance, extension, and polymorphism facilitate customization opportunities.

Important UVM base classes used by the M-PHY VIP include the following:

- ❖ `uvm_agent`
- ❖ `uvm_component`
- ❖ `uvm_object`
- ❖ `uvm_callback`
- ❖ `uvm_sequencer`

3.2.2 UVM Components

Objects derived from `uvm_component` are objects in an UVM compliant verification environment. The testbench and stack layers exchange transactions through `uvm_component` in three distinct ways:

- ❖ **Sequencer Ports**
 - ◆ Sequencers and drivers are connected through TLM ports. Sequencers place transactions that they create in input ports.
 - ◆ The sequencer includes an “implementation” of a pull port which it “exports” for connection, so that `sequence_item_pull_ports` can connect to it. The M-PHY components and drivers contain `sequence_item_pull_ports` which are connected to the sequencer provided exports.
- ❖ **“Callbacks”**
 - ◆ Before a component “gets” something from a data source, it does a “get” callback. On the opposite side, before a component “puts” something to a data link, it does a “put” callback. For example, a “get” operation can be any of the following:
 - ❖ Get data from a sequence item pull port
 - ❖ Get recognition of data
 - ❖ Get or peek from a lower level component
 - ◆ For example, a “put” operation can be any of the following:
 - ❖ Get or peek export to a higher level component
 - ❖ Sequence execution that results in sequence items going to lower level components
 - ◆ Callbacks are defined in a callback facade class (associated with each sequencer), and accessed by registering (with the associated component) an instance of a class extended from that facade class.
 - ◆ Each `uvm_component` supports additional callbacks to access the data at internal dataflow points.



Note For a complete callback list, see the M-PHY VIP Class Reference HTML documentation.

❖ UVM Events

- ◆ Events are based on the `uvm_event` class. Some `uvm_component` signal "significant events" through UVM events and include transactions as `uvm_data` objects with these events. Testbenches can be configured to wait for `uvm_event` instances, making a call to the `uvm_component`'s UVM event service instance. One can retrieve data by making a call to the `get_trigger_data()` method on the UVM event.

UVM components associate coverage (cov) callbacks with ports, in addition to "get" and "put" callbacks. These callbacks connect functional coverage to these ports. Coverage callbacks are called after the corresponding get and put methods, if none of the methods set "drop_it" to 1.

3.2.3 M-PHY VIP Objects

The M-PHY VIP defines several classes designed for UVM environment. This section introduces the major M-PHY VIP objects.

The M-PHY VIP classes extend base classes to handle specific needs of the protocol and provide predefined constraints. The predefined constraints can be used "as is" to produce a wide range of stimulus, or extended to create specific test conditions. For information about constraints, see "[Constraints](#)".

An object and its constraints are referred to as a factory object or factory, when used to control the production of, or randomization of a transaction data object. Sequencers which create sequences use factories to create streams of randomized objects. Sequencers are typically responsible for creating sequence items based on factories.

The remainder of this section describes the following M-PHY VIP objects:

- ❖ Configuration Objects
- ❖ Transaction Objects
- ❖ Callbacks
- ❖ Status Objects

3.2.3.1 Configuration Objects

The configuration data objects convey the agent level and port level testbench configuration. The configuration of agents is done in the `build()` phase of environment or the test case. If you want to change the required configuration later, use `reconfigure()` method of the M-PHY Agent.

The configuration can be of two types:

- ❖ **Static configuration**

Static configuration parameters specify a configuration value which cannot be changed when the system is running. For example, coverage enable, number of lanes, or any other capability attributes.

- ❖ **Dynamic configuration**

Dynamic configuration parameters specify configuration value which can be changed at any time, regardless of whether the system is running or not. For example, `tx_mode`, `tx_hsgear`, `tx_pwmgear`.

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

The M-PHY VIP has configurable agent level configuration class object. After configuring the agent level configuration, the user can pass the object to the constructor of the M-PHY agent for configurations to take effect.

M-PHY Agent Configuration (svt_mphy_agent_configuration)

The M-PHY Agent Configuration class is extended from `svt_mphy_configuration` class, as it inherits the parent class. It has access to lower level configuration classes to configure the components in the M-PHY Agent.

This configuration class mainly contains the set of configuration which defines the overall structure of the agent. For example,

- ❖ Monitor enable
- ❖ Layer trace or coverage
- ❖ Active or Passive Component

For details on the attributes of the M-PHY Agent Configuration and their default values, see the M-PHY VIP Class Reference HTML documentation.

M-PHY Configuration (svt_mphy_configuration)

The M-PHY Configuration class contains fields to configure M-PHY protocol features and verification features.

This class contains the parameters that mainly define the flow of the model. For example,

- ❖ Capabilities defined in Table 50 and 54 of *MIPI Alliance Specification for M-PHY Version 4.1- 01 December 2016*.
- ❖ Timer values
- ❖ Direction (M-TX/M-RX)
- ❖ Interfaces

M-PHY requires the modules to start in DISABLED state with Reset values of all configuration attributes identified in the specification. It requires Control SAP transactions for runtime updates of these attributes. To shorten the simulation cycles, M-PHY provides initialization configuration attributes, for example, `init_state`, `init_mode` and `init_hsgear`. These attributes allow M-PHY VIP to come up pre-configured to start, for example, in STALL (state), HS_Mode (mode), and HS_G1 (high speed gear).

Similarly, there are reset configuration attributes which allow the M-PHY VIP to have different values of configuration attributes than those defined in the specification. Particularly, in HS mode, sync range is set to 'COARSE' after reset, which can result in long SYNC patterns. Value of this attribute can be set to 'FINE' to allow shorter values to be set.

For details on the attributes of the M-PHY Configuration, and their default values, see the M-PHY VIP Class Reference HTML documentation.

**Note**

Configuration data objects are extended from the `svt_configuration` class, which is extended from the `uvm_sequence_item` base class. These objects implement all of the methods specified for the `uvm_sequence_item` class. For more information, see the M-PHY VIP Class Reference HTML documentation.

M-PHY Lane Management Agent Configuration (`svt_mphy_mport_lm_configuration`)

The M-PHY Lane Management Configuration class is extended from `svt_mphy_agent_configuration` class, as it inherits the parent class. It has access to lower level configuration classes to configure the components in the Lane Management.

This configuration class mainly contains the set of configuration which defines the overall structure of the agent. For example,

- ❖ Active Lanes
- ❖ Frame Demarcator
- ❖ Active or Passive Component

For details on the attributes of the M-PHY Lane Management Configuration and their default values, see the M-PHY Class Reference HTML documentation.

3.2.3.2 Transaction Objects

The transaction objects are extended from `svt_sequence_item` base class. These objects define a unit of M-PHY protocol information that is passed across the M-PHY lane. The attributes of transaction objects are public and are accessed directly for setting and getting values. Most of the transaction attributes can be randomized. The transaction object represents the desired activity to be simulated on the M-PHY Lane, or the actual activity that was monitored.

M-PHY transaction data objects are used to,

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

The class properties are public and accessed directly to set and read values. The transaction data objects support randomization and provide the following built-in constraints:

- ❖ `valid_ranges` constraints

These constraints limit the generated values to those acceptable to the drivers. They ensure basic VIP operation and should never be disabled. Following is an example for `valid_ranges` constraints:

```
constraint valid_ranges {
    solve primitive_layer before primitive_name;
    solve primitive_name before primitive_type;

    (primitive_layer == M_LANE) -> primitive_name inside {
        SYMBOL,
        PREPARE,
        SYNC,
        BURSTEND,
        SAVESTATE
    };
}
```

- ❖ `reasonable_*` constraints

These constraints can be disabled individually or as a block. Following is an example for `reasonable_*` constraints:

```
constraint reasonable_sync_length
{
    sync_length inside {[1:31]};
}
```

These constraints limit the simulation by,

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

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

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

M-PHY VIP defines the following transaction classes:

- ❖ M-PHY Transaction (`svt_mphy_transaction`)

The `svt_mphy_transaction` class extends from the `svt_sequence_item` class. The transaction class contains all the Data and Control SAP attributes, for example, `primitive_layer`, `data_n_ctrl`, `mib_attribute`, and so on, as defined in *Section 8.2 of MIPI Alliance Specification for M-PHY Version 4.1-01 December 2016*.

Data SAP and Control SAP transactions are distinguished by the `primitive_layer` attribute with the value of `M_LANE` and `M_CTRL` respectively.

**Note**

For details on the attributes of the M-PHY Configuration and their default values, see the M-PHY VIP Class Reference HTML documentation

3.2.3.3 Callbacks

The M-PHY Driver supports multiple callbacks for controlling randomization, viewing process flow data points, and covering data and activities. Each M-PHY Driver layer is associated with a callback class that contains a set of callback methods.

M-PHY VIP provides the following callback classes:

- ❖ `svt_mphy_txrx_callback`

The `svt_mphy_txrx_callback` class contains the callback methods called by the M-TX and M-RX modules. This callback operates on the Data and Control SAP objects. Following set of methods are incorporated as part of this callback class.

- ◆ Data transaction callbacks
 - ◇ `post_data_tx_trans_in_port_get`

Callback issued by the component after pulling a data TX transaction descriptor out of its transaction input, but before acting on the transaction in any way.

✧ `pre_data_tx_trans_out_port_put`

Callback issued by the component after recognizing a data TX transaction, just prior to placing the transaction.

✧ `post_data_rx_trans_in_port_get`

Callback issued by the component after pulling a data RX transaction descriptor out of its transaction input, but before acting on the transaction in any way.

✧ `pre_data_rx_trans_out_port_put`

Callback issued by the component after recognizing a data RX transaction, just prior to placing the transaction.

✧ `data_tx_trans_ended`

Callback issued by the component when a data TX transaction has just been ended.

✧ `data_tx_trans_started`

Callback issued by the component when a data TX transaction has just been started.

✧ `data_rx_trans_ended`

Callback issued by the component when a data RX transaction has just been ended.

✧ `data_rx_trans_started`

Callback issued by the component when a data RX transaction has just been started.

◆ Control transaction callbacks

✧ `post_control_tx_trans_in_port_get`

Callback issued by the component after pulling a control TX transaction descriptor out of its transaction input, but before acting on the transaction in any way.

✧ `pre_control_tx_trans_out_port_put`

Callback issued by the component after recognizing a control TX transaction, just prior to placing the transaction.

✧ `post_control_rx_trans_in_port_get`

Callback issued by the component after pulling a control RX transaction descriptor out of its transaction input, but before acting on the transaction in any way.

✧ `pre_control_rx_trans_out_port_put`

Callback issued by the component after recognizing a control RX transaction, just prior to placing the transaction.

✧ `control_tx_trans_ended`

Callback issued by the component when a control TX transaction has just been ended.

✧ `control_tx_trans_started`

Callback issued by the component when a control TX transaction has just been started.

✧ `control_rx_trans_ended`

Callback issued by the component when a control RX transaction has just been ended.

✧ `control_rx_trans_started`

Callback issued by the component when a control RX transaction has just been started.

3.2.3.4 Status Object

Status object represents the current state, as defined by the transactor implementing and updating the status information. It represents the runtime state of configuration attributes as identified in Table 51,52,55 and 56 *MIPI Alliance Specification for M-PHY Version 4.1- 01 December 2016*. Status object represents the configuration space of the M-TX or M-RX module and also implements a shadow memory and INLINE-CR. Its data members represent OFFLINE or INLINE set depending on its FSM state.

The notifies are used to indicate a change in status, as defined by the transactor responsible for indicating the notification.

Attention

Handle to status object is available to the user, but it should only be used as a read-only object. Its data members are public, and hence make sure that these are not updated by the testbench code, lest it interferes with VIP behavior and causes incorrect behavior.



For more information, see the M-PHY VIP Class Reference HTML documentation.

3.2.4 Interfaces and Modports

SystemVerilog models the signal connections using interfaces and modports. The interfaces define a set of signals which build a port connection. The modports define a set 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 M-PHY Agent provides SystemVerilog interface which can be used to connect the VIP to a DUT. The top level interface is `svt_mphy_if`, which contains M-TX & M-RX interfaces to connect to M-RX and M-TX of DUT interfaces respectively.

For more information, refer to Class Reference HTML documentation.

3.2.5 Constraints

M-PHY VIP uses objects with constraints for transactions and configurations. The tests in an UVM flow are primarily defined by constraints. Constraints define the range of randomized values that are used to create each object during the simulation.

Classes that provide random attributes allow you to constrain the contents of the resulting object. When you call the `randomize()` method, which is a built-in method, all random attributes are randomized using all the constraints that are enabled.

Constraint randomization is sometimes misunderstood and seen as a process whereby the simulation engine takes the control of class members away from the user. In fact, the opposite is true. Randomization is an additional way for the user to assign class members and there are several ways to control the process.

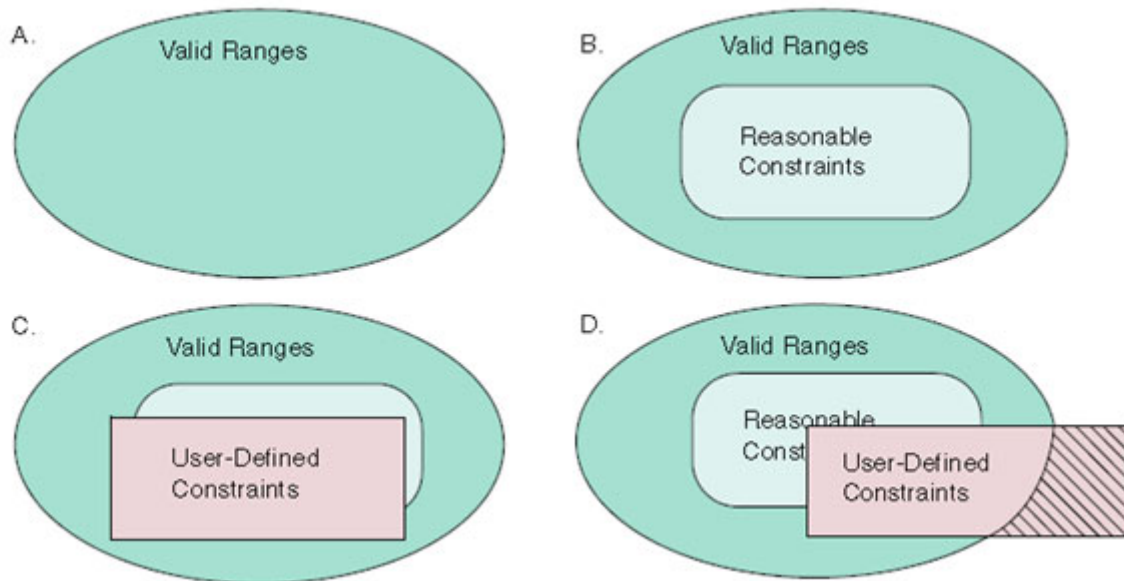
When working with randomization, apply the following basics or techniques:

- ❖ Randomization only occurs when the `randomize()` method in an object is called. It completely depends upon the test code, when, or even if, this occurs.
- ❖ Constraints form a rule set to follow when randomization is performed. By controlling the constraints, the testbench has influence over the outcome. Direct control can be exerted by constraining a member to a single value. Constraints can also be enabled or disabled.

- ❖ Each rand member has a rand mode that can be turned ON or OFF, giving individual control of what is randomized.
- ❖ A user can assign a member to a value at any time. Randomization does not affect the other methods of assigning class members.

Figure 3-2 shows the scope of constraints that are part of the M-PHY VIP.

Figure 3-2 Constraints: Valid Ranges, Reasonable, and User-Defined



3.2.5.1 Valid Range Constraints

Valid Range constraints are provided with M-PHY VIP. These constraints,

- ❖ Keep values within a range, that the components can handle
- ❖ Are not tied to protocol limits
- ❖ Are On by default, and should not be turned off or modified

3.2.5.2 Reasonable constraints

Reasonable constraints are provided with M-PHY VIP. These constraints,

- ❖ Keep values within protocol limits (typically) to generate worthwhile traffic
- ❖ Keep simulations to a reasonable length and size, in some cases
- ❖ Are defined to be “reasonable” by Synopsys (user can override)
- ❖ May result in conditions that are a subset of the protocol
- ❖ Are “On” by default and can be turned “Off” or modified (user should review these constraints)

3.2.5.3 User-Defined Constraints

User-Defined constraints provide you a way to define specific tests.

Constraints that lie outside the valid ranges are not included during randomization. All constraints that are enabled are included in the simulation. The constraint solver resolves any conflicts.

3.2.6 Functional Coverage

All functional coverage items are provided by the monitor callback class. M-PHY VIP coverage implementation contains the following types of Functional Coverage:

- ❖ Protocol layer coverage
- ❖ Physical layer coverage

[Table 3-1](#) lists the functional coverage features provided with M-PHY VIP.

Table 3-1 Coverage Features

Protocol Layer Coverage	M-PHY
Transaction Coverage	Control TX/RX Coverage Data TX/RX Coverage
FSM State Coverage	LS Mode HS Mode LS-HS RCT HS-LS RCT Valid State transitions
Physical Layer Coverage	M-PHY
Toggle Coverage	RMMI Interface <ul style="list-style-type: none"> ❖ RMMI Controller (TX/RX) ❖ RMMI PHY (TX/RX)

The functional coverage data collected is based on the instances. This means, that each instance of VIP gathers and maintains its own functional coverage information.

For detailed information about covergroups in M-PHY, see M-PHY VIP Class Reference HTML.

3.3 Covergroup Organization

Covergroups are organized as the collection of coverpoints and the cross of these coverpoints. M-PHY VIP contains the following covergroups:

- ❖ Protocol Layer Coverage
 - ◆ Transaction Coverage
 - ❖ svt_mphy_mtx_ctrl_capability_attr_rd_wr
 - ❖ svt_mphy_mrx_ctrl_capability_attr_rd_wr
 - ❖ svt_mphy_mtx_data_transaction
 - ❖ svt_mphy_mrx_data_transaction
 - ❖ svt_mphy_mtx_mkr_filler_transaction
 - ❖ svt_mphy_mrx_mkr_filler_transaction
 - ❖ svt_mphy_mtx_ls_ctrl_transaction
 - ❖ svt_mphy_mrx_ls_ctrl_transaction
 - ❖ svt_mphy_mtx_hs_ctrl_transaction
 - ❖ svt_mphy_mrx_hs_ctrl_transaction

- ✧ svt_mphy_mtx_sync_data_transaction
- ✧ svt_mphy_mrx_data_error
- ✧ svt_mphy_mtx_linereset_reset_transaction
- ✧ svt_mphy_mrx_linereset_reset_transaction
- ✧ For M-PHY_2.0_Version:
 - svt_mphy_mtx_ctrl_mphy_20_line_reset
 - svt_mphy_mrx_ctrl_mphy_20_line_reset
- ◆ FSM State coverage
 - ✧ svt_mphy_serial_ls_state_machine
 - ✧ svt_mphy_serial_hs_state_machine
 - ✧ svt_mphy_serial_ls_to_hs_state_machine
 - ✧ svt_mphy_serial_hs_to_ls_state_machine
 - ✧ svt_mphy_serial_state_machine_transition
- ❖ Physical layer coverage
 - ◆ Toggle Coverage
 - ✧ RMMI Signalling

Example 3-1 Typical Example of a Covergroup Definition

```
symbol_req_data_value : coverpoint mphy_transaction.data_value
                        iff(mphy_transaction.data_n_ctrl == 0)
{
  option.weight = 0;
  option.comment = "Coverage of MPHY symbol_req_data_value";
  bins data_value_small_range = {[0:50]};
  bins data_value_mid_range   = {[51:200]};
  bins data_value_large_range = {[201:255]};
}
```

The above definition can be referred to define range bins. The range bins are defined by dividing the total range of possible values into a few buckets. In [Examples 3-1](#), the buckets are divided as low, medium and high ranges.

Example 3-2 Typical Example of a Cross Between Two Coverpoints

```
tx_data_trans : cross mtz_direction,
                    mtz_primitive_layer,
                    mtz_primitive_type,
                    mtz_primitive_name,
                    symbol_req_data_n_ctrl,
                    symbol_req_data_value
{
  option.weight = 1;
  option.comment = "cross coverage of mlane_tx_data_trans";
  ignore_bins burst_end   = binsof(mtz_primitive_name.burstend);
  ignore_bins indication = binsof(mtz_primitive_type.indication);
  ignore_bins savestate = binsof(mtz_primitive_name.savestate);
  ignore_bins prepare = binsof(mtz_primitive_name.prepare);
}
```

Examples 3-2 shows the cross of coverpoints defining a data transaction along with the required attributes.

3.3.1 Enable or Disable Functional Coverage

The default functional coverage can be enabled by setting the attributes in the M-PHY agent configuration class.

```
//To enable transaction coverage  
svt_mphy_agent_configuration:: enable_transaction_cov=1
```

```
//To enable toggle coverage for RMMI interface  
svt_mphy_agent_configuration:: enable_toggle_cov=1
```

```
//To enable protocol check pass coverage  
svt_mphy_agent_configuration:: enable_pass_cov=1
```

```
//To enable protocol check fail coverage  
svt_mphy_agent_configuration:: enable_fail_cov=1
```

Figure 3-3 Coverage Plan

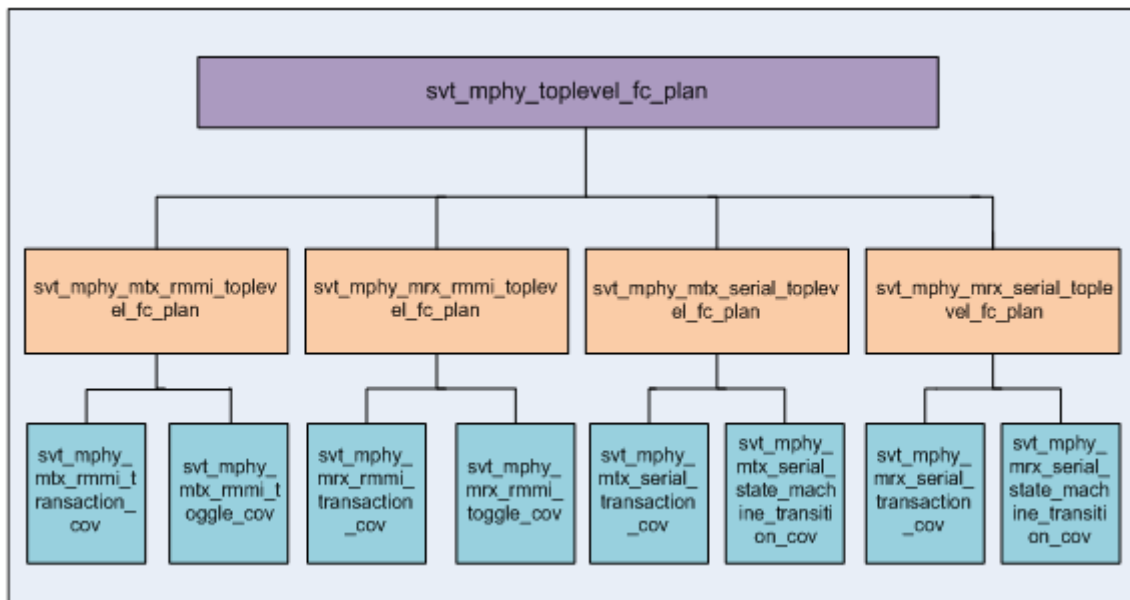


Figure 3-4 User Control to Disable Coverage Plan

1	2	3	4	5	6	7
top plan	include	value	reference	feature	subfeature	subfeature
svt_mphy_toplevel_fc_plan		mphy_svt_group				
1	svt_mphy_mrx_rmmi_toplevel_fc_plan.xml					
2	svt_mphy_mrx_rmmi_toplevel_fc_plan.xml					
3	svt_mphy_mrx_rmmi_toplevel_fc_plan.xml					
4	svt_mphy_mrx_rmmi_toplevel_fc_plan.xml					
5	svt_mphy_mrx_rmmi_toplevel_fc_plan.xml					
6						
7			Chapter 4.8			
8	skip			RMMI		
9					M_TX	subplan svt_mphy_mrx_rmmi_toplevel_fc_plan @INST_GROUP_COV_attr=="INST_NAME_attr="mrx"
10					M_RX	subplan svt_mphy_mrx_rmmi_toplevel_fc_plan @INST_GROUP_COV_attr=="INST_NAME_attr="mrx"
11			Chapter 8			
12				Serial		
13					M_TX	subplan svt_mphy_mrx_serial_toplevel_fc_plan @INST_GROUP_COV_attr=="instance">@INST_NAME
14					M_RX	subplan svt_mphy_mrx_serial_toplevel_fc_plan @INST_GROUP_COV_attr=="instance">@INST_NAME
15						
16						

The user has a control to disable the coverage plan. For example, if RMMI coverage is not required, then user can add "skip" as mentioned in Figure 3-4. Similarly, if the user wants to skip M_RX in RMMI coverage, "skip" needs to be added inline with M_RX.

3.3.1.1 Coverage Generation for TX or RX Sublink and Loopback Environment

The following two variables are added in svt_mphy_mport_lm_agent_configuration class to enable or disable TX or RX sublink coverage:

- ❖ enable_tx_lm_cov
It is used to enable or disable Master or Slave TX path coverage.
- ❖ enable_rx_lm_cov
It is used to enable or disable Master or Slave RX path coverage.

To enable TX sublink coverage with SERIAL or RMMI interface (back-to-back topology):

```
svt_mphy_mport_lm_agent_configuration::master_cfg.enable_tx_lm_cov = 1;
svt_mphy_mport_lm_agent_configuration::master_cfg.enable_rx_lm_cov = 0;
svt_mphy_mport_lm_agent_configuration::slave_cfg.enable_tx_lm_cov = 0;
svt_mphy_mport_lm_agent_configuration::slave_cfg.enable_rx_lm_cov = 1;
```

To enable RX sublink coverage with SERIAL or RMMI interface (back-to-back topology):

```
svt_mphy_mport_lm_agent_configuration::master_cfg.enable_tx_lm_cov = 0;
svt_mphy_mport_lm_agent_configuration::master_cfg.enable_rx_lm_cov = 1;
svt_mphy_mport_lm_agent_configuration::slave_cfg.enable_tx_lm_cov = 1;
svt_mphy_mport_lm_agent_configuration::slave_cfg.enable_rx_lm_cov = 0;
```

To enable coverage for Loopback environment:

```
svt_mphy_mport_lm_agent_configuration::master_cfg.enable_tx_lm_cov = 1;
svt_mphy_mport_lm_agent_configuration::master_cfg.enable_rx_lm_cov = 1;
svt_mphy_mport_lm_agent_configuration::slave_cfg.enable_tx_lm_cov = 0;
svt_mphy_mport_lm_agent_configuration::slave_cfg.enable_rx_lm_cov = 0;
```




4

M-PHY VIP Agent Overview

M-PHY is a suite of advanced verification components and data objects based on SystemVerilog UVM compliant technology. This chapter describes the UVM components supported by M-PHY Verification IP. Refer to the Class Reference HTML for a description of objects, classes, and attributes mentioned in this chapter.

This chapter discusses the following two agent based use models for M-PHY:

- ❖ [M-PHY TX or RX Agent UVM Component Stack](#)
- ❖ [M-PORT Lane Management Agent UVM Component](#)
- ❖ [“M-PHY Model Agent UVM Component”](#)

4.1 M-PHY TX or RX Agent UVM Component Stack

UVM system-level verification environments are constructed with an UVM Agent which contains three UVM Components:

- ❖ Driver
- ❖ Monitor
- ❖ Sequencer

The M-PHY VIP has two UVM agents:

- ❖ M-PHY TX Agent
- ❖ M-PHY RX Agent

Each of these agents has Driver, Monitor & Sequencer components. The M-PHY Agents are configured using an object of `svt_mphy_agent_configuration`.

Based on the signal level interface, the following agent configurations are available:

- ❖ M-TX Serial
 - ◆ M-PHY M-TX Driver (`svt_mphy_tx`)
 - ◆ M-PHY Monitor (`svt_mphy_monitor`)
 - ◆ M-PHY Data Sequencer (`svt_mphy_transaction_sequencer`)
 - ◆ M-PHY Control Sequencer (`svt_mphy_transaction_sequencer`)
 - ◆ M-TX Serial Interface

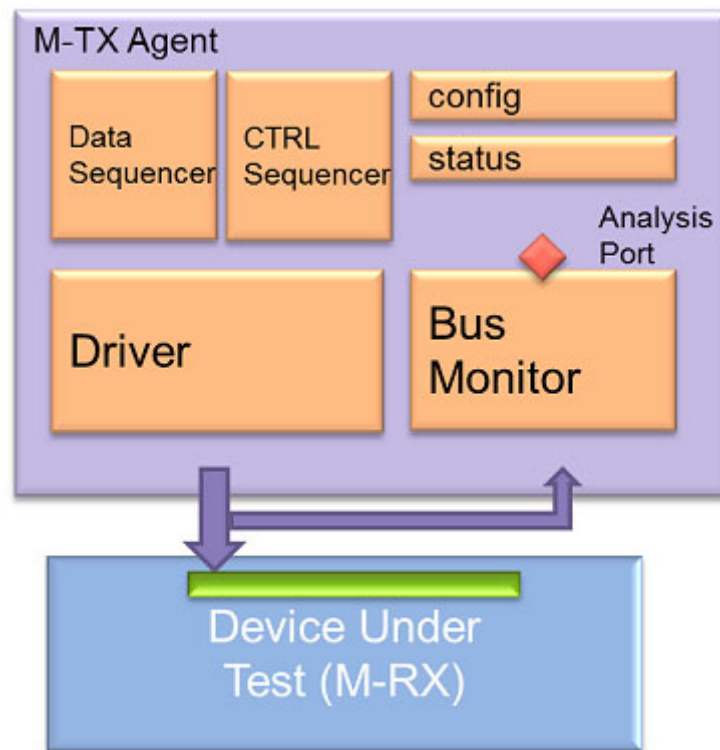
- ❖ M-RX Serial
 - ◆ M-PHY M-RX Receiver (svt_mphy_rx)
 - ◆ M-PHY Monitor (svt_mphy_monitor)
 - ◆ M-PHY Control Sequencer (svt_mphy_transaction_sequencer)
 - ◆ M-RX Serial Interface
- ❖ M-TX RMMI Local
 - ◆ M-PHY M-TX Driver (svt_mphy_tx)
 - ◆ M-PHY Monitor (svt_mphy_monitor)
 - ◆ M-PHY Data Sequencer (svt_mphy_transaction_sequencer)
 - ◆ M-PHY Control Sequencer (svt_mphy_transaction_sequencer)
 - ◆ M-TX DUT PHY Interface
- ❖ M-TX RMMI Remote
 - ◆ M-PHY M-TX Driver (svt_mphy_tx)
 - ◆ M-PHY M-RX Receiver (svt_mphy_rx)
 - ◆ M-PHY Monitor (svt_mphy_monitor)
 - ◆ M-PHY Data Sequencer (svt_mphy_transaction_sequencer)
 - ◆ M-PHY Control Sequencer (svt_mphy_transaction_sequencer)
 - ◆ M-RX DUT Controller Interface
- ❖ M-RX RMMI Local
 - ◆ M-PHY M-RX Receiver (svt_mphy_rx)
 - ◆ M-PHY Monitor (svt_mphy_monitor)
 - ◆ M-PHY Control Sequencer (svt_mphy_transaction_sequencer)
 - ◆ M-RX DUT PHY Interface
- ❖ M-RX RMMI Remote
 - ◆ M-PHY M-RX Receiver (Local) (svt_mphy_tx)
 - ◆ M-PHY M-TX Driver (Remote) (svt_mphy_rx)
 - ◆ M-PHY Monitor (svt_mphy_monitor)
 - ◆ M-PHY Control Sequencer (svt_mphy_transaction_sequencer)
 - ◆ M-TX DUT Controller Interface

4.1.1 M-PHY TX Agent

M-PHY TX Agent has two independent sequencers:

- ❖ Data Sequencer for Data SAP Transactions
- ❖ CTRL Sequencer for Control SAP Transactions

The monitor provides analysis ports for Data SAP and Control SAP which can be connected to a monitor of [Lane Management layer](#) or a Scoreboard. [Figure 4-1](#) shows the M-PHY TX Agent.

Figure 4-1 M-PHY TX Agent

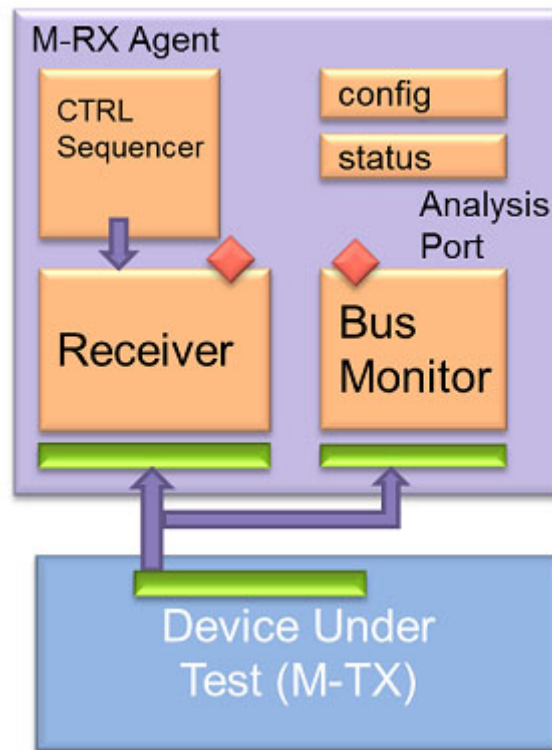
The Driver and Monitor components within the agent, call the callback methods at various phases of execution of the M-PHY transaction. The details of callbacks are covered in the later sections. After the M-PHY transaction is completed on the stack, the sequence item is provided to the analysis port for use by the testbench.

4.1.2 M-PHY RX Agent

M-PHY RX Agent only has a CTRL sequencer for Control SAP.

The analysis ports of the monitor provide Data SAP and Control SAP objects. These analysis ports can be connected to a monitor of Lane Management layer or a Score-board.

[Figure 4-2](#) shows the M-PHY RX Agent.

Figure 4-2 M-PHY RX Agent

4.1.3 M-PHY Agent Modes

The M-PHY Agent can be configured in three different modes by setting "is_active" & "enable_monitor" members of the M-PHY agent configuration.

- ❖ Active Mode

- ◆ With Monitor

This mode is set when "is_active" and "enable_monitor" variables are set to value of "1". In this mode, the driver, monitor and sequencers of respective layers are created and sequencers are connected to the relevant driver.

- ◆ Without Monitor

This mode is set when "is_active" is set to value of "1" and "enable_monitor" is set to value of "0". In this mode, the driver and sequencers of respective layers are created and sequencers are connected to the relevant driver.

- ❖ Passive Mode

This mode is set when "is_active" variable is set to value of "0" and "enable_monitor" is set to value of "1". In this mode, only the monitor is available as a part of the M-PHY agent.

4.1.3.1 Usage

A typical usage of a M-PHY Agent is as follows:

```
svt_mphy_tx_agent      mtx0;  
svt_config_int_db#(int)::set(this,  "mtx0", "is_active", 0);  
mtx0 = svt_mphy_tx_agent::type_id::create("mtx0",this);  
svt_mphy_rx_agent      mrx0;  
svt_config_int_db#(int)::set(this,  "mrx0", "is_active", 1);  
mrx0 = svt_mphy_rx_agent::type_id::create("mrx0",this);
```

M-PHY TX agent "mtx0" will be created only with Monitor.

M-PHY RX Agent will have Sequencer, Driver and Monitor.

4.1.4 M-PHY TX Agent Driver

In the M-PHY agent, the Driver is an UVM component object in an UVM compliant environment which,

- ❖ Processes and exchanges the transfers with the testbench
- ❖ Communicates with the remote M-PHY using M-PHY link

The Driver operates for transmitting, and processing M-PHY transactions. It implements the M-PHY protocol layer and can be configured to work as one of the following components:

- ❖ M-TX RMMI Controller
- ❖ M-TX RMMI PHY
- ❖ M-TX Serial

4.1.4.1 Driver Feature Support

The following is a list of supported protocol layer verification features:

- ❖ Port Support
 - ◆ Data Port
 - ◆ Control Port
- ❖ Configurable Stimulus to Input port
- ❖ Testbench visibility and control through callbacks
- ❖ Protocol Checkers
- ❖ Randomization Factories

4.1.4.2 Driver Ports

UVM Ports are the mechanism through which the M-PHY component connect to each other and/or to the testbench. The Protocol component contains various types of ports:

- ❖ Transaction Input Ports

These are used by the sequencer to send sequences into the protocol layer. You cannot connect to them. They are only used by the sequencer.
- ❖ Observed Ports

These ports are used for either generating inputs into response (output ports), or for creating scoreboards and coverage checks. You can also collect data from these ports.

The Protocol Layer component has the following UVM Port interfaces:

❖ Transaction Input Port

This is used to supply stimulus transfers for the M-PHY VIP, used only by the respective sequencers. User cannot connect to them. The following is the list of transaction input ports:

- ◆ ctrl_get_port
Control SAP Transaction to configure M-TX
- ◆ data_in_get_port
Data SAP Transaction to M-TX

4.1.4.3 Driver Data Objects

The following list of objects represents the information that the protocol layer component receives, sends, or processes:

- ❖ M-PHY Transaction (svt_mphy_transaction)
This class is used for both Data and Control SAP transactions with primitive_layer = M_LANE for Data SAP and primitive_layer = M_CTRL for Control SAP.

4.1.4.4 Driver Callbacks

The driver supports multiple callbacks for controlling randomization, viewing the process flow data points, and covering data and activities. Each M-PHY Agent Component is associated with a callback class that contains a set of callback methods. M-PHY VIP provides the following callback classes:

- ❖ svt_mphy_txrx_callback

4.1.5 M-PHY RX Agent Receiver

In the M-PHY RX Agent, the Receiver is an UVM component object in an UVM compliant environment which,

- ❖ Processes and exchanges the transfer with the testbench
- ❖ Communicates with the remote M-PHY using M-PHY Link

The Receiver operates for receiving and processing M-PHY transactions. It implements the M-PHY protocol layer and can be configured to work as one of the following components:

- ❖ M-RX RMMI Controller
- ❖ M-RX RMMI PHY
- ❖ M-RX Serial PHY

4.1.5.1 Receiver Feature Support

The following is a list of supported protocol layer verification features:

- ❖ Port Support
 - ◆ Data Port
 - ◆ Control Port
- ❖ Configurable Stimulus to Input port
- ❖ Testbench visibility and control through callbacks
- ❖ Protocol Checkers
- ❖ Randomization Factories

4.1.5.2 Receiver Ports

UVM Ports are the mechanism through which the M-PHY components connect to each other and/or to the testbench. The Protocol component contains various types of ports:

- ❖ Transaction Input Ports

These are used by the sequencer to send sequences into the protocol layer. You cannot connect to them. They are only used by the sequencer.

- ❖ Observed Ports

These ports are used for either generating inputs into response (output ports), or for creating scoreboards and coverage checks. You can also collect data from these ports.

The Protocol Layer component has the following UVM Port interfaces:

- ❖ Transaction Input Port

This is used to supply stimulus transfers for the M-PHY VIP, used only by the respective sequencers. User cannot connect to them. The following is the list of transaction input ports:

- ◆ ctrl_in_get_port

Control SAP Transaction to configure M-RX

- ◆ ctrl_out_put_port

Control SAP Transaction to indicate LINE-RESET and RESET

- ◆ data_put_port

Data SAP Transaction from M-RX

4.1.5.3 Receiver Data Objects

The following list of objects represents the information that the protocol layer component receives, sends, or processes:

- ❖ M-PHY Transaction (svt_mphy_transaction)

This class is used for both Data and Control SAP transactions with `primitive_layer = M_LANE` for Data SAP and `primitive_layer = M_CTRL` for Control SAP.

4.1.5.4 Receiver Callbacks

The receiver supports multiple callbacks for controlling randomization, viewing the process flow data points, and covering data and activities. Each M-PHY Agent Component is associated with a callback class that contains a set of callback methods. M-PHY VIP provides the following callback classes:

- ❖ svt_mphy_txrx_callback

4.1.6 M-PHY Agent Monitor

In the M-PHY agent, the Monitor is an UVM component object in an UVM compliant environment that monitors the signals on the interface. The Monitor checks the transaction transmitted from M-PHY TX and RX, and vice-versa. It sends the transaction packets to the scoreboard for comparison. The transaction error checks are also performed by the monitor.

In the TX Agent, the Monitor receives M-PHY traffic transmitted on the lane from local TX to remote RX.

In the RX Agent, the Monitor receives M-PHY traffic transmitted on the lane from remote TX to local RX.

4.1.6.1 Monitor Feature Support

The following is a list of supported verification features:

- ❖ Transaction Port Support
- ❖ Testbench visibility and control through callbacks
- ❖ Protocol Checkers

4.1.6.2 Monitor Ports

The Protocol Layer component contains the UVM Port interfaces. The M-PHY Monitor has analysis ports that are used to get the transmitted M-PHY Transaction out of the monitor for use by the testbench.

The following is the list of ports:

- ❖ `data_out_put_port`: For Data SAP
- ❖ `ctrl_out_put_port`: For Control SAP

4.1.6.3 Monitor Data Objects

The following list of objects represents the information for Protocol layer component associated with Monitor analysis ports:

- ❖ M-PHY Transaction (`svt_mphy_transaction`)

For more details, see Class reference HTML documentation.

4.1.6.4 Monitor Callbacks

The Monitor supports multiple callbacks for viewing process flow data points, and covering data and activities. The Monitor is associated with a callback class in correspondence to each of the M-PHY Driver layer components that contains a set of callback methods.

Monitor provides the following callbacks classes:

- ❖ `svt_mphy_monitor_callback`

For more details, see Class reference HTML documentation.

4.1.6.5 Monitor Protocol Checkers

The test environment can selectively enable and disable check levels, as defined and supported by the components. The M-PHY VIP provides the following checks in the M-PHY Agent Monitor

- ❖ `svt_mphy_err_check`: M-PHY Serial and RMMI common (TX & RX) checks
- ❖ `svt_mphy_tx_err_check`: M-PHY Serial TX and RMMI TX checks
- ❖ `svt_mphy_rx_err_check`: M-PHY Serial RX and RMMI RX checks

where SERIAL TX represents checks related to SERIAL driver

- ❖ SERIAL RX represents checks related to SERIAL receiver
- ❖ RMMI TX represents checks related to RMMI M-TX signaling interface
- ❖ RMMI RX represents checks related to RMMI M-RX signaling interface

By default, the checkers are enabled.

For more details, see [Class reference HTML documentation](#).

4.1.7 M-PHY Agent Sequencers

In UVM, all the stimuli come through sequences and sequencers. The M-PHY Agent has sequencers for each layer of the M-PHY stack, which in turn has different sequencers for individual traffic classes to generate respective M-PHY transactions.

Below are the sequencers associated with the M-PHY Agent:

- ❖ Data Sequencer
`svt_mphy_transaction_sequencer`
- ❖ Control Sequencer
`svt_mphy_transaction_sequencer`

For a list of checks, refer to the [Class Reference HTML](#).

4.2 M-PORT Lane Management Agent UVM Component

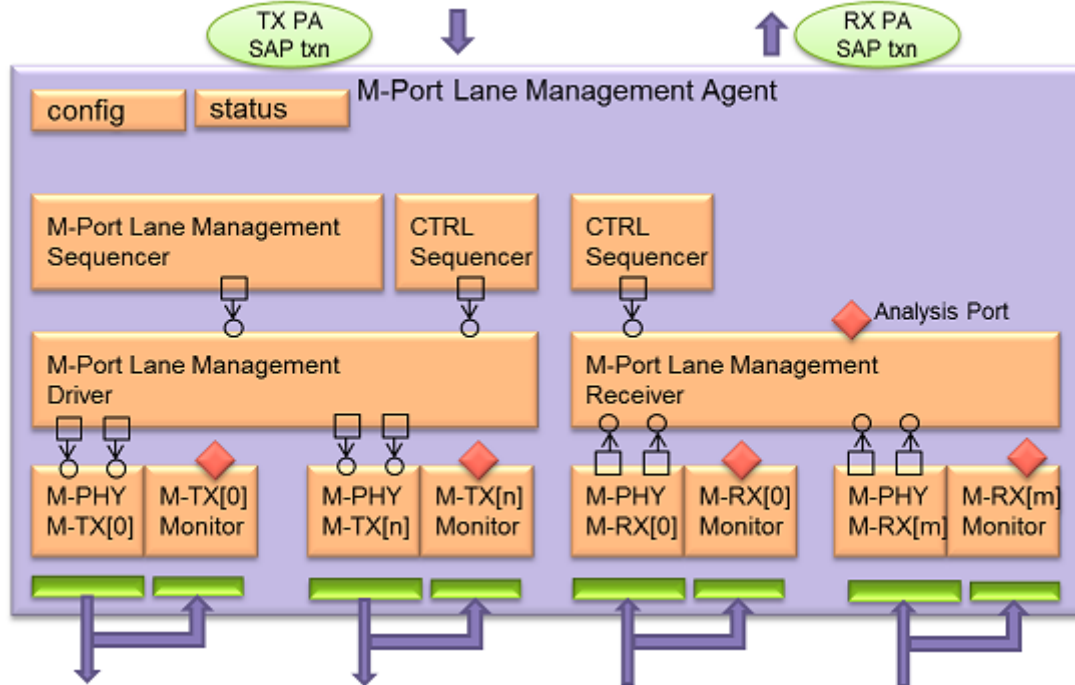
The M-PHY M-Port Lane Management Agent consists of Sequencers, Drivers and Monitor components.

M-Port Lane Management Agent is configured using an object of `svt_mphy_mport_lm_configuration`.

Following are the objects used to configure the M-Port Lane Management Agent:

- ❖ M-Port Lane Management Agent Driver
- ❖ M-Port Lane Management Agent Receiver
- ❖ M-TX Driver
- ❖ M-RX Receiver
- ❖ M-TX Monitor
- ❖ M-RX Monitor
- ❖ M-Port Lane Management Data Sequencer
- ❖ M-PHY Control Sequencer

[Figure 4-3](#) shows the block diagram for M-Port Lane Management Agent.

Figure 4-3 M-Port Lane Management Agent

4.2.1 M-PHY M-Port Lane Management Agent Driver

M-Port Lane Management Driver is an UVM component object in an UVM-compliant environment which,

- ❖ Processes and exchanges the transfers with the testbench
- ❖ Communicates with the M-PHY TX Drivers using ports

M-PHY M-Port consists of the following two sub-links, with each sub-link consisting of configurable number of M-PHY modules:

- ❖ TX Sub-link
- ❖ RX sub-link

M-Port Lane Management Driver communicates with the testbench to receive a payload, such as an LLI PHIT or UniPro PA-SAP or a DigRF TASLC message, splits it into configurable 'n' TX Lanes and drives them to the connected M-TX transactors.

The data payload is an object of type `svt_mphy_mport_lm_transaction` and consists of Data Symbols and optional MARKER and FILLER Symbols.

On receiving the Data SAP, the M-Port Lane Management Driver sends a PREPARE Symbol on active lanes, followed by SYNC as required, followed by a MARKER0 Symbol, and then sends Data Symbols along with the optional FILLER and MARKER symbols received as part of the Data SAP. If it is the END-of-BURST, it sends BURSTEND Symbol on all active lanes or else keeps sending FILLER symbols till it receives the next Data SAP. Using Data SAPs, multiple frames can be sent as the part of a BURST and multiple BURSTS can be sent with intervening SAVE states depending upon the MODE.

Control SAP transactions can be used to send M-PHY CTRL SAPs to get M-TX capability attributes and get and set M-TX configuration attributes.

4.2.1.1 Driver Feature Support

The following is a list of supported protocol layer verification features:

- ❖ Port Support
 - ◆ Data Port
 - ◆ Control Port
- ❖ Configurable Stimulus to Input port
- ❖ Testbench visibility and control through callbacks
- ❖ Randomization factories

4.2.1.2 Driver Ports

UVM Ports are the mechanism through which the M-PHY component connects to each other and/or to the testbench. The Protocol component contains various types of ports:

- ❖ Transaction Input Ports

These are used by the sequencer to send sequences into the protocol layer. You cannot connect to them. They are only used by the sequencer. For information on how to use sequence related classes, refer to UVM documentation.
- ❖ Observed Ports

Data is collected from these ports and these ports are also used for either generating inputs into response (output ports), or for creating scoreboards and coverage checks.

The M-Port Lane Management Layer component has the following UVM Port interfaces:

- ❖ Transaction Input Port

This is used to supply stimulus transfers for the M-PHY VIP, used only by respective sequencers. User cannot connect to them.

 - ◆ ctrl_in_get_port

Control SAP Transaction to configure M-TX and M-RX Drivers
 - ◆ data_in_get_port

Data SAP Transaction to M-Port Lane Management Driver

4.2.1.3 Driver Data Objects

The following list of objects represents information that the Protocol layer component receives, sends, or processes:

- ❖ M-Port Lane Management Data Transaction (svt_mphy_mport_lm_transaction)
- ❖ M-PHY Control SAP Transaction (svt_mphy_transaction)

4.2.2 M-PHY M-Port Lane Management Agent Receiver

M-Port Lane Management Receiver is an UVM component object in an UVM-compliant environment which,

- ❖ Processes and exchanges the transfers with the testbench
- ❖ Communicates with the M-PHY RX Receivers using ports

M-PHY M-Port consists of the following two sub-links, with each sub-link consisting of configurable number of M-PHY modules:

- ❖ TX Sub-link
- ❖ RX Sub-link

M-Port Lane Management Receiver communicates with the testbench to send a payload, such as an LLI PHIT or UniPro PA-SAP or a DigRF TASLC message it received from configurable 'm' M-RX transactors.

The data payload is an object of type `svt_mphy_mport_lm_transaction` and consists of Data Symbols and optional MARKER and FILLER Symbols.

On receiving the M-PHY Data SAPs, including Data, FILLER and MARKER from 'm' M-RX transactors, the M-Port Lane Management Receiver merges into M-Port Lane Management Data object and makes it available on Ports for testbench or Protocol layer components to observe or receive.

Control SAP transactions can be used to send M-PHY CTRL SAPs to get M-RX capability attributes and get and set M-RX configuration attributes.

4.2.2.1 Receiver Feature Support

The following is a list of supported protocol layer verification features:

- ❖ Port Support
 - ◆ Data Port
 - ◆ Control Port
- ❖ Configurable Stimulus to Input port
- ❖ Testbench visibility and control through callbacks
- ❖ Randomization factories

4.2.2.2 Receiver Ports

UVM Ports are the mechanism through which the M-PHY component connects each other and/or to the testbench. The Protocol component contains various types of ports:

- ❖ Transaction Input Ports

These are used by the sequencer to send sequences into the protocol layer. You cannot connect to them. They are only used by the sequencer. For information on how to use sequence related classes, refer to UVM documentation.
- ❖ Observed Ports

Data is collected from these ports. These ports are used for either generating inputs into response (output ports), or for creating scoreboards and coverage checks.

The M-Port Lane Management Layer component has the following UVM Port interfaces:

- ❖ Transaction Input Port

This is used to supply stimulus transfers for the M-PHY VIP, used only by respective sequencers. User cannot connect to them.

- ◆ `ctrl_in_get_port`
Control SAP Transaction to configure M-TX and M-RX Drivers
- ◆ `data_out_put_port`
Data SAP Transactions from M-Port Lane Management Receiver

4.2.2.3 Receiver Data Objects

The following list of objects represents information that the Protocol layer component receives, sends, or processes:

- ❖ M-Port Lane Management Data Transaction (`svt_mphy_mport_lm_transaction`)
- ❖ M-PHY Control SAP Transaction (`svt_mphy_transaction`)

4.2.3 M-PHY M-Port Lane Management Agent Monitor

M-Port Lane Management Monitor is an UVM component object in an UVM-compliant environment which,

- ❖ Processes and exchanges the transfers with the testbench
- ❖ Communicates with the M-PHY monitors using ports

M-PHY M-Port consists of the following two sub-links, with each sub-link consisting of configurable number of M-PHY modules:

- ❖ TX Sub-link
- ❖ RX Sub-link

M-Port Lane Management Monitor communicates with the testbench to send payload, such as an LLI PHIT or UniPro PA-SAP or a DigRF TASLC message it received from configurable m M-TX or M-RX monitors.

The data payload is an object of type `svt_mphy_mport_lm_transaction` and consists of Data Symbols and optional MARKER and FILLER Symbols.

On receiving the M-PHY Data SAPs from m M-TX or M-RX monitors, the M-Port Lane Management Monitor merges into M-Port Lane Management Data object and makes it available on Ports for testbench or upper layer components to observe or receive.

4.2.3.1 Monitor Feature Support

The following is a list of supported protocol layer verification features:

- ❖ Port Support
- ❖ Data Ports
- ❖ Control Ports
- ❖ Analysis ports
- ❖ Testbench visibility and control through callbacks

4.2.3.2 Monitor Ports

UVM Ports are the mechanism through which the M-PORT component connects each other and/or to the testbench. The M-PORT monitor component contains various types of ports:

The M-Port Lane Management Layer monitor component has the following UVM Port interfaces:

4.2.3.3 Transaction Input Port

These are used to receive SAPs from the M-PHY monitors. User cannot connect to them.

- ❖ `tx_ctrl_in_get_port`
To get control SAP transaction M-TX monitor
- ❖ `rx_ctrl_in_get_port`
To get control SAP transaction M-RX monitor
- ❖ `tx_data_in_get_port`
To get data SAP transaction M-TX monitor
- ❖ `rx_data_in_get_port`
To get data SAP transaction M-RX monitor

4.2.3.4 Transaction Output Port

These analysis ports are used to send SAPs to the testbench or upper layer. User can connect to them.

- ❖ `tx_data_out_put_port`
To send TX data SAP transaction
- ❖ `rx_data_out_put_port`
To send RX data SAP transaction

4.2.3.5 Monitor Data Objects

The following list of objects represents information that the Protocol layer component receives, sends, or processes:

- ❖ M-Port Lane Management Data Transaction (`svt_mphy_mport_lm_transaction`)
- ❖ M-PHY Control SAP Transaction (`svt_mphy_transaction`)

4.2.3.6 Monitor Callbacks

The MPHY M-PORT Monitor supports multiple callbacks for viewing process flow data points, and covering data and activities. The Monitor is associated with a callback class in correspondence to each of the M-PHY LM MPORT Driver layer components that contains a set of callback methods.

Monitor provides the following callbacks classes:

`svt_mphy_mport_lm_monitor_callback`

For more details, see Class reference HTML documentation.

4.2.3.7 Monitor checkers

The MPHY M-PORT Monitor provides checkers to check the behavior of MPHY monitors. The test environment can selectively enable and disable check levels, as defined and supported by the components. The M-PHY VIP provides the following checks in the M-PHY M-PORT Agent Monitor:

`svt_mphy_mport_lm_err_check`

By default, the checkers are enabled.

For more details, see Class reference HTML documentation.

4.3 M-PHY Model Agent UVM Component

The M-PHY Model Agent is a PHY agent which is connected to RMMI interface on one side and to SERIAL interface on the other side. It consists of RMMI and SERIAL M-PHY Driver or Receiver components. The M-PHY Model Agent is configured using an object of `svt_mphy_agent_configuration` class.

M-PHY Model agent can be configured in the following two ways:

- ❖ “M-PHY TX-Model Agent”

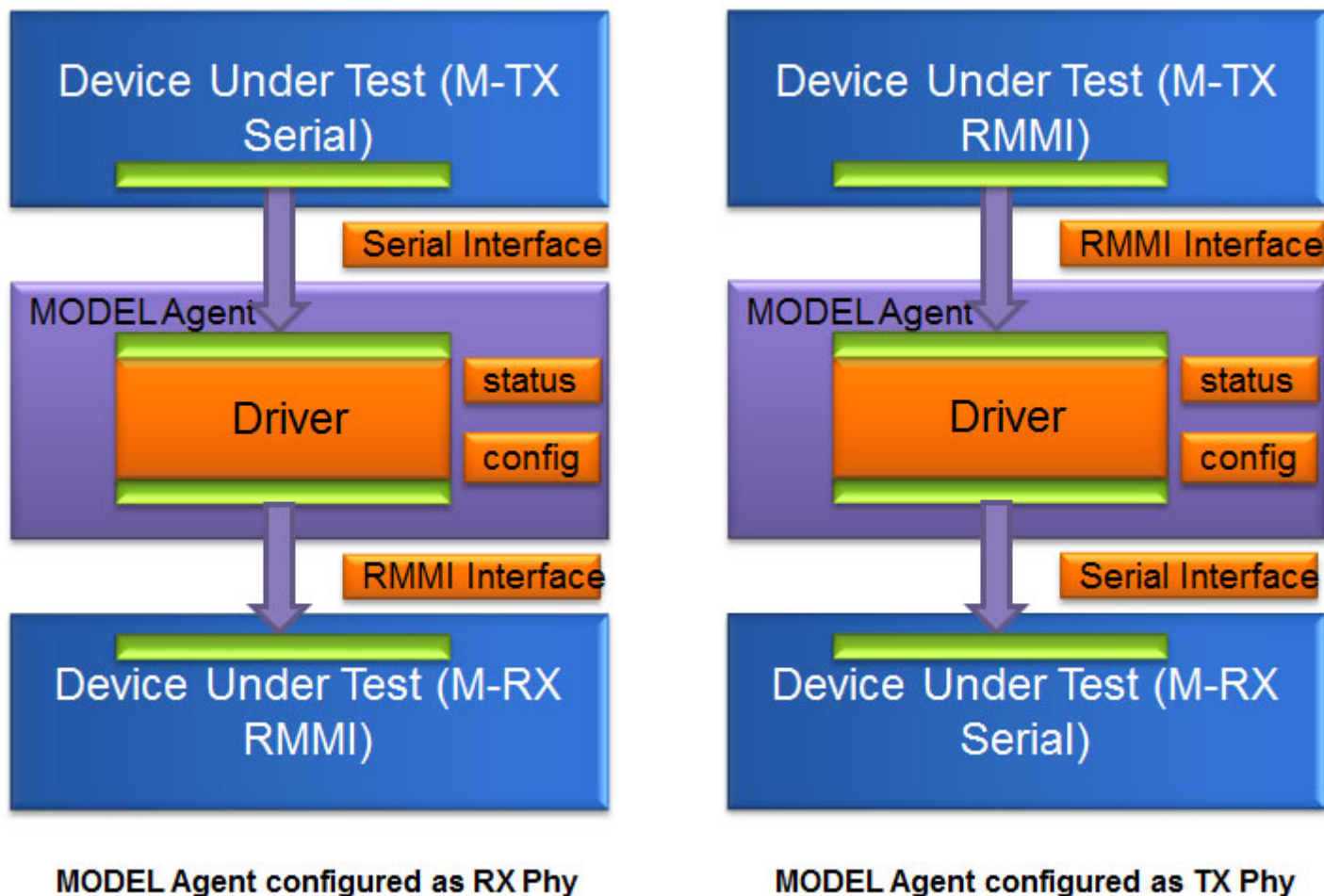
It takes input from RMMI interface and drives it on SERIAL interface, that is, RMMI to SERIAL conversion.

- ❖ “M-PHY RX-Model Agent”

It takes input from SERIAL interface and drives it on RMMI interface, that is, SERIAL to RMMI conversion.

Figure 4-4 shows the block diagram for M-PHY Model Agent with different configurations.

Figure 4-4 M-PHY Model Agent



4.3.1 M-PHY TX-Model Agent

M-PHY TX Model Agent is an UVM component object in an UVM-compliant environment which,

- ❖ Processes and exchanges the transfers between RMMI and SERIAL interfaces
- ❖ Processes and exchanges the transfers between RMMI and SERIAL TX drivers

M-PHY TX-Model Agent includes two M-PHY TX drivers:

1. M-PHY M-TX RMMI Driver (svt_mphy_tx)

This driver is connected to M-PHY M-TX DUT Controller Interface

"mphy_rmmi_mtx_dut_controller_if" on one side and to M-TX SERIAL driver on the other side. It reads the DATA and CTRL information from M-TX DUT Controller interface and passes it to M-TX SERIAL Driver.

2. M-PHY M-TX SERIAL Driver (svt_mphy_tx)

This driver is connected to M-PHY M-TX RMMI Driver on one side and to M-TX SERIAL interface "mphy_serial_tx_if" on other side. It takes the DATA and CTRL information in the form of transaction object from M-TX RMMI driver and drives it on M-TX SERIAL interface "mphy_serial_tx_if".

This agent requires the following configuration parameters to be set properly to work as a TX Model:

```
cfg.component_type = svt_mphy_configuration::MODEL;
cfg.dir = svt_mphy_configuration::M_TX;
```

4.3.2 M-PHY RX-Model Agent

M-PHY RX Model Agent is an UVM component object in an UVM-compliant environment which,

- ❖ Processes and exchanges the transfers between SERIAL and RMMI interfaces
- ❖ Processes and exchanges the transfers between SERIAL and RMMI RX receivers

M-PHY RX-Model Agent includes two M-PHY RX receivers:

1. M-PHY M-RX RMMI Receiver (svt_mphy_rx)

This receiver is connected to M-PHY M-RX DUT Controller Interface

"mphy_rmmi_mrx_dut_controller_if" on one side and to M-RX SERIAL receiver on the other side. It reads CTRL information from M-RX DUT Controller interface and passes it to M-RX SERIAL Driver.

Also, it takes DATA from M-RX SERIAL receiver and drives it on M-RX RMMI DUT Controller Interface "mphy_rmmi_mrx_dut_controller_if".

2. M-PHY M-RX SERIAL Receiver (svt_mphy_rx)

This receiver is connected to M-PHY M-RX RMMI receiver on one side and to M-RX SERIAL interface "mphy_serial_rx_if" on the other side. Also, it reads DATA from M-RX SERIAL interface "mphy_serial_rx_if" and passes it to M-RX RMMI interface.

This agent requires the following configuration parameters to be set properly to work as a RX Model:

```
cfg.component_type = svt_mphy_configuration::MODEL;  
cfg.dir = svt_mphy_configuration::M_RX;
```


5

Using M-PHY Verification IP

This chapter discusses the UVM concepts and techniques for quickly achieving a basic constrained random testbench that incorporates the M-PHY VIP. The code snippets illustrate these methods in practical use. The testbench shows typical M-PHY VIP and SystemVerilog UVM usage, and highlights the concepts and techniques described. These techniques can be used with any of the VIP products.

This chapter discusses the following topics:

- ❖ [Creating a Test Environment](#)
- ❖ [Instantiating the VIP](#)
- ❖ [Configuring the VIP Models](#)
- ❖ [Generating Constrained Random Stimulus](#)
- ❖ [Controlling the Test](#)
- ❖ [Lane-to-Lane Skew](#)
- ❖ [Exception Support](#)
- ❖ [Inserting PPM Drift and Clock Recovery](#)
- ❖ [M-PHY Specification Version 2.0 Feature Support](#)
- ❖ [M-PHY Specification Version 3.0 Feature Support](#)
- ❖ [M-PHY Specification Version 3.1 Feature Support](#)
- ❖ [Line Reset](#)
- ❖ [Lane to Lane Skew in Ctrl Transactions](#)
- ❖ [M-PHY Specification Version 4.0 and Later Feature Support](#)
- ❖ [Set Clock Period for PWM and HS Mode](#)

5.1 Creating a Test Environment

To create a test environment for M-PHY VIP, the following scenarios exist:

- ❖ [Creating a Test Environment Without Lane Management](#)
- ❖ [Creating a Test Environment With Lane Management](#)

5.1.1 Creating a Test Environment Without Lane Management

The `uvm_env` base class provides a testbench template to organize both the flow and the code associated with a test. You can create a test environment by defining a new class extended from `uvm_env`. The environment class contains (or affects) the entire test environment as it provides an overall structure. Typically, the majority of code in an UVM testbench is contained in the environment, which can be reused by other tests or projects. Due to this, the `uvm_env` class has a large impact on user coding, so understanding the `uvm_env` class is central to effectively writing the environment code.

Another result of the overarching nature of `uvm_env` is that it is an excellent vehicle for organizing the discussion of UVM and M-PHY VIP techniques and methods. Therefore the subjects presented here are organized and presented in the context of `uvm_env`.

The `uvm_env` base class has several methods that correspond to the major steps all tests follow. Most of these methods are declared in the base class as virtual and should be extended by the user.

To create a user environment, define a new class extended from `uvm_env` and extend the methods above to add test-specific code. To retain the core functionality of the base class methods, each extended method must call `super.method` as the first line of code.

The code snippet below shows the user extension of `uvm_env` by creating the class `mphy_svt_env`.

The "`mphy_svt_env`" class instantiates all the components of the testing environment, which may include VIP products, user designs, generators, scoreboards, and so on. At the end of this extended class is a list of the methods that are extended.

For details on creating the user UVM environment, refer to the M-PHY Basic Example shipped with the VIP.

```

////////////////////////////////////
// Verification Environment
////////////////////////////////////
class mphy_svt_env extends uvm_env;
// Instantiate the objects that compose the environment
// M-PHY VIP models

//An instance of M-PHY TX VIP Agent to act as a M-TX
svt_mphy_tx_agent    mtx_agent;

//An instance of M-PHY VIP RX Agent to act as a M-RX
svt_mphy_rx_agent    mrx_agent;

// Configuration object for master & slave agent
svt_mphy_agent_configuration    mtx_cfg;
svt_mphy_agent_configuration    mrx_cfg;

//Master & slave Interfaces
virtual svt_mphy_serial_tx_if    mtx_if;
virtual svt_mphy_serial_rx_if    mrx_if;

// List of methods in this class

```

```
extern function new (string name, uvm_component parent);
extern virtual function build_phase();
extern virtual function connect_phase();
extern virtual task main_phase();
endclass: mphy_svt_env
```

When the `run_test` method is called from the testbench top, the methods of the `mphy_svt_env` and other `uvm_components` are called automatically in the below order.

`build_phase -> connect_phase -> main_phase`

Calling this method (`run_test`) launches the entire test sequence inside a testbench top module:

```
initial
begin
run_test();
$finish;
end
```

Notice how small the testbench top is, when UVM is used. Most of the code is in the environment, which is a reusable component. The test-specific code is minimized and is kept common so it is not replicated unnecessarily. This yields a smaller code base to maintain.

The user inherits the structure and base functionality from `uvm_env` and can customize it when needed. In addition, the customization is controlled by the user. This is a common theme throughout the UVM and VIP products. In the sections that follow, the individual steps in the test sequence are shown in detail.

5.1.2 Creating a Test Environment With Lane Management

The `uvm_env` base class provides a testbench template to organize both the flow and the code associated with a test. You can create a test environment by defining a new class extended from `uvm_env`. The environment class contains (or affects) the entire test environment as it provides an overall structure.

Typically, the majority of code in an UVM testbench is contained in the environment, which can be reused by other tests or projects. Due to this, the `uvm_env` class has a large impact on user coding, so understanding the `uvm_env` class is central to effectively writing the environment code. Another result of the overarching nature of `uvm_env` is that it is an excellent vehicle for organizing the discussion of UVM and M-PHY VIP techniques and methods. Therefore the subjects presented here are organized and presented in the context of `uvm_env`.

The `uvm_env` base class has several methods that correspond to the major steps all tests follow. Most of these methods are declared in the base class as virtual and should be extended by the user. To create a user environment, define a new class extended from `uvm_env` and extend the methods above to add test-specific code. To retain the core functionality of the base class methods, each extended method must call `super.method` as the first line of code.

The code snippet below shows the user extension of `uvm_env` by creating the class `mphy_mport_lm_svt_env`. The "`mphy_mport_lm_svt_env`" class instantiates all the components of the testing environment, which may include VIP products, user designs, generators, scoreboards, and so on. At the end of this extended class is a list of the methods that are extended. For details on creating the user UVM environment, refer to the M-PHY Basic Example shipped with the VIP.

For details on creating the user UVM environment, refer to the M-PHY Basic Example shipped with the VIP.

```

////////////////////////////////////
// Verification Environment
////////////////////////////////////
class mphy_mport_lm_svt_env extends uvm_env;
// Instantiate the objects that compose the environment
// M-PHY Lane Management VIP models
// An instance of M-PHY Lane Management VIP Agent to act as Master
svt_mphy_mport_lm_agent master_agent;

//An instance of M-PHY Lane Management VIP Agent to act as a Slave
svt_mphy_mport_lm_agent slave_agent;

// Configuration object for master & slave agent
svt_mphy_mport_lm_configuration master_cfg;
svt_mphy_mport_lm_configuration slave_cfg;

// List of methods in this class
extern function new (string name, uvm_component parent);
extern virtual function build_phase();
extern virtual function connect_phase();
extern virtual task main_phase();

endclass: mphy_svt_env

```

When the `run_test` method is called from the testbench top, the methods of the `mphy_mport_lm_svt_env` and other `uvm_components` are called automatically in the below order.

`build_phase` -> `connect_phase` -> `main_phase`

Calling this method (`run_test`) launches the entire test sequence inside a testbench top module:

```

initial
begin
run_test();
$finish;
end

```

Notice how small the testbench top is, when UVM is used. Most of the code is in the environment, which is a reusable component. The test-specific code is minimized and is kept common so it is not replicated unnecessarily. This yields a smaller code base to maintain. The user inherits the structure and base functionality from `uvm_env` and can customize it when needed. In addition, the customization is controlled by the user. This is a common theme throughout UVM and VIP products. In the sections that follow, the individual steps in the test sequence are shown in detail.

5.2 Instantiating the VIP

To instantiate the M-PHY VIP, the following scenarios exist:

- ❖ [Instantiating the VIP Without Lane Management](#)
- ❖ [Instantiating the VIP With Lane Management](#)

5.2.1 Instantiating the VIP Without Lane Management

The M-PHY VIP models are instantiated in the testbench and appropriate configuration objects are passed to the agent's object. These components can be configured as active or passive components.

```
//An instance of M-PHY TX VIP Agent to act as a M-TX
svt_mphy_tx_agent    mtx_agent;

//Master interfaces
virtual svt_mphy_serial_tx_if    mtx_if;

//Creation of "mtx_agent" object
mtx_agent = svt_mphy_tx_agent::type_id::create("mtx_agent",this);

//Connect the virtual interface of the environment to master agent interface
uvm_config_vif_db#(virtual svt_mphy_serial_tx_if)::set(this,  "mtx_agent",  "vif",
mtx_if);

//Connect the virtual interface of the environment to the physical interface present in
top file
this.mtx_if = mphy_top.mtx_if;

//mphy_top file
svt_mphy_serial_tx_if    mtx_if();
svt_mphy_serial_rx_if    DUT_if();

// TX Sublink connection
assign DUT_if.mphy_rx_if.rx_dp = mtx_if.tx_dp;
assign DUT_if.mphy_rx_if.rx_dn = mtx_if.tx_dn;
```

For more information, refer to the Basic UVM Example shipped with the M-PHY VIP Model.

5.2.2 Instantiating the VIP With Lane Management

The Lane Management M-PHY VIP models are instantiated in the testbench and appropriate configuration objects are passed to the agent's object.

```
//An instance of Lane Management M-PHY VIP Agent to act as a Master
svt_mphy_mport_lm_agent master_agent;

//An instance of Agent configuration object
svt_mphy_mport_lm_configuration master_cfg;

//Creation of "master_agent" object
master_agent = svt_mphy_mport_lm_agent::type_id::create("master_agent",this);
```

5.3 Configuring the VIP Models

To configure the M-PHY VIP, the following scenarios exist:

- ❖ [Configuring the VIP Models Without Lane Management](#)
- ❖ [Configuring the VIP Models With Lane Management](#)
- ❖ [Configuring the Passive Serial Monitors \(TX Monitor is Connected to TX DUT or RX Monitor is Connected to RX DUT\)](#)

5.3.1 Configuring the VIP Models Without Lane Management

The VIP models are configured using configuration objects that are ready for use. The configuration objects can be randomized and passed as an argument to a method. The objects come with constraints so that they adhere to the protocol limits. They can be controlled or extended to create specific test conditions, or used as is to produce a wide range of stimulus.

```
//Passing configuration to "mtx_agent" object
```

```
uvm_config_db#(svt_mphy_agent_configuration)::set(this, "mtx_agent", "cfg",mtx_cfg);
```

The configuration objects can be randomized and then some of the attributes of the `mtx_cfg` object can be manually assigned. This is one approach to control the attribute values.

In most cases, the default values of the configuration class are manually assigned and then used for the other attributes.

The test case can create the configuration object, assign the values required for the particular test, and also assign the configuration object to the environment's configuration.

```
test_cfg = sv_t_mphy_agent_configuration::type_id::create("test_cfg");
uvm_config_db#(svt_mphy_agent_configuration)::set(this, "mphy_svt_env", "cfg",
test_cfg);
```

If the configuration object is not provided by the test case, the environment can create its own configuration object.

The examples below show how to configure the attributes of the M-PHY Agent configuration.

5.3.1.1 Configuring the VIP Component as Active

The M-PHY TX Agent can be configured as an active component by setting the "is_active" parameter.

```
//The Top Env has
svt_mphy_agent_configuration tx_cfg;

/** MPHY Transmitter active agent instance. */
svt_mphy_tx_agent tx_agent;

/** Active Host agent instance */
tx_agent = sv_t_mphy_tx_agent::type_id::create("tx_agent",this);

//Cust sys configuration class has sv_t_mphy_agent_configuration master_cfg
tx_cfg = sv_t_mphy_agent_configuration::type_id::create("tx_cfg");
tx_cfg.is_active = 1;

uvm_config_db#(svt_mphy_agent_configuration)::set(this,"tx_agent", "tx_cfg", tx_cfg);
```


The M-PHY RX Agent can also be configured as an active component as mentioned above by replacing "tx" by "rx".

5.3.1.2 Configuring the VIP Component as Passive for RMMI Interface

The M-PHY Agent can be configured as a passive component by setting the "is_active" parameter to 0.

For RMMI Local Interface

1. When the VIP Agent is Master M-PHY, DUT is M-PHY Slave and the M-PHY slave agent is used as a passive only to monitor DUT, the following configuration parameters need to be set:

```
svt_mphy_rx_agent          passive_rx_agent;
svt_mphy_agent_configuration passive_rx_cfg;
svt_mphy_status            passive_rx_status;

passive_rx_agent = svt_mphy_rx_agent::type_id::create("passive_rx_agent ",this);

passive_rx_cfg.is_active=0;
passive_rx_cfg.enable_monitor = 1;
passive_rx_cfg.use_debug_interface = 0;
```

**Note**

For Passive mode, along with "is_active", user also needs to make sure that two more parameters, "enable_monitor" and "use_debug_interface" are configured properly as shown in the code above.

```
uvm_config_db#( svt_mphy_agent_configuration )::set(this, "passive_rx_agent", "cfg",
passive_rx_cfg);

uvm_config_db#(svt_mphy_status)::set(this, "passive_rx_agent", "shared_status",
passive_rx_status);
```

The Passive RX Agent will be connected to "rmmtx_dut_controller_if" and will monitor all the interface signals.

2. When the VIP Agent is Slave M-PHY, DUT is M-PHY Master and the M-PHY Master agent is used as a passive only to monitor DUT, the following configuration parameters need to be set:

```
svt_mphy_tx_agent          passive_tx_agent;
svt_mphy_agent_configuration passive_tx_cfg;
svt_mphy_status            passive_tx_status;

passive_tx_agent = svt_mphy_tx_agent::type_id::create("passive_tx_agent ",this);

passive_tx_cfg.is_active=0;
passive_tx_cfg.enable_monitor = 1;
passive_tx_cfg.use_debug_interface = 0;
```

**Note**

For Passive mode, along with "is_active", user also needs to make sure that two more parameters, "enable_monitor" and "use_debug_interface" are configured properly as shown in the code above.

```
uvm_config_db#( svt_mphy_agent_configuration )::set(this, "passive_tx_agent", "cfg",
passive_tx_cfg);
```

```
uvm_config_db#(svt_mphy_status)::set(this, "passive_tx_agent", "shared_status",
passive_tx_status);
```

The Passive TX Agent will be connected to "rmmi_mtx_dut_phy_if" and will monitor all the interface signals.

For RMMI Remote Interface

1. When M-PHY VIP Agent is Master M-PHY, DUT is M-PHY Slave and the M-PHY slave agent is used as a passive only to monitor DUT, the following configuration parameters need to be set:

```
svt_mphy_rx_agent          passive_rx_agent;
svt_mphy_agent_configuration passive_rx_cfg;
svt_mphy_status            passive_rx_status;
passive_rx_agent = svt_mphy_rx_agent::type_id::create("passive_rx_agent ",this);

passive_rx_cfg.is_active=0;
passive_rx_cfg.enable_monitor = 1;
passive_rx_cfg.use_debug_interface = 0;
```



Note

For Passive mode, along with "is_active", user also needs to make sure that two more parameters, "enable_monitor" and "use_debug_interface" are configured properly as shown in the code above.

```
uvm_config_db#( svt_mphy_agent_configuration )::set(this, "passive_rx_agent", "cfg",
passive_rx_cfg);
```

```
uvm_config_db#(svt_mphy_status)::set(this, "passive_rx_agent", "shared_status",
passive_rx_status);
```

The Passive RX Agent will be connected to "rmmi_mrx_dut_phy_if" and will monitor all the interface signals.

2. When M-PHY VIP Agent is Slave M-PHY, DUT is M-PHY Master and the M-PHY Master agent is used as a passive only to monitor DUT, the following configuration parameters need to be set:

```
svt_mphy_tx_agent          passive_tx_agent;
svt_mphy_agent_configuration passive_tx_cfg;
svt_mphy_status            passive_tx_status;

passive_tx_agent = svt_mphy_tx_agent::type_id::create("passive_tx_agent ",this);

passive_tx_cfg.is_active=0;
passive_tx_cfg.enable_monitor = 1;
passive_tx_cfg.use_debug_interface = 0;
```



Note

For Passive mode, along with "is_active", user also needs to make sure that two more parameters, "enable_monitor" and "use_debug_interface" are configured properly as shown in the code above.

```
uvm_config_db#( svt_mphy_agent_configuration )::set(this, "passive_tx_agent", "cfg",
passive_tx_cfg);
```

```
uvm_config_db#(svt_mphy_status)::set(this, "passive_tx_agent", "shared_status",
passive_tx_status);
```

The Passive TX Agent will be connected to "rmmi_mrx_dut_controller_if" and will monitor all the interface signals.

5.3.1.3 Configuring the VIP Component as Passive for Serial Interface

The M-PHY VIP has two independent agents M-TX and M-RX (which can work for SERIAL or RMMI). Each agent has its own monitor. The M-PHY Agent can be configured as an active component by setting the "is_active" parameter.

For example:

1. **VIP Agent is Master M-PHY, DUT is M-PHY Slave and the M-PHY slave agent is used as a passive only to monitor DUT**

The following configuration parameters need to be set:

```
svt_mphy_rx_agent passive_rx_agent;  
svt_mphy_agent_configuration passive_rx_cfg;  
svt_mphy_status passive_rx_status;  
passive_rx_agent = svt_mphy_rx_agent::type_id::create("passive_rx_agent ",this);  
passive_rx_cfg.is_active=0;  
passive_rx_cfg.enable_monitor = 1;  
passive_rx_cfg.use_debug_interface = 0;
```



Note

For Passive mode, along with "is_active", user also needs to make sure that two more parameters, "enable_monitor" and "use_debug_interface" are configured properly as shown in the code above.

```
uvm_config_db#( svt_mphy_agent_configuration )::set(this, "passive_rx_agent", "cfg",  
passive_rx_cfg);  
uvm_config_db#(svt_mphy_status)::set(this, "passive_rx_agent", "shared_status",  
passive_rx_status);
```

The Passive RX Agent will be connected to "mphy_serial_rx_if" and will monitor all the interface signals.

In passive mode, user needs to pass the control information to the monitor to make it work in sync with other modules. For this purpose, a "uvm_analysis_imp (mon_ctrl_in_export)" is provided in M-PHY monitor to get control information/transaction in passive mode. User can connect this analysis port and pass the control transaction through the testbench.

Use model is shown in the following code snippet.

- ❖ **VIP M-PHY Master agent configured as Passive**

```
uvm_analysis_port #(svt_mphy_transaction) tx_mon_ctrl_in_port;  
...  
...  
tx_mon_ctrl_in_port = new("tx_mon_ctrl_in_port",this);  
...  
...  
tx_mon_ctrl_in_port.connect(  
    passive_tx_agent.tx_monitor.mon_ctrl_in_export);  
...  
tx_mon_ctrl_in_port.write(mphy_ctrl_trans);
```

- ❖ **VIP M-PHY Slave agent configured as Passive**

```

uvm_analysis_port #(svt_mphy_transaction) rx_mon_ctrl_in_port;
...
...
rx_mon_ctrl_in_port = new("rx_mon_ctrl_in_port",this);
...
...
rx_mon_ctrl_in_port.connect(
    passive_rx_agent.rx_monitor.mon_ctrl_in_export);
...
rx_mon_ctrl_in_port.write(mphy_ctrl_trans);

```

2. VIP Agent is Slave M-PHY, DUT is M-PHY Master and the M-PHY master agent is used as a passive only to monitor DUT

The procedure will be same as [Step 1](#), the exception being that "rx" would be changed to "tx".

5.3.2 Configuring the VIP Models With Lane Management

The VIP models are configured using configuration objects that are ready for use. The configuration objects can be randomized and passed as an argument to a method. The objects come with constraints so that they adhere to the protocol limits. They can be controlled or extended to create specific test conditions, or used as is to produce a wide range of stimulus.

```

uvm_config_db#(svt_mphy_mport_lm_configuration)::set(this,"master_agent", "tx_cfg",
this.sys_cfg.master_cfg);

```

```

master_agent = sv_t_mphy_mport_lm_agent::type_id::create("master_agent",this);

```

The test case can create the configuration object, assign the values required for the particular test, and also assign the configuration object to the environment's configuration.

```

// Set the updated MPHY Shared Configuration class for the ENV
uvm_config_db#(mphy_lm_shared_cfg)::set(this, "env", "shared_cfg", this.shared_cfg);

```

5.3.2.1 Configuring the VIP Component as Active

The M-PHY LM MPORT Master Agent can be configured as an active component by setting the "is_active" parameter.

```

//The Top Env has
svt_mphy_mport_lm_agent_configuration  master_cfg;

/** MPHY Transmitter active agent instance. */
svt_mphy_mport_lm_agent master_agent;

/** Active Host agent instance */
master_agent = sv_t_mphy_mport_lm_agent::type_id::create("master_agent",this);

//Cust sys configuration class has sv_t_mphy_mport_lm_configuration master_cfg
master_cfg = sv_t_mphy_mport_lm_configuration::type_id::create("master_cfg");
master_cfg.is_active = 1;
uvm_config_db#(svt_mphy_mport_lm_configuration)::set(this,"master_agent", "tx_cfg",
this.sys_cfg.master_cfg);

```

The M-PHY LM MPORT Slave Agent can also be configured as an active component as mentioned above by replacing "tx" by "rx".

5.3.2.2 Configuring the VIP Component as Passive for RMMI Interface

The M-PHY LM MPORT Agent can be configured as passive component by setting the "is_active" parameter in its configuration. Some other parameters also need to be set as described in the use case.

Use Case:

1. Master LM MPORT VIP Agent configured as Passive

```
svt_mphy_mport_lm_agent    passive_master_agent;
svt_mphy_mport_lm_agent_configuration passive_master_cfg;
/** Copying the Active master agent cfg in Passive master agent cfg instance */

assert($cast(passive_master_cfg, master_cfg.clone()));
assert($cast(passive_master_cfg.tx_cfg, master_cfg.tx_cfg.clone()));
assert($cast(passive_master_cfg.rx_cfg, master_cfg.rx_cfg.clone()));
```

The following fields also need to be modified after copying the cfg into passive_monitor_cfg:

```
/** Set is_active to 0 to make agent as passive */
passive_master_cfg.is_active=0;

/** Set enable_monitor to 1 to enable monitor */
passive_master_cfg.enable_monitor = 1;

/** Set use_debug_interface to 0 to disable debug interface*/
passive_master_cfg.use_debug_interface = 0;
```



Note

For Passive mode, along with "is_active", user also needs to make sure that two more parameters, "enable_monitor" and "use_debug_interface" are configured properly as shown in the code above.

```
uvm_config_db#(svt_mphy_mport_lm_agent_configuration)::set(this, "passive_master_agent",
"cfg", passive_master_cfg);

/** Passive agent instance */
passive_master_agent =
svt_mphy_mport_lm_agent::type_id::create("passive_master_agent", this);
```

2. Slave LM MPORT VIP Agent configured as Passive

```
svt_mphy_mport_lm_agent    passive_slave_agent;
svt_mphy_mport_lm_agent_configuration passive_slave_cfg;

/** Copying the Active slave agent cfg in Passive slave agent cfg instance */

assert($cast(passive_slave_cfg, slave_cfg.clone()));
assert($cast(passive_slave_cfg.tx_cfg, slave_cfg.tx_cfg.clone()));
assert($cast(passive_slave_cfg.rx_cfg, slave_cfg.rx_cfg.clone()));
```

The following fields also need to be modified after copying the cfg into passive_slave_cfg:

```
/** Set is_active to 0 to make agent as passive */
passive_slave_cfg.is_active=0;

/** Set enable_monitor to 1 to enable monitor */
passive_slave_cfg.enable_monitor = 1;
```

```
/** Set use_debug_interface to 0 to disable debug interface*/
passive_slave_cfg.use_debug_interface = 0;
```

**Note**

For Passive mode, along with "is_active", user also needs to make sure that two more parameters, "enable_monitor" and "use_debug_interface" are configured properly as shown in the code above.

```
uvm_config_db#(svt_mphy_mport_lm_agent_configuration)::set(this, "passive_slave_agent",
"cfg", passive_slave_cfg);
```

```
/** Passive agent instance */
passive_slave_agent =
svt_mphy_mport_lm_agent::type_id::create("passive_slave_agent", this);
```

5.3.2.3 Configuring the VIP Component as Passive for Serial Interface

When the user wants to monitor at Lane Management layer, then the user can instantiate a single LM agent (at TX or RX) and configure it as passive.

The M-PHY LM MPORT Agent can be configured as active/passive component by setting the "is_active" parameter.

For example:

1. VIP LM MPORT Agent is Master, DUT is Slave and the VIP LM MPORT slave agent is used as passive only to monitor DUT

The user can configure Passive LM agent as:

```
svt_mphy_mport_lm_agent      passive_slave_agent;
svt_mphy_mport_lm_agent_configuration  passive_slave_cfg;

/** Set is_active to 0 to make agent as passive */
passive_slave_cfg.is_active=0;

/** Set enable_monitor to 1 to enable monitor */
passive_slave_cfg.enable_monitor = 1;

/** Set use_debug_interface to 0 to disable debug interface*/
passive_slave_cfg.use_debug_interface = 0;

uvm_config_db#(svt_mphy_mport_lm_agent_configuration)::set(this, "passive_slave_agent",
"cfg", passive_slave_cfg);

/** Passive Slave agent instance */
passive_slave_agent =
svt_mphy_mport_lm_agent::type_id::create("passive_slave_agent", this);
```

**Note**

For Passive mode, along with "is_active", user also needs to make sure that two more parameters, "enable_monitor" and "use_debug_interface" are configured properly as shown in the code above.

In passive mode, user needs to pass control information to M-PHY monitor to make it work in sync with other modules. To do this, a "uvm_analysis_imp (mon_ctrl_in_export)" is provided in M-PHY monitor to get control information/transaction in passive mode. User can connect this analysis port and pass control transaction through LM MPORT testbench.

```
//Analysis port to get Control transaction in case of passive
// For Master agent of TX_Sublink
    uvm_analysis_port #(svt_mphy_transaction) master_tx_mon_ctrl_in_port[];

// For Master agent of RX_Sublink
    uvm_analysis_port #(svt_mphy_transaction) master_rx_mon_ctrl_in_port[];
```

Use model is shown in the following code snippet:

❖ **VIP M-PHY LM MPORT Master agent configured as Passive for TX_SUBLINK**

```
uvm_analysis_port #(svt_mphy_transaction) master_tx_mon_ctrl_in_port[];
...
...
for(int i = 0; i < master_cfg.tx_cfg.active_lane;i++) begin
    $sformat(inst_name,"master_tx_mon_ctrl_in_port[%0d]",i);
    master_tx_mon_ctrl_in_port[i] = new(inst_name, this);
end
...
...
for (int i = 0; i < master_cfg.tx_cfg.active_lane; i++)begin

master_tx_mon_ctrl_in_port[i].connect(passive_master_agent.tx_monitor[i].mon_ctrl_in_exp
ort);

end
...
...
for (int i = 0; i < master_cfg.tx_cfg.active_lane; i++) begin
master_tx_mon_ctrl_in_port[i].write(mphy_ctrl_trans);
end
```

❖ **VIP M-PHY LM MPORT Master agent configured as Passive for RX_SUBLINK**

```
uvm_analysis_port #(svt_mphy_transaction) master_rx_mon_ctrl_in_port[];
...
...
for(int i = 0; i < master_cfg.rx_cfg.active_lane;i++) begin
    $sformat(inst_name,"master_rx_mon_ctrl_in_port[%0d]",i);
    master_rx_mon_ctrl_in_port[i] = new(inst_name, this);
end
...
...
for (int i = 0; i < master_cfg.rx_cfg.active_lane; i++)begin
master_rx_mon_ctrl_in_port[i].connect(passive_master_agent.rx_monitor[i].mon_ctrl_in_exp
ort);

end
...
...
for (int i = 0; i < master_cfg.rx_cfg.active_lane; i++) begin
```

```
        master_rx_mon_ctrl_in_port[i].write(mphy_ctrl_trans);  
    end
```


2. VIP LM MPORT Agent is Slave, DUT is Master and the M-PHY LM MPORT master agent is used as a passive only to monitor DUT

The user can configure Passive LM agent as:

```
svt_mphy_mport_lm_agent      passive_master_agent;
svt_mphy_mport_lm_agent_configuration  passive_master_cfg;

/** Set is_active to 0 to make agent as passive */
passive_master_cfg.is_active=0;

/** Set enable_monitor to 1 to enable monitor */
passive_master_cfg.enable_monitor = 1;
/** Set use_debug_interface to 0 to disable debug interface*/
passive_master_cfg.use_debug_interface = 0;

uvm_config_db#(svt_mphy_mport_lm_agent_configuration)::set(this, "passive_master_agent",
"cfg", passive_master_cfg);

/** Passive agent instance */
passive_master_agent =
svt_mphy_mport_lm_agent::type_id::create("passive_master_agent", this);
```



Note

For Passive mode, along with "is_active", user also needs to make sure that two more parameters, "enable_monitor" and "use_debug_interface" are configured properly as shown in the code above.

In passive mode, the user needs to pass control information to M-PHY monitor to make it work in sync with other modules. To do this, a "uvm_analysis_imp (mon_ctrl_in_export)" is provided in M-PHY monitor to get control information/transaction in passive mode. User can connect this analysis port and pass control transaction through LM MPORT testbench.

```
// For Slave agent of TX_Sublink
uvm_analysis_port #(svt_mphy_transaction) slave_rx_mon_ctrl_in_port[];
// For Slave agent of RX_Sublink
uvm_analysis_port #(svt_mphy_transaction) slave_tx_mon_ctrl_in_port[];
```

Use model is shown in following code snippet:

❖ VIP M-PHY LM MPORT Slave agent configured as Passive for TX_SUBLINK

```
uvm_analysis_port #(svt_mphy_transaction) slave_rx_mon_ctrl_in_port[];
...
...
...

for(int i = 0; i < slave_cfg.rx_cfg.active_lane;i++) begin
    $sformat(inst_name,"slave_rx_mon_ctrl_in_port[%0d]",i);
    slave_rx_mon_ctrl_in_port[i] = new(inst_name, this);
end
...
...
...

for (int i = 0; i < slave_cfg.rx_cfg.active_lane; i++) begin
```

```

slave_rx_mon_ctrl_in_port[i].connect(passive_slave_agent.rx_monitor[i].mon_ctrl_in_export);
end
...
...
...

for (int i = 0; i < slave_cfg.rx_cfg.active_lane; i++) begin
    slave_rx_mon_ctrl_in_port[i].write(mphy_ctrl_trans);
end

```

❖ VIP M-PHY LM MPORT Slave agent configured as Passive for RX_SUBLINK

```

uvm_analysis_port #(svt_mphy_transaction) slave_tx_mon_ctrl_in_port[];
...
...
...

for(int i = 0; i < slave_cfg.tx_cfg.active_lane;i++) begin
    $sformat(inst_name,"slave_tx_mon_ctrl_in_port[%0d]",i);
    slave_tx_mon_ctrl_in_port[i] = new(inst_name, this);
end
...
...
...

for (int i = 0; i < slave_cfg.tx_cfg.active_lane; i++) begin

slave_tx_mon_ctrl_in_port[i].connect(passive_slave_agent.tx_monitor[i].mon_ctrl_in_export);
end
...
...
...

for (int i = 0; i < slave_cfg.tx_cfg.active_lane; i++) begin
    slave_tx_mon_ctrl_in_port[i].write(mphy_ctrl_trans);
end

```

5.3.2.4 Configuring the Shared Status of VIP Components

Each transactor of M-PHY VIP has an attribute `shared_status` of type `svt_mphy_status`. This `shared_status` attribute reflects the state of M-PHY transactor, that is, the mode, gear and series at which the transactor is working.

Base test defines `top_status` attribute which has the reference to the `shared_status` of all transactors at TX path and RX path.

A reference to the `top_status` is created by example tests (which are extended from Base test) with the virtual sequencer class of LM MPORT environment or M-PHY environment using `uvm_config_db SET/GET` method. In case `top_status` of the sequencer is not set by example tests, sequence collection file issues a fatal error message.

Top module also defines a `top_status` which is reference to the `top_status` defined in base test. This reference is created by using `uvm_config_db SET/GET` functions inside the top environment file. This `top_status` is used by clock data recovery logic for sampling the appropriate data from incoming serial data stream.

5.3.3 Configuring the Passive Serial Monitors (TX Monitor is Connected to TX DUT or RX Monitor is Connected to RX DUT)

Serial monitors probe DP or DN lines on serial interface and create M-PHY transactions. Serial monitor also requires control transactions to update its status object. These control transactions are not visible on serial interface. They can be accessed by passing the control SAP transactions from the serial transactor to the serial monitor using callbacks.

Serial monitor requires a serial UI clock, the following use models are used to access the serial UI clock:

- ❖ Serial monitor does not have its own CDR logic. It shares the serial UI clock from the serial transactor.
- ❖ To read the data from the serial interface, passive serial monitor enables its own CDR logic present inside `svt_mphy_serial_tx_if` or `svt_mphy_serial_rx_if` file.

The first use model is valid for the following configuration:

- ❖ TX serial DUT is connected to RX serial VIP and only passive serial RX or active serial RX monitors are connected
- ❖ TX serial VIP is connected to RX serial DUT and only passive serial TX or active serial TX monitors are connected

The second use model is valid for all the configurations defined in the first use model and it is also applicable for the following configurations:

- ❖ TX serial DUT is connected to RX serial VIP and passive serial TX monitor is also connected
- ❖ TX serial VIP is connected to RX serial DUT and passive serial RX monitor is also connected
- ❖ TX serial DUT is connected to RX serial DUT and passive serial TX and passive serial RX monitors are connected on serial interface

For first use model, refer section 5.3.1.3 and 5.3.2.4 “Configuring the VIP as passive for Serial interface”.

For second use model, perform the following steps:

1. Create an object of serial interface in `top.sv` (say `passive_mtx_serial_if`)
2. Connect DP or DN of `passive_mtx_serial_if` with the DP or DN of DUT interface
3. Pass `passive_mtx_serial_if` object to `mphy_lm_env` using `uvm_config_db` set/get method
4. Create an object of passive serial agent in `mphy_lm_env` (say `psv_ser_agent`)
5. Pass `passive_mtx_serial_if` object to the configuration object (say `shared_cfg.passive_serial_cfg`) of passive serial agent

```
this.shared_cfg.passive_serial_cfg.tx_cfg.tx_serial_int_if[i] =  
passive_mtx_serial_if[i];
```

6. Set `is_active` field to 0 and `enable_monitor` to 1 in `passive_serial_cfg`

```
/** Set is_active to 0 to make agent as passive */
```

```
this.shared_cfg.passive_serial_cfg.is_active = 0;
```

```
/** Set enable_monitor to 1 to enable monitor */
```

```
this.shared_cfg.passive_serial_cfg.enable_monitor = 1;
```

7. Set `enable_passive_serial_monitor` to 1

```
this.shared_cfg.passive_serial_cfg.tx_cfg.enable_passive_serial_mon = 1; // enables the  
CDR logic inside svt_mphy_serial_tx_if for passive TX serial monitor
```

```
this.shared_cfg.passive_serial_cfg.rx_cfg.enable_passive_serial_mon = 1; // enables the  
CDR logic inside svt_mphy_serial_rx_if for passive RX serial monitor
```

**Note**

This variable enables the CDR logic inside svt_mphy_serial_tx_if and svt_mphy_serial_rx_if interface. By default enable_passive_serial_mon is set to '0'.

5.4 Connecting the Interface to DUT

To connect the M-PHY Interface to the DUT, there exist two scenarios:

- ❖ [Connecting the Interface to DUT Without Lane Management](#)
- ❖ [Connecting Lane Management Interface to DUT](#)

5.4.1 Connecting the Interface to DUT Without Lane Management

The M-PHY interface is connected to the DUT as shown in the following code snippet:

```
DUT_mphy_serial_rx_if  SERIAL_INTF_MRX
svt_mphy_serial_tx_if  SERIAL_INTF_MTX

// Mapping the Control Input signals of the SERIAL RX Interface with local wires
assign SERIAL_INTF_MRX.rx_dp      = mrx_serial_rx_dp;
assign SERIAL_INTF_MRX.rx_dn      = mrx_serial_rx_dn;

// Mapping the Control Output Signals of the SERIAL TX Interface with local wires
assign mtz_serial_tx_dp          = SERIAL_INTF_MTX.tx_dp;
assign mtz_serial_tx_dn          = SERIAL_INTF_MTX.tx_dn;

/** Simple Wire Pass-through connections */
assign mrx_serial_rx_dp = mtz_serial_tx_dp ;
assign mrx_serial_rx_dn = mtz_serial_tx_dn ;
```

5.4.2 Connecting Lane Management Interface to DUT

The configuration file contains an array of interface. Each element of this interface represents a lane in sublink.

```
/**
 * Signal level interface
 */
MPHY_SERIAL_TX_IF tx_serial_int_if[];
```

Depending on the maximum number of lanes for which lane management layer is configured using the macros `SVT_MPHY_MPORT_LM_TX_MAX_LANE` and `SVT_MPHY_MPORT_LM_RX_MAX_LANE`, the appropriate elements of the interface array are connected. The following example shows the connection for a lane management master with two lanes:

```
//Connect the virtual interface of the environment to master agent interface
//for agent with two lanes
master_cfg.tx_serial_int_if[0] = test_top.master_mtx_serial_if_0;
master_cfg.tx_serial_int_if[1] = test_top.master_mtx_serial_if_1;

//Connect the virtual interface of the environment to the physical interface present in
//top file
svt_mphy_serial_tx_if      master_mtx_serial_if_0();
svt_mphy_serial_tx_if      master_mtx_serial_if_1();

// DUT Interface
DUT_mphy_serial_rx_if      slave_mrx_serial_if_0();
DUT_mphy_serial_rx_if      slave_mrx_serial_if_1();
```

```
// TX Sublink connection
assign slave_mrx_serial_if_0.mphy_rx_if.rx_dp = master_mtx_serial_if_0.tx_dp;
assign slave_mrx_serial_if_0.mphy_rx_if.rx_dn = master_mtx_serial_if_0.tx_dn;
assign slave_mrx_serial_if_1.mphy_rx_if.rx_dp = master_mtx_serial_if_1.tx_dp;
assign slave_mrx_serial_if_1.mphy_rx_if.rx_dn = master_mtx_serial_if_1.tx_dn;
```

5.5 Generating Constrained Random Stimulus

The constrained random generation uses the built-in `randomize()` method that all SystemVerilog objects possess. The most common use of random generation is to produce a series of random transactions, which follow a set of applied constraints. The M-PHY SV-UVM basic example shows a UVM sequencer that is used to drive directed and random M-PHY sequences.

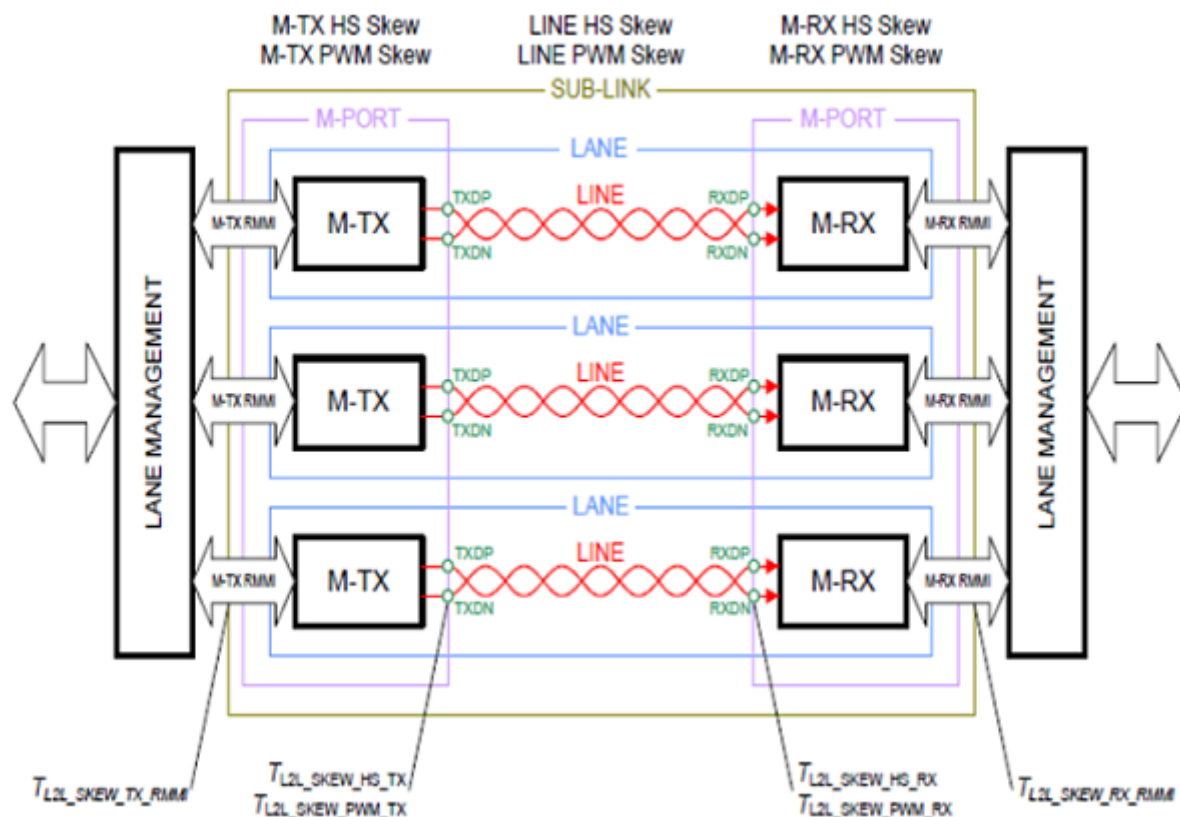
5.6 Controlling the Test

All the components and sequences in UVM use a common mechanism of raising and dropping the objections to control the execution of simulation. The simulation completes when all the raised objections have been dropped.

5.7 Lane-to-Lane Skew

Lane-to-Lane Skew is introduced in Lane Management for M-PHY Verification IP. Skew is defined as the time difference between SymbolClk at the MTX RMMIs in a SUB-LINK.

[Figure 5-1](#) shows the graphical representation of skew at the M-PHY Interface, as per *MIPI Alliance Specification for MPHY Version 4.1 - 01 February 2016*.

Figure 5-1 Skew at M-PHY Interface

5.7.1 Skew Insertion

A configuration field is defined in `svt_mphy_configuration` class for skew insertion. This field inserts skew on the interface whenever a positive value is assigned to it.

Variable Name	Variable Type	Description
<code>t_skew</code>	int	Inserts skew on the interface. For Serial -- In the form of UI For RMMI -- In the form of SI

The variable `t_skew` inserts a delay for the assigned number of clocks just before the start of PREPARE for every burst.

At the lane management layer, a dynamic array is defined in `svt_mphy_mport_lm_configuration` class.

Variable Name	Variable Type	Description
<code>lane_to_lane_skew[]</code>	int	This dynamic array has skew values for all active lanes on index basis. For example, at index 0 -- skew for 0th lane, at index 1 -- skew for 1st lane and so on.

Both positive and negative skew are supported. The skew with least value is considered as a reference for all the lanes. From lane management agent, the skew from `lane_to_lane_skew[]` is assigned to the configuration of each lane of M-PHY on index basis.

5.7.2 Use Case with Mport LM Agent on SERIAL and RMMI Interface

For skew insertion, a dynamic array is created with size less than or equal to the number of active lanes in a test case. Skews for all lanes should be specified in a dynamic array based on the index.

If skew is needed only on 2 lanes (0, 1) out of 4 active lanes, then dynamic array with size 2 needs to be created. The remaining two lanes will have zero skew by default.

For both SERIAL and RMMI interfaces, skew value should be given in the form of UI only. In case of RMMI interface, any skew value from 1 to 10 UI will be interpreted as 1 SI, a skew value between 11 to 20 will be interpreted as 2 SI and so on. In the case of SERIAL interface, skew will be inserted in UI form directly.

Use Case with All Positive Skew

If the active lanes are 4, then a dynamic array of size 4 is created and skew for individual lanes is assigned on each index.

The following example has positive skew for all lanes. Lane with least skew, that is, lane [0] will be the reference for all other lanes. At run time, PREPARE sequence for lane [0] will start, first followed by lane [3], then lane [2] and lane [1].

```
shared_cfg.master_cfg.tx_cfg.lane_to_lane_skew = new [4];
shared_cfg.master_cfg.tx_cfg.lane_to_lane_skew[0] = 0;
shared_cfg.master_cfg.tx_cfg.lane_to_lane_skew[1] = 4;
shared_cfg.master_cfg.tx_cfg.lane_to_lane_skew[2] = 2;
shared_cfg.master_cfg.tx_cfg.lane_to_lane_skew[3] = 1;
```

Use Case with All Negative Skew

If the active lanes are 4 and the user wants to insert a negative skew on lane [0] and lane [2], then a dynamic array of size 3 is created and a skew for individual lanes is assigned on each index.

The following example has negative skew for lane [1] and lane [2]. The lane with highest negative skew will act as a reference for all other lanes. PREPARE sequence for lane [2] will start first followed by lane [0]. Lane [1] and lane [3] will also start at the same time.

```
shared_cfg.master_cfg.tx_cfg.lane_to_lane_skew = new [3];
shared_cfg.master_cfg.tx_cfg.lane_to_lane_skew[0] = -1;
shared_cfg.master_cfg.tx_cfg.lane_to_lane_skew[1] = 0;
shared_cfg.master_cfg.tx_cfg.lane_to_lane_skew[2] = -2;
```

Use Case with Both Positive and Negative Skew

If the user wants to insert both positive and negative skew on specific lanes, then the dynamic array will have both types of skew.

The lane with highest negative skew will be a reference for all other lanes. In the above case, lane [4] will start first.

If active lanes are 5:

```
shared_cfg.master_cfg.tx_cfg.lane_to_lane_skew = new [5];
shared_cfg.master_cfg.tx_cfg.lane_to_lane_skew[0] = 0;
shared_cfg.master_cfg.tx_cfg.lane_to_lane_skew[1] = -5;
```



```
shared_cfg.master_cfg.tx_cfg.lane_to_lane_skew[2] = 2;  
shared_cfg.master_cfg.tx_cfg.lane_to_lane_skew[3] = -1;  
shared_cfg.master_cfg.tx_cfg.lane_to_lane_skew[4] = -1;
```

Use Case for No Skew

For no skew case, a dynamic array is not required. By default, all the lanes will start at the same time.

5.7.3 Inserting Random Skew

Two new configuration fields are added in `svt_mphy_configuration` class for random skew insertion.

Table 5-1 New Configuration Fields for Random Skew

Variable Name	Variable Type	Description
<code>min_skew</code>	rand int	Minimum skew value inserted in MTX in the form of number of clocks. For Serial -- In the form of UI For RMMI -- In the form of SI
<code>max_skew</code>	rand int	Maximum skew value inserted in MTX in the form of number of clocks. For Serial -- In the form of UI For RMMI -- In the form of SI

Constraint for random skew is as follows, `min_skew` shall be less than `max_skew`.

5.7.3.1 Use Case to Insert Random Skew

Set `min_skew` and `max_skew` configuration parameters to different values in `mphy_configuration` in `shared_cfg`.

For example,

```
mphy_cfg.min_skew = 0;  
mphy_cfg.max_skew = 50;
```

Random skew between `min_skew` and `max_skew` are calculated before each burst for the corresponding lane to achieve the variable skew.

Any lane can start the `PREPARE` period first, the skew varies for every `BURST`.



Note

To maintain consistency, use only one use model between `lane_to_lane_skew[]/t_skew` and `min_skew/max_skew`. If `min_skew/max_skew` fields are used and you have also assigned a value to `lane_to_lane_skew[]/t_skew` then it is ignored and only `min_skew/max_skew` is used for skew insertion in that test.

5.8 Exception Support

The M-PHY VIP supports error injection by exceptions. An Exception List Factory can be assigned to the transactor, which uses the factory to create a list of exceptions for transactions. Any transaction that is put()

into a port will have its exception list replaced by one generated by the TX transactor's Exception List Factory.

The TX transactor supports test bench-guided error injection for input transactions, which are associated with an individual transaction and are applied to the transaction as a part of transmission.

The test bench can generate the exceptions directly or use automatically generated exceptions by the transactor.

Transactions can have one or more exceptions associated with them, although they don't need to have exceptions. The exceptions associated with a transaction are stored in *svt_mphy_transaction_exception_list*, which is made up of individual exceptions of type *svt_mphy_transaction_exception*.

5.8.1 Exception Class Hierarchy

The M-PHY VIP provides classes for describing an individual exception, and each class instance can be populated manually or via randomization and inserted into an exception list to be associated with a transaction. The user can create exceptions for transactions.

The following classes are used to define exceptions:

❖ Exception Class

The exception class, for example, *svt_mphy_transaction_exception* implements a class for describing an individual exception. The *svt_mphy_transaction_exception* instance can be populated manually or via randomization and inserted into an exception list to be associated with a transaction.

❖ Exception List Class

The exception list class, for example, *svt_mphy_transaction_exception_list* implements a class for maintaining a list of transaction exceptions. The *svt_mphy_transaction_exception_list* instance can be populated manually or via randomization and then associated with a transaction.

5.8.2 Exception Architecture

Two new classes *svt_mphy_mport_lm_transaction_exception* and *svt_mphy_transaction_exception* are defined for Lane Management (LM) Layer and M-PHY Layer respectively which are extended from the *svt_exception* class.

Two new classes *svt_mphy_mport_lm_transaction_exception_list* and *svt_mphy_transaction_exception_list* are defined for LM layer and M-PHY layer respectively which are extended from *svt_exception_list* class.

All the functions, like, *get_*_val* and *has_** are defined in “*_exception” and “*_exception_list” classes to introduce the exceptions at LM and M-PHY level.

Exception firing on LM layer and M-PHY layer are independent of each other and there is no collision on LM layer and M-PHY layer exceptions.

The Lane Management layer inserts exceptions on LM transaction. The LM transactor splits LM transaction in M-PHY transactions. Then, the M-PHY layer corrupts the M-PHY transactions using M-PHY exceptions.

Figure 5-2 shows the LM MPORT architecture for M-PHY.

The diagram illustrates a 3-lane L1.5 LM layer. It features three horizontal lanes, each containing an M-TX (Master Transmitter) and an M-RX (Master Receiver) block. The lanes are connected by SUB-LINK lines (yellow) and LANE lines (blue). The LANE lines are labeled with PINs, TXDP, TXDN, RXDP, RXDN, and PINs. The M-TX and M-RX blocks are connected to the LANE lines via M-TXP and M-RXP ports. The entire structure is flanked by LANE MANAGEMENT blocks (grey) on both sides, which are connected to the LANE lines. The LANE MANAGEMENT blocks are connected to the LANE lines via LANE lines (blue) and SUB-LINK lines (yellow). The LANE MANAGEMENT blocks are also connected to the LANE lines via LANE lines (blue) and SUB-LINK lines (yellow). The LANE MANAGEMENT blocks are connected to the LANE lines via LANE lines (blue) and SUB-LINK lines (yellow). The LANE MANAGEMENT blocks are connected to the LANE lines via LANE lines (blue) and SUB-LINK lines (yellow).

This section defines the architecture development flow from top to bottom, that is from the test case level to the common files.

- ❖ Test Case Code
- ❖ M-PHY/Lane Management Transactor Code
- ❖ Exceptions Messaging

- ❖ User_Test
 - ◆ M-PHY exception list:
 - ❖ A new exception_list is extended from svt_mphy_transaction_exception_list and passed to M-PHY transactors.
 - ❖ This exception list enables or disables the M-PHY exceptions by constraining the weights on M-PHY exceptions.
 - ❖ This exception list is also constrained to chose a specific type of transactions.
 - ◆ LM layer exception list:
 - ❖ A new exception_list is extended from svt_mphy_mport_lm_transaction_exception_list and passed to LM transactor.

- ❖ This exception list enables or disables the LM exceptions by constraining the weights on LM exceptions.
- ❖ This exception list is also constrained to chose a specific payload index or a specific LM transaction.

Following is the code snippet to extend the M-PHY exception list:

```
class transaction_rsvd_ctrl_codes_error_exception_list extends
svt_mphy_transaction_exception_list;
    svt_mphy_transaction_exception xact_exc = new("xact_exc");

function new(string name = "transaction_rsvd_ctrl_codes_error_exception_list",
svt_mphy_transaction_exception xact_exc = null);
    super.new(name,xact_exc);
    xact_exc = this.xact_exc;
    xact_exc.CTRL_CODES_RSVD_ERROR_wt = 1;
    xact_exc.RD_ERROR_wt = 0;
    xact_exc.EN_3B4B_5B6B_ERROR_wt = 0;
    this.randomized_exception = xact_exc;
endfunction : new
endclass
```



Note The code to pass the extended M-PHY exception list to M-PHY transactors is shown in [Use Case 6](#).

5.8.3.2 M-PHY/Lane Management Transactor Code

- ❖ The object to the exception list in Lane Management transactor is set by the user.
- ❖ In case of Lane Management layer, all the Lane Management transactions received from the sequencer and in case of M-PHY layer, M-PHY transactions received from LM transactors, are passed to the *randomize_transaction_exception_list* function. Input parameter to this function is *lm_xact* / *xact* and *exception_list*. This function randomizes the list of exceptions depending upon user constraints and exception list weights.
- ❖ If the exception list is not constraint in the test case, then a random exception is picked.
- ❖ If the triggered exception is of LM layer, then LM transactors trigger the exception using *get*_val* function on LM transactions.
- ❖ If the triggered exception is of M-PHY layer, then M-PHY transactors trigger the exception using *get*_val* function.

5.8.3.3 Exceptions Messaging

- ❖ When the exception list is non-null and the transactor issues an exception due to randomization:
The transactor issues a note stating “CTRL_CODES_RSVD_ERROR exception triggered on M-PHY transaction”.
- ❖ When the exception list is non-null and the transactor is not able to insert an exception due to randomization:
The transactor issues a note stating “CTRL_CODES_RSVD_ERROR exception not triggered on M-PHY transaction”.

All the error_checks fired in exception test cases are demoted to “note”. At the end of simulation, final result provides the number of checks demoted along with the number of UVM_ERROR and UVM_WARNING.

5.8.4 M-PHY Exceptions List

Table 5-2 shows the list of M-PHY exceptions provided in this release, with reference to *MIPI Alliance Specification for MPHY Version 4.1 - 01 December 2016*.

Table 5-2 M-PHY Exceptions List

Exception Name	Description	Checks Implemented
CTRL_CODE_RSVD_ERROR	<p>This is a M-PHY Data and LM Data Exception.</p> <p>As per Table 5-5 Control Symbols, K28.0, K28.2, K28.4, K28.7 and K23.7 are reserved control symbols and shall not be used in normal operation. This exception replaces the CTRL symbols (Markers and filler) with reserved symbol.</p> <p>Valid For: Serial and RMMI interfaces</p> <p>Spec. Reference: Line no.151</p>	res_error_check
RD_ERROR	<p>This is a M-PHY Data Exception.</p> <p>Running disparity is applied for DC balance in the transmitted serial bit stream. The M-TX shall follow the RD rules for a BURST, from the first SYNC symbol up to, and including, the last 8b10b symbol preceding TAIL-OF-BURST. RD bit is wrongly set in the middle of burst to have RD error exception.</p> <p>Valid For: Serial and RMMI interface</p> <p>Spec. Reference: Line no.150, 153</p>	rd_error_check in Serial interface, decoded_symbol_error_check in RMMI interface for RMMI_REMOTE, For RMMI Local, transaction object field "rd_error" is set to 1. This transaction object can be used by upper layer to know about exception insertion.

Table 5-2 M-PHY Exceptions List (Continued)

Exception Name	Description	Checks Implemented
CODE_3B4B_5B6B_ERROR	<p>This is a M-PHY Data Exception.</p> <p>Each byte is coded following the 3b4b and 5b6b sub block coding rules. In this exception, 3b4b and 5b6b coding rules are violated.</p> <p>Valid For: Serial interface</p> <p>Spec. Reference: Line no.153, 154</p>	<p>sub_block_3b4b_error_check in Serial interface,</p> <p>sub_block_5b6b_error_check in Serial interface,</p> <p>rd_error_check in Serial interface,</p> <p>decoded_symbol_error_check in RMMI interface for RMMI_REMOTE,</p> <p>For RMMI Local, transaction object fields</p> <p>"en_3b4b_error" & "en_5b6b_error" are set to 1. This transaction object can be used by upper layer to know about exception insertion.</p>
PREPARE_LEN_ERROR	<p>This is a M-PHY Protocol Exception.</p> <p>Prepare length for LS and HS mode is calculated as per Table 6. Exception generates prepare length violation.</p> <p>Valid For: Serial and RMMI interfaces</p> <p>Spec. Reference: Table 6</p>	<p>prepare_length_violation_check in Serial interface,</p> <p>invalid_symbol_in_prepare_check in Serial interface,</p> <p>invalid_fsm_state_transition_check in RMMI interface</p>
SYNC_LEN_ERROR	<p>This is a M-PHY Protocol Exception.</p> <p>Sync length for LS G6-G7 and HS mode is calculated as per Table 6. Exception generates sync length violation.</p> <p>Valid For: Serial and RMMI interfaces</p> <p>Spec. Reference: Table 6</p>	<p>sync_length_violation_check</p>

Table 5-2 M-PHY Exceptions List (Continued)

Exception Name	Description	Checks Implemented
NO_SYNC_ERROR	<p>This is a M-PHY Protocol Exception.</p> <p>For HS-MODE, the PREPARE sub-state period shall be followed by a SYNC sequence. Exception blocks the internal or the external SYNC symbols.</p> <p>Valid For: Serial and RMMI interfaces</p> <p>Spec. Reference: Line number 227</p>	<p>sync_not_rcvd_after_prepare_in_hs_check in Serial and RMMI interface,</p> <p>invalid_fsm_state_transition_check in Serial interface,</p> <p>sync_length_violation_check in RMMI interface,</p> <p>marker0_check in Serial and RMMI interface</p>
ILLEGAL_SYNC_SYMBOL_ERROR	<p>This is a M-PHY Protocol Exception.</p> <p>The SYNC sequence shall be a serialized subset of 8b10b data symbols with a high edge density for fast synchronization. Exception will transmit the invalid SYNC symbols.</p> <p>Valid For: Serial and RMMI interfaces (not for RMMI Remote)</p> <p>Spec. Reference: Line number 227</p>	<p>invalid_sync_symbol_check in Serial and RMMI interface,</p> <p>sync_not_rcvd_after_prepare_in_hs_check in Serial and RMMI interface,</p> <p>marker0_check in Serial and RMMI interface,</p> <p>sync_length_violation_check in RMMI interface</p>
INVALID_PWM_BURST_CLOSURE_CAPABILITY_ERROR	<p>This is a M-PHY Protocol Exception.</p> <p>The attribute value shall not violate range of values of applicable capabilities, if any, for that attribute. This exception will violate the capability range of values.</p> <p>Valid For: Serial interface</p> <p>Spec. Reference: Line number 839</p>	<p>invalid_cap_range_check</p>
FRAME_SEPARATOR_ON_ALL_LANES_ERROR	<p>This is a LM Protocol Exception.</p> <p>DEMARKETERS on each lane of a sub-link must be present. This exception will miss the uniform alignment of DEMARCATERs. For example, Lane 1 will miss demarcater out of 4 active lanes.</p> <p>Valid For: Serial and RMMI interfaces</p> <p>Spec. Reference: N/A</p>	<p>frame_separator_on_all_lanes_check, Data_Integrity_Failure (uvm_error will be reported)</p>
DATA_INTEGRITY_ERROR	<p>This is a LM Protocol Exception.</p> <p>The data is alternatively transferred between the lanes until a DEMARCATER or Marker2 comes. This exception will un-align the data. For example, data will go as D0 D1 D3 D2 D4 D5 D6.</p> <p>Valid For: Serial and RMMI interfaces</p> <p>Spec. Reference: N/A</p>	<p>Data_Integrity_Failure (uvm_error will be reported)</p>

Table 5-3 shows the list of M-PHY RMMI signal level exceptions provided in the M-PHY VIP, with reference to *MIPI Alliance Specification for M-PHY Version 4.1 - 01 December 2016*.

Table 5-3 M-PHY Signal Level Exceptions

Exception Name	Description	Checks Implemented
DATANCTRL_SIGNAL_ERROR	<p>This is a M-PHY data signal exception.</p> <p>This exception is enabled under the following scenarios:</p> <ul style="list-style-type: none"> When TX/RX Burst signal is set to 0, this exception will set all bits of DataNCtrl to "1". When a data symbol is transmitted/received, this exception will set all bits of TX/RX DataNCtrl to "1". When 10b8b decoding is bypassed, this exception will set all bits of TX/RX DataNCtrl to "1". <p>Valid For: RMMI interface Spec. Reference: Table 58</p>	invalid_datanctrl_signal_val_check
CFGUPDT_SIGNAL_ERROR	<p>This is a M-PHY control signal exception.</p> <p>This exception is enabled under the following scenarios:</p> <ul style="list-style-type: none"> For more than one TX/RX CfgClk cycle, this exception will set TX/RX CfgUpdt to "1". When TX/RX LineReset is set to "1", this exception will set TX/RX CfgUpdt to "1". When TX/RX RESET is set to "1", this exception will set TX/RX CfgUpdt to "1". <p>Valid For: RMMI interface Spec. Reference: Table 57</p>	invalid_cfgupdt_signal_val_check

For information on all other data and control signal exceptions, see `error_kind_enum` in `svt_mphy_transaction_exception` class in M-PHY Class Reference HTML documentation.

Table 5-4 lists the M-PHY error sequences.

Table 5-4 Error Sequences

Exception Name	Description	Checks Implemented
INVALID_CONFIGURATION_ATTRIBUTE_RANGE_SEQUENCE	<p>This is a M-PHY Protocol Exception.</p> <p>Supported value range for an attribute shall not exceed the range of values specified in the "Range" column of that attribute. Check for tx_mode, tx_hsrateseries_tx_hsgear, tx_amplitude, tx_hs_sync_len, rx_mode, rx_hsrateseries, rx_hsgear, rx_bypass_8b10b_enable, rx_enter_hibern8, rx_hs_terminate_enable, rx_ls_terminate_enable. This error sequence will violate the range of configuration attributes values.</p> <p>Valid For: RMMI interface Spec. Reference: Line no. 851, Table 49, 53.</p>	invalid_attr_val_check

Table 5-4 Error Sequences (Continued)

Exception Name	Description	Checks Implemented
SYNC_DATA_MK0_SEQUENCE	<p>This is a M-PHY Protocol Exception.</p> <p>In HS-BURST or PWM-BURST for PWM-G6 or PWM-G7, the SYNC sequence is followed by payload that shall start with MARKER0. This error sequence will add a DATA symbol between SYNC and MK0.</p> <p>Valid For: Serial and RMMI interfaces</p> <p>Spec. Reference: Line no. 245</p>	invalid_sync_symbol_check, sync_not_rcvd_after_prepare_in_hs_check, marker0_check

5.8.5 M-PHY Exceptions Use Cases

This section explains the use model to handle M-PHY exceptions with the help of different use cases.

Use Case 1

Have weights between good transactions and bad transactions. Weights are defined for GOOD transactions and no exception is fired on it.

Extend *svt_mphy_exception_list* in the test case and modify the weights between good transactions and bad transactions. Good transactions are fired when NO_OP exception is hit.

In the following code snippet case, NO_OP exception has 50% weight age. The xact_exc is an object to *svt_mphy_exception* class.

```
xact_exc.NO_OP_wt = 5;  
xact_exc.CTRL_CODES_RSVD_ERROR_wt = 3;  
xact_exc.RD_ERROR_wt = 2;
```

This use case will fire exceptions randomly on a transaction.

More than 1 exception can also be fired in a transaction. To fire more than 1 exception on a specific transaction or a specific burst, refer [Use Case 5](#).

Use Case 2

If user has specified num_exception = 1 to insert exception on all the transactions and also does not want any BAD transaction, then weights are defined for GOOD transactions. No exception will be fired.

Extend *svt_mphy_exception_list* class in the test case and define the weights as '0' for the exceptions not to be fired. In the following example code, transactions will not have RD error, 3B4B error, 5B6B error and reserved control code error. The xact_exc is an object to *svt_mphy_exception* class.

```
xact_exc.NO_OP_wt = 1;  
xact_exc.CTRL_CODES_RSVD_ERROR_wt = 0;  
xact_exc.RD_ERROR_wt = 0;  
xact_exc.EN_3B4B_5B6B_ERROR_wt = 0;
```

Even if "num_exception" is 1, no exception would be fired.

Use Case 3

If the user wants to fire an exception on all the transactions of a specific kind, say, all the control codes, then constraint num_exception variable to one for that exception.

Extend *svt_mphy_exception_list* in the test case and enable the exceptions to be fired. Constraint “num_exception” variable to ‘1’ which is defined in *svt_exception_list* class. This exception will replace all the control codes with the reserved control codes.

```
xact_exc.NO_OP_wt = 0;
xact_exc.CTRL_CODES_RSVD_ERROR_wt = 1;
xact_exc.RD_ERROR_wt = 0;
xact_exc.EN_3B4B_5B6B_ERROR_wt = 0;
```

```
constraint test_constraints1
{
  num_exceptions == 1;
}
```

If this constraint is not added, then at the time when every transaction exception list is randomized, “num_exception” field is also randomized to 1/0. If num_exception is ‘1’, an exception is fired for that transaction. If it is ‘0’, the exception is not fired.

Use Case 4

Specify a specific type of transaction example, say, Marker0 symbol transaction where user wants or does not want to fire an exception.

Extend *svt_mphy_exception_list* class in the test case and enable the exceptions to be fired. Constraint “num_exception” variable to ‘1’ for that specific transaction. This exception will replace all the control code symbols except Marker0 with the reserved control codes.

```
xact_exc.NO_OP_wt = 0;
xact_exc.CTRL_CODES_RSVD_ERROR_wt = 1;
xact_exc.RD_ERROR_wt = 0;
xact_exc.EN_3B4B_5B6B_ERROR_wt = 0;
xact_exc.SYNC_LEN_ERROR_wt = 0;
xact_exc.PREPARE_LEN_ERROR_wt = 0;

constraint test_constraints1 {
  ((xact_exc.xact.data_n_ctrl == 1) && (xact_exc.xact.marker_number ==
`SVT_MPHY_CTRL_MARKER0)) -> (num_exceptions == 0);
}
```

Use Case 5

Specify more than two exceptions to fire on a specific transaction or a specific burst.

Extend *svt_mphy_exception_list* class in the test case and enable the exceptions to be fired. Constraint num_exception variable to one for those specific transaction. This exception will fire sync length violation and prepare length violation exception.

```
xact_exc.NO_OP_wt = 0;
xact_exc.CTRL_CODES_RSVD_ERROR_wt = 0;
xact_exc.RD_ERROR_wt = 0;
xact_exc.EN_3B4B_5B6B_ERROR_wt = 0;
xact_exc.SYNC_LEN_ERROR_wt = 1;
xact_exc.PREPARE_LEN_ERROR_wt = 1;
```

```
constraint test_constraints1 {
```

```

        ((xact_exc.xact.primitive_name == svt_mphy_transaction::PREPARE) &&
(xact_exc.xact.primitive_type == svt_mphy_transaction::REQUEST)) -> (num_exceptions ==
1);
    }

constraint test_constraints2 {
    ((xact_exc.xact.primitive_name == svt_mphy_transaction::SYNC) &&
(xact_exc.xact.primitive_type == svt_mphy_transaction::REQUEST)) -> (num_exceptions ==
1);
}

```

This use case will fire the exceptions on all PREPARE REQUEST and SYNC REQUEST SAP M-PHY transactions.

Use Case 6

Specify any of the above mentioned five use cases to fire exception randomly on all the M-PHY lanes or a specific M-PHY lane connected to LM transactor.

Extend *svt_mphy_exception_list* class in the test case and enable the exceptions to be fired. In this case, RD error is inserted in M-PHY transactions.

```

xact_exc.NO_OP_wt = 0;
xact_exc.CTRL_CODES_RSVD_ERROR_wt = 0;
xact_exc.RD_ERROR_wt = 1;
xact_exc.EN_3B4B_5B6B_ERROR_wt = 0;

```

Pass the extended exception list to all the M-PHY transactors to have exception on all the M-PHY lanes (assuming 2 M-PHY lanes are active) as mentioned in the following code snippet.

```

svt_config_object_db#(svt_mphy_transaction_exception_list)::set(this,"env.tx_agent.mphy_tx[0]","tx_transaction_exception_list",transaction_exc_list);

svt_config_object_db#(svt_mphy_transaction_exception_list)::set(this,"env.tx_agent.mphy_tx[1]","tx_transaction_exception_list",transaction_exc_list);

```

Use Case 7

User can fire two exceptions simultaneously on the same payload byte.

To issue such exception, user has to perform the following steps:

1. Enable the weights

```

function new(string name = "transaction_rd_error_exception_list",
svt_mphy_transaction_exception xact_exc = null);
    super.new(name,xact_exc);
    xact_exc = this.xact_exc;
    xact_exc.set_constraint_weights(0);
    xact_exc.RD_ERROR_wt = 1;    // enable RD error exception
    xact_exc.EN_3B4B_5B6B_ERROR_wt = 1; // enable 3b4b_5b6b error exception
    EXCEPTION_LIST_SHORT_wt = 1; // enable this weight to fire 2 exceptions
simultaneously
    this.randomized_exception = xact_exc;
endfunction : new

```

2. Enable number of exceptions == 2

```

constraint test_constraints2 {
    (xact_exc.xact.data_n_ctrl == 0) -> (num_exceptions == 2);
}

```

3. Populate type of exception in exception list

```

constraint test_constraints3 {
    foreach (typed_exceptions[i]) {
        if(i == 0) {
            typed_exceptions[i].error_kind ==
            svt_mphy_transaction_exception::EN_3B4B_5B6B_ERROR;
            typed_exceptions[i].mask_or_value == 10'b1000001010;
        }
        else if (i == 1) {
            typed_exceptions[i].error_kind == svt_mphy_transaction_exception::RD_ERROR;
        }
    }
}

```

4. Define the following macro to two in top.sv file

```

define SVT_MPHY_TRANSACTION_EXCEPTION_LIST_MAX_NUM_EXCEPTIONS 2

```

5.8.6 Lane Management Exceptions Use Cases

This section explains the use model to handle Lane Management exceptions with the help of different use cases.



Note First 5 Use cases for M-PHY exceptions mentioned in [M-PHY Exceptions Use Cases](#) are also valid for LM exceptions.

Following is the summary of the first five M-PHY use cases:

- ❖ Weights are defined for GOOD/BAD LM transactions.
- ❖ Weights are defined between all LM exceptions.
- ❖ A specific kind of exception can be fired on all LM transactions by asserting “num_exception = 1” defined in lm_exception_list. For example, all demarcaters are replaced by reserved control codes.
- ❖ Constraining the LM transaction such that “num_exception = 1”, for a specific field of LM transaction. For example, a demarcater MK5 is only replaced by reserved control codes.
- ❖ More than two exceptions are fired on the same LM transaction.

Use Case 1

Insert LM exception on a specific M-PHY lane

Define a variable, say, exception_lane_index in svt_mphy_lm_mport_transaction.sv class. Place a valid constraint on exception_lane_index to have a value less than cfg.active_lanes in the sequence.

Pass this exception_lane_index value to LM transaction by constraining this field in the sequence. A variable lane_index is also required in LM transaction class.

Exception valid for this use case is:

- ❖ FRAME_SEPARATOR_ON_ALL_LANES_ERROR

5.9 Inserting PPM Drift and Clock Recovery

The `svt_mphy_configuration` class defines a `mphy_ppm_drift` variable which can have a valid range of PPM drift between -2000 PPM to +2000 PPM of UI. By default, the value of `mphy_ppm_drift` is 0. Therefore, PPM drift is disabled in basic testcases.

To enable PPM drift, define a value between -2000 to +2000 to `mphy_ppm_drift` variable in case `svt_mphy_configuration` class is not randomized. A positive value inserts a positive PPM drift on every UI of serial data and vice versa. To check the drifted data on serial lines, refer `tx_dp` signal on serial interface.

Clock data recovery (CDR) logic is defined at the RX path of each lane for the Serial interface. This logic generates an output clock which gets aligned to incoming serial data stream. In HS mode, CDR logic aligns the output clock at the center of HS bit duration. In case of PPM drift, when output clock edges come closer to serial HS bit transitions, the output clock corrects itself to the centre of HS bit transitions using CDR. In LS PWM mode, negative edge of `RX_DP` represents the start of UI and a positive edge of `RX_DP` samples the serial data stream. Therefore, in LS PWM mode, output clock is extracted from LS PWM data stream. In case of PPM drift, extracted output clock itself gets aligned to the positive edge and negative edge of LS PWM serial data stream.



Note

Since CDR logic works on data transitions and there is no transitions in PREPARE of BURST, so this logic is unable to correct the PPM drift in the PREPARE of burst. A Prepare length of 10 SI or more will miss 1-2 UI of HS data at serial interface when SYNC starts. User must ensure to configure `init_hs_prepare_length` or pass a CTRL SAP with `TX_HS_PREPARE_LENGTH` less than 10 SI. This shall be taken care when `mphy_ppm_drift` is not equal to 0.

5.10 M-PHY Specification Version 2.0 Feature Support

According to *MIPI Alliance Specification for M-PHY Version 2.0 - 04 April 2012*, two of the existing capability attributes of MRX listed in Table 52 have been changed into 6 new capabilities attributes. Also, one new MRX configuration attribute has been added to Table 53 of *MIPI Alliance Specification for M-PHY Version 2.0 - 04 April 2012*.

To maintain backward incompatibility of *MIPI Alliance Specification for M-PHY Version 2.0* with *MIPI Alliance Specification for M-PHY Version 1.40.00 - 30 November 2011*, a new enum has been added in `svt_mphy_configuration` class.

Variable Name	Variable Type	Description
<code>mphy_protocol_version</code>	<pre>/** Indicates the MPHY Specification Version */ typedef enum { MPHY_VERSION_14, MPHY_VERSION_20, MPHY_VERSION_30 }mphy_protocol_version_enum;</pre>	Specifies different version of M-PHY Specifications

To ensure backward compatibility, default value of this enum is `MPHY_VERSION_14`.

This table shows the description of MRX capability attributes in *MIPI Alliance Specification for M-PHY Version 1.40.00*.

Table 5-5 MIPI Alliance Specification for M-PHY Version 1.40.00, Table 52

Attribute Name	Attribute ID	Description	FSM	Type	Bits	Range
RX_HS_SYNC_LEN GTH_Capability	0x8B	High Speed Synchronization pattern length in SI. Existence depends on: RX_HSMODE_Capability	Both	Int	B[7:6] B[5:0]	SYNC_range FINE = 0, COARSE = 1 SYNC_length ² 0 to 15
RX_HS_PREPARE_L ENGTH_Capability ³	0x8C	HS Prepare length multiplier for M-RX. Existence depends on: RX_HSMODE_Capability	Both	Int	B[3:0] ¹	0 to 15

This table shows the description of attributes as per *MIPI Alliance Specification for M-PHY Version 2.0 - 04, April 2012*.

Table 5-6 MIPI Alliance Specification for M-PHY Version 2.0, Table 5-2

Attribute Name	Attribute ID	Description	FSM	Type	Bits	Range
RX_HS_G1_SYNC_L ENGTH_Capability	0x8B	High Speed Gear 1 Synchronization pattern length in SI. Existence depends on: RX_HSMODE_Capability	Both	Int	B[7:6] B[5:0]	SYNC_range FINE = 0, COARSE = 1 SYNC_length ² 0 to 15
RX_HS_G1_PREPA RE_LENGTH_Capabi lity ³	0x8C	HS-G1 PREPARE length multiplier for M-RX. Existence depends on: RX_HSMODE_Capability	Both	Int	B[3:0] ¹	0 to 15
RX_HS_G2_SYNC_L ENGTH_Capability	0x94	High Speed Gear 2 Synchronization pattern length in SI. Existence depends on: RX_HSGEAR_Capability	Both	Int	B[7:6] B[5:0]	SYNC_range FINE = 0, COARSE = 1 SYNC_length ² 0 to 15
RX_HS_G3_SYNC_L ENGTH_Capability	0x95	High Speed Gear 3 Synchronization pattern length in SI. Existence depends on: RX_HSGEAR_Capability	Both	Int	B[7:6] B[5:0]	SYNC_range FINE = 0, COARSE = 1 SYNC_length ² 0 to 15

Attribute Name	Attribute ID	Description	FSM	Type	Bits	Range
RX_HS_G2_PREPARE_LENGTH_Capability ³	0x96	HS-G2 PREPARE length multiplier for M-RX. Existence depends on: RX_HSGEAR_Capability	Both	Int	B[3:0] ¹	0 to 15
RX_HS_G3_PREPARE_LENGTH_Capability ³	0x97	HS-G3 PREPARE length multiplier for M-RX. Existence depends on: RX_HSGEAR_Capability	Both	Int	B[3:0] ¹	0 to 15

The following is the new MRX configuration attribute added to *MIPI Alliance Specification for M-PHY Version 2.0 - 04, April 2012*.

Attribute Name	Attribute ID	Description	FSM	Type	Bits	Range
RX_Termination_Force_Enable	0xA9	Force connection of differential termination resistance, R_{DIF_RX} , to enabled state, for RX S-Parameter test purposes	Both	Bool	B[0] ¹	NO = 0, YES = 1

All new MRX capability attributes are defined in `svt_mphy_configuration` class and MRX configuration attribute is present in `svt_mphy_status` class.

For detailed explanation of new attributes, refer to M-PHY Class Reference HTML documentation.

5.10.1 Use Case to Enable M-PHY VIP for Specification Version 2.0 Features

To enable new MRX capability and configuration attributes of *MIPI Alliance Specification for M-PHY Version 2.0*, user needs to specify the protocol version in shared configuration.

For example,

```
shared_cfg.master_cfg.tx_cfg.mphy_protocol_version =
svt_mphy_configuration::MPHY_VERSION_20;
shared_cfg.master_cfg.rx_cfg.mphy_protocol_version =
svt_mphy_configuration::MPHY_VERSION_20;
shared_cfg.slave_cfg.tx_cfg.mphy_protocol_version =
svt_mphy_configuration::MPHY_VERSION_20;
shared_cfg.slave_cfg.rx_cfg.mphy_protocol_version =
svt_mphy_configuration::MPHY_VERSION_20;
```

MIPI Alliance Specification for M-PHY Version 2.0 has different high speed prepare and sync capability fields for all HS gears, so all these capabilities need to be specified in M-PHY VIP shared configuration file.

All these capabilities hold the MRX capability and MTX configuration relationship as defined in Table 65 in *MIPI Alliance Specification for M-PHY Version 2.0*.

- ❖ To run the test case in HS_G1, user needs to set these new capabilities for `rx_cfg` in shared configuration:

```
rx_cfg.hs_g1_sync_length_capability = 1;           // Range of this field is 0-15
```



```
rx_cfg.hs_g1_sync_range_capability = 0;           // Sync range can be 0 (FINE) and
                                                    1(Coarse)
rx_cfg.hs_g1_prepare_length_capability = 1;       // Range of this field is 0-15
```

- ❖ To run the test case in HS_G2, user needs to set these new capabilities for rx_cfg in shared configuration:

```
rx_cfg.hs_g2_sync_length_capability = 1;          // Range of this field is 0-15.
rx_cfg.hs_g2_sync_range_capability = 0;          // Sync range can be 0 (FINE) and
                                                    1(Coarse)
rx_cfg.hs_g2_prepare_length_capability = 1;       // Range of this field is 0-15.
```

- ❖ To run the test case in HS_G3, user needs to set these new capabilities for rx_cfg in shared configuration:

```
rx_cfg.hs_g3_sync_length_capability = 1;          // Range of this field is 0-15
rx_cfg.hs_g3_sync_range_capability = 0;          // Sync range can be 0 (FINE) and
                                                    1(Coarse)
rx_cfg.hs_g3_prepare_length_capability = 1;       // Range of this field is 0-15
```

5.11 M-PHY Specification Version 3.0 Feature Support

According to *MIPI Alliance Specification for M-PHY Version 3.0, Revision 03*, the M-PHY VIP supports the following features:

- ❖ Prepare length calculation (Section 4.7.2.2, Table 7 of *MIPI Alliance Specification for M-PHY Version 3.0, Revision 03*)

As specified in Section 8.10 in *MIPI Alliance Specification for M-PHY Version 3.0, Revision 03*, a relation has to be maintained between TX configuration and RX capabilities. This relation has to be held for the reduced value of prepare_length as well.

- ❖ Changes in Hibern8 and Activate timer calculation (Section 4.7.1.3, Table 5 of *MIPI Alliance Specification for M-PHY Version 3.0, Revision 03*)
- ❖ New configuration and capability parameters added in Table 49, 50, 53. The following are these parameters:
 - ◆ TX_Advanced_Granularity_Capability, TX_Advanced_Hibern8Time_Capability, TX_HS_Equalizer_Setting_Capability (Table 49)
 - ◆ TX_Advanced_Granularity_Setting, TX_Advanced_Granularity, TX_HS_Equalizer_Setting (Table 50)
 - ◆ RX_Advanced_Granularity_Capability, RX_Advanced_Hibern8Time_Capability, RX_Advanced_Min_ActivateTime_Capability (Table 53)

In order to enable the MPHY3.0 features, the user needs to set the configuration parameter mphy_protocol_version to "MPHY_VERSION_30".

5.12 M-PHY Specification Version 3.1 Feature Support

According to *MIPI Alliance Specification for M-PHY Version 3.1*, the M-PHY VIP supports the following features:

- ❖ T_ACTIVATE_TX time calculation (Section 4.7.1.3, Table 5 of *MIPI Alliance Specification for M-PHY Version 3.1*):

The calculation for T_ACTIVATE_TX time is based only on TX_Min_ActivateTime and granularity attributes. It is independent of the T_ACTIVATE time value.

❖ LS Prepare Length calculation

- a. Configuration of LS_PREPARE duration of the M-TX (Section 4.7.2.1 of *MIPI Alliance Specification for M-PHY Version 3.1*):

The PREPARE capabilities of the remote M-RX and the OMC in the LINE are used to configure the PREPARE duration of the local M-TX using the following method:

```
IF (OMC is present)
M-TX LS_PREPARE_LENGTH >= MAX(RX_LS_PREPARE_LENGTH_capability,
MC_LS_PREPARE_LENGTH)
ELSE
M-TX LS_PREPARE_LENGTH >= RX_LS_PREPARE_LENGTH_capability
```

- b. Calculation of TPWM_PREPARE (Section 4.7.2.2, Table 5 of *MIPI Alliance Specification for M-PHY Version 3.1*):

TPWM_PREPARE calculation is based only on M-TX LS_PREPARE_LENGTH and current PWM_GEAR without any dependency on OMC presence.

❖ Constraint on TAIL-OF-BURST for HS_BURST to LINE_CFG:

As per the new specification: During exit from BURST to LINE_CFG, DIF-P for greater than or equal to 20 UIHS plus TPWM_PREPARE (for configured PWM_GEAR and PREPARE_LENGTH) of the local M-TX shall not exceed the minimum value of TLINE-RESET-DETECT.

❖ min_save_config_time calculation (Section 8.4, Table 49 of *MIPI Alliance Specification for M-PHY Version 3.1*):

The time of 20 SI has been removed from the timing calculation of Min Save Config time. Now, min_save_config_time is calculated as: (TX_Min_SAVE_Config_Time_Capability * 40) ns

❖ Additional Configuration attributes

Two additional configuration attributes has been added in Section 8.4, Table 50 of *MIPI Alliance Specification for M-PHY Version 3.1*

- a. TX_Min_SLEEP_NoConfig_Time(Attribute ID – 0x38)
- b. TX_Min_STALL_NoConfig_Time(Attribute ID – 0x39).

❖ Reset value of MC_Output_Amplitude OMC attribute (Section 8.4, Table 50 of *MIPI Alliance Specification for M-PHY Version 3.1*):

As per new specification, the reset value for this attribute depends on MC_RX_LA_Capability attribute. If MC_RX_LA_Capability is true then reset value is LARGE_AMPLITUDE otherwise reset value is SMALL_AMPLITUDE.

This Feature involve the Setting of MC_Output_Amplitude from PA layer because MC_Output_amplitude will get set to LARGE or SMALL based on MC_RX_LA_Capability which is only known to PA layer.

For more information, see the MPHY Bugzilla:-

https://members.mipi.org/mipi-bugzilla/bugs/show_bug.cgi?id=4466

❖ Change in the range of values of some attributes (Section 8.4, Table 49 of *MIPI Alliance Specification for M-PHY Version 3.1*):

TX_PHY_Editorial_Release_Capability (Attribute ID – 0x0E) Range for this attribute has been changed from (1 to 99) to (0 to 99).

RX_PHY_Editorial_Release_Capability (Attribute ID - 0x91) Range for this attribute has been changed from (1 to 99) to (0 to 99).

MC_PHY_Editorial_Release_Capability (Attribute ID - 0xE2) Range for this attribute has been changed from (1 to 99) to (0 to 99).

- ❖ Driving Condition of RMMI control interface signal TX_DIFNDrive (Section A.3.1, Table 61 of *MIPI Alliance Specification for M-PHY Version 3.1*):

If TX_Controlled_ActTimer is 0, the protocol shall set this signal to 1 for at least T_ACTIVATE before issuing a pulse on TX_LineReset or on exiting HIBERN8.

5.12.1 Use Case to Enable M-PHY VIP for Specification Version 3.1 Features

To enable new features of *MIPI Alliance Specification for M-PHY Version 3.1*, user needs to specify the protocol version in shared configuration.

For example,

```
shared_cfg.master_cfg.tx_cfg.mphy_protocol_version =
svt_mphy_configuration::MPHY_VERSION_31;
shared_cfg.master_cfg.rx_cfg.mphy_protocol_version =
svt_mphy_configuration::MPHY_VERSION_31;
shared_cfg.slave_cfg.tx_cfg.mphy_protocol_version =
svt_mphy_configuration::MPHY_VERSION_31;
shared_cfg.slave_cfg.rx_cfg.mphy_protocol_version =
svt_mphy_configuration::MPHY_VERSION_31;
```

5.13 Line Reset

According to *MIPI Alliance Specification for M-PHY Version 4.1 - 01 December 2016*, the semantics of the M-CTRL-LINERESET.request primitive has been changed as follows:

- ❖ M-CTRL-LINERESET.request primitive has been changed to M-CTRL-LINERESET.request(TActivateControl). A new field t_activate_control is added in svt_mphy_transaction class for the input of LINERESET M-CTRL primitive.

Table 5-7 Table for t_activate_control

Variable Name	Variable Type	Description
t_activate_control	typedef enum bit { PROTOCOL_CONTROLLED = `SVT_MPHY_PROTOCOL_ CONTROLLED, PHY_CONTROLLED = `SVT_MPHY_PHY_CONTR OLLED} t_activate_control_ enum;	Indicates which Layer controls TActivate time and driving DIF-N. TActivateControl is an optional parameter. ❖ The default value of TActivateControl is ProtocolControlled ❖ When TActivateControl value is set to PhyControlled, TActivate timer is run by MTX.

- ❖ The t_activate_control variable is an optional parameter. So, a reasonable constraint is added to constraint its value to `SVT_MPHY_PROTOCOL_CONTROLLED.

- ◆ If M-CTRL-LINERESET.request with TActivateControl set to PhyControlled is issued when the M-TX is in BURST state, the Protocol Layer shall be aware that the payload of an ongoing BURST is interrupted immediately, and the M-TX might not trigger the proper BURST closure condition. M-TX shall immediately drive DIF-N on the LINE for TActivate before driving the LINE-RESET condition.
- ◆ After exiting LINE-RESET state, the Protocol Layer shall keep M-TX for a given LANE in SLEEP state for the greater of the local TX_Min_SAVE_Config_Time_Capability and, if known, for the remote RX_Min_SAVE_Config_Time_Capability.
- ❖ To control the TActivate timer using the Protocol layer or MTX, an interface pin TX_Controlled_ActTimer is defined. The direction of TX_Controlled_ActTimer is input for svt_mphy_rmmi_mtx_dut_controller_if interface and output for svt_mphy_rmmi_mtx_dut_phy_if interface.

Table 5-8 Table for TX_Controlled_ActTimer

Pin Name	Detection Type	Width	Description
TX_Controlled_ActTimer	Level	1	<p>TX_Controlled_ActTimer informs M-TX, which Layer controls the TActivate time.</p> <p>TX_Controlled_ActTimer is an optional signal.</p> <p>The Protocol Layer shall set TX_Controlled_ActTimer to "1" to inform M-TX to drive DIF-N for at least TActivate time, irrespective of the current M-TX state, before driving DIF-P for TLINE_RESET after TX_LineReset is asserted.</p> <p>M-TX shall also control the TActivate time upon HIBERN8 exit.</p> <p>When TX_Controlled_ActTimer is set to "0", the Protocol Layer shall wait for TActivate time after M-TX sets TX_SaveState_Status_N to "0" before asserting TX_LineReset. If TX_Controlled_ActTimer is set to "0", when TX_LineReset is asserted, M-TX shall immediately drive the LINE-RESET condition.</p> <p>The Protocol Layer shall also control the TActivate time upon HIBERN8 exit.</p>

5.13.1 Interface Behavior on Reception of LINERESET CTRL SAP in BURST State

This section describes the interface behavior on reception of LINERESET CTRL SAP in BURST state.

5.13.1.1 SERIAL Interface Behavior

Behavior of M-TX

If LINERESET CTRL SAP is received when MTX is in burst state, then MTX will stop the BURST only after sending the ongoing 10 bit serial data. MTX will not send any burst closure sequence and DIF_N will be driven on lines for TActivate time before driving LINERESET condition.

The M-TX Phy will also run the TActivate timer upon HIBERN8 exit.

Behavior of MRX

If MRX receives any corrupted stream of 10 bit data value, then the checker will transmit error messages.

When TActivate timer is running, MRX will detect DIF_N as burst closure condition and will send BURSTEND indication to PA layer.

5.13.1.2 TX RMMI LOCAL Interface Behavior

If MTX Phy detects TX_LineReset asserted during BURST state and TX_Controlled_ActTimer is high, then MTX Phy will de-assert Tx_PhyDIRDY signal and will start running TActivate timer before running LINERESET timer.

It is the responsibility of MTX CONTROLLER to de-assert TX_Burst and TX_ProtDORDY after asserting TX_LineReset. M-TX Phy shall also run the TActivate timer upon HIBERN8 exit, if TX_Controlled_ActTimer signal is high. The M-TX Phy shall not run the TActivate timer upon HIBERN8 exit if TX_Controlled_ActTimer signal is low.

5.13.2 Use Case for Line Reset

In this use case to send LINERESET CTRL SAP, the constraint mode is set to Off to send it as PHY_CONTROLLED:

```
\uvm_create(req);
req.reasonable_t_activate_control.constraint_mode(0);
\uvm_rand_send_with(req, {req.dir_kind == svt_mphy_transaction::M_TX;
                          req.primitive_layer == svt_mphy_transaction::M_CTRL;
                          req.primitive_name == LINERESET;
                          req.primitive_type == REQUEST;
                          req.t_activate_control == svt_mphy_transaction::PHY_CONTROLLED ;})
get_response(rsp);
```

By default, the value for t_activate_control is PROTOCOL_CONTROLLED. So, we can send LINERESET CTRL SAP without setting Off the reasonable constraint mode.

```
\uvm_do_with(req, {req.dir_kind == svt_mphy_transaction::M_TX;
                  req.primitive_layer == svt_mphy_transaction::M_CTRL;
                  req.primitive_name == LINERESET;
                  req.primitive_type == REQUEST;
                  })
get_response(rsp);
```



Note It is recommended to have minimum one clock difference between t_linereset_detect and t_linereset duration.

5.14 Lane to Lane Skew in Ctrl Transactions

A configuration field is added in svt_mphy_configuration class for skew insertion in CTRL transactions on RMMI interface.

This field inserts skew on interface whenever a positive value is assigned to it. This skew delays the assertion of TX/RX_CfgUpdt only.

Table 5-9 ctrl_xact_skew Variable Description

Variable Name	Variable Type	Description
ctrl_xact_skew	rand int	Integer to insert skew in ctrl xacts in MTX and MRX RMMI interfaces. Its default value is zero.

The `ctrl_xact_skew` variable inserts a delay for the assigned number of `CfgClk` just before the `CfgUpdt` signal. At the lane management layer, a dynamic array is defined in `svt_mphy_mport_lm_configuration` class.

Table 5-10 lane_to_lane_ctrl_xact_skew[] Variable Description

Variable Name	Variable Type	Description
lane_to_lane_ctrl_xact_skew[]	rand int	Each element of this dynamic array represents skew in the ctrl xact (CFGREADY) on RMMI interface

5.14.1 Use Case

If the active lanes are 4, then a dynamic array of size 4 is created in test case and skew for individual lanes is assigned on each index.

The following example has positive skew for all lanes. Lane with least skew, that is, `lane [0]` will be the reference for all other lanes. At run time, `TX/RX_CfgUpdt` signal for `lane [0]` will be asserted first, it will be followed by `lane [3]`, `lane [2]` and `lane [1]`.

```
shared_cfg.master_cfg.tx_cfg.lane_to_lane_skew = new [4];
shared_cfg.master_cfg.tx_cfg.lane_to_lane_skew[0] = 0;
shared_cfg.master_cfg.tx_cfg.lane_to_lane_skew[1] = 4;
shared_cfg.master_cfg.tx_cfg.lane_to_lane_skew[2] = 2;
shared_cfg.master_cfg.tx_cfg.lane_to_lane_skew[3] = 1
```

5.15 M-PHY Specification Version 4.0 and Later Feature Support

As per *MIPI Alliance Specification for MPHY Version 4.1 - 01 December 2016* the M-PHY VIP supports the following features:

- ❖ The New attributes are as follows:
 - ◆ `TX_HS_ADAPT_LENGTH`
 - ◆ `TX_ADAPT_Control`
 - ◆ `RX_HS_G4_PREPARE_LENGTH_Capability`
 - ◆ `RX_HS_G4_SYNC_LENGTH_Capability`
 - ◆ `RX_HS_Equalizer_Setting_Capability`
 - ◆ `RX_HS_ADAPT_LENGTH_FINE_Capability`
 - ◆ `RX_HS_ADAPT_LENGTH_COARSE_Capability`
 - ◆ `RX_ADAPT_Control`
- ❖ `ADAPT sub-state`

- ❖ ADAPT sequence on serial interface



Note ADAPT feature is supported for serial interface.

To enable new features of *MIPI Alliance Specification for M-PHY Version 4.0*, it is recommended to specify the protocol version in shared configuration. Similarly, you can update the M-PHY Version 4.1.

For example,

```
shared_cfg.master_cfg.tx_cfg.mphy_protocol_version =
svt_mphy_configuration::MPHY_VERSION_40;
shared_cfg.master_cfg.rx_cfg.mphy_protocol_version =
svt_mphy_configuration::MPHY_VERSION_40;
shared_cfg.slave_cfg.tx_cfg.mphy_protocol_version =
svt_mphy_configuration::MPHY_VERSION_40;
shared_cfg.slave_cfg.rx_cfg.mphy_protocol_version =
svt_mphy_configuration::MPHY_VERSION_40;
```

5.15.1 Use Case for Adapt Feature

As per the specification, for HS-G4, the PREPARE sub-state period may be followed by the ADAPT sub-state. The ADAPT sub-state is intended for bit synchronization of the M-RX to the embedded clock data stream and for M-RX channel equalizer adaptation. The ADAPT sequence shall start with an MK0 followed by an 8b10b encoded PRBS9 pattern completed by one b0 bit. Adding this bit enables complete encoding of the ADAPT sequence. The PRBS9 pattern is generated by a linear feedback shift register with feedback on the 5th and 9th taps as described by [ITU01]. The 8b10b encoded ADAPT sequence repeats every 650 bits.

To generate ADAPTSTART request for M-TX a new field `adaptstart_req` has been added in `svt_mphy_mport_lm_transaction` class.

M-TX will start Adapt sequence on serial interface when a transaction object with `adaptstart_req` set to 1 and `data_payload_size` as zero is sent from sequence.

For example,

```
`uvm_create(req);
req.reasonable_adaptstart_req.constraint_mode(0);
`uvm_rand_send_with(req, {
    req.adaptstart_req == 1'b1;
    req.data_payload_size == 0;
})
```

For more information on complete sequence example, see the `svt_mphy_lm_ls_hs_hs_g4_adapt_virtual_sequence` in `svt_mphy_mport_lm_virtual_sequence_collection.sv` file.



Note ADAPT sequence will get generated only when MTX is configured in HS-MODE and HS-G4, in other gears it will not be generated and SAP with `adaptstart_req == 1'b1` will get ignored by MTX.

5.16 Set Clock Period for PWM and HS Mode

By default MPHY VIP uses the following values for various gears and speed modes:

Table 5-11 Values for Various Gears and Speed Modes

Speed Modes and Gears	Clock Period
<code>pwm_g0_clk_period</code>	500000.00000 ps
<code>pwm_g1_clk_period</code>	300000.00000 ps
<code>pwm_g2_clk_period</code>	150000.00000 ps
<code>pwm_g3_clk_period</code>	76000.00000 ps
<code>pwm_g4_clk_period</code>	38000.00000 ps
<code>pwm_g5_clk_period</code>	18000.00000 ps
<code>pwm_g6_clk_period</code>	8000.00000 ps
<code>pwm_g7_clk_period</code>	4000.00000 ps
<code>hs_g1_series_a_clk_period</code>	801.28000 ps
<code>hs_g2_series_a_clk_period</code>	400.64000 ps
<code>hs_g3_series_a_clk_period</code>	200.32000 ps
<code>hs_g1_series_b_clk_period</code>	686.06000 ps
<code>hs_g2_series_b_clk_period</code>	343.03000 ps
<code>hs_g3_series_b_clk_period</code>	171.510000 ps

When the DUT uses different values for the above `clk_periods`, you can also use different clock period to drive the data. You may customize the clock period, as per the following configuration.

For example, if you want to set `pwm_g1_clk_period` as 200000ps instead of 300000ps, you can add the following statements in the mphy shared configuration file (`mphy_lm_shared_cfg.sv`).

```
master_cfg.tx_cfg.pwm_g1_clk_period = 200000ps; //Our default is 300000ps;
master_cfg.rx_cfg.pwm_g1_clk_period = 200000ps; //Our default is 300000ps;
slave_cfg.tx_cfg.pwm_g1_clk_period = 200000ps; //Our default is 300000ps;
slave_cfg.rx_cfg.pwm_g1_clk_period = 200000ps; //Our default is 300000ps;
```

**Note**

Similarly, we can update the clock period value of any other speed mode or gear.

6

VIP Tools

6.1 Using Native Protocol Analyzer for Debugging

6.1.1 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.

6.1.2 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+all`

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 following runtime switch.

Configuration Variable Setting:

Set `enable_xml_gen` parameter of `svt_mphy_mport_lm_agent_configuration` to enable the generation of PA and/or FSDB files.

For example,

```
master_cfg.enable_xml_gen = 1;
```

It is recommended to generate XML (0), FSDB (1) or BOTH (2) with the following user-configurable variable `pa_format_type`.



By default, `pa_format_type` is set to XML and this attribute can be modified as required.

```
master_cfg.pa_format_type = svt_xml_writer::FSDB;  
// 0 is XML, 1 FSDB and 2 both XML and FSDB, default it will be zero
```

Runtime Switch:

```
+svt_enable_pa=fsdb
```

Enables FSDB output of transaction and memory information for display in Verdi.

6.1.3 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.

6.1.4 Documentation

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

`$VERDI_HOME/doc/Verdi_Transaction_and_Protocol_Debug.pdf`

6.1.5 Limitations

Interactive support is available only for VCS.

7

Verification Topologies

This chapter describes several agent testbench topologies. In addition to a brief description of the testbench topologies, each section lists the configuration parameters required to instantiate the VIP agent.

This chapter discusses the following topologies:

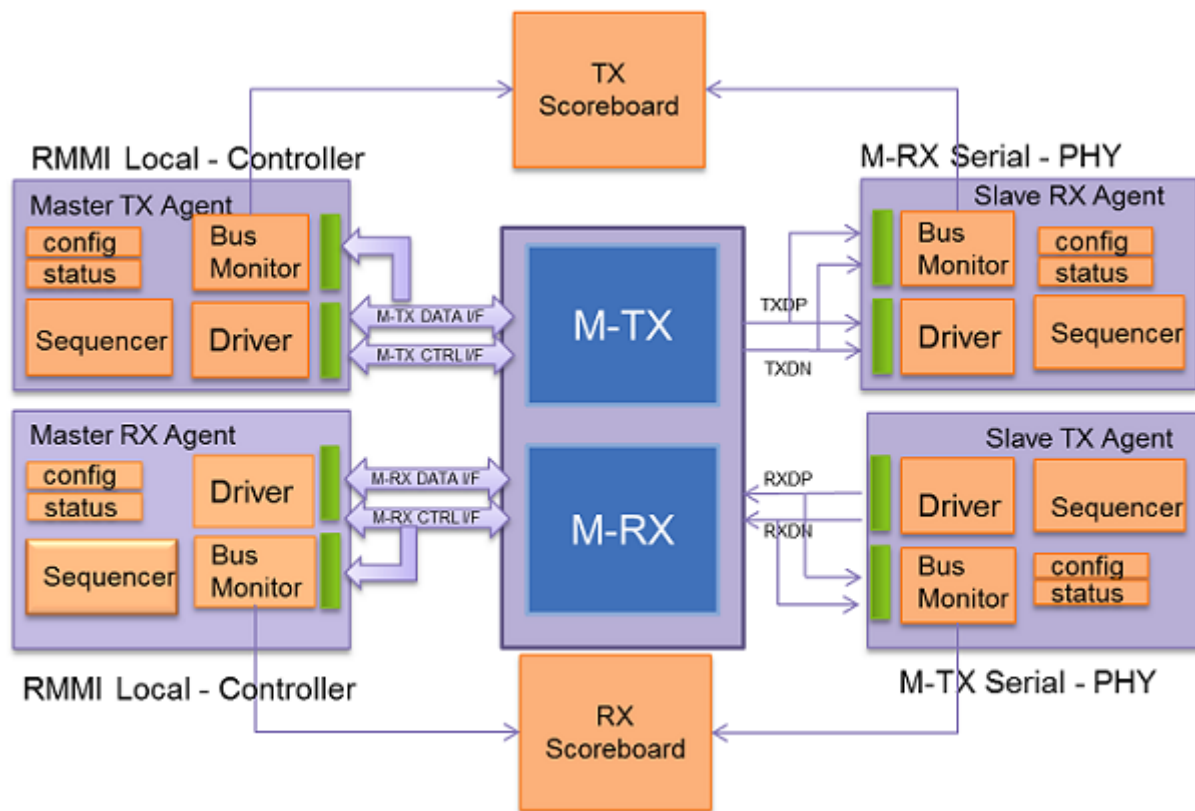
- ❖ [Single-Lane M-PHY DUT \(Signal Interface on Both Sides\)](#)
- ❖ [Single-Lane M-PHY DUT \(Serial Interface\)](#)
- ❖ [Single-Lane M-PHY DUT \(RMMI Remote Interface\)](#)
- ❖ [Multi-Lane M-PHY DUT \(Serial or RMMI Remote Interface\)](#)
- ❖ [Multi-Lane M-PHY DUT \(Serial or RMMI Local Interface\)](#)
- ❖ [Multi-Lane M-PHY VIP with MPHY Model](#)
- ❖ [Multi-Lane M-PHY VIP With Test Modes](#)

7.1 Single-Lane M-PHY DUT (Signal Interface on Both Sides)

This verification topology incorporates an Active mode M-PHY M-TX VIP Agent with RMMI DUT PHY interface, to generate M-PHY transactions for the PHY DUT. An M-PHY M-RX VIP Agent is configured with Serial Interface connected to the TX Serial Interface of DUT PHY.

Another topology incorporates an Active mode M-PHY M-TX VIP Agent with Serial DUT PHY interface, to generate M-PHY transactions for the PHY DUT. An M-PHY M-RX VIP Agent is configured with RMMI Interface connected to the M-RX DATA and CTRL Interface of DUT PHY, to monitor the M-PHY transactions.

Figure 7-1 Single-Lane M-PHY DUT (Signal Interface on Both Sides)



VIP configuration for this topology (Master TX Agent):

```
master_tx_config.is_active = 1;
master_tx_config.dir = svt_mphy_configuration::M_TX;
master_tx_config.interface_type = svt_mphy_configuration::RMMI_LOCAL;
master_tx_config.component_type = svt_mphy_configuration::CONTROLLER;
```

VIP configuration for this topology (Slave RX Agent):

```
slave_rx_config.is_active = 1;
slave_rx_config.dir = svt_mphy_configuration::M_RX;
slave_rx_config.interface_type = svt_mphy_configuration::SERIAL;
slave_rx_config.component_type = svt_mphy_configuration::PHY;
```

VIP configuration for this topology (Master RX Agent):

```
master_rx_config.is_active = 1;
master_rx_config.dir = svt_mphy_configuration::M_RX;
master_rx_config.interface_type = svt_mphy_configuration::RMMI_LOCAL;
master_rx_config.component_type = svt_mphy_configuration::CONTROLLER;
```

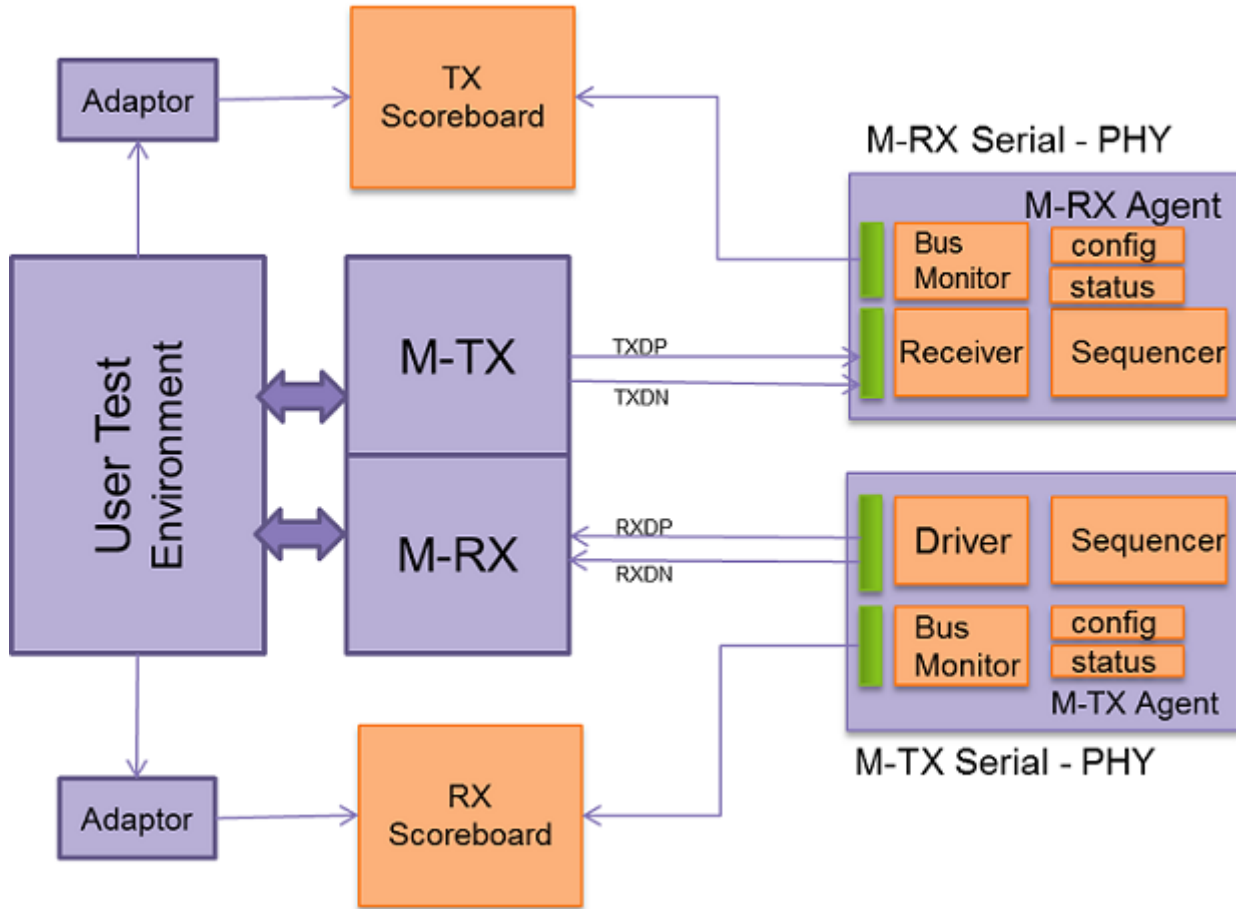
VIP configuration for this topology (Slave TX Agent):

```
slave_tx_config.is_active = 1;
slave_tx_config.dir = svt_mphy_configuration::M_TX;
slave_tx_config.interface_type = svt_mphy_configuration::SERIAL;
slave_tx_config.component_type = svt_mphy_configuration::PHY;
```

7.2 Single-Lane M-PHY DUT (Serial Interface)

This verification topology incorporates an Active mode M-PHY VIP agent with RX serial interface to receive M-PHY transactions from PHY DUT on the TX lane. It incorporates the M-PHY VIP with TX serial interface to generate and send M-PHY transactions to PHY DUT on the RX lane.

Figure 7-2 Single-Lane M-PHY DUT (Serial Interface)



VIP configuration for this topology (TX Agent):

```
tx_config.is_active = 1;
tx_config.dir = svt_mphy_configuration::M_TX;
tx_config.interface_type = svt_mphy_configuration::SERIAL;
tx_config.component_type = svt_mphy_configuration::PHY;
```

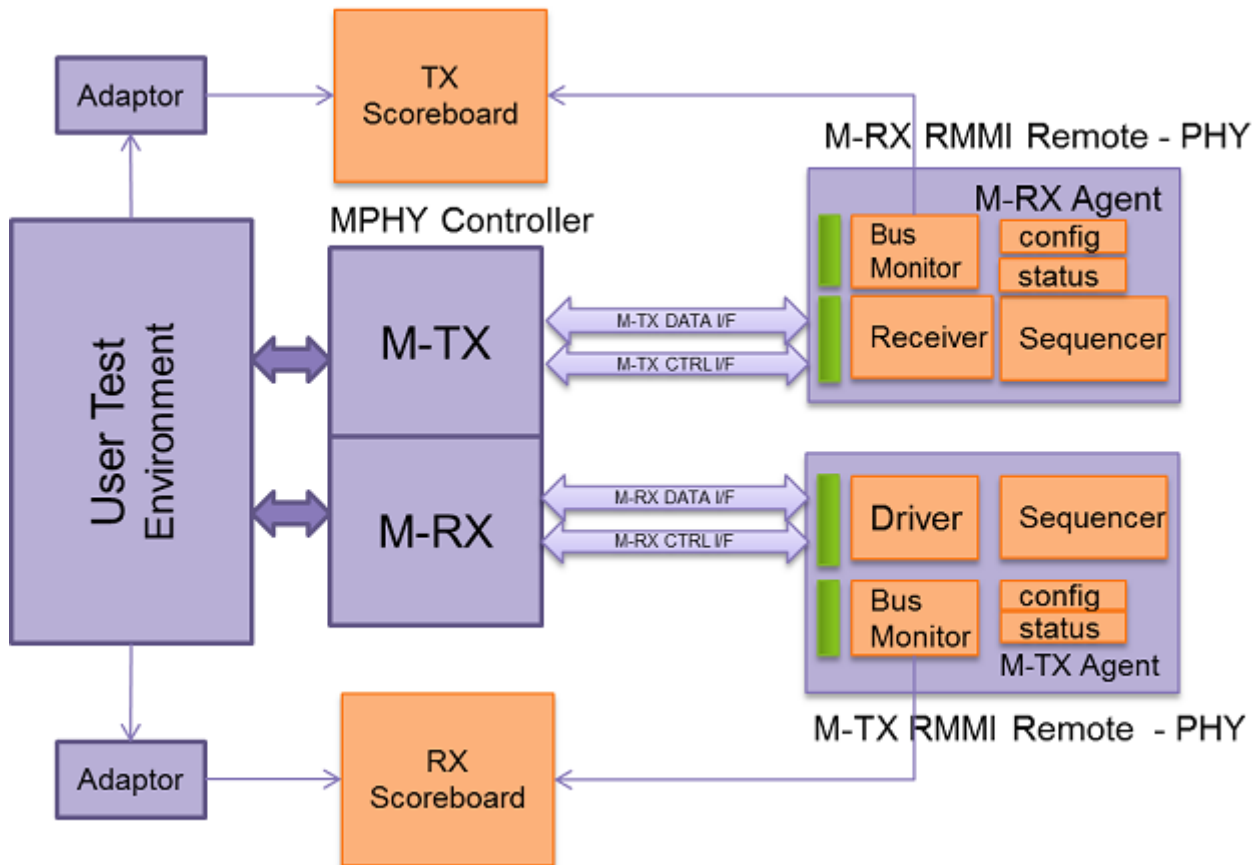
VIP configuration for this topology (RX Agent):

```
rx_config.is_active = 1;
rx_config.dir = svt_mphy_configuration::M_RX;
rx_config.interface_type = svt_mphy_configuration::SERIAL;
rx_config.component_type = svt_mphy_configuration::PHY;
```

7.3 Single-Lane M-PHY DUT (RMMI Remote Interface)

This verification topology incorporates an Active mode M-PHY VIP agent with RX RMMI_REMOTE interface to receive M-PHY transactions from Controller DUT on the TX lane. It incorporates the M-PHY VIP with TX RMMI_REMOTE interface to generate and send M-PHY transactions to Controller DUT on the RX lane.

Figure 7-3 Single-Lane M-PHY DUT (RMMI Remote Interface)



VIP configuration for this topology (TX Agent):

```
this.tx_config.is_active = 1;
this.tx_config.dir = svt_mphy_configuration::M_TX;
tx_config.interface_type = svt_mphy_configuration::RMMI_REMOTE;
tx_config.component_type = svt_mphy_configuration::PHY;
```

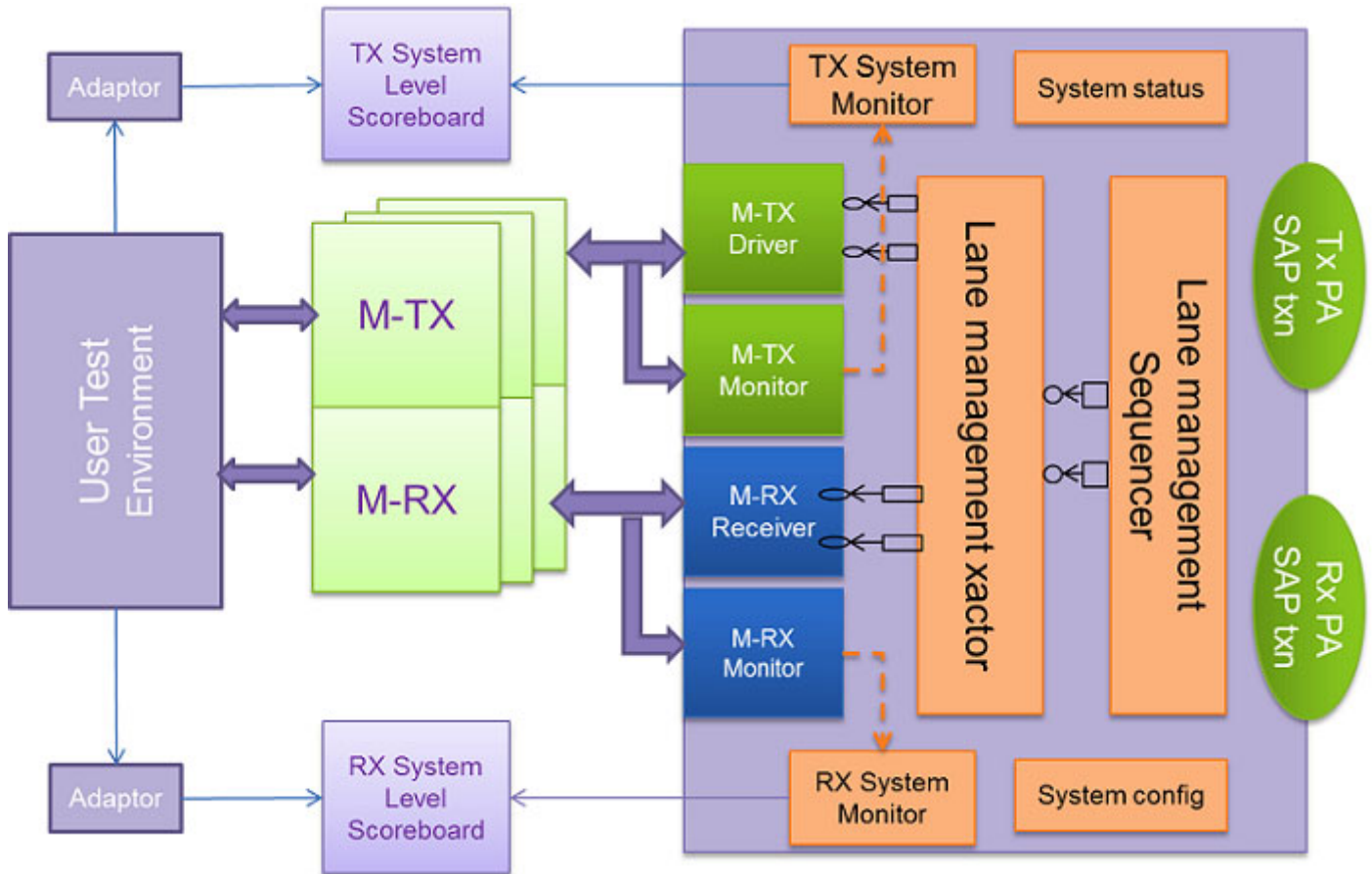
VIP configuration for this topology (RX Agent):

```
rx_config.is_active = 1;
rx_config.dir = svt_mphy_configuration::M_RX;
rx_config.interface_type = svt_mphy_configuration::RMMI_REMOTE;
rx_config.component_type = svt_mphy_configuration::PHY;
```

7.4 Multi-Lane M-PHY DUT (Serial or RMMI Remote Interface)

This verification topology incorporates an Active mode M-PHY M-PORT Lane Management VIP agent with TX and RX Serial interface with multi lane support. Configuration number of TX and RX lanes are supported and MPORT transactor takes care of lane splitting on TX sub-link and lane merging on TX sublink.

Figure 7-4 Multi-Lane M-PHY DUT (Serial or RMMI Remote Interface)



VIP configuration for this topology:

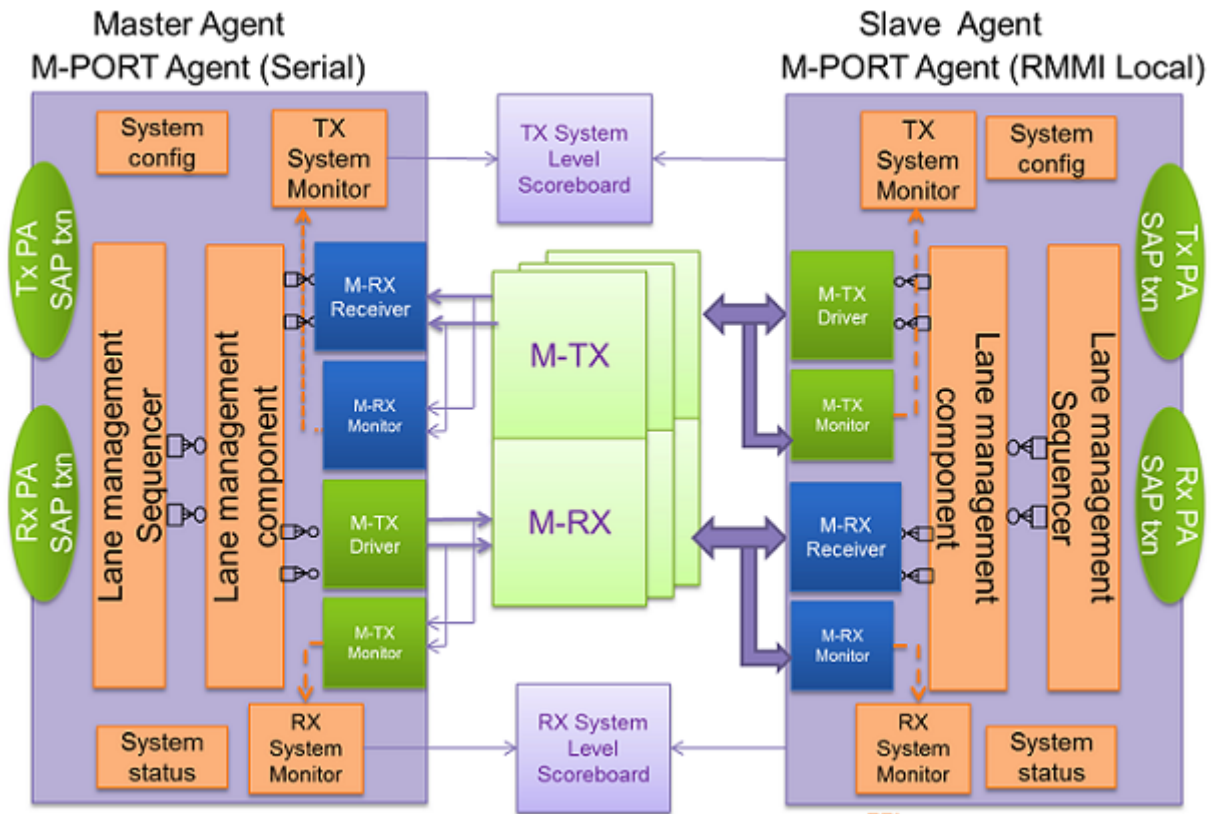
```
tx_cfg.is_active = 1;
tx_cfg.enable_monitor = 1;
tx_cfg.dir = svt_mphy_mport_lm_configuration::M_TX;
tx_cfg.interface_type = svt_mphy_configuration::SERIAL;
tx_cfg.component_type = svt_mphy_configuration::PHY;
```

```
rx_cfg.is_active = 1;
rx_cfg.enable_monitor = 1;
rx_cfg.dir = svt_mphy_mport_lm_configuration::M_RX;
rx_cfg.interface_type = svt_mphy_configuration::SERIAL;
rx_cfg.component_type = svt_mphy_configuration::PHY;
```

7.5 Multi-Lane M-PHY DUT (Serial or RMMI Local Interface)

This verification topology incorporates an Active mode Serial M-PHY M-PORT Lane Management VIP Agent with TX and RX Serial interface with multi lane support.

Figure 7-5 Multi-Lane M-PHY DUT (Serial or RMMI Local Interface)



VIP configuration for this topology (Master Agent):

```
master_config.is_active = 1;
master_config.tx_cfg.dir = svt_mphy_configuration::M_TX;
master_config.tx_cfg.interface_type = svt_mphy_configuration::SERIAL;
master_config.tx_cfg.component_type = svt_mphy_configuration::PHY;
```

```
master_config.is_active = 1;
master_config.rx_cfg.dir = svt_mphy_configuration::M_RX;
master_config.rx_cfg.interface_type = svt_mphy_configuration::SERIAL;
master_config.rx_cfg.component_type = svt_mphy_configuration::PHY;
```

VIP configuration for this topology (Slave Agent):

```
slave_config.is_active = 1;
slave_config.rx_cfg.dir = svt_mphy_configuration::M_RX;
slave_config.rx_cfg.interface_type = svt_mphy_configuration::RMMI_LOCAL;
slave_config.rx_cfg.component_type = svt_mphy_configuration::CONTROLLER;
```



```

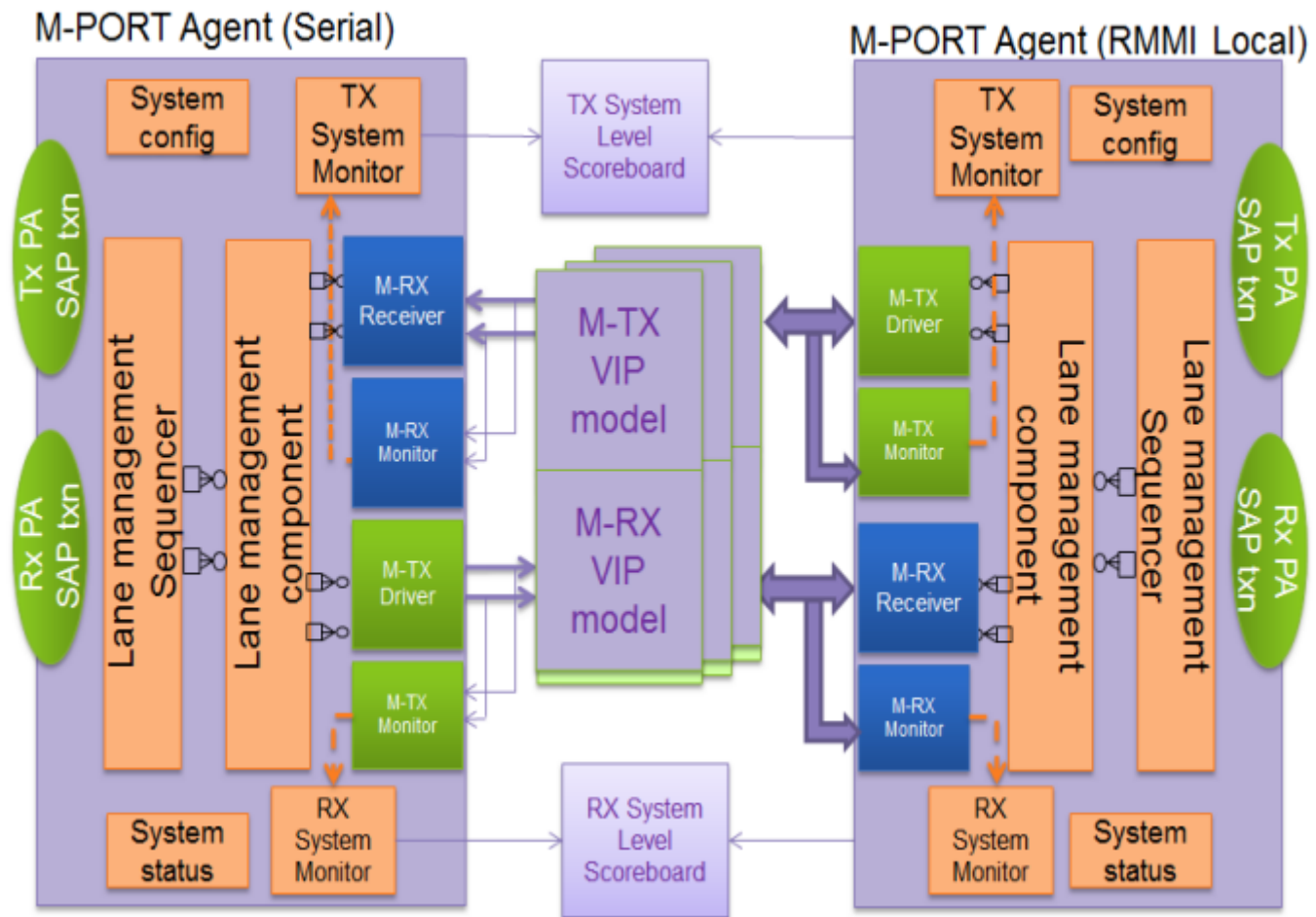
slave_config.is_active = 1;
slave_config.tx_cfg.dir = svt_mphy_configuration::M_TX;
slave_config.tx_cfg.interface_type = svt_mphy_configuration::RMMI_LOCAL;
slave_config.tx_cfg.component_type = svt_mphy_configuration::CONTROLLER;

```

7.6 Multi-Lane M-PHY VIP with MPHY Model

This verification topology incorporates an Active mode Serial M-PHY M-PORT Lane Management VIP Agent with TX and RX Serial interface with multi lane support. MPHY TX and RX models are connected on one side to serial interface and on other side to RMMI interface.

Figure 7-6 Multi-Lane M-PHY VIP with MPHY MODEL



VIP configuration for this topology (Master Agent):

```

master_config.is_active = 1;
master_config.tx_cfg.dir = svt_mphy_configuration::M_TX;
master_config.tx_cfg.interface_type = svt_mphy_configuration::SERIAL;
master_config.tx_cfg.component_type = svt_mphy_configuration::PHY;

```

```

master_config.is_active = 1;
master_config.rx_cfg.dir = svt_mphy_configuration::M_RX;

```

```
master_config.rx_cfg.interface_type = svt_mphy_configuration::SERIAL;
```

VIP configuration for this topology (Slave Agent):

```
slave_config.is_active = 1;  
slave_config.rx_cfg.dir = svt_mphy_configuration::M_RX;  
slave_config.rx_cfg.interface_type = svt_mphy_configuration::RMMI_LOCAL;  
slave_config.rx_cfg.component_type = svt_mphy_configuration::CONTROLLER;  
  
slave_config.is_active = 1;  
slave_config.tx_cfg.dir = svt_mphy_configuration::M_TX;  
slave_config.tx_cfg.interface_type = svt_mphy_configuration::RMMI_LOCAL;  
slave_config.tx_cfg.component_type = svt_mphy_configuration::CONTROLLER;
```

VIP configuration for this topology (MPHY TX MODEL Agent):

```
master_mtx_model_cfg.component_type = svt_mphy_configuration::MODEL;  
master_mtx_model_cfg.dir = svt_mphy_configuration::M_TX;
```

VIP configuration for this topology (MPHY RX MODEL Agent):

```
master_mrx_model_cfg.component_type = svt_mphy_configuration::MODEL;  
master_mrx_model_cfg.dir = svt_mphy_configuration::M_RX;
```

7.7 Multi-Lane M-PHY VIP With Test Modes

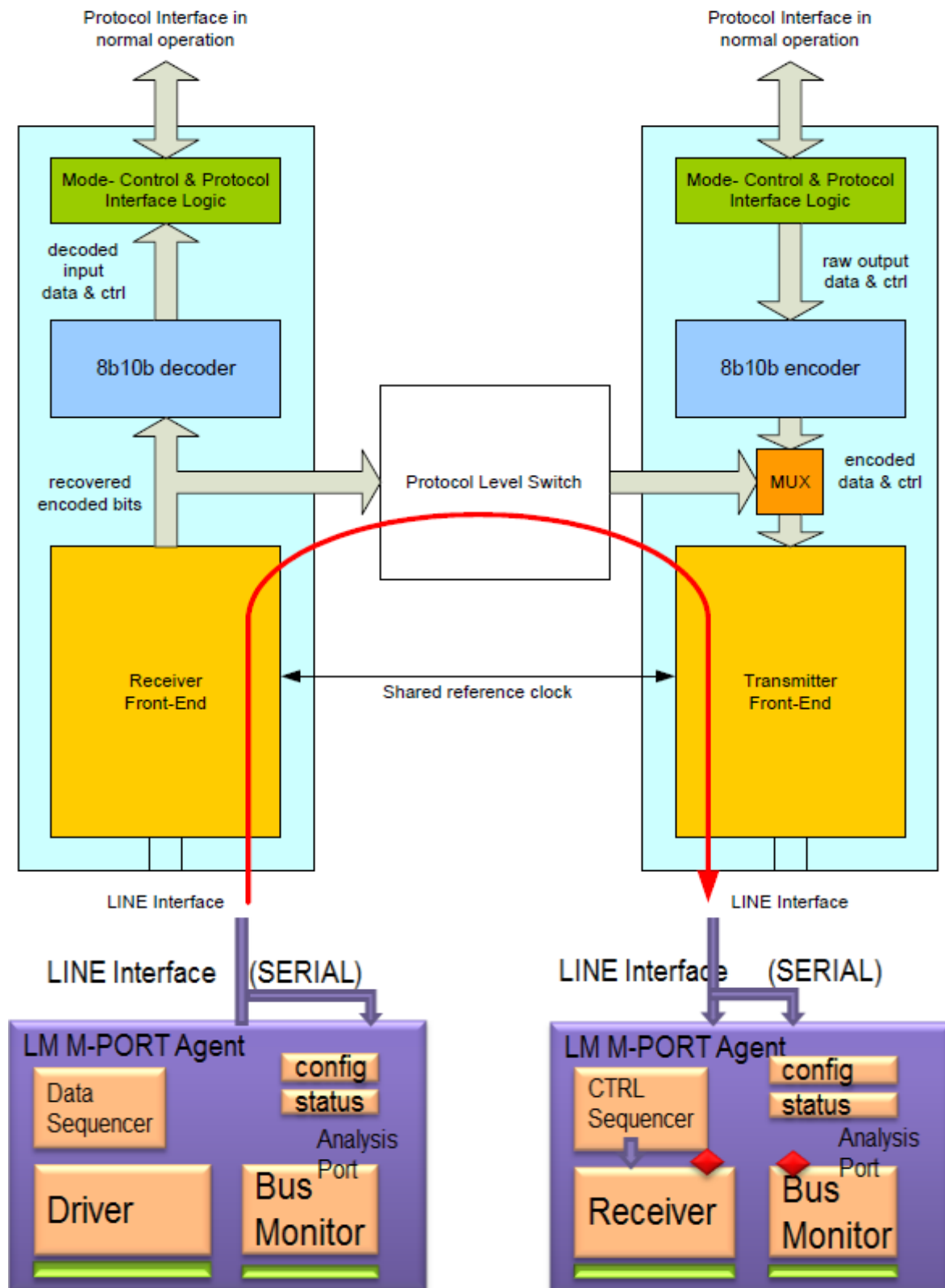
The M-PHY Verification IP supports the Test Mode functionality, wherein it supports the following two configurations:

- ❖ [“Near-End Loopback Mode”](#)
- ❖ [“Far-End Loopback Mode”](#)

7.7.1 Near-End Loopback Mode

This verification topology incorporates a Serial M-PHY M-PORT Lane Management VIP Agent with TX and RX Serial interface. M-PHY TX and RX are connected via serial interface as shown in [Figure 7-7](#).

Figure 7-7 Multi-Lane M-PHY VIP Connected as Near-End Loopback Configuration



VIP configuration for this topology (Master Agent):

```

master_config.tx_cfg.dir = svt_mphy_configuration::M_TX;
master_config.tx_cfg.interface_type = svt_mphy_configuration::SERIAL;
master_config.tx_cfg.component_type = svt_mphy_configuration::PHY;

```

VIP configuration for this topology (Slave Agent):

```

slave_config.rx_cfg.dir = svt_mphy_configuration::M_RX;
slave_config.rx_cfg.interface_type = svt_mphy_configuration::SERIAL;
slave_config.rx_cfg.component_type = svt_mphy_configuration::PHY;

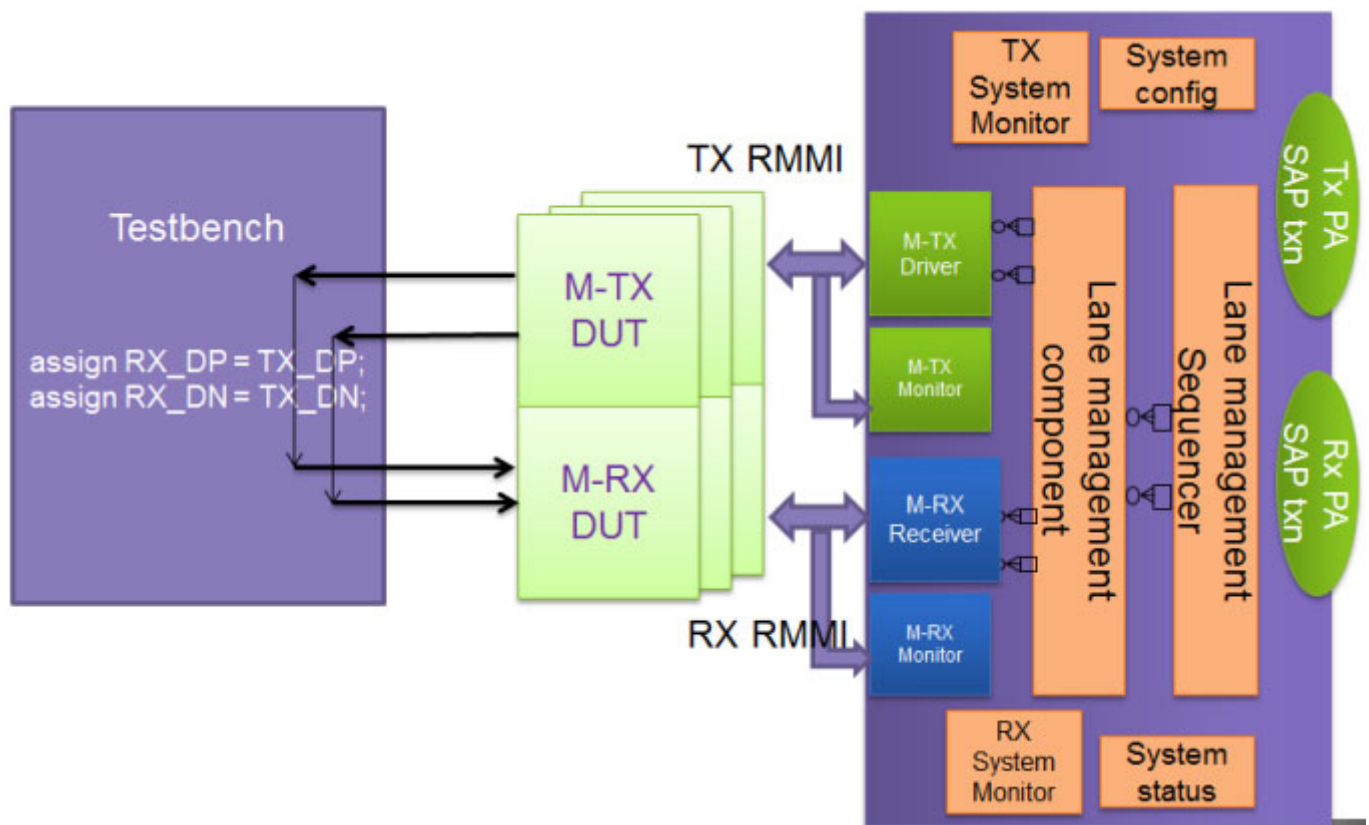
```

For rest of the configuration settings and usage notes, refer to “[Test Modes](#)”.

7.7.2 Far-End Loopback Mode

This verification topology incorporates an Active mode RMMI M-PHY M-PORT Lane Management VIP Agent with TX and RX RMMI interface. M-PHY TX and RX models are connected on one side to serial interface and on other side to RMMI interface.

Figure 7-8 Multi-Lane M-PHY VIP Connected as Far-End Loopback Configuration

**VIP configuration for this topology (Master Agent):**

```

master_config.is_active = 1;
master_config.tx_cfg.dir = svt_mphy_configuration::M_TX;
master_config.tx_cfg.interface_type = svt_mphy_configuration::LOCAL;
master_config.tx_cfg.component_type = svt_mphy_configuration::CONTROLLER;

```



```
master_config.rx_cfg.dir = svt_mphy_configuration::M_RX;  
master_config.rx_cfg.interface_type = svt_mphy_configuration::LOCAL;  
master_config.rx_cfg.component_type = svt_mphy_configuration::CONTROLLER;
```

For rest of the configuration settings and usage notes, refer to [“Test Modes”](#).

8

Usage Notes

This chapter discusses the usage notes for M-PHY. It contains the following topics:

- ❖ [Verification Environment Initialization](#)
- ❖ [Timescale Usage](#)
- ❖ [M-PHY Package Usage](#)
- ❖ [M-PHY Clocking Requirements](#)
- ❖ [Enabling the DIF_Z functionality](#)
- ❖ [Configurable RESET Duration](#)
- ❖ [Configurable Prepare and Sync Length Through PREPARE SAP](#)
- ❖ [Configurable Number of Fillers in a Transaction at Lane Management](#)
- ❖ [Enable or Disable External SYNC at Lane Management](#)
- ❖ [First SYNC Symbol Selection Between D10.5 and D26.5](#)
- ❖ [EVENT_MPHY_HIBERN8_ENTERED and EVENT_MPHY_HIBERN8_EXITED Events Usage](#)
- ❖ [EVENT_MPHY_DATA_STARTED and EVENT_MPHY_DATA_ENDED Events Usage](#)
- ❖ [M-RX Capability and M-TX Configuration Relation](#)
- ❖ [Transfer Data on TX Sublink and RX Sublink](#)
- ❖ [Midstream De-Assertion of RX_PhyDORDY or TX_ProtDORDY](#)
- ❖ [PWM Gear0 Requirements](#)
- ❖ [Aligned and Unaligned Transfers](#)
- ❖ [HIBERN8 Time Capability Configuration](#)
- ❖ [Dynamic Change in Active Lanes](#)
- ❖ [Bypass Mode](#)
- ❖ [Configuration Register Read Value Check](#)
- ❖ [Test Modes](#)
- ❖ [Line Control Command \(LCC\) with Media Convertor](#)
- ❖ [User-Specific Burst Closure Pattern](#)
- ❖ [Configuration Register Read on Configurable Number of Clocks](#)

- ❖ Inline Config Feature (Optional)
- ❖ De-Assertion of TX_SaveState_Status_N Signal (Optional)
- ❖ T_activate Timer on Exit to HIBERN8
- ❖ Control Symbols MK1,MK3,MK4,MK5 & MK6 insertion in a transaction at Lane Management
- ❖ Unaligned EOB at M-RX PHY
- ❖ RX_Burst Assertion at First MK0 in LS-MODE
- ❖ User Defined Attributes
- ❖ Send Multiple External SYNC Symbols in a Transaction at Lane Management
- ❖ Configure Number of Clock to Enable rx_hibern8exit_type_I Signal Check (RMMI Interface).
- ❖ Configure Initial Value of Running Disparity
- ❖ Clock Jitter
- ❖ Time Scale Independent Timers
- ❖ Support of Dynamic Reconfiguration for MPHY
- ❖ Dynamic Symbol Length Variation
- ❖ Setting of Symbol Length
- ❖ Configurable Extra Bits in PREPARE State
- ❖ Configure Number of Clock to Enable invalid_cfgrdyn_signal_val_check Signal Check (RMMI Interface)
- ❖ Check for RX_LCCRdDet Signal
- ❖ Dithering Support
- ❖ Allow LCC_READ in all PWM gears.
- ❖ Difference Between Prepare State and RX_Burst
- ❖ PWM Transmit Ratio Usage
- ❖ Enable External REF Clock (refclk) Support in VIP
- ❖ Max Lane Skew
- ❖ TX_DP and TX_DN Lines Driving User Control
- ❖ User-Defined Payload Pattern
- ❖ Enabling Debug Port View
- ❖ Corrupt 8b10b ADAPT Pattern
- ❖ Control De-Assertion of TX_ADAPT_REQ
- ❖ Corrupt 650bits of ADAPT Pattern Length
- ❖ 80bit Symbol Length
- ❖ TX/RX Reset Mode Support
- ❖ Support for LCC in MPHY5.0
- ❖ Support for All PWM Gear in MPHY5.0

8.1 Verification Environment Initialization

The Y usage model provides configuration parameters to initialize the verification environment, as per specific user needs.

The initialization is required in the following two cases:

- ❖ “Configuration Initialization at the Start of Simulation”
- ❖ “Configuration Initialization After Reset”

For instance, user can choose to start the model from an initial FSM state of SLEEP or STALL or HIBERN8 or DISABLED.

The user also has an option to modify the configuration parameter by CTRL SAPs as specified in the *MIPI Alliance Specification for MPHY Version 4.1 - 01 December 2016*.

8.1.1 Configuration Initialization at the Start of Simulation

The initialization at the start of simulation can be done using a set of configuration parameters. [Table 8-1](#) lists these configuration parameters with their purpose.

Table 8-1 Configuration Parameters

Configuration Parameter	Purpose
init_bypass_8b10b_enable	8b10b encoding bypassed
init_fsm_state	The initial FSM state from which model starts
init_hibern8_control	This is used in conjunction with init_fsm_state. In case init_fsm_state is SLEEP or STALL, a high on this parameter will move model into HIBERN8 state immediately
init_hsgear	The initial high speed gear. This is used in conjunction with init_mode. In case the init_mode is HS_MODE, then the initial value for HS_GEAR is chosen using this configuration parameter
init_hsrateseries	The initial high speed series rate. This is used in conjunction with init_mode. In case the init_mode is HS_MODE, then the initial value for HSRATE_SERIES is chosen using this configuration parameter.
init_hs_prepare_length	Initial value of prepare length in HS_MODE
init_hs_sync_length	Initial value of sync length in HS_MODE
init_hs_sync_range	Initial value for sync range in HS_MODE
init_ls_prepare_length	Initial value of prepare length in LS_MODE
init_mode	Initial value of mode
init_pwmgear	The initial PWM gear. This is used in conjunction with init_mode. In case the init mode is LS_MODE, then the PWM gear is chosen using this configuration parameter
init_pwm_g6_g7_sync_length	Initial value of sync length for LS_MODE PWM gears G6 and G7
init_pwm_g6_g7_sync_range	Initial value for sync range for LS_MODE

Table 8-1 Configuration Parameters (Continued)

Configuration Parameter	Purpose
init_pwm_burst_closure_extension	This value is used as initial value for TX_Burst_Closure_Extension
init_lcc_enable	Initial value for TX_LCC_Enable

All the above attributes are declared as rand in `svt_mphy_configuration` class. The user can modify their values and initialize the VIP accordingly. These parameters are only declared for some important configuration attributes present in *Table 51 and 55 of MIPI Alliance Specification for MPHY Version 4.1 - 01 December 2016*. Default value for all these parameters are the reset values defined in specification for attributes. These attributes reduces the default values defined in the spec for less simulation time.

In case the initial FSM state is set to `DISABLED`, you need to provide `RESET_CTRL_SAP` which will change the FSM state to `HIBERN8` after running the reset timer. The exit from `HIBERN8` can be achieved by sending `CFGSET_REQUEST` of `M-CTRL - TX_HIBERN8_Control` to `EXIT` followed by `M-CTRL-CFGREADY_REQUEST`.

If DUT does not support the changing of configuration attributes at initialization then all `init_*` parameters should be left with default values and should not be randomized.

MPHY VIP can directly start in `STALL` or `SLEEP` state apart from `DISABLED` state. If DUT also supports this feature then user needs to set the value of `init_fsm_state` to `STALL` or `SLEEP` and set other `init_*` parameters accordingly.

For example,

To start the MODEL directly from `SLEEP` state and in `PWM_G3`, following parameters need to set:-

```
init_fsm_state = svt_mphy_types::SLEEP;
init_mode     = svt_mphy_types::LS_MODE;
init_pwmgear  = svt_mphy_types::PWM_G3;
init_ls_prepare_length = 10;
```

**Note**

All the `init_*` parameters are applicable till first RESET. After `LOCAL RESET` or `LINERESET` `reset_*` parameters are used. `reset_*` parameters are described in the following section [“Configuration Initialization After Reset”](#). If MODEL is starting from `DISABLED` state then also these parameters are not applicable.

8.1.2 Configuration Initialization After Reset

The *MIPI Alliance Specification for MPHY Version 4.1 - 01 December 2016* clearly defines the reset values for each configuration parameter in Table 49 and 53. The model allows the user to override these values and specify its own set of reset values. [Table 8-2](#) defines the set of configuration parameters which can be used to override the reset values.

All `reset_*` parameters are declared as rand in `svt_mphy_configuration` class initialized with the spec defined reset values. With the help of these parameters user can change the spec defined reset values in VIP.

These parameters overrides the spec defined reset values of configuration attributes after `LOCAL RESET` and `LINERESET`.

For correct operation, values assigned to these parameters should match the reset values of DUT. If DUT does not support different reset values then these parameters can be left without assigning any values and should not be randomized too.

Table 8-2 Configuration Parameters to Override Reset Values

Configuration Parameter	Description
reset_bypass_8b10b_enable	8b10b encoding bypassed
reset_hsgear	The reset value for high speed gear. This is used in conjunction with <code>reset_mode</code> . In case the <code>reset_mode</code> is <code>HS_MODE</code> , then the initial value for <code>HS_GEAR</code> is chosen using this configuration parameter
reset_hsrateseries	The reset value for high speed series rate. This is used in conjunction with <code>init_mode</code> . In case the <code>init_mode</code> is <code>HS_MODE</code> , then the initial value for <code>HSRATE_SERIES</code> is chosen using this configuration parameter
reset_hs_prepare_length	Reset value of prepare length in <code>HS_MODE</code>
reset_hs_sync_length	Reset value of sync length in <code>HS_MODE</code>
reset_hs_sync_range	Reset value for sync range for <code>HS_MODE</code>
reset_ls_prepare_length	Reset value of prepare length in <code>HS_MODE</code>
reset_mode	Reset value for mode
reset_pwmgear	The reset value for low speed gear. This is used in conjunction with <code>reset_mode</code> . In case the <code>reset_mode</code> is <code>LS_MODE</code> , then the initial value for <code>PWM_GEAR</code> is chosen using this configuration parameter
reset_pwm_g6_g7_sync_length	Initial value of sync length for <code>LS_MODE</code> PWM gears G6 and G7
reset_pwm_g6_g7_sync_range	Reset value for sync range for <code>LS_MODE</code>
reset_pwm_burst_closure_extension	This value is used for attribute <code>TX_Burst_Closure_Extension</code> after <code>RESET</code> or <code>LINERESET</code> .
reset_lcc_enable	Reset value for <code>TX_LCC_ENABLE</code>

Table 8-3 Non Rand Configuration Parameters To Override Reset Values

Configuration Parameter	Description
reset_hs_slewrates	Reset value of hs slewrates is 'h00. There is no specific value of reset in specification. Implementation should ensure that the <code>TX_HS_SlewRate</code> value does not violate other parameter specifications
reset_driver_polarity	Reset value of driver polarity is <code>NORMAL</code> for local <code>RESET</code>

Configuration Parameter	Description
<code>reset_sync_source</code>	Reset value of sync source is <code>INTERNAL_SYNC</code>
<code>reset_advanced_granularity_step</code>	Reset value of advanced granularity step is zero
<code>reset_lcc_sequencer</code>	Reset value of lcc sequencer is 0 (no READ or WRITE operation requested)
<code>reset_termination_force_enable</code>	Reset value of termination force enable is NO
<code>reset_amplitude</code>	Reset value of amplitude is <code>LARGE_AMPLITUDE</code>
<code>reset_adapt_control</code>	Reset value of adapt control is SYNC
<code>reset_min_activate_time</code>	Reset value of min activate time is 15
<code>reset_hs_terminated_enable</code>	Reset value of hs terminated enable is OFF
<code>reset_ls_terminated_enable</code>	Reset value of ls terminated enable is OFF
<code>reset_hs_equalizer_setting</code>	Reset value of hs equalizer setting is 0 for local RESET
<code>reset_advanced_granularity</code>	Reset value of advanced granularity is 15
<code>reset_min_sleep_noconfig_time</code>	Reset value of min sleep noconfig time 15
<code>reset_min_stall_noconfig_time</code>	Reset value of min stall noconfig time 255
<code>reset_hs_adapt_length</code>	Reset value of hs adapt length 17
<code>reset_hs_adapt_range</code>	Reset value of hs adapt range COARSE

8.2 Timescale Usage

The M-PHY supports Serial and RMMI interface. The signaling in these interfaces is dependent on the timescale (timeunit and timeprecision). In these interfaces, HS_Gear 3 Series B runs at 5.8Gbps and therefore a proper time unit is required.

This section defines the minimum requirement of timescale for the M-PHY to function properly.

Serial Interface

Specify the timescale to 100ps/100ps or more. Timeprecision is taken care in the serial interface files.

RMMI Interface

Specify the timescale to 100ps/100ps or more (for example, 10ps/10ps)



Note

You must ensure that the DUT timescale does not override the M-PHY timescale. So, user must include M-PHY package (or any other protocol package using M-PHY) just after the timescale inclusion. Thereafter, a DUT package can be included.

8.3 M-PHY Package Usage

M-PHY defines the following two packages:

- ❖ `"svt_mphy_component.uvm.pkg"`

❖ “svt_mphy.uvm.pkg”

svt_mphy_component.uvm.pkg

This package includes M-PHY data classes, M-PHY transactor classes and M-PHY monitor classes. It can be used by other higher level protocol layers (LLI, UNIPRO, USB) to have M-PHY functionality imported inside their top packages.

Use model to include and import this package is shown below using a LLI example.

1. **Step 1:** Create a top level LLI package and include and import M-PHY component package

```
`include "svt.uvm.pkg"
`include "svt_mphy_component.uvm.pkg"
package svt_mipi_lli_uvm_pkg;
import uvm_pkg::*;
`include "uvm_macros.svh"
import svt_uvm_pkg::*;
import svt_mphy_component_uvm_pkg::*;
endpackage
```

2. **Step 2:** Import M-PHY component package wherever LLI top package is imported

```
`include "svt_mipi_lli.uvm.pkg"

module test_top;

/** Import UVM Package */
import uvm_pkg::*;
`include "uvm_macros.svh"

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

/** Import MPHY UVM COMPONENT Package */
import svt_mphy_component_uvm_pkg::*;

/** Import MIPI LLI VIP Package */
import svt_mipi_lli_uvm_pkg::*;
endmodule
```

svt_mphy.uvm.pkg

This package imports svt_mphy_component.uvm.pkg, M-PHY agent classes, LM data classes, LM transactors, LM monitors and LM agent classes. To create verification topologies as discussed in Chapter-7 [Verification Topologies](#), svt_mphy.uvm.pkg is imported to create M-PHY agents or M-PHY MPORT LM agents.

Use model to include and import this package is shown below.

1. **Step 1:** Create a top level M-PHY package and include and import M-PHY component package

```
`include "svt_mphy_component.uvm.pkg"
package svt_mphy_uvm_pkg;
import uvm_pkg::*;
`include "uvm_macros.svh"
import svt_uvm_pkg::*;
import svt_mphy_component_uvm_pkg::*;
endpackage
```

2. **Step 2:** Import M-PHY component package and M-PHY top package in module

```
`include "svt_mphy.uvm.pkg"
```

```

module test_top();

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

`include "uvm_macros.svh"

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

/** Import the MPHY COMPONENT Package */
import svt_mphy_component_uvm_pkg::*;

/** Import the MPHY Package */
import svt_mphy_uvm_pkg::*;
endmodule

```



Note These packages are available for all three methodologies, that is, UVM, OVM, and VMM.

8.4 M-PHY Clocking Requirements

This section provides the details of M-PHY VIP clocking requirement related to the operating frequencies for LS_MODE and HS_MODE.

MPHY VIP requires a reference clock in LS_MODE for TYPE_II signaling. Its frequency can be 26 MHz, 38.4 MHz or 52 MHz as per *MIPI Alliance Standard for DigRF Version 1.10.00-22 July 2011*, Table 4.

User can set any one of the operating frequencies by modifying the define value as shown below in the top.sv file of testbench.

- ❖ For 19.2 MHz: define SVT_MPHY_SYS_CLK_PERIOD 52.08ns
- ❖ For 26 MHz: define SVT_MPHY_SYS_CLK_PERIOD 38.46ns
- ❖ For 38.4 MHz: define SVT_MPHY_SYS_CLK_PERIOD 26.04ns
- ❖ For 52 MHz: define SVT_MPHY_SYS_CLK_PERIOD 19.23ns

M-PHY VIP supports 19.2 MHz operating frequency.

For HS_MODE signaling and LS_MODE TYPE_I signaling, MPHY VIP doesn't require reference clock.

The following subsections provide details regarding clocking requirements for the M-PHY Verification IP for Serial and RMMI modes:

- ❖ ["Serial Mode Clocking Requirements"](#)
- ❖ ["RMMI Mode Clocking Requirements"](#)

8.4.1 Serial Mode Clocking Requirements

In Serial Mode, the M-PHY TX VIP model expects UI clock defined as clk. The clock frequency of clk must be aligned to the various gears and modes as defined in Table 8 and 9 of *MIPI Alliance Specification for MPHY Version 4.1 - 01 December 2016*.

By default, the clock clk needs to be provided by the user. In case user wants to generate the clock clk from the model itself, the user must enable the following define in the simulation.

Run the simulation with SVT_MPHY_SERIAL_INTERNAL_CLK = 1

To recover the clk at RX path of M-PHY VIP model for PWM signaling, that is, signaling for LS M-PHY TYPE-I, posedge of clk always aligns itself to the negedge of rx_dp signal present on serial interface. The negedge of clk distinguishes between major and minor ratio and provides logic_0 and logic_1.

Clock recovery is done by the model by enabling the following define:

```
SVT_MPHY_SERIAL_INTERNAL_CLK = 1
```

In case of SYS signaling, that is, signaling for LS M-PHY TYPE_II or HS signaling, user shall implement the CDR logic and provide the recovered clock to the M-PHY VIP interfaces. To enable CDR logic from the model and recover the clock from incoming serial data stream, enable the following define:

```
SVT_MPHY_SERIAL_INTERNAL_CLK = 1
```

This logic checks for the rx_dp signal edges and when rx_dp edges come closer to clk edges, clk aligns itself to the centre of rx_dp unit interval (UI) duration, so that correct data is sampled.

**Note**

At RX path, CDR logic is required by the RX transactors of M-PHY VIP at serial interface to recover the clock from the serial data stream. At TX path, TX transactors of M-PHY VIP expect a local clock clk which is aligned to gears and modes at which serial interface is running. This logic is defined inside svt_mphy_serial_tx_if and svt_mphy_serial_rx_if files and used when CDR mechanism is enabled from the model using SVT_MPHY_SERIAL_INTERNAL_CLK

**Note**

PWM gears have different frequency ranges as per *Draft MIPI MPHY Specification Version 4.1 - 01 December 2016, Table 10*. For MPHY RX VIP to run in sync with MPHY TX DUT at serial interface, user must set the following defines to a value at which MPHY TX DUT is running:
SVT_MPHY_PWM_G0_CLK_PERIOD, SVT_MPHY_PWM_G1_CLK_PERIOD,
SVT_MPHY_PWM_G2_CLK_PERIOD, SVT_MPHY_PWM_G3_CLK_PERIOD,
SVT_MPHY_PWM_G4_CLK_PERIOD, SVT_MPHY_PWM_G5_CLK_PERIOD,
SVT_MPHY_PWM_G6_CLK_PERIOD and SVT_MPHY_PWM_G7_CLK_PERIOD

8.4.2 RMMI Mode Clocking Requirements

In RMMI Mode, there are four clocks specified in the M-PHY Specification. This section defines the model requirement for the following clocks:

❖ RX_CfgClk or TX_CfgClk

In RMMI mode, the CfgClk is an input to the model. The M-PHY specification doesn't define any specific frequency or frequency range for this clock.

It also doesn't define any phase relation with the symbol clock. The M-PHY model works with any clock frequency provided by the user. All the control transactions are processed using this clock. The user has to ensure that there is sufficient CfgClk cycles provided for successful RCT before starting the new burst.

The M-PHY model has inbuilt EVENT_MPHY_RECONFIGURE_TRIGGER notification for UVM which can be used in the testbench to ensure the RCT completion.

❖ RX_SymbolClock or TX_SymbolClock

The TX_SymbolClock is generated by the TX controller and passed to TX PHY. RX_SymbolClock is generated by RX PHY and passed to the RX controller.

When the model is used as TX RMMI Controller or RX RMMI PHY, it generates the TX_SymbolClock and RX_SymbolClock respectively.

The clock frequency generated by the model is aligned to various gears and modes as defined in Table 9 and 10 in *Draft MIPI MPHY Specification Version 4.1 - 01 December 2016*.

In case the model is used as TX RMMI PHY or RX RMMI controller, the symbol clock is expected as an input to the model.

As per *MIPI Alliance Specification for MPHY Version 4.1 - 01 December 2016*, TX_SymbolClk is generated by MTX Controller and it is an input to MTX PHY. An enhancement, TX_SymbolClk being generated as an output from MTX PHY is added in MPHY VIP. This feature can be enabled by setting the configuration parameter **enable_phy_clock_gen** in MPHY VIP cfg. When **enable_phy_clock_gen** is set to one TX_SymbolClk will be output from MTX PHY VIP and will be input to MTX Controller DUT. Thus we should swap the clock connections in the top.sv file as shown below:

Remove the following connection:

```
assign mtx_dut_controller_if[i].TX_SymbolClk =
mtx_dut_phy_if[i].TX_SymbolClk;
```

Add the following connection:

```
assign mtx_dut_phy_if[i].TX_SymbolClk =
mtx_dut_controller_if[i].TX_SymbolClk;
```

- ◆ As per *MIPI Alliance Specification for MPHY Version 4.1 - 01 December 2016* and later, RX_SymbolClk and TX_SymbolClk generation may get disabled when the FSM state is not in LINE-CFG, PWM-BURST, SYS-BURST, or HS-BURST states.

To enable this behavior in M-PHY VIP, user needs to set the following bit in shared cfg file:

```
allow_symbol_clock_disable = 1
```

In DUT environment, if the DUT is not generating SymbolClk in save states, then allow_symbol_clock_disable bit shall be set to one in the shared cfg.

- ◆ TX_SymbolClk is generated by MTX CONTROLLER. If MTX DUT PHY generates its own TX_SymbolClk, then it can be looped back into MPHY VIP CONTROLLER.

To enable this behavior, user needs to set the following bit in shared cfg:

```
external_symbol_clock = 1
```

Also, DUT TX_SymbolClk needs to be connected to the ext_TX_SymbolClk pin in HDL interconnect.

When this behavior is enabled, DUT is responsible for the switching of TX_SymbolClk (on different gears) and both MTX DUT PHY and VIP DUT CONTROLLER will work on this clock.

❖ RX_RefClock or TX_BitClk

These clocks are not used by the model.

❖ SYS Clock

SYS clock runs on 19.2Mhz, 26 Mhz, 38.4 Mhz, or 52 Mhz. This clock is used for SYS signaling on serial interface. Therefore, TX_SymbolClock or RX_SymbolClock runs at 1/10 times the SYS clock frequency, if symbol_length=10.

To set SYS clock at 26 Mhz and RX_SymbolClk at 2.6Mhz at RMMI, use define in top level testbench file:


```
`define SVT_MPHY_SYS_CLK_PERIOD 38.46ns
```

8.4.2.1 How to Assign Same Clock to all Lanes

- ❖ When PHY VIP generates clock, it is possible that `SymbolClk` on each lane might have a phase shift due to different configuration on each lane.
- ❖ To overcome this issue, you can assign Lane0 MPHY VIP clock to other lanes as shown in this table using the MPHY configuration parameters `enable_phy_clock_gen` and `external_symbol_clock`.

	<code>enable_phy_clock_gen</code>	<code>external_symbol_clock</code>	Clock connection	Comments
TX Lane 0	1	0	assign mtx_dut_phy_if[0].TX_SymbolClk = mtx_dut_controller_if[0].TX_SymbolClk;	PHY VIP TX will generate clock to DUT.
RX Lane 0	1	0	assign mrxdut_phy_if[0].RX_SymbolClk = mrxdut_controller_if[0].RX_SymbolClk;	PHY VIP RX will generate clock to DUT
TX other lanes[1:N]*	0	1	assign master_mtx_dut_controller_if[N].TX_SymbolClk = master_mtx_dut_controller_if[0].TX_SymbolClk;	Assign Lane0 clock generated by PHY VIP to all other lanes of VIP
RX other lanes[1:N]*	0	1	assign master_mrx_dut_controller_if[N].ext_RX_SymbolClk = master_mrx_dut_controller_if[0].RX_SymbolClk;	Assign Lane0 clock generated by PHY VIP to all other lanes of VIP

**Note**

N = Lane id other than lane0

8.5 Enabling the DIF_Z functionality

According to MPHY specs, DIF_Z is driven by M_RX in DISABLED and HIBERN8 state. M-TX drives high-Z in DISABLED and HIBERN8 state.

To enable DIF_Z functionality, user sets the following:

```
SVT_MPHY_ENABLE_DIFZ = 1
```

DIF_Z is defined as DP or DN being driven to weak0 by M-RX.

By DEFAULT in MPHY VIP, M_RX sense DP and DN lines (it doesn't drive anything on DP and DN) and M-TX maintains high-Z on both the DP and DN lines. So M-RX sense the high-Z in the default behavior.

**Note**

This functionality is valid only for serial interfaces.

8.6 Configurable RESET Duration

The M-PHY Verification IP provides configurable RESET duration feature. This feature enables the user to control the reset timer. If the user wants to extend the reset duration in the DISABLED state to a configurable length, then RESET REQUEST CTRL SAP is appended with one extra field which is defined as real in the transaction class. This feature is applicable to both Lane Management and without Lane Management.

The following new variables have been added to support this feature:

Variable Name	Variable Type	Description
reset_duration	real	SVT_MPHY_TRANSACTION_RESET_ON (-1): To assert reset and leave it asserted. SVT_MPHY_TRANSACTION_RESET_OFF (0): To de-assert reset if it was high. Any value greater than zero (defined in terms of time, such as, 500ns): To give specific timer value different from configuration defined reset_timer. SVT_MPHY_TRANSACTION_DEFAULT_RESET (-2): Default value to process reset in protocol defined manner.

8.6.1 Use Case to Assert and De-Assert RESET Through CTRL SAP

To assert the RESET, user needs to send RESET REQUEST SAP as shown in the following code snippet:

```
// RESET CTRL SAP to assert the reset signal
svt_mphy_transaction req;
`uvm_create(req)
req.reset_duration = `SVT_MPHY_TRANSACTION_RESET_ON;
`uvm_rand_send_with (req,
{ req.dir_kind == svt_mphy_transaction::M_TX;
  req.primitive_layer == svt_mphy_transaction::M_CTRL;
  req.primitive_name == svt_mphy_transaction::RESET;
  req.primitive_type == svt_mphy_transaction::REQUEST;
})
get_response(rsp);
```

The RESET REQUEST SAP will assert the RESET and will leave it asserted until the SAP to de-assert RESET is given. This is shown in the following code snippet:

```
// RESET CTRL SAP to De-assert the reset signal
svt_mphy_transaction req;
`uvm_create(req)
req.reset_duration = `SVT_MPHY_TRANSACTION_RESET_OFF;
```

```
`uvm_rand_send_with (req,  
    { req.dir_kind == svt_mphy_transaction::M_TX;  
      req.primitive_layer == svt_mphy_transaction::M_CTRL;  
      req.primitive_name == svt_mphy_transaction::RESET;  
      req.primitive_type == svt_mphy_transaction::REQUEST;  
    })  
get_response(rsp);
```

The time difference between RESET_ON SAP and RESET_OFF SAP should be greater than svt_mphy_configuration, that is, protocol defined reset_timer. If this condition is violated, then xactors will issue a warning.

8.6.2 Use Case to Assert and De-Assert RESET by Increasing the Reset Duration

In this use case, we can override the reset_timer value.

The following code snippet illustrates the use model:

```
svt_mphy_transaction req;  
`uvm_create(req)  
req.reset_duration = 1us;  
`uvm_rand_send_with (req,  
    { req.dir_kind == svt_mphy_transaction::M_TX;  
      req.primitive_layer == svt_mphy_transaction::M_CTRL;  
      req.primitive_name == svt_mphy_transaction::RESET;  
      req.primitive_type == svt_mphy_transaction::REQUEST;  
    })  
get_response(rsp);
```

In the above example, 1us, which is the reset_duration timer value will be used in place of svt_mphy_configuration defined reset_timer value to process RESET CTRL SAP.

8.6.3 Use Case Without any Overriding

This use case uses the default processing of RESET CTRL SAP as per protocol specifications. It uses svt_mphy_configuration defined reset_timer value to assert and de-assert RESET. The following code snippet illustrates the use model:

```
svt_mphy_transaction req;  
`uvm_do_with (req,  
    { req.dir_kind == svt_mphy_transaction::M_TX;  
      req.primitive_layer == svt_mphy_transaction::M_CTRL;  
      req.primitive_name == svt_mphy_transaction::RESET;  
      req.primitive_type == svt_mphy_transaction::REQUEST;  
    })  
get_response(rsp);
```

8.7 Configurable Prepare and Sync Length Through PREPARE SAP

The M-PHY Verification IP provides configurable Prepare and Sync length feature. This feature provides control to the user to change prepare and sync length during the start of a new burst, through PREPARE SAP. This feature is not applicable to Lane Management.

The following new fields have been added in transaction class to support this feature:

Variable Name	Variable Type	Description
enable_prepare_sync_override	bit	0: To disable this feature (Default) 1: To enable this feature
override_prepare_length	bit [4:0]	HS or PWM prepare length
override_sync_length	bit [5:0]	HS or PWM (G6 or G7) sync length
override_sync_range	bit	HS or PWM(G6 or G7) Sync range (0:fine,1:coarse)

If enable_prepare_sync_override is set to high in PREPARE REQUEST DATA SAP, then prepare and sync length for that burst will be calculated from override_prepare_length, override_sync_length and override_sync_range and not from values of prepare and sync attributes defined in the configuration attribute.

If bit enable_prepare_sync_override is not defined or set to zero in PREPARE REQUEST DATA SAP, then prepare and sync length will be calculated from attributes defined in the configuration attributes.

8.7.1 Use Case With Override Feature

The following code snippet illustrates the use model:

```
svt_mphy_transaction req;
`uvm_create(req)
req.enable_prepare_sync_override = 1'b1; //To enable this feature
`uvm_rand_send_with (req,
    { req.dir_kind == svt_mphy_transaction::M_TX;
      req.primitive_layer == svt_mphy_transaction::M_LANE;
      req.primitive_name == svt_mphy_transaction::PREPARE;
      req.primitive_type == svt_mphy_transaction::REQUEST;
      req.override_prepare_length== 10;    // User defined prepare length
      req.override_sync_length    == 5;    //User defined sync length
      req.override_sync_range    == 0;    //User defined sync range
    })
get_response(rsp);
```

8.7.2 Use Case Without any Overrides for Prepare Length, Sync Length and Sync Range

The following code snippet illustrates the use model:

```
svt_mphy_transaction req;
`uvm_do_with (req,
    { req.dir_kind == svt_mphy_transaction::M_TX;
      req.primitive_layer == svt_mphy_transaction::M_LANE;
      req.primitive_name == svt_mphy_transaction::PREPARE;
      req.primitive_type == svt_mphy_transaction::REQUEST;
    })
get_response(rsp);
```

If PREPARE SAP is sent as shown above, then it will be processed in the protocol defined manner, that is, prepare and sync length will be calculated from configuration attributes for the given burst.

8.8 Configurable Number of Fillers in a Transaction at Lane Management

The M-PHY Verification IP provides configurable number of fillers functionality. This functionality provides control to the user to configure how many times fillers should be inserted in one transaction. This feature is provided at Lane Management.

User can configure the number of fillers via setting the `num_of_filler` and `filler_position` fields in the testcase. The `filler_position` field is set as a dynamic array to insert filler multiple times in one transaction.

Number of fillers should satisfy the following equality to ensure that total `num_of_filler` < `payload_length`:
 $\text{num_fillers} \leq ((\text{payload_length or lanes}) - 1);$

The following code snippet illustrates the use model:

```
svt_mphy_mport_lm_transaction      req;

int      local_num_fillers  = 2;
int      local_filler_pos [] ;

for(int i=1; i <= num_fillers; i++) begin
    local_filler_pos[(i - 1)] = (lanes * i);
end

`uvm_do_with(req, {
    req.num_of_filler      == local_num_fillers;
    foreach ( local_filler_pos[k])
        req.filler_position[k] == local_filler_pos[k];
})

get_response(rsp);
```

For example, if `num_of_filler` = 2

`filler_position` = say 2 and 3

 active number of lanes = 4

The following fillers would be inserted twice in one transaction:

1. after 2nd payload byte
2. after 3rd payload byte

8.9 Enable or Disable External SYNC at Lane Management

The M-PHY Verification IP provides enable or disable external SYNC feature. This feature provides control to the user to configure whether the SYNC source is external or internal.

To configure whether the SYNC source is external or internal, user needs to set the `enable_ext_sync` field using `uvm_config_db` set method. For example:

```
uvm_config_db#(bit)::set(this,    "*.tx_sublink_sequencer",
"svt_mphy_lm_random_mode_virtual_sequence.enable_ext_sync",1);
```

where,

`enable_ext_sync` = 1 means SYNC source is external

enable_ext_sync = 0 means SYNC source is internal

User can refer or use the M-PHY VIP LM MPORT sequence

"svt_mphy_lm_random_mode_virtual_sequence" to enable or disable the external SYNC feature.

8.10 First SYNC Symbol Selection Between D10.5 and D26.5

According to *MIPI Alliance Specification for MPHY Version 4.1 - 01 December 2016*, Section 4.7.2.2, the default SYNC sequence shall be an alternating D10.5 and D26.5 pattern that may start with either of the two symbols.

To start the SYNC sequence from D26.5 symbol:

```
cfg.first_internal_sync_symbol = svt_mphy_types::SYNC_SYMBOL_D_26_5
```

To start the SYNC sequence from D10.5 symbol:

```
cfg.first_internal_sync_symbol = svt_mphy_types::SYNC_SYMBOL_D_10_5
```

Once the user has set this parameter, all the SYNC sequences with every new burst will start with the same parameter. This selection is applicable for sync_source = INTERNAL_SYNC only.

8.11 EVENT_MPHY_HIBERN8_ENTERED and EVENT_MPHY_HIBERN8_EXITED Events Usage

The M-PHY VIP model has inbuilt events EVENT_MPHY_HIBERN8_ENTERED and EVENT_MPHY_HIBERN8_EXITED for UVM, which can be used in the testbench to know about HIBERN8 state entry and exit. The EVENT_MPHY_HIBERN8_ENTERED event will be triggered when model enters HIBERN8 state and EVENT_MPHY_HIBERN8_EXITED event will be triggered when model exits from HIBERN8 state.

User can use these events as shown in the following code snippet:

```
bit hibern8_bit=1'b0;
genvar m;
generate
  for (m=0; m<`MAX_NUM_LANES; m=m+1) begin
    assign mrx_dut_phy_if[m].RX_Hibern8Exit_Type_I = hibern8_bit ;
  end
endgenerate

always begin
  wait (top_status != null);
  top_status.slave_tx_status.EVENT_MPHY_HIBERN8_EXITED.wait_trigger();
  hibern8_bit= 1'b1;
  $display("%t Debug: got event hibern8 exited",$time);
end

always begin
  wait (top_status != null);
  top_status.slave_tx_status.EVENT_MPHY_HIBERN8_ENTERED.wait_trigger();
  hibern8_bit= 1'b0;
  $display("%t Debug: got event hibern8 entered",$time);
end
```

8.12 EVENT_MPHY_DATA_STARTED and EVENT_MPHY_DATA_ENDED Events Usage

The M-PHY VIP model has inbuilt events `EVENT_MPHY_DATA_STARTED` and `EVENT_MPHY_DATA_ENDED` for UVM, which can be used in the testbench for burst start and end information. The `EVENT_MPHY_DATA_STARTED` event is triggered when the model starts the burst and `EVENT_MPHY_DATA_ENDED` event is triggered when model ends the burst.

You can use these events as per the following code snippet:

```
always begin
    wait (top_status != null);
    top_status.slave_tx_status.EVENT_MPHY_DATA_STARTED.wait_trigger();
    $display("%t Debug: got BURST started",$time);
end
always begin
    wait (top_status != null);
    top_status.slave_tx_status.EVENT_MPHY_DATA_ENDED.wait_trigger();
    $display("%t Debug: got event BURST ended",$time);
end
```

8.13 M-RX Capability and M-TX Configuration Relation

M-PHY VIP supports the relationship between M-TX Configuration and M-RX Capability Attributes as defined in Table 65 in *MIPI Alliance Specification for MPHY Version 4.1 - 01 December 2016*.

The Protocol Adapter layer needs to ensure that capabilities of MRX should have a value smaller than or equal to MTX configuration attributes.

8.14 Transfer Data on TX Sublink and RX Sublink

M-PHY VIP supports `tx_sublink_sequencer` to transfer data on TX sublink and `rx_sublink_sequencer` to transfer data on RX sublink.

- ❖ To transfer the data from TX sublink, user needs to connect the sequencer in the test-case as follows:
 - `uvm_config_db#(uvm_object_wrapper)::set(this, "env.tx_sublink_sequencer.main_phase", "default_sequence", svt_mphy_lm_hs_burst_linereset_local_reset_virtual_sequence::type_id::get());`
- ❖ To transfer the data from RX sublink, user needs to connect the sequencer in the test-case as follows:
 - `uvm_config_db#(uvm_object_wrapper)::set(this, "env.rx_sublink_sequencer.main_phase", "default_sequence", svt_mphy_lm_hs_burst_linereset_local_reset_virtual_sequence::type_id::get());`

8.15 Midstream De-Assertion of RX_PhyDORDY or TX_ProtDORDY

M-PHY VIP supports this feature that gives control to the user to de-assert `RX_PhyDORDY` or `TX_ProtDORDY` signal in the middle of a burst. This feature is not applicable to lane management layer.

The following new fields have been added in the transaction class to support this feature:

Variable name	Variable type	Description
phy_busy_bytes	rand int	0: Default behavior Greater than zero (>0): To de-assert RX_PhyDORDY or TX_ProtDORDY signal for given num of bytes.
busy_data	rand bit[9:0]	Symbol sent on TX_Symbol or RX_Symbol bus during phy_busy_bytes duration.

This feature can be enabled using any SYMBOL REQUEST DATA SAP during a burst. There is a reasonable constraint on phy_busy_bytes and busy_data for value zero, so that in a normal case they don't affect anything. There would not be any effect if phy_busy_bytes is set to any value in PREPARE, SYNC, BURSTEND SAP or in CTRL SAPs.

Data value which will be driven on TX_Symbol or RX_Symbol bus when TX_ProtDORDY or RX_PhyDORDY is zero is defined by transaction variable busy_data which has default value zero. It is the responsibility of PA layer to give appropriate value of phy_busy_bytes in High speed and Low speed mode.

There exists a reasonable constraint on variable busy_data for value zero.

This feature has different behavior on RMMI interfaces. On serial interface it does not have any effect.

8.15.1 MTX as RMMI REMOTE

In the case when MTX acts as RMMI REMOTE interface, RX_PhyDORDY will get de-asserted before transmitting the SYMBOL REQUEST DATA SAP in which phy_busy_bytes variable is defined. The phy_busy_bytes variable will de-assert the RX_PhyDORDY signal for the given number of bytes.

SYMBOL REQUEST SAP which has phy_busy_bytes greater than zero will be ended only after all the delay present in it got driven on lines.

8.15.2 MTX as RMMI LOCAL

This feature can be used to insert FILLERS on serial lines. When phy_busy_bytes is greater than zero in any SYMBOL REQUEST SAP, then it will de-assert TX_ProtDORDY signal which indicates TX Phy to insert fillers on serial lines.

The value given to busy_data variable will be present on TX_Symbol bus till TX_ProtDORDY is zero, then only SYMBOL REQUEST SAP will be driven.

8.15.3 Effect on Other Signals

This feature of M-PHY VIP has the following effect on the other signals:

- ❖ TX_Burst or RX_Burst signal will remain asserted during this time.
- ❖ RX_DataNCtrl or TX_DataNCtrl signal can have any value because there is no new data present on SYMBOL bus.
- ❖ TX_Symbol or RX_Symbol bus will carry data value defined in busy_data variable till RX_PhyDORDY or TX_ProtDORDY is zero.

8.15.4 Use Case to De-Assert RX_PhyDORDY or TX_ProtDORDY Through DATA SAP

The following code snippet illustrates the use model:


```
`uvm_create(req)
req.phy_busy_bytes == 5; // Delay of 5 bytes
`uvm_rand_send_with(req, {
    req.dir_kind      == M_TX;
    req.primitive_layer == M_LANE;
    req.primitive_name == SYMBOL;
    req.primitive_type == REQUEST;
    req.data_n_ctrl    == 0;
    req.marker_n_filler == 0;
    req.data_value     == 1;
    req.busy_data      == 4; // This data will be driven on Symbol
                             bus for 5 bytes as defined in
                             phy_busy_bytes
    })
get_response(rsp);
```

If the user doesn't want to send any busy_data. By default, zero value will be driven on lines.

The following is the Use Case illustrating this condition.


```
`uvm_create(req)
req.phy_busy_bytes == 1; // Delay of 5 bytes
`uvm_rand_send_with(req, {
    req.dir_kind      == M_TX;
    req.primitive_layer == M_LANE;
    req.primitive_name == SYMBOL;
    req.primitive_type == REQUEST;
    req.data_n_ctrl    == 0;
    req.marker_n_filler == 0;
    req.data_value     == 5;
    })
get_response(rsp);
```

8.16 PWM Gear0 Requirements

M-PHY VIP supports PWM signaling with FIXED_MINOR format for Gear0. According to *MIPI Alliance Specification for MPHY Version 4.1 - 01 December 2016*, For the FIXED-MINOR format, the duration of $T(PWM_MINOR)$ is specified as absolute time duration, while $T(PWM_MAJOR)$ scales with the bit period.

For $T(PWM_MINOR)$ duration, following field is defined in `svt_mphy_configuration` class. Now this new field will be used for the minor portion calculation in `PWM_G0`.

Table 8-4 $T(PWM_MINOR)$ Duration

Name	Type	Description	Constraint
pwm_g0_minor_period	real	Time period for PWM_G0 minor portion ($TPWM_MINOR$).  Note Default value is 50ns.	Valid range for this field is from 37ns to 111ns.

For example,

```
tx_cfg.pwm_g0_minor_period = 100ns;
```

or

```
tx_cfg.set_timer("pwm_g0_minor_period",100, "ns");
```

PWM_MINOR ratio remains fixed to 1/3 for PWM_G1 through PWM_G7.

**Note**

Since the usage of configuration field `fixed_minor_factor` is deprecated for PWM_G0 from release 1.27a onwards. Thus, configuration parameter `fixed_minor_factor` (default value 0.34) is used to set the ratio of FIXED_MINOR portion for PWM_G1 to PWM_G7. However, for testing purpose if user wants to modify the FIXED_MINOR ratio in FIXED-RATIO format (PWM_G1 to PWM_G7) then it can be implemented by changing the value of configuration field `fixed_minor_factor`. After implementing this modification, check failures are expected from the serial monitor.

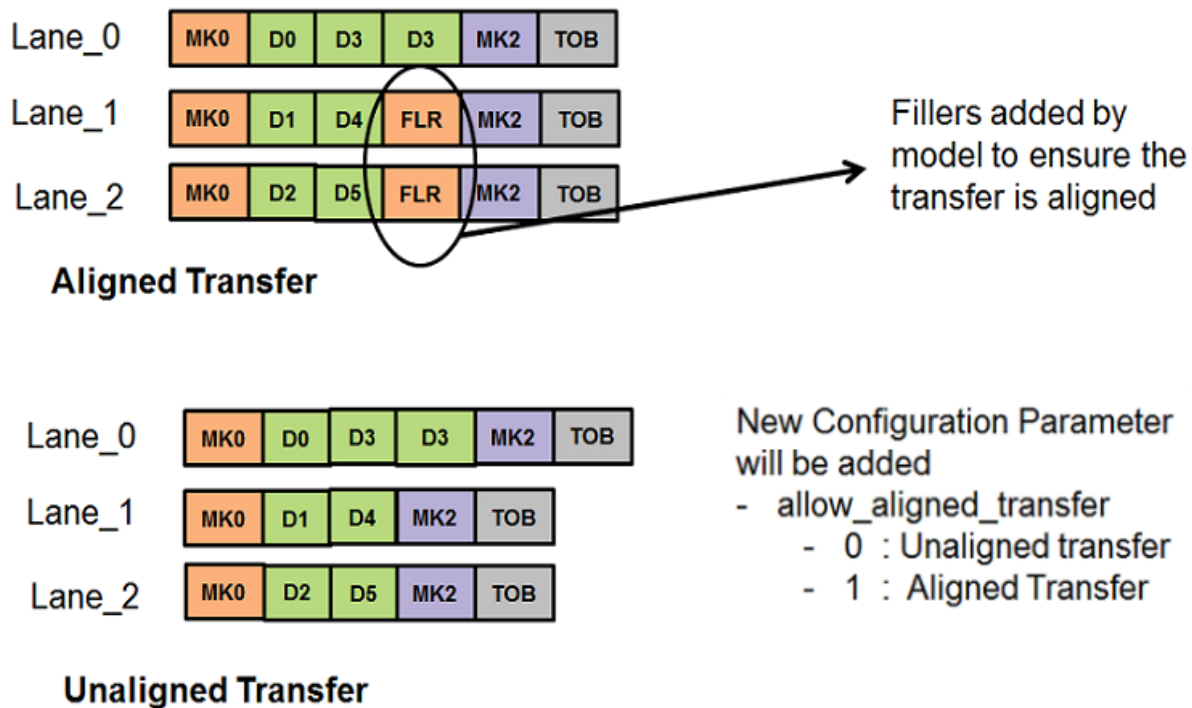
8.17 Aligned and Unaligned Transfers

M-PHY Verification IP supports the functionality for aligned and unaligned transfers. A configuration parameter `allow_aligned_transfer` is added to allow the user to control the aligned versus unaligned transfers. By default, this parameter is set to aligned transfer.

```
constraint reasonable_allow_aligned_transfer
{
    allow_aligned_transfer == 1'b1;
}
```

To enable unaligned transfers, turn this constraint off and pass `allow_align_transfer = 0`.

[Figure 8-1](#) illustrates the aligned and unaligned transfer.

Figure 8-1 Aligned and Unaligned Transfers

8.18 HIBERN8 Time Capability Configuration

As mentioned in *MIPI Alliance Specification for MPHY Version 4.1 - 01 December 2016* (Section 4.7.1.3 paragraph 216), "The Protocol Layer shall not request a MODULE exit HIBERN8 before a minimum period in HIBERN8 of THIBERN8, which is defined as the larger of local **Tx_Hibern8Time_Capability** and remote **RX_Hibern8Time_Capability**."

While configuring "hibern8time_capability" for M-PHY VIP, user needs to make sure that the configured value should be "the larger of local **Tx_Hibern8Time_Capability** and remote **RX_Hibern8Time_Capability**".

8.19 Dynamic Change in Active Lanes

M-PHY VIP supports the functionality for dynamically changing the number of Active Lanes in the Lane Management agent.

M-PHY VIP (multi-lane svt_mphy_mport_lm_agent) will allow the testbench to dynamically change the number of lanes for its TX and RX paths during the simulation. The dynamically changed active lanes can be equal to or less than the configured lanes during the build_phase of the UVM environment.

The change in number of active lanes will be applied only during the SAVE states of the M-PHY FSM.

The testbench is required to re-configure the number of active lanes of other VIP agents that sit across the M-PHY link. Currently, the VIP requires the following number of active lanes (this must be up to the number of M-PHY instances instantiated in testbench top) using the svt_mphy_mport_lm_agent_configuration class:

```

/** Set the number of active lanes Master TX sub link*/
master_cfg.tx_cfg.active_lane = 2;

/** Set the number of active lanes Master RX sub link*/
master_cfg.rx_cfg.active_lane = 2;

/** Set the number of active lanes Slave TX sub link*/
slave_cfg.tx_cfg.active_lane = 2;

/** Set the number of active lanes Slave RX sub link*/
slave_cfg.rx_cfg.active_lane = 2;

```

The testcase may override this value in the build_phase.



Note

The testbench sets the number of TX lanes of the master_cfg equal to the number of RX lanes of the slave_cfg and vice-versa. This requirement will continue to hold.

With this proposed enhancement, the VIP will allow the testbench to set any one or more lanes as active during simulation.

For Example,

If the build_phase is configured with two active lanes on the TX-SUBLINK marked as lane0 and lane1, the number of active lanes can be dynamically modified to using two lanes (no change from initial configuration) or one lane. When one lane is selected, lane0 would be selected by default.

8.19.1 Use Model

M-PHY VIP provides a reconfigure method in the multi-lane svt_mphy_mport_lm_agent class that will take a handle of the svt_mphy_mport_lm_agent_configuration class.

```

env.master_agent.reconfigure(shared_cfg.master_cfg);
env.slave_agent.reconfigure(shared_cfg.slave_cfg);

```

The user needs to modify the active_lane field of the LM configuration and then call reconfigure method from the testbench. The VIP will apply new settings when exiting from a SAVE state of the M-PHY FSM. After this, the transmissions will be on the reconfigured lanes.



Note

The reconfigured active lanes can only be from amongst the lanes that were set in the build_phase. The VIP will exit with a FATAL error, if the number of lanes after reconfiguration is zero.

Before reconfiguration, user needs to make sure that master and slave both are in the same SAVE (HIBERN8 or SLEEP or STALL) State.

For example:

Start with two lanes (configured during build_phase) STALL -> HS_BURST-> STALL-> **(a)** reconfigure to one lane -> SLEEP-> LS_BURST -> SLEEP-> **(b)** reconfigure again to two lanes -> LS_BURST -> SLEEP

At time **(a)**, both lanes are in HS_MODE. After reconfiguration, user needs to send LS_MODE CTRL transaction to configure single lane in LS_MODE.



It would only configure single lane in LS_MODE.

At time **(b)**, single lane would be in LS_MODE. After reconfiguration, user needs to send HS_MODE CTRL transaction to configure both lanes in LS_MODE.

Following is the code snippet of reconfigure() function call for TX-SUBLINK (Master-TX & Slave-RX) through testbench:

```
wait(test_top.top_status.master_tx_status.fsm_state == svt_mphy_types::SLEEP);
    `uvm_note("TX_FSM_STATE", "Reached SLEEP State, reconfiguring to 1 lane");
wait(test_top.top_status.slave_rx_status.fsm_state == svt_mphy_types::SLEEP);
    `uvm_note("RX_FSM_STATE", "Reached Sleep State, reconfiguring to 1 lane");
    shared_cfg.master_cfg.tx_cfg.active_lane = 1;
    shared_cfg.passive_master_cfg.tx_cfg.active_lane = 1;
    shared_cfg.slave_cfg.rx_cfg.active_lane = 1;
    shared_cfg.passive_slave_cfg.rx_cfg.active_lane = 1;
env.master_agent.reconfigure(shared_cfg.master_cfg);
env.passive_master_agent.reconfigure(shared_cfg.passive_master_cfg);
env.slave_agent.reconfigure(shared_cfg.slave_cfg);
env.passive_slave_agent.reconfigure(shared_cfg.passive_slave_cfg);
```

The svt_mphy_mport_lm_agent_configuration configuration class has a new field active_lane_positions.

```
/* Current active lanes of agent */
bit [12:0] active_lane_positions;
```

This bit-vector defines the position of the active lanes. The bit position with a binary value one is considered as an active position and with binary value zero is considered as an inactive position.

The default value of this vector is set to all one, that is, all lanes are considered to be active.

The user can reconfigure the VIP to set the number of active lanes to a value less than active_lane in build time and choose the active lanes using this vector. For example, if active_lane at build time is four, the user may reconfigure the active lanes to two and set this vector value to specify which two lanes out of total four lanes are active. A value of 0101 will specify that lanes zero and two are active.

The user can modify this field of the LM agent configuration. For example, to set Lane1 as active, set the value to 0010 and then call reconfigure method from the testbench. The VIP will apply new settings when exiting from a SAVE state of the M-PHY FSM. After this, the transmissions will be on the reconfigured lanes.

The number of bits which are one in vector active_lane_positions must be more than or equal to active_lane. The VIP will exit with FATAL error, when number of one's in vector active_lane_positions are less than active_lane.

8.19.1.1 Active Lane Switching

The M-PHY Verification IP provides the support for Active Lane switching. This feature provides control to the user to connect any TX active lane to any RX active lane on reconfiguration.

To configure which TX lane should be connected to which RX lane, user needs to configure "active_lane_positions" configuration 12-bit vector using svt_mphy_mport_lm_agent_configuration class.

The following is the code snippet of `reconfigure()` function call for TX-SUBLINK (Master-TX & Slave-RX) through testbench which connects TX active lane 1 to RX active lane 2:

```
wait(test_top.top_status.master_tx_status.fsm_state == svt_mphy_types::SLEEP);
`uvm_note("TX_FSM_STATE", "Reached SLEEP State, reconfiguring to 1 lane");

wait(test_top.top_status.slave_rx_status.fsm_state == svt_mphy_types::SLEEP);
`uvm_note("RX_FSM_STATE", "Reached Sleep State, reconfiguring to 1 lane");

shared_cfg.master_cfg.tx_cfg.active_lane = 1;
shared_cfg.master_cfg.tx_cfg.active_lane_positions = 12'b1111_1111_0010;
shared_cfg.passive_master_cfg.tx_cfg.active_lane = 1;
shared_cfg.passive_master_cfg.tx_cfg.active_lane_positions = 12'b1111_1111_0010;
shared_cfg.slave_cfg.rx_cfg.active_lane = 1;
shared_cfg.slave_cfg.rx_cfg.active_lane_positions = 12'b1111_1111_0100;
shared_cfg.passive_slave_cfg.rx_cfg.active_lane = 1;
shared_cfg.passive_slave_cfg.rx_cfg.active_lane_positions = 12'b1111_1111_0100;

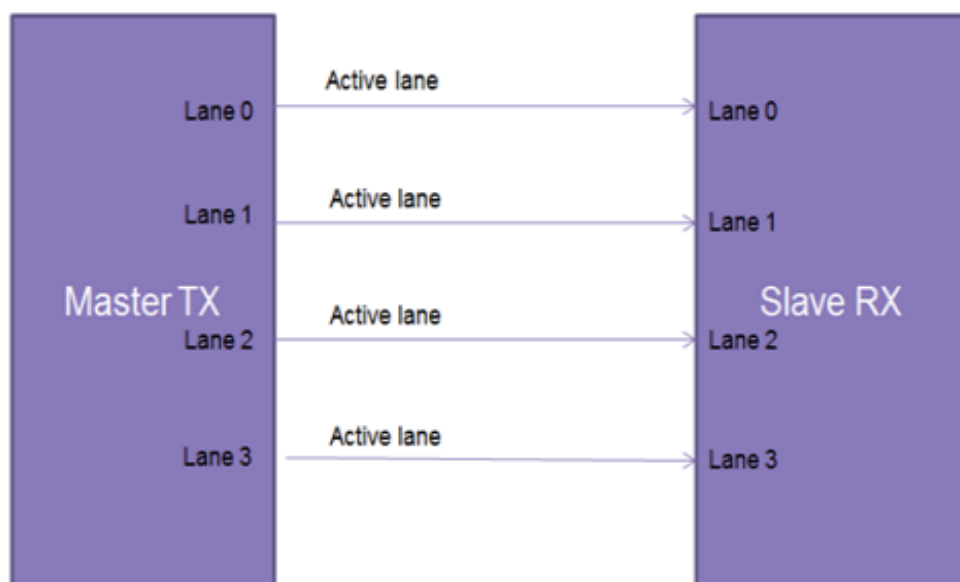
env.master_agent.reconfigure(shared_cfg.master_cfg);
env.passive_master_agent.reconfigure(shared_cfg.passive_master_cfg);
env.slave_agent.reconfigure(shared_cfg.slave_cfg);
env.passive_slave_agent.reconfigure(shared_cfg.passive_slave_cfg);
```

For more information on 12 bit vector "active_lane_positions", refer to ["Use Model"](#).

This figure shows the active lane configuration before reconfiguration with Lane Switching feature.

Figure 8-2 Active Lane Configuration Before Reconfiguration

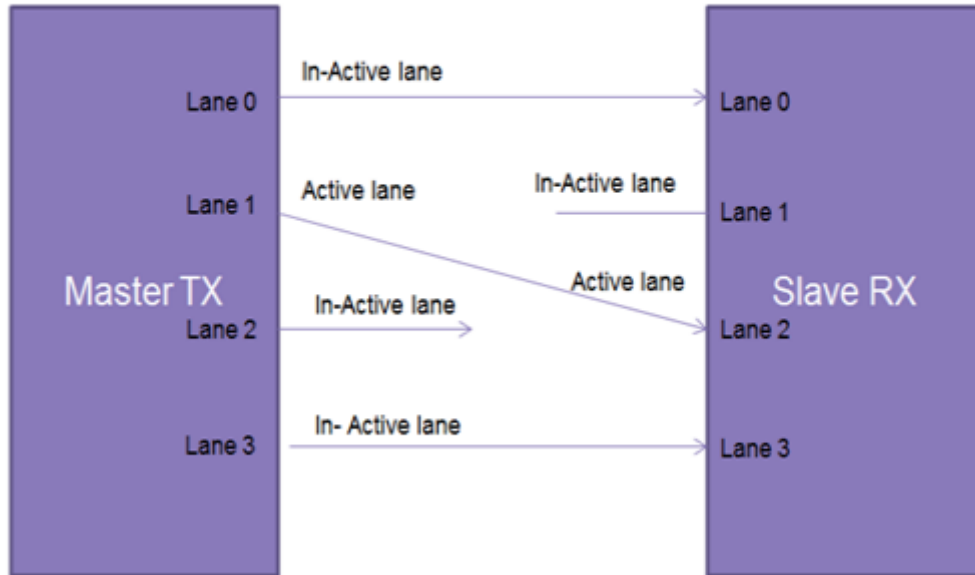
Active Lane =4, before reconfiguration



The following figure shows the active lane configuration after reconfiguration with Lane Switching feature.

Figure 8-3 Active Lane Configuration After Reconfiguration

Active Lane = 1, after reconfiguration



Note

MPHY VIP supports the Lane switching feature only for single lane, that is, reconfiguring active lanes to 1.

8.19.1.2 Use Model Restrictions

This version of the M-PHY VIP will allow change of number of active lanes through reconfigure method only. Any other configuration changes will not be applicable in this release.

It is the duty of the testbench to reconfigure the master and slave agents correctly so that a SUBLINK is correctly configured with equal TX and RX lanes. The simulation behavior will be incorrect in case this is not followed correctly. The VIP will not be able to check that the settings are done correctly or not (the testbench however can), so, simulation errors or hang can be expected.

8.20 Bypass Mode

M-PHY Verification IP supports the Bypass Mode functionality.

In case of Bypass mode,

- ❖ When 8b10b encoding is bypassed, the j bit is transmitted first.

If protocol drives raw data on RMMI 10 bit interface or on SYMBOL.request SAP, it should be as per the following:

- a (MSB)**
- b**
- c**

d
e
i
f
g
h
j (LSB)

The data that appears on serial lines is as follows:

j h g f i e d c b a

- ❖ At receiver end, the LSB of RX_Symbol shall correspond to the earliest received bit.

On receiving j h g f i e d c b a on serial lines, M-RX must transmit j on LSB of RMMI lines is as follows:

a (MSB)
b
c
d
e
i
f
g
h
j (LSB)



Note

Bypass control sequences are provided in sequence collection file for MPORT LM. The sequence names are `svt_mphy_lm_ls_hs_ls_bypass_8b10b_data_transaction_virtual_sequence` and `svt_mphy_lm_ls_hs_ls_bypass_8b10b_data_transaction_rx_sublink_virtual_sequence`. To pass CTRL SAPs, use `svt_mphy_tx_ctrl_bypass_8b10b_sequence` and `svt_mphy_tx_ctrl_bypass_8b10b_sequence` sequences

8.20.1 Bypass Mode Limitations

The following limitations exist, which a user has to strictly follow for correct data transfer in BYPASS Mode:

- ❖ TX sync_length must be equal to RX sync_length_capability to avoid data integrity failures.
- ❖ TX prepare_length must be equal to RX prepare_length_capability.
- ❖ Control codes shall not be transmitted in one HOT coding format as the raw data.
- ❖ Transmitted raw data symbols must have an edge sensitivity of four or five or more, that is, number of transitions from zero to one and from one to zero must be at least four.
- ❖ A burst can only have one frame and demarcator shall not be used for passing LM transactions, that is, for every LM transaction, end_of_burst =1 must be sent.

**Note**

First two points are not applicable if configuration field `mk0_in_bypass_mode` is set for MRX configuration.

8.20.2 Marker 0 in Bypass mode

To get Symbol boundary alignment in bypass mode a configuration parameter is added in `svt_mphy_configuration` class.

**Note**

Its default value is set as 0.

Table 8-5 Variable

Variable Name	Variable Type
<code>mk0_in_bypass_model</code>	bit

8.20.2.1 Description and Use Case:-

If this bit is set for `MTX_SERIAL` and `MTX_RMMI_LOCAL` (MTX Controller) then these drivers automatically insert 8b10b encoded MK0 symbol

(`SVT_MPHY_MK0_SYMBOL_FOR_BYPASS_MODE (0fa)`) in bypass mode. By default `'h0fa` is sent. This symbol is generated from VIP driver so there is no change required in sequences. In bypass mode all the symbols are sent with LSB as first bit so `'h0fa` symbol will be sent as `'h17c` on serial lines.

If you want to send `'h305` as MK0 symbol then this can be achieved by setting this value for replaceable `define SVT_MPHY_MK0_SYMBOL_FOR_BYPASS_MODE`

```
tx_cfg.mk0_in_bypass_mode = 1;
```

If this bit is set for MRX SERIAL configuration then MRX first wait for MK0 symbol to get a lock on SYMBOL boundary, then start sending DATA to PA layer along with received MK0. In this case MRX will wait for `'h17c` or `'h283`.

```
rx_cfg.mk0_in_bypass_mode = 1;
```

8.21 Configuration Register Read Value Check

The M-PHY Verification IP provides configuration register read value check via the monitor which compares the PHY attribute read value on RMMI lines with the controller effective configuration bank value on every CFGGET request (register read command).

Whenever there is a mis-match between PHY attribute read value and Controller effective configuration bank value, the `invalid_attr_val_check` checker gets fired.

8.22 Test Modes

M-PHY Verification IP supports the Test Modes functionality. This section describes the following for Test Modes:

- ❖ “Test Mode Patterns”
- ❖ “Test Mode Configuration”
- ❖ “Data Pattern Transmission Modes”

8.22.1 Test Mode Patterns

There are two types of CJTPAT patterns from the PA layer, one is defined by *MIPI UniPro specification Version 1.41.00 4-May-2012* and the other is defined in *MIPI Alliance Specification for Low Latency Interface 2 (LLI-2), Draft Version 0.1 Revision 05 – 31 August 2012*. Hence, the `svt_mphy_types::data_pattern` is extended for CJTPAT_A and CJTPAT_B values for Unipro and LLI CJTPAT patterns respectively.

Similarly, there are two types of CRPAT patterns from the PA layer, one is defined by *MIPI UniPro specification Version 1.41.00 4-May-2012* and the other is defined in *MIPI Alliance Specification for Low Latency Interface 2 (LLI-2), Draft Version 0.1 Revision 05 – 31 August 2012*. Hence, the `svt_mphy_types::data_pattern` is extended for CRPAT_A and CRPAT_B values for Unipro and LLI CRPAT patterns respectively.

8.22.1.1 Generating CJTPAT Test Mode Patterns

To generate CJTPAT test mode patterns, user needs to perform the following steps:

- ❖ Set the default sequence to `svt_mphy_lm_hs_ls_data_transaction_loopback_CJTPAT_virtual_sequence`.
- ❖ Set the sequence `svt_mphy_types::data_pattern == CJTPAT_A`
Or
Set the sequence `svt_mphy_types::data_pattern == CJTPAT_B`
- ❖ Set the `repeat_count` of the sequence same as the number of `cfg.active_lane`.
- ❖ Actual sequence which generates the patterns is `svt_mphy_tx_sap_lm_loopback_cjtpat_data_pattern_sequence` which is embedded in the above virtual sequence with LM loopback mode.

8.22.1.2 Generating CRPAT Test Mode Patterns

To generate CRPAT test mode patterns, user needs to perform the following steps:

- ❖ Set the default sequence to `svt_mphy_lm_hs_ls_data_transaction_loopback_CRPAT_virtual_sequence`.
- ❖ Set the sequence `svt_mphy_types::data_pattern == CRPAT_A`
Or
Set the sequence `svt_mphy_types::data_pattern == CRPAT_B`
- ❖ Set the `repeat_count` of the sequence same as the number of `cfg.active_lane`.
- ❖ Actual sequence which generates the patterns is `svt_mphy_tx_sap_lm_loopback_crpat_data_pattern_sequence` which is embedded in the above virtual sequence with LM loopback mode.

8.22.1.3 Generating KAT Test Mode Patterns

To generate KAT test mode patterns, user needs to perform the following steps:

- ❖ Set the default sequence to `svt_mphy_lm_hs_ls_data_transaction_loopback_KAT_virtual_sequence`.

- ❖ Set the sequence `svt_mphy_types::data_pattern == KAT_1`
Or
Set the sequence `svt_mphy_types::data_pattern == KAT_3`
- ❖ Set the `repeat_count` of the sequence same as the number of `cfg.active_lane`.
- ❖ Actual sequence which generates the patterns is `svt_mphy_tx_sap_lm_loopback_kat_data_pattern_sequence` which is embedded in the above virtual sequence with LM loopback mode.

8.22.1.4 Generating D10_2 Test Mode Patterns

To generate D10_2 test mode patterns, user needs to perform the following steps:

- ❖ Set the default sequence to `svt_mphy_lm_hs_ls_data_transaction_loopback_D10_2_virtual_sequence`.
- ❖ Set the `repeat_count` of the sequence same as the number of `cfg.active_lane`.
- ❖ Set the `payload_length` of the sequence which would decide as to how many times the patterns should be repeated.
- ❖ Actual sequence which generates the patterns is `svt_mphy_tx_sap_lm_loopback_D10_2_data_pattern_sequence` which is embedded in the above virtual sequence with LM loopback mode.

8.22.1.5 Generating D30_7 Test Mode Patterns

To generate D30_7 test patterns, user needs to perform the following steps:

- ❖ Set the default sequence to `svt_mphy_lm_hs_ls_data_transaction_loopback_D30_7_virtual_sequence`.
- ❖ Set the `repeat_count` of the sequence same as the number of `cfg.active_lane`.
- ❖ Set the `payload_length` of the sequence which would decide as to how many times the patterns should be repeated.
- ❖ Actual sequence which generates the patterns is `svt_mphy_tx_sap_lm_loopback_D30_7_data_pattern_sequence` which is embedded in the above virtual sequence with LM loopback mode.

8.22.1.6 Generating Random Data Payload Bytes

To send random data payload bytes, user only needs to set `payload_length` using testbench where `payload_length` represents the total number of bytes in a single transaction.



Note You can refer to any M-PHY VIP LM MPORT sequence for this feature.

8.22.1.7 Generating Data Pattern With Bypass 8b10b Encoding or Decoding Enabled

To enable bypass 8b10b encoding, set the `init_bypass_8b10b_enable` configuration parameter to one as mentioned above in this section. Refer section [Step 8.22.1.1](#) for details regarding the use model of this feature.

8.22.2 Test Mode Configuration

M-PHY Verification IP supports the Test Modes functionality wherein it has the following two configurations:

- ❖ “Near-End Loopback”
- ❖ “Far-End Loopback”

Loopback mode runs in both LS and HS modes. To enable loopback mode, user needs to set a configuration parameter `mphy_loopback_mode_enable` to 1 using `svt_mphy_mport_lm_agent_configuration` class



Note Loopback mode is only supported with Symbol Length equal to 10bit.

For example,

```
svt_mphy_mport_lm_agent_configuration    master_cfg;
svt_mphy_mport_lm_agent_configuration    slave_cfg;

/** Set the Loopback mode for Master TX */
master_cfg.tx_cfg.mphy_loopback_mode_enable = 1'b1;

/** Set the Loopback mode for Master RX */
master_cfg.rx_cfg.mphy_loopback_mode_enable = 1'b1;

/** Set the Loopback mode for Slave TX */
slave_cfg.tx_cfg.mphy_loopback_mode_enable = 1'b1;

/** Set the Loopback mode for Slave RX */
slave_cfg.rx_cfg.mphy_loopback_mode_enable = 1'b1;
```

8.22.2.1 Near-End Loopback

In Near-end loopback mode, 8b10b encoding or decoding may be bypassed.

To bypass 8b10b encoding or decoding:

- ❖ User needs to set a configuration parameter `init_bypass_8b10b_enable` to 1 using `svt_mphy_mport_lm_agent_configuration` class.

For example,

```
svt_mphy_mport_lm_agent_configuration    master_cfg;
svt_mphy_mport_lm_agent_configuration    slave_cfg;

/** Enable/disable bypass 8b10b encoding for Master TX */
master_cfg.tx_cfg.init_bypass_8b10b_enable = 1'b1;

/** Enable/disable bypass 8b10b encoding for Master RX */
master_cfg.rx_cfg.init_bypass_8b10b_enable = 1'b1;

/** Enable/disable bypass 8b10b encoding for Slave TX */
slave_cfg.tx_cfg.init_bypass_8b10b_enable = 1'b1;

/** Enable/disable bypass 8b10b encoding for Slave RX */
slave_cfg.rx_cfg.init_bypass_8b10b_enable = 1'b1;
```

**Note**

You can start the burst directly without issuing register write command in configuration registers. (OR)

You can issue TX_BYPASS_8B10B_Enable =1 and RX_BYPASS_8B10B_Enable =1 through control transactions in SAVE state.

**Note**

In both the cases (bypass mode enabled and disabled), you need to send 8bit data through the testbench. In the case of bypass mode enabled (that is, 8b10 bit encoding is OFF), to ensure that data has sufficient edge density, the M-PHY VIP internally does 8b10b encoding on TX side and 8b10b decoding on RX side only in Loopback mode.

8.22.2.2 Far-End Loopback

In Far-End loopback mode, 8b10b encoding is always enabled.

8.22.3 Data Pattern Transmission Modes

The M-PHY Verification IP provides CJTPAT and CRPAT data or test pattern transmission in the following two modes or ways:

- ❖ “Burst Mode”
- ❖ “Continuous Mode”

8.22.3.1 Burst Mode

In Burst mode, the test pattern starts with MK0 and ends with MK2 and then goes to SAVE state (SLEEP or STALL). To send a transaction in burst mode, user needs to set the transaction field end_of_burst to one.

The code snippet for this mode is:

```
\uvm_do_with(req,
{
    req.end_of_burst    == 1;
})
```

Here, req is the svt_mphy_mport_lm_transaction transaction object. User may also use or refer to M-PHY LM MPORT virtual sequence svt_mphy_lm_hs_ls_data_transaction_loopback_CJTPAT_virtual_sequence.

8.22.3.2 Continuous Mode

In continuous mode, the test pattern starts with MK0 and a new Test Pattern begins immediately with MK0 when the previous one ends, as shown in [Figure 8-4](#).

Figure 8-4 Continuous Mode



By default, the M-PHY VIP supports burst mode transmission. To enable or use continuous mode, user needs to set the configuration parameter mphy_continuous_mode_enable to one using svt_mphy_mport_lm_agent_configuration class.

For example:

```
svt_mphy_mport_lm_agent_configuration master_cfg;
svt_mphy_mport_lm_agent_configuration slave_cfg;

/** Set for Master TX */
master_cfg.tx_cfg. mphy_continuous_mode_enable = 1'b1;

/** Set for Master RX */
master_cfg.rx_cfg. mphy_continuous_mode_enable= 1'b1;

/** Set for Slave TX */
slave_cfg.tx_cfg. mphy_continuous_mode_enable= 1'b1;

/** Set for Slave RX */
slave_cfg.rx_cfg. mphy_continuous_mode_enable= 1'b1;
```

You can also use or refer to the following M-PHY LM MPORT virtual sequences:

- ❖ svt_mphy_lm_hs_ls_data_transaction_loopback_CJTPAT_virtual_sequence
- ❖ svt_mphy_lm_hs_ls_data_transaction_loopback_CRPAT_virtual_sequence



Note

The information to enable the continuous mode is passed through tx_cfg of the virtual sequence to insert the end_of_burst. For example, if the payload_length is set to three, the actual CJTPAT sequence would have three CJTPAT patterns in continuous mode and would insert the end_of_burst after the third one. In continuous mode, the end_of_burst would be automatically appended at the end of payload_length times test patterns.

8.23 Line Control Command (LCC) with Media Convertor

The M-PHY FSM switches to LINE_CFG state based on the following parameters:

- ❖ TX_LCC_Enable
 - ◆ By default, TX_LCC_Enable = 1. If the user has set this bit to 0 by issuing CFGSET on TX_LCC_Enable, user must set this bit to 1 by re-issuing CFGSET on TX_LCC_Enable.
 - ◆ If the testcase starts from DISABLED state or a RESET is issued in any state, reset_lcc_enable overrides TX_LCC_Enable bit. By default, reset_lcc_enable = 1. User sets the bit to zero in the testbench or testcase to skip LINE-CFG state.
 - ◆ If the testcase starts from any state other than DISABLED state, init_lcc_enable overrides TX_LCC_Enable bit. By default, init_lcc_enable = 1. User sets the bit to zero in the testbench or testcase to skip the LINE-CFG state.
- ❖ CFGREADY.request SAP
 - ◆ To switch the M-PHY VIP to LINE_CFG state, protocol layer has to send CFGREADY.request SAP in the middle of burst.

8.23.1 TX_LCC_Sequencer Configuration Settings

The following are the configuration settings for TX_LCC_Sequencer:

- ❖ By default, TX_LCC_Sequencer = 0.

If you have not enabled LCC_READ and LCC_WRITE by setting TX_LCC_Sequencer bits and LINE_CFG state machine is enabled (issuing CFGREADY.request SAP in the middle of burst and TX_LCC_Enable = 1), M-PHY FSM moves to LINE_CFG. It will transition to LINE_INIT state and then to LCC_MODE state and then exit the LINE_CFG sub-state machine.

- ❖ If TX_LCC_Sequencer[0] is set to 1, READ_CAPABILITY is set to 8'hFF (all 1's).

In case OMC is not present in the simulation environment, M-PHY VIP will drive READ_CAPABILITY if user sets `cfg.omc_present = 0`. LCC_READ operation will not happen if PWM_Gear is not PWM_G1. M-PHY VIP drops that LINE_CFG LCC_READ state transition request (on receiving CFGREADY.request SAP in the middle of burst) and resets this bit to 0. It also issues an error.

- ❖ If TX_LCC_Sequencer[1] is set to 1, READ_MFG_INFO is set to 8'hFF (all 1's).

In case OMC is not present in the simulation environment, M-PHY VIP drives READ_MFG_INFO if user sets `cfg.omc_present = 0`. LCC_READ operation will not happen if PWM_Gear is not PWM_G1. M-PHY VIP drops that LINE_CFG LCC_READ state transition request (on receiving CFGREADY.request SAP in the middle of burst) and resets this bit to 0. It also issues an error.

- ❖ If TX_LCC_Sequencer[2] is set to 1, READ_VEND_INFO is set to 8'hFF (all 1's).

In case OMC is not present in the simulation environment, M-PHY VIP drives the READ_VEND_INFO if user sets `cfg.omc_present = 0`. LCC_READ operation will not happen if PWM_Gear is not PWM_G1. M-PHY VIP drops that LINE_CFG LCC_READ state transition request (on receiving CFGREADY.request SAP in the middle of burst) and resets this bit to 0. It also issues an error.

- ❖ If TX_LCC_Sequencer[7] is set to 1, LCC_WRITE command is issued. M-PHY VIP sets this bit to 0, when it enters LCC_MODE state.

**Note**

LCC feature works fine for 10 and 20 bit RMMI. It is not verified on 40 bit RMMI. It works on M-PHY TX and RX MODEL also. Monitors are enabled for LCC feature. For more information on checks, see the VC Verification IP for MIPI M-PHY Class Reference Guide.

8.23.2 Switching from LS_MODE to HS_MODE and Vice Versa

To switch from LS_MODE to HS_MODE, the following settings must be applied in the LS_BURST transaction:

- ❖ Pass a CTRL SAP transaction to change the TX_Mode to HS_MODE. It is passed in the middle of burst B1 and the effect appears in the next burst B2.
- ❖ Pass a CTRL SAP transaction to select the HS_GEAR in burst B1. It is not mandatory thus, it takes the default value if the transaction is not issued.
- ❖ Pass a CFGREADY.request SAP in the middle of burst B1.

**Note**

A proper LCC_MODE is observed before exiting LINE_CFG state in burst B1. It reflects a LCC_MODE value with reference to HS gears and not PWM gears if switching from LS_MODE to HS_MODE.

8.23.3 Switching from LS_MODE to HS_MODE via HIBERN8 state

To switch from LS_MODE to HIBERN8 to HS_MODE, the following settings must be applied in the LS_BURST transaction:

- ❖ Pass a CTRL SAP transaction to enter HIBERN8 state in the middle of LS_BURST (TX_Hibern8_Control == Enter)
- ❖ Pass a CTRL SAP transaction to change the TX_Mode to HS_MODE. It is passed in the middle of Burst. Its effect will appear in next burst.
- ❖ A CTRL SAP transaction to select the HS_GEAR can't be issued. It will take the default value if the transaction is not issued.
- ❖ Pass a CFGREADY.request SAP in the middle of Burst.

8.23.4 LCC_READ operation when OMC is not present

To verify OMC functionality when it is not present between M_TX and M-RX modules, configure `cfg.omc_present = 0`

By default, `cfg.omc_present = 1`.

By setting `cfg.omc_present = 0`, the OMC TX functionality is driven by M_TX module.

For example, in LCC_READ operation (READ1 to READ4), all 1's are replaced by the O-TX capabilities values defined in M-TX Module. Therefore, M_TX passes O_TX capabilities instead of all 1's when `cfg.omc_present = 0`.

8.23.5 Points to Note

- ❖ A proper LCC_MODE is observed before exiting LINE_CFG state. In the above scenario, LCC_MODE = HIBERN8-STALL
- ❖ FSM state moves from LINE_CFG to STALL in the above scenario and after Thibern8_enter_tx time, it goes to HIBERN8 state
- ❖ If a CTRL SAP with HS_Gear change is issued along with Tx_Hibern8_Control SAP, LCC_MODE will not reflect changed HS_Gear value. It will reflect LCC_MODE = HIBERN8-STALL value only.

8.24 User-Specific Burst Closure Pattern

The M-PHY Verification IP provides user specific burst closure pattern feature. This feature provides control to the user whether to use default burst closure pattern as per *MIPI Alliance Specification for MPHY Version 4.1 - 01 December 2016* or to use user-specific 10bit burst closure pattern.

To enable or use user-specific burst closure pattern, user needs to set the `mphy_ext_burst_closure_enable` configuration parameter to 1 using `svt_mphy_mport_lm_agent_configuration` class.

For example:

```
svt_mphy_mport_lm_agent_configuration master_cfg;
svt_mphy_mport_lm_agent_configuration slave_cfg;

/** Set for Master TX */
master_cfg.tx_cfg. mphy_ext_burst_closure_enable = 1'b1;

/** Set for Master RX */
master_cfg.rx_cfg. mphy_ext_burst_closure_enable = 1'b1;
```



```
/** Set for Slave TX */  
slave_cfg.tx_cfg.mphy_ext_burst_closure_enable = 1'b1;
```

```
/** Set for Slave RX */  
slave_cfg.rx_cfg.mphy_ext_burst_closure_enable = 1'b1;
```

User can provide 10 bit pattern as for data sequence in `ext_burst_closure_pattern` field. This pattern is provided to the transaction which has `req.end_of_burst` as 1.

If user wants to switch off this feature in run time then that can be done by calling `reconfigure` method.

```
this.cfg.mphy_ext_burst_closure_enable = 0; // Switch OFF this feature  
env.master_agent.reconfigure(cfg);
```

8.25 Configuration Register Read on Configurable Number of Clocks

The M-PHY Verification IP provides the support for configuration register read on configurable number of clocks. This feature provides control to the user to decide the number of clocks after which the configuration register read should happen.

To use this feature, user needs to configure the `mphy_cfgget_num_of_clock` configuration parameter using `svt_mphy_mport_lm_agent_configuration` class.

For example, to read the registers after 3 clock cycles, user must set the following:

```
svt_mphy_mport_lm_agent_configuration master_cfg;  
svt_mphy_mport_lm_agent_configuration slave_cfg;
```

```
/** Set for Master TX */  
master_cfg.tx_cfg.mphy_cfgget_num_of_clock = 3;  
/** Set for Master RX */  
master_cfg.rx_cfg.mphy_cfgget_num_of_clock = 3;
```

```
/** Set for Slave TX */  
slave_cfg.tx_cfg.mphy_cfgget_num_of_clock = 3;
```

```
/** Set for Slave RX */  
slave_cfg.rx_cfg.mphy_cfgget_num_of_clock = 3;
```

By default, the value of `mphy_cfgget_num_of_clock` is 1, that is, the configuration register read will happen after one clock.

8.26 Inline Config Feature (Optional)

The M-PHY Verification IP supports the Inline Config feature which is an optional feature according to *MIPI Alliance Specification for MPHY Version 4.1 - 01 December 2016*. The `TX_InLineCfg` and `RX_InLineCfg` pin are optional pin. In order to disable the Inline Config feature, the user needs to set the following configuration parameter:

```
cfg.disable_inline_cfg = 1'b1;
```

The Inline Config behavior is enabled by default. When the Inline Config feature is disabled, the pins `TX_InLineCfg` and `RX_InLineCfg` will be driven to a low (1'b0) value.

8.27 De-Assertion of TX_SaveState_Status_N Signal (Optional)

The M-PHY Verification IP supports the deassertion of TX_SaveState_Status_N signal after two clock cycles delay which is an optional feature according to *MIPI Alliance Specification for MPHY Version 4.1 - 01 December 2016*. The TX_SaveState_Status_N signal can be optionally delayed by two clock cycles. The following is the use model for this feature:

```
cfg.delay_save_state_n = 1, add two clock delays
cfg.delay_save_state_n = 0, add no clock delay
```

8.28 T_activate Timer on Exit to HIBERN8

The M-PHY Verification IP provides the following feature:

- ❖ Running T_activate timer via protocol or M-TX - This VIP protocol feature provides control to the user to run T_activate timer either from protocol layer or from M-TX Phy layer on exit to HIBERN8.

According to *MIPI Alliance Specification for MPHY Version 4.1 - 01 December 2016*, the local M-TX shall drive DIF-N for a period of TACTIVATE on exit of HIBERN8.



Note

- The M-PHY VIP provides PHY-CONTROLLED T_activate timer feature for MPHY version 2.0 and version 3.0. For M-PHY version 1.4, protocol layer has to run this timer.
- When t_activate_control is set to PHY_CONTROLLED and any control transaction from protocol layer is received by M-TX during the timer, then M-TX will successfully process that control transaction and will return to the power saving state of the configured operating mode.

If protocol layer initiates a BURST transfer during the T_activate timer, then M-TX executes T_activate timer first and then it would start the BURST.

T_activate timer can be PROTOCOL_CONTROLLED or PHY_CONTROLLED and is user configurable. There is a field in svt_mphy_transaction class t_activate_control which can be configured as either PROTOCOL_CONTROLLED or PHY_CONTROLLED on exit to HIBERN8.

The t_activate_control variable is an optional parameter. Thus, a reasonable constraint is added to constraint its value to SVT_MPHY_PROTOCOL_CONTROLLED.

The description of t_activate_control is as follows:

Table 8-6 Table for t_activate_control

Variable Name	Variable Type	Description
t_activate_control	typedef enum bit { PROTOCOL_CONTROLLED = `SVT_MPHY_PROTOCOL_ CONTROLLED, PHY_CONTROLLED = `SVT_MPHY_PHY_CONTR OLLED} t_activate_control_ enum;	Indicates which Layer controls TActivate time and driving DIF-N. TActivateControl is an optional parameter. ❖ The default value of TActivateControl is ProtocolControlled ❖ When TActivateControl value is set to PhyControlled, TActivate timer is run by MTX.

8.28.1 Use Case

❖ PHY_CONTROLLED

Following code snippet shows how to set `t_activate_control` transaction field as `PHY_CONTROLLED` in `HIBERN8_EXIT` control SAP:

```
svt_mphy_transaction req;  
`uvm_create(req);  
req.reasonable_t_activate_control.constraint_mode(0);  
req.dir_kind = svt_mphy_transaction::M_TX;  
req.primitive_layer = svt_mphy_transaction::M_CTRL;  
req.primitive_name = svt_mphy_transaction::CFGSET;  
req.primitive_type = svt_mphy_transaction::REQUEST;  
req.mib_attribute = `SVT_MPHY_TX_HIBERN8_CONTROL;  
req.mib_value = 0;  
req.t_activate_control = t_activate_ctrl;  
`uvm_send(req);  
get_response(rsp);
```

To control the TActivate timer using the Protocol layer or MTX, there is an interface pin `TX_Controlled_ActTimer`. The direction of `TX_Controlled_ActTimer` is input for `svt_mphy_rmmi_mtx_dut_controller_if` interface and output for `svt_mphy_rmmi_mtx_dut_phy_if` interface.

The description of `TX_Controlled_ActTimer` is as follows:

Table 8-7 Table for TX_Controlled_ActTimer

Pin Name	Detection Type	Width	Description
TX_Controlled_ActTimer	Level	1	<p>TX_Controlled_ActTimer informs M-TX, which Layer controls the TActivate time.</p> <p>TX_Controlled_ActTimer is an optional signal.</p> <p>The Protocol Layer shall set TX_Controlled_ActTimer to one to inform M-TX to drive DIF-N for at least TActivate time, irrespective of the current M-TX state, before driving DIF-P for TLINE_RESET after TX_LineReset is asserted.</p> <p>M-TX shall also control the TActivate time upon HIBERN8 exit.</p> <p>When TX_Controlled_ActTimer is set to zero, the Protocol Layer shall wait for TActivate time after M-TX sets TX_SaveState_Status_N to zero before asserting TX_LineReset. If TX_Controlled_ActTimer is set to zero, when TX_LineReset is asserted, M-TX shall immediately drive the LINE-RESET condition.</p> <p>The Protocol Layer shall also control the TActivate time upon HIBERN8 exit.</p>

8.29 Control Symbols MK1,MK3,MK4,MK5 & MK6 insertion in a transaction at Lane Management

The M-PHY Verification IP provides the following feature:

Sending control symbols MK1, MK3, MK4, MK5, and MK6 in a LM transaction. This feature provides control to the user to send control symbol (MK1, MK3, MK4, MK5, and MK6) embedded in the payload stream. This feature is provided at Lane Management.

User can configure the number of markers via setting the `num_of_markers`, `select_marker` (which can have values MK1, MK3, MK4, MK5, MK6, and RANDOM) which decides whether to send single marker multiple times or send randomly multiple markers out of above five markers) and `marker_position` fields in the testcase using `uvm_config_db` set or get methods. The `marker_position` field is set as a dynamic array to insert markers multiple times in one LM transaction.



Note

`marker_position` and `select_marker` are two newly added LM MPORT transaction fields.



Note

Out of all M-PHY specified control symbols MK1, MK3, MK4, MK5, and MK6 can be inserted by the user, except `frame_demarcator` which can be any of the above mentioned five control symbols.

- ❖ When `frame_demarcator` is MK1, then LM MPORT TX sends any of the four control symbols (MK3, MK4, MK5, and MK6) within burst with `data_n_ctrl = 1`.
- ❖ When `frame_demarcator` is MK3, then LM MPORT TX sends any of the four control symbols (MK1, MK4, MK5, and MK6) within burst with `data_n_ctrl = 1` and so on for MK4, MK5, and MK6.

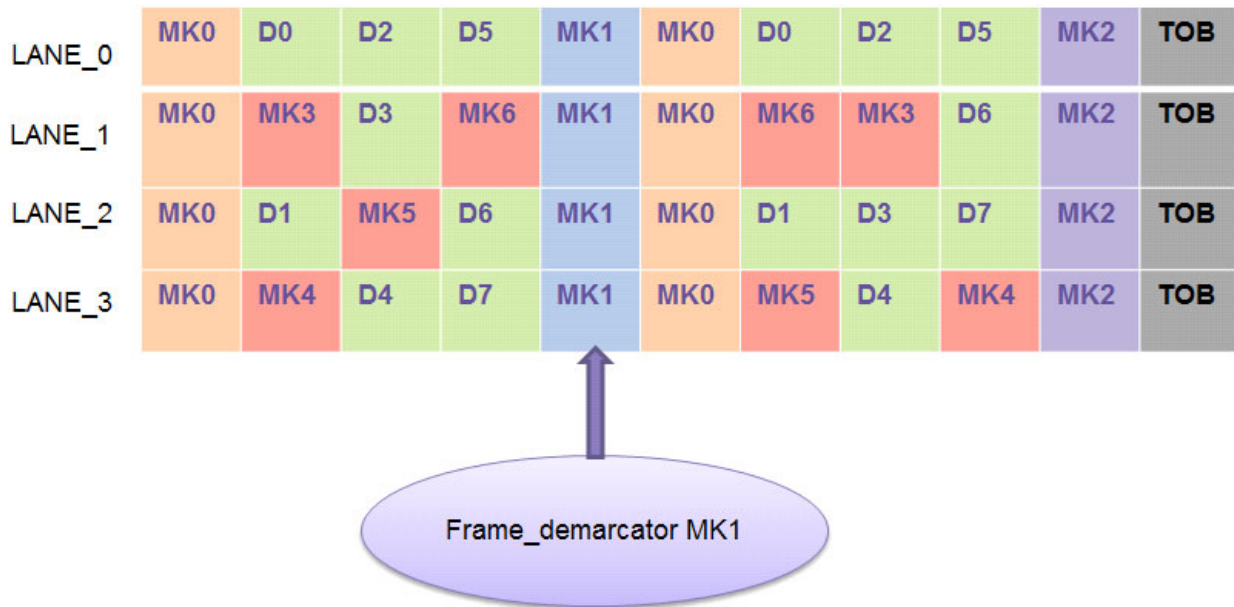
To configure `frame_demarcator`, user can use LM MPORT agent configuration class

`svt_mphy_mport_lm_agent_configuration` as:

```
/** Master configuration */
svt_mphy_mport_lm_agent_configuration master_cfg;
/** Slave configuration */
svt_mphy_mport_lm_agent_configuration slave_cfg;
/** Set the frame demarcator for Master TX*/
master_cfg.tx_cfg.frame_demarcator = svt_mphy_mport_lm_configuration::MK1;
/** Set the frame demarcator for Slave RX*/
slave_cfg.rx_cfg.frame_demarcator = svt_mphy_mport_lm_configuration::MK1;
```

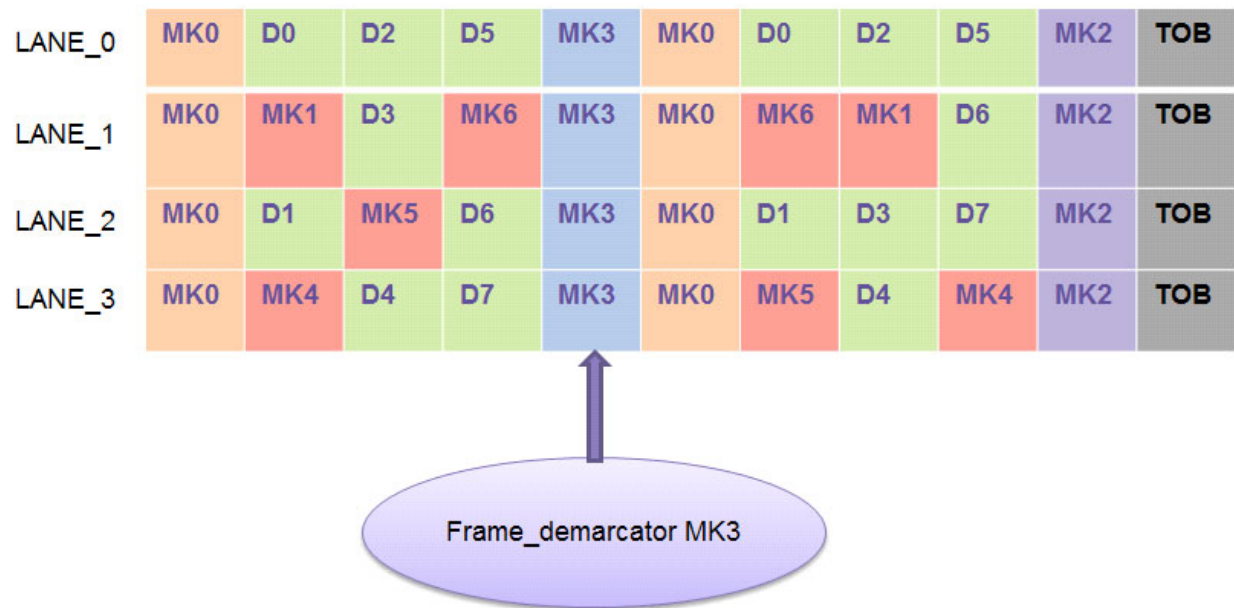
For example: If `frame_demarcator` is MK1, then MK3, MK4, MK5, and MK6 can be send as shown in figure 8-5.

Figure 8-5 When frame_demarcator is MK1



- ❖ If frame_demarcator is MK3, then MK1, MK4, MK5 and MK6 can be send as data payload byte as shown in figure 8-6.

Figure 8-6 When frame_demarcator is MK3



8.29.1 Points to Remember

1. User needs to set or configure above fields `num_of_markers`, `marker_position`, and `select_marker` to insert control symbols in data stream using `uvm` set or get method for corresponding virtual sequence.
2. `marker_position` array size determines the insertion of control symbol or MARKERS.
3. If user does not provide any position or does not fill `marker_position` array but configures `num_of_markers`, then `num_of_markers` fields will be used for `marker_position` array size and MPHY VIP will fill this array with any random position or values through virtual sequence.



Note

The values inside the `marker_position` array must be within the payload length provided.

The following code snippet displays the use model:

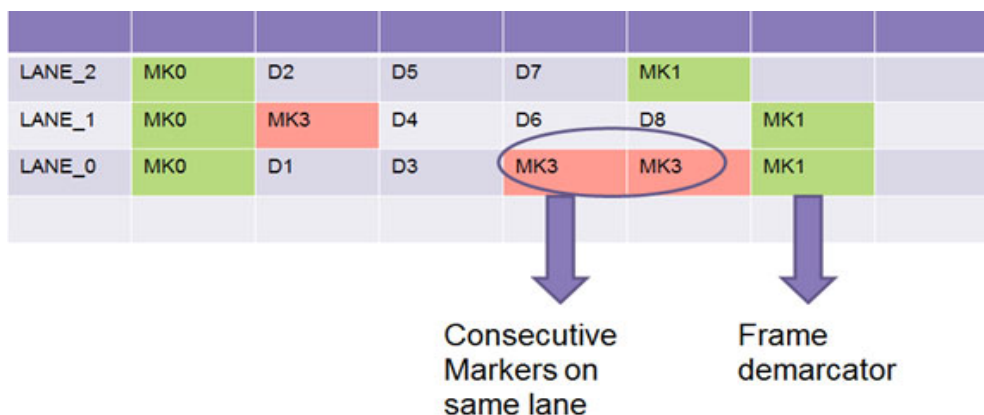
```
svt_mphy_tx_sap_lm_data_marker_insertion_sequence data_seq;
...
...
...
        marker_position = new[num_of_markers];
        for(int i=0; i < marker_position.size(); i++) begin
            marker_position[i] = $urandom_range(0:payload_length-1);
        end
...
...
...
`uvm_do_on_with(data_seq, p_sequencer.tx_data_sequencer, {data_seq.payload_length ==
payload_length_s;
marker_position.size();
select_marker;
marker_position[k];});
                                data_seq.seq_marker_position.size() ==
                                data_seq.lanes == tx_cfg.active_lane;
                                data_seq.seq_select_marker ==
                                foreach (marker_position[k])
                                    data_seq.seq_marker_position[k] ==
```

You can refer MPHY VIP virtual sequence

`svt_mphy_lm_hs_marker_insertion_virtual_sequence` for this feature.

1. If user does not configure `num_of_markers` and `marker_position` and keep them as zero or null, then the MPHY VIP will not insert any MARKER during the transaction.
2. You can configure the above mentioned fields based on the following:

```
Payload_length = 8
marker_position = {1,5,7}
num_of_marker = 3;
select_marker = MK3;
active_lanes = 3;
```

Figure 8-7 The output for the user configured field

- ❖ If user configures `marker_position` as {7,1,5} then MPHY VIP will treat this as {1,5,7} and will insert markers at position {1,5,7}.
- ❖ Since, `payload_length`, `num_of_markers`, and `marker_position` are user configurable fields then it is the responsibility of the user to decide that the transfer should be aligned or un-aligned. This example leads to an un-aligned transfer due to insertion of three markers. So to make this an aligned transfer, see the [“Aligned and Unaligned Transfers”](#) section.
- 3. On the LM MPORT receiver side, `marker_position` array values indicates the position at which markers have been inserted.
- 4. LM MPORT monitor treats these markers as data bytes. The total number of data bytes received by LM MPORT monitor includes these markers. Thus, MK1, MK3, MK4, MK5, and MK6 control symbols can only be transmitted within the BURST.

8.30 Unaligned EOB at M-RX PHY

M-RX PHY in MPHY VIP ends the data on RX_Symbol bus aligned due to auto filler insertion feature of M-TX PHY.

To unalign data after the burst by M-RX Phy, the following new field has been added in `svt_mphy_configuration` class. When this field is set and data is sent unaligned from PA layer, there will not be any auto fillers at EOB and data on RX_SYMBOL bus is unaligned.

If data MK0, D1, D2, and EOB (40 bit RMMI) is sent from PA layer then data on MRX RMMI pins will be as follows:

- ❖ MK0, D1, D2, EOB
- ❖ `RX_PhyDORDY[3]` will be zero

Variable name	Variable type	Description
allow_unaligned_eob	rand bit	0: Default behavior. 1: To de-assert RX_PhyDORDY at EOB if unaligned data bytes are given to MPHY by PA layer.

There is a default constraint on this field to make sure data is not unaligned in normal case at RX RMMI lines.

```
constraint default_allow_unaligned_eob {
    allow_unaligned_eob == 0;
}
```

8.30.1 Use Case

You can set this field to one in shared_cfg.

// If cfg is not randomized.

```
cfg.allow_unaligned_eob = 1;
```

// If cfg is randomized then user need to OFF the default constraint.

```
cfg.default_allow_unaligned_eob.constraint_mode(0);
cfg.allow_unaligned_eob = 1;
```

8.31 RX_Burst Assertion at First MK0 in LS-MODE

MPHY VIP (RMMI back to back), MRX asserts RX_Burst signal at the start of PREPARE period and data on RX_Symbol starts only when PREPARE and SYNC length (PWM G6 and PWM G7) expires.

To assert the RX_Burst signal at first MK0 (skipping PREPARE), following new field has been added in svt_mphy_configuration class. When this field is set then RX_Burst will assert at MK0 in LS mode. This behavior will be same for any prepare length value.

Variable name	Variable type	Description
enable_rx_burst_assertion_at_mk0	rand bit	0: Default behavior. 1: To assert RX_Burst at MK0 detection

There is a default constraint on this field to make sure RX_Burst asserts at PREPARE period in normal case at RX RMMI lines.

```
constraint default_enable_rx_burst_assertion_at_mk0 {
    enable_rx_burst_assertion_at_mk0 == 0;
}
```

To see similar behavior in case of HS-MODE, TX_HS_PREPARE_LENGTH and TX_HS_SYNC_LENGTH should be set to zero. It can be done by directly setting the following lengths:


```
init_hs_prepare_length = 0  
init_hs_sync_range = 0  
init_hs_sync_length = 0
```

8.31.1 Use Case

User can set this field to one in shared_cfg.

```
// If cfg is not randomized.
```

```
cfg.enable_rx_burst_assertion_at_mk0 = 1;
```

```
// If cfg is randomized then user need to OFF the default constraint.
```

```
cfg.default_enable_rx_burst_assertion_at_mk0.constraint_mode(0);
```

```
cfg.enable_rx_burst_assertion_at_mk0 = 1;
```

8.32 User Defined Attributes

MPHY spec defines certain attributes for different operations in MPHY. If there is a CFGGET or CFGSET REQUEST on attributes outside of MPHY Spec then MPHY VIP will send a checker.

MPHY VIP has added the feature which gives user control to increase the attribute's addresses as per requirement. MPHY VIP will not send any check if there is any CFGGET or CFGSET on these attributes.

A class svt_mphy_extended_attribute.sv is added in mphy_svt/src/ directory for this purpose.

The class members are as follows:

Variable name	Variable type	Description
attribute_addr	bit [SVT_MPHY_SAP_ATTR_ADDR_WIDTH - 1:0]	Contains address for user defined attributes.
range_discreet	bit	0: range defined by min_val and max_val 1: Discreet values filled in attribute_val_list array.
min_val	bit [SVT_MPHY_SAP_ATTR_ADDR_WIDTH - 1:0]	Default: 0
max_val	bit [SVT_MPHY_SAP_ATTR_ADDR_WIDTH - 1:0]	Default: 255
reset_val	bit [SVT_MPHY_SAP_ATTR_ADDR_WIDTH - 1:0]	Reset value. Set to zero for all attributes.
attribute_val_list[\$]	bit [SVT_MPHY_SAP_ATTR_ADDR_WIDTH - 1:0]	This queue is used to have discrete values for attribute address defined in attribute_addr field when range_discreet is set to one.
rw_access	bit	0: Both Read and write access 1: read access only

A queue of class svt_mphy_extended_attribute.sv is created in svt_mphy_configuration.sv class.

Field name	Variable type	Description
extended_attributes[\$]	svt_mphy_extended_attribute	This queue will hold the user defined attributes.

This queue can be used to add any number of attributes. User can fill this queue in shared_cfg file so that all user defined attributes are valid in all tests.

range_discreet field is used to specify whether the attribute has discreet values or fixed range. If range_discreet is set to one then user can specify discreet values for attribute in attribute_val_list queue. Otherwise min_val and max_val field can be used to specify minimum and maximum value for the attribute.

rw_access field is given to specify whether attribute has both read and write access or read access only. By default its value is set to zero (both read and write access).

For each CFGGET operation on these attributes, MPHY VIP will return the value stored for that attribute by previously requested CFGSET command otherwise it will return value stored in reset_val field.

All user defined attributes will reset to reset_val on LOCAL RESET and LINERESET. There should not be any conflict between MPHY spec defined attributes and user defined attributes. If it occurs then MPHY spec define attribute will take precedence.

Checks implemented for this feature are as follows:

1. There will be a check in monitor which will check for the valid range of values for every read or write operation, as per the attribute_val_list, min_val, and max_val.
2. There will be a check if there is any write operation on READ only attribute.

Use case will add user-defined attribute through test bench.

You need to add all attributes through test bench (in shared_cfg file for one time), for MPHY VIP to be aware about the valid set of user defined attributes during run time.

Default value for define SVT_MPHY_SAP_ATTR_ADDR_WIDTH is set to eight. If user defined attributes have address or data ranges greater than eight bit then change the value of this define to appropriate value from top file. Also default value of port defines SVT_MPHY_TX_PORT_DATA_WIDTH and SVT_MPHY_RX_PORT_DATA_WIDTH is set to eight. To drive attribute greater than eight bits on TX_AddrID or RX_AddrID bus set the port defines to appropriate value.

Defines SVT_MPHY_SAP_ATTR_ADDR_WIDTH, SVT_MPHY_TX_PORT_DATA_WIDTH, and SVT_MPHY_RX_PORT_DATA_WIDTH should be set to same value for proper CFGGET or CFGSET operation.

This example creates an attribute with address 10'h121 and 10'h122 with minimum value zero and maximum value 100. Other fields like reset_val, range_discreet, rw_access are at default value zero.

```
for (int i = 0; i < 2; i++) begin
    mphy_tx_cfg.extended_attributes[i] = new;
    mphy_tx_cfg.extended_attributes[i].attribute_addr= 10'h121 + i;
    mphy_tx_cfg.extended_attributes[i].min_val = 0;
    mphy_tx_cfg.extended_attributes[i].max_val = 100;
end
```

Use case to SET/GET on these attribute is same as for other MPHY defined attributes.

```
`uvm_do_with(req, {req.dir_kind == svt_mphy_transaction::M_TX;
                    req.primitive_layer == svt_mphy_transaction::M_CTRL;
                    req.primitive_name == svt_mphy_transaction::CFGGET;
```

```
req.primitive_type == svt_mphy_transaction::REQUEST;  
req.mib_attribute == 10'h121 ;})          // user defined attribute  
address  
    get_response(rsp);
```

**Note**

This release does not support checkers and coverage. These features will be available in future releases.

8.33 Send Multiple External SYNC Symbols in a Transaction at Lane Management

The M-PHY Verification IP provides the following feature:

Sending multiple external SYNC symbols in a LM transaction. This feature provides control to the user to send any SYNC symbol through the testbench.

This feature is provided at Lane Management. The user can fill `sync_symbol_array` field in the testcase and pass it to the MPHY VIP virtual sequence using `uvm_config_db` set or get methods. The `sync_symbol_array` field is set as a dynamic array to insert SYNC symbols multiple times in one LM transaction.

**Note**

`lm_sync_symbol_array` is a newly added LM MPORT transaction field and `sync_symbol_array` is a newly added MPHY transaction field.

You can send any SYNC symbol defined in Table 6 of MIPI Alliance Specification for MPHY Version 4.1 - 01 December 2016.

8.33.1 Points to Remember

1. User needs to set or configure `num_of_sync_symbols` and `sync_symbol_array` fields to send SYNC symbols using `uvm_config_db` set or get method for corresponding virtual sequence.
2. `num_of_sync_symbols` value indicates the size of `sync_symbol_array`.
3. If user does not provide any value or does not fill `sync_symbol_array` field but configures `num_of_sync_symbols`, then `num_of_sync_symbols` field will be used for `sync_symbol_array` array size and MPHY VIP will fill this array with any random SYNC symbols through corresponding virtual sequence.
4. If user does not configure `num_of_sync_symbols` and `sync_symbol_array` and keep them as zero or null, then the MPHY VIP will not send SYNC symbols externally during the transaction.

8.33.2 Use Model

The following code snippet illustrates the use model:

In testbench or testcase, user can set or configure the `sync_symbol_array` field as follows:

```
svt_mphy_types::sync_symbol_enum sync_symbol_array[ ];  
svt_mphy_types::sync_symbol_enum temp_sync_symbol;  
string inst_name ;  
...  
...
```

```

...
sync_symbol_array = new[5];
for(int i=0; i < sync_symbol_array.size(); i++) begin
    std::randomize(temp_sync_symbol) with {temp_sync_symbol inside
{SVT_MPHY_SYNC_SYMBOLS}};
    sync_symbol_array[i] = temp_sync_symbol ;
end
...
...
uvm_config_db#(int unsigned)::set(this,  "*.tx_sublink_sequencer",
"svt_mphy_lm_hs_ext_sync_array_virtual_sequence.num_of_sync_symbols",
sync_symbol_array.size());
    foreach(sync_symbol_array[i])begin
        $sformat(inst_name,
"svt_mphy_lm_hs_ext_sync_array_virtual_sequence.sync_symbol_array[%0d]", i);
        svt_config_int_db#(int unsigned)::set(this,  "*.tx_sublink_sequencer",inst_name
,sync_symbol_array[i]);
    end

```

In MPHY VIP virtual sequence, the user will get the `num_of_sync_symbols` and `sync_symbol_array` fields as follows:

```

uvm_config_db#(int
unsigned)::get(p_sequencer,"",{get_type_name(),".num_of_sync_symbols"},
num_of_sync_symbols);
sync_symbol_array = new[num_of_sync_symbols];
foreach(sync_symbol_array[i])begin
    $sformat(inst_name, {get_type_name(),".sync_symbol_array[%0d]"}, i);
    status = svt_config_int_db#(int unsigned)::get(p_sequencer,"",inst_name,
sync_symbol_array[i]);
end
// Check if user does not set "sync_symbol_array" then fill it locally if(!status)begin
    for(int i=0; i < sync_symbol_array.size(); i++) begin
        std::randomize(temp_sync_symbol) with {temp_sync_symbol inside
{`SVT_MPHY_SYNC_SYMBOLS}};
        sync_symbol_array[i] = temp_sync_symbol ;
    end
end
...
...
uvm_do_on_with(data_seq, p_sequencer.tx_data_sequencer,{data_seq.payload_length ==
payload_length_s;
    data_seq.seq_sync_symbol_array.size()
== sync_symbol_array.size();
    data_seq.lanes ==
tx_cfg.active_lane;
    foreach
(sync_symbol_array[k])
data_seq.seq_sync_symbol_array[k] == sync_symbol_array[k];});

```



Note You can refer MPHY VIP virtual sequence

`svt_mphy_lm_hs_ext_sync_array_virtual_sequence` for this feature.

8.34 Configure Number of Clock to Enable rx_hibern8exit_type_I Signal Check (RMMI Interface).

The M-PHY Verification IP provides the following feature:

- ❖ Configure number of clock to enable rx_hibern8exit_type_I signal check. This feature provides control to the user to enable rx_hibern8exit_type_I signal check on RMMI interface.
- ❖ In svt_mphy_configuration class check_hibern8_num_of_clock field is added to enable rx_hibern8exit_type_I signal check after configurable number of clocks.
- ❖ Default value for this field is set to one in svt_mphy_configuration class.
- ❖ You can configure this field from test case based on this example.

Example :

```
/* Master agent configuration */
svt_mphy_mport_lm_agent_configuration master_cfg;
/* Slave agent configuration */
svt_mphy_mport_lm_agent_configuration slave_cfg;

master_cfg.tx_cfg.check_hibern8_num_of_clock = 3;
slave_cfg.rx_cfg.check_hibern8_num_of_clock = 3;
```

8.35 Configure Initial Value of Running Disparity

Configuring initial value of Running Disparity feature enables the user to set initial value of Running Disparity. A new configuration field (of type bit) init_rd_value is added in svt_mphy_configuration class.

This field can have two values either 0 or 1 (0 means RD = -1 & 1 means RD = +1). Default value of this field is 0 for example, RD = -1.

8.35.1 Use Model

The following code snippet illustrates the use model:

In testbench or testcase, user can set or configure the init_rd_value field as follows:

- ❖ For TX_Sublink

```
/* Master agent configuration for TX_Sublink*/
svt_mphy_mport_lm_agent_configuration master_tx_cfg;
/* Slave agent configuration for TX_Sublink*/
svt_mphy_mport_lm_agent_configuration slave_rx_cfg;

master_tx_cfg.tx_cfg.init_rd_value = 1'b1;
slave_rx_cfg.rx_cfg.init_rd_value = 1'b1;
```
- ❖ For RX_Sublink

```
/* Master agent configuration for RX_Sublink*/
svt_mphy_mport_lm_agent_configuration master_rx_cfg;
/* Slave agent configuration for RX_Sublink*/
svt_mphy_mport_lm_agent_configuration slave_tx_cfg;
```

```
master_rx_cfg.rx_cfg.init_rd_value = 1'b1;
slave_tx_cfg.tx_cfg.init_rd_value = 1'b1;
```

8.36 Clock Jitter

VIP supports insertion of jitter in the generated transmit clock on Serial interface as well as on RMMI interface.

The insertion of jitter in the generated transmit clock in different modes are as follows:

- ❖ For Serial Mode the jitter would be reflected on the differential TX lines. For example, on TX_P and TX_N.
- ❖ For RMMI mode, the jitter observed is RX_Symbol_Clk only and not data (RX_Symbol), when svt_mphy_rmmi_mrx_dut_controller_if interface is configured. Data would be driven with respect to the actual (non-jittered) clock.
- ❖ In both cases jitter is enabled when enable_clock_jitter is set to 1.

8.36.1 Use Model With no PPM Limit

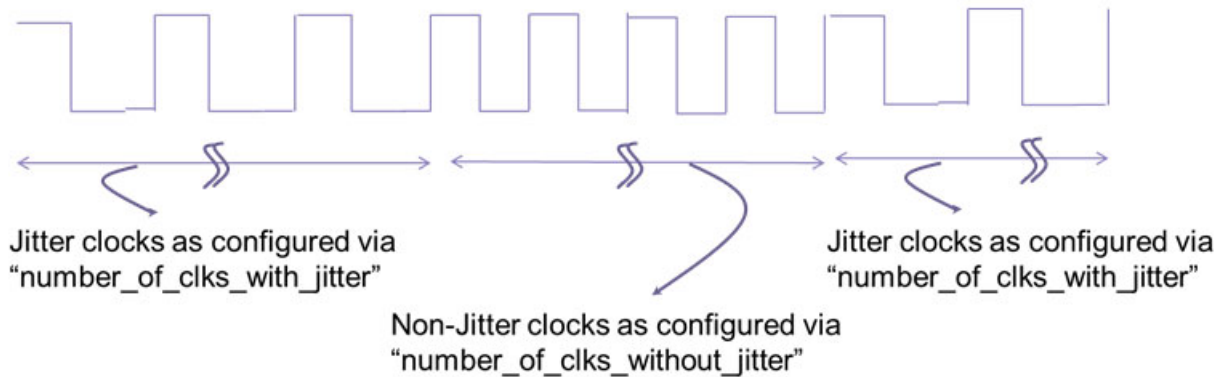
8.36.1.1 Insertion of 17% of CONSTANT_PERIODIC_JITTER

```
interface_type = SERIAL;
enable_clock_jitter = 1;
number_of_clks_with_jitter = 100;
```

- ❖ SERIAL mode of operation:
percentage_of_jitter_per_clk = 17;
- ❖ RMMI mode of operation:
percentage_of_jitter_per_symbol_clk = 6.25;

The final generated clock is as shown in the diagram:

Figure 8-8 The final generated clock for CONSTANT_PERIODIC_JITTER



Note

Configuration parameter for jitter are random in nature. To set any of the variable to a desired value, the respective default constraint should be turned OFF. For additional information, refer to the UVM Class Reference Guide available at:
[\\$DESIGNWARE_HOME/vip/svt/mphy_svt/latest/doc/class_ref/class_ref/mphy_svt_uvm_class_reference/html/index.html](#)

8.36.1.2 Insertion of 17% of RANDOM_JITTER

❖ Serial mode of operation

```
interface_type = SERIAL;
enable_clock_jitter = 1;
percentage_of_jitter_per_clk = 17;
```

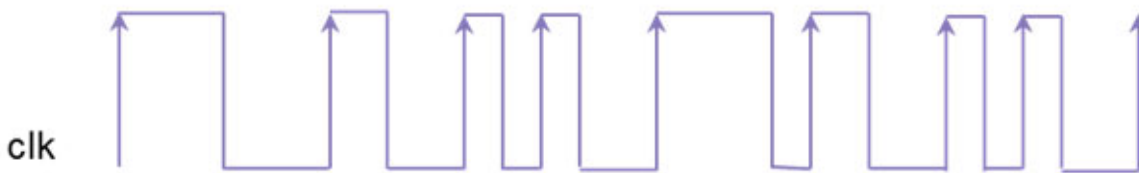
❖ RMMI mode of operation (RMMI REMOTE, TLM_REMOTE)

```
interface_type = SERIAL;
enable_clock_jitter = 1;
percentage_of_jitter_per_symbol_clk = 6.25;
```

The input percentage of jitter is a real value. The jitter introduced per clock is randomly selected by the VIP from the following values:

```
percentage_of_jitter_per_symbol_clk = percentage_of_jitter_per_symbol_clk + 0.2;
percentage_of_jitter_per_symbol_clk = percentage_of_jitter_per_symbol_clk + 0.4;
percentage_of_jitter_per_symbol_clk = percentage_of_jitter_per_symbol_clk + 0.6;
percentage_of_jitter_per_symbol_clk = percentage_of_jitter_per_symbol_clk + 0.8;
percentage_of_jitter_per_symbol_clk = percentage_of_jitter_per_symbol_clk;
percentage_of_jitter_per_symbol_clk = percentage_of_jitter_per_symbol_clk - 0.2;
percentage_of_jitter_per_symbol_clk = percentage_of_jitter_per_symbol_clk - 0.4;
percentage_of_jitter_per_symbol_clk = percentage_of_jitter_per_symbol_clk - 0.6;
percentage_of_jitter_per_symbol_clk = percentage_of_jitter_per_symbol_clk - 0.8;
```

The final generated clock is as follows in the following diagram:

Figure 8-9 The final generated clock for RANDOM_JITTER

In the above diagram every clock has a random jitter within the value of “percentage of jitter clk” configuration variable.

**Note**

In the above diagram every clock has a random jitter within the value of `percentage_of_jitter_clk` configuration variable.

**Note**

Configuration parameter for jitter are random in nature. To set any of the variable to a desired value, the respective default constraint should be turned OFF. For additional information, refer to the UVM Class Reference Guide available at:
[\\$DESIGNWARE_HOME/vip/svt/mphy_svt/latest/doc/class_ref/mphy_svt_uvm_class_reference/html/index.html](#)

8.36.1.3 Insertion of 17% of DETERMINISTIC_JITTER

```
interface_type = SERIAL;
enable_clock_jitter = 1;
raise_of_jitter_per_clk = 1;
```

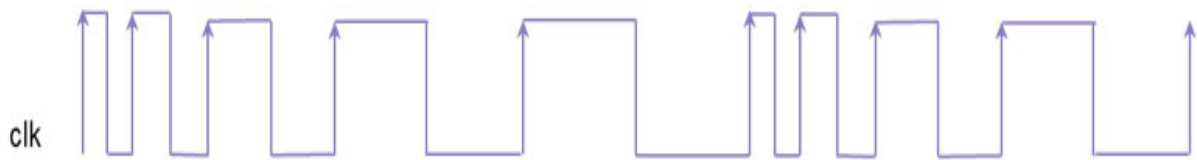
- ❖ SERIAL mode of operation:

```
percentage_of_jitter_per_clk = 17;
```

- ❖ RMMI mode of operation:

```
percentage_of_jitter_per_symbol_clk = 6.25;
```

The final generated clock is as shown in this diagram:

Figure 8-10 The final generated clock for DETERMINISTIC_JITTER**Note**

In the above diagram, every clock has an incremental value of jitter as per the `raise_of_jitter_per_clk` parameter, and the pattern rotates throughout the simulation.

**Note**

Configuration parameter for jitter are random in nature. To set any of the variable to a desired value, the respective default constraint should be turned OFF. For additional information, refer to the UVM Class Reference Guide available at:
[\\$DESIGNWARE_HOME/vip/svt/mphy_svt/latest/doc/class_ref/mphy_svt_uvm_class_reference/html/index.html](#)

8.36.1.4 Insertion of Max 10% of ZERO_MEAN_JITTER

Zero mean jitter is only applicable to SERIAL interface.

Set the following configuration, to enable zero mean jitter:

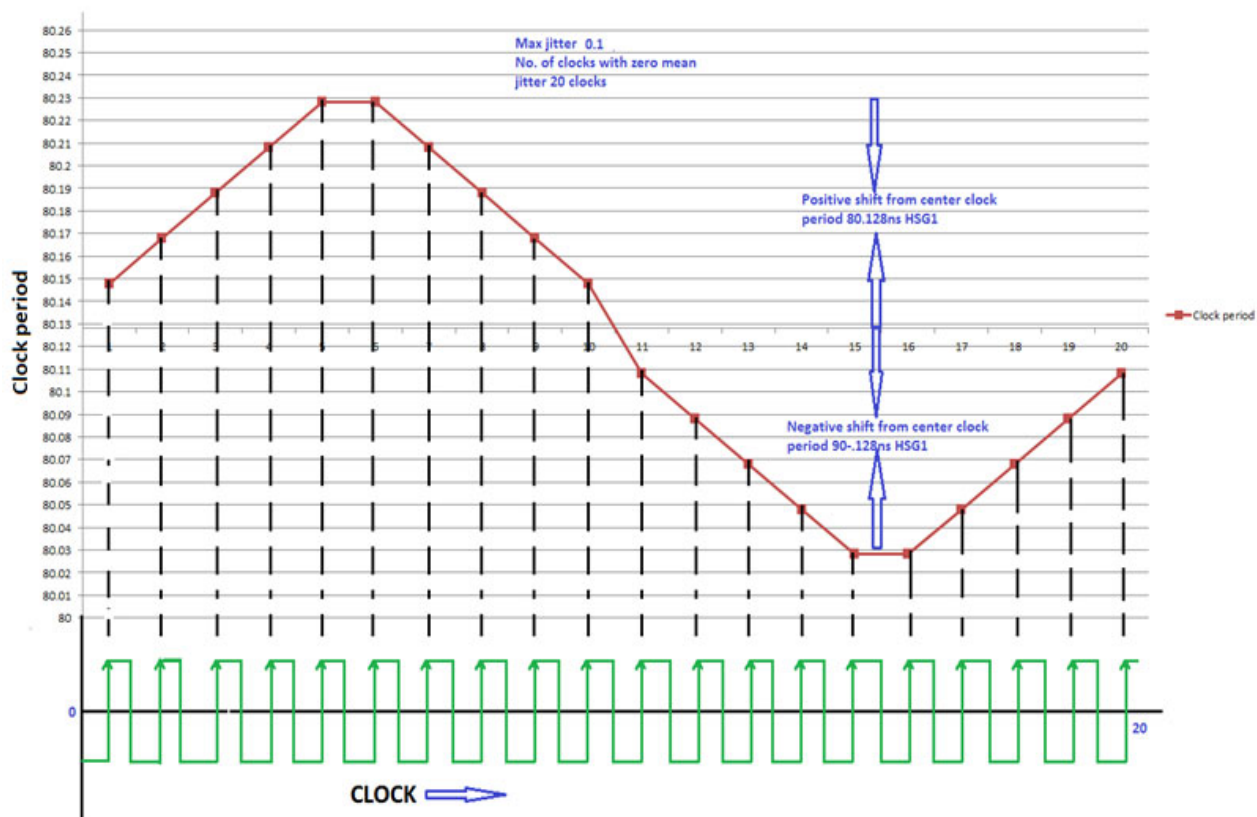
1. `<mphy_cfg>.enable_clock_jitter = 1'b1;`
2. `<mphy_cfg>.percentage_of_jitter_per_clk = 10;`
3. `<mphy_cfg>.number_of_clks_with_jitter = 20;`
4. `<mphy_cfg>.mphy_jitter_type=svt_mphy_configuration::ZERO_MEAN_JITTER;`

If the mean is calculate for particular number of clocks, and the result is the center clock period then it is referred as Zero Mean Jitter.

For example,

$\sum (P_1, P_2, \dots, P_n) / n = P_c$ (where n = number of clock, P_c = center period).

In the following figure, $P_c = \text{HSG1} = 80.128\text{ns}$ and $n = 20$ clock.

Figure 8-11 The Final Generated Clock for ZERO_MEAN_JITTER

Above figure shows example for HSG1 with max jitter 0.1 for 20 clocks.



Note For zero mean jitter:

Max value for `percentage_of_jitter_per_clk` should be less than or equal to 20 for a threshold of 2000ppm.

Number of clock with jitter should be multiple of 4 for zero mean jitter, for a period of time.

8.36.2 Use Model With PPM Limit

- ❖ MAX PPM with MPHY 5.0 is 150ppm and older version is 2000ppm
- ❖ To limit jitter value well inside PPM limit user need to define macro `SVT_MPHY_USE_MAX_PPM`.

These sections provide the MPHY configuration settings that are required to set different jitter types.

8.36.2.1 CONSTANT_PERIODIC_JITTER

- ❖ `interface_type = SERIAL;`
- ❖ `enable_clock_jitter = 1;`
- ❖ `mphy_jitter_type = svt_mphy_configuration::CONSTANT_PERIODIC_JITTER;`
- ❖ `jiter_control = <VALUE>;` // (division factor used to divide max PPM). You can set to 1 for max value.

- ❖ `number_of_clks_with_jitter = <VALUE> ;` // This indicates number clocks with jitter which will be repeated periodically.

8.36.2.2 RANDOM_JITTER

- ❖ `interface_type = SERIAL;`
- ❖ `enable_clock_jitter = 1;`
- ❖ `mphy_jitter_type = svt_mphy_configuration:: RANDOM_JITTER;` // No extra setting required. VIP will send random jitter within the limit.

8.36.2.3 DETERMINISTIC_JITTER

- ❖ `interface_type = SERIAL;`
- ❖ `enable_clock_jitter = 1;`
- ❖ `mphy_jitter_type = svt_mphy_configuration:: DETERMINISTIC_JITTER;`
- ❖ `number_of_clks_with_jitter = <VALUE> ;` // Number of clock used to increase PPM steady till it reaches MAX PPM and then roll over to start again.

8.36.2.4 ZERO_MEAN_JITTER

- ❖ `interface_type = SERIAL;`
- ❖ `enable_clock_jitter = 1;`
- ❖ `mphy_jitter_type = svt_mphy_configuration:: ZERO_MEAN_JITTER;`
- ❖ `number_of_clks_with_jitter = <VALUE> ;` // This indicates number clocks required to complete one cycle so mean results in zero jitter. Must be in multiple of 4. Recommended value 20 or more.

8.37 Time Scale Independent Timers

To set the timer value independent of time scale, use `set_timer` method of the `svt_mphy_configuration` class. For example, if you want to set the `reset_timer` to a value of 1us, then use the following:

```
this.sys_cfg.tx_cfg.set_timer("reset_timer",1, "us");
```

Instead of

```
this.sys_cfg.tx_cfg.reset_timer = 1us;
```

8.38 Support of Dynamic Reconfiguration for MPHY

Re-configure the MPHY M-TX and M-RX components with single lane models or with Lane Management model.

The following components are updated with `reconfigure` method, and allows you to change the static capabilities of MPHY model are as follows:

- ◆ `svt_mphy_tx`
- ◆ `svt_mphy_rx`
- ◆ `svt_mphy_moinitor`
- ◆ `svt_mphy_mport_lm_tx_common`
- ◆ `svt_mphy_mport_lm_rx_common`
- ◆ `svt_mphy_tx_agent`
- ◆ `svt_mphy_rx_agent`

- ◆ svt_mphy_model_agent
- ◆ svt_mphy_mport_lm_agent

1. Capabilities which could be re-configured are as follows:

- ◆ RX_HS_G1_SYNC_LENGTH_Capability 0x8B
- ◆ RX_HS_G1_PREPARE_LENGTH_Capability 0x8C
- ◆ RX_LS_PREPARE_LENGTH_Capability 0x8D
- ◆ RX_PWM_G6_G7_SYNC_LENGTH_Capability 0x93
- ◆ RX_HS_G2_SYNC_LENGTH_Capability 0x94
- ◆ RX_HS_G3_SYNC_LENGTH_Capability 0x95
- ◆ RX_HS_G2_PREPARE_LENGTH_Capability 0x96
- ◆ RX_HS_G3_PREPARE_LENGTH_Capability 0x97

The following code is used for usage of reconfigure method to modify the following RX_PWM_PREPARE_LENGTH_CAPABILITY are as follows:

```
class ENV extends uvm_env;
svt_mphy_mport_lm_agent master_agent;
-----

endclass

class TEST extends base_test;
---
task run();
begin
----
this.cfg.ls_prepare_length_capability = 7;
env.master_agent.reconfigure(cfg);
---
endtask

endclass
```

2. Clock periods of all PWM (PWM_G0 - PWM_G7) gears can also be reconfigured using the reconfigure method. With reconfigure MTX, you can transmit the data for one PWM gear at different frequencies in the same simulation.

Following is an example to reconfigure the PWM_G1 clock period:

```
class ENV extends uvm_env;
svt_mphy_mport_lm_agent master_agent;
-----

endclass

class TEST extends base_test;
---
task run();
begin
----
this.cfg.pwm_g1_clk_period = 350ns; // Default clock period is 300ns in VIP
env.master_agent.reconfigure(cfg);
---
endtask

endclass
```

```
endtask
```

```
endclass
```

**Note**

The logical code modifies the prepare length capability attribute and allows you to validate the dynamic prepare capabilities for DUT reception.

8.39 Dynamic Symbol Length Variation

A new configuration field is added in `svt_mphy_configuration` class for symbol length variation at run time.

Table 8-8 Variable Description

Variable Name	Variable Type	Description
<code>symbol_length_in_burst</code>	rand int	Signifies the symbol length to be considered in run time for LS mode. This value would be considered while transmitting and receiving symbols from TX and RX RMMI interfaces. Value of this field is applicable only when <code>enable_symbol_length_change_in_burst</code> (for LS mode) or <code>enable_symbol_length_change_in_hs</code> (for HS mode) is set. For example, If value of this parameter is set to 10 and <code>cfg.symbol_length</code> is set to 40 then for all LS-BURSTS <code>symbol_length</code> will change to 10.

This feature is disabled by default. It gets enabled by setting `enable_symbol_length_change_in_burst` or `enable_symbol_length_change_in_hs` configuration field to 1.

8.39.1 Use Model

LS MODE:

You must set the following fields in `shared_cfg` file or test case to enable this feature:

```
cfg.enable_symbol_length_change_in_burst = 1;  
cfg.symbol_length_in_burst = 10;
```

With the above settings, all LS-BURST is sent and received at `symbol_length` 10 in complete simulation.

All HS_BURST implements the symbol length set in `cfg.symbol_length` field.

HS MODE:

You must set the following fields in `shared_cfg` file or test case to enable this feature:

```
cfg.enable_symbol_length_change_in_hs = 1;  
cfg.enable_symbol_length_in_hs_gear = svt_mphy_types::HS_G2;  
cfg.symbol_length_in_burst = 20;
```

With the above settings, all HSG2 BURST is sent and received at `symbol_length` 20 in complete simulation.

All other HS gear and LS BURST implements the symbol length set in `cfg.symbol_length` field.

8.40 Setting of Symbol Length

MPHY VIP supports the following three symbol lengths specified in MPHY specification 10,20,and 40. The value of the symbol length can be modified by setting the value of configuration field `symbol_length`. For example,

```
mphy_cfg.symbol_length = 10;
mphy_cfg.symbol_length = 20;
```

MPHY VIP uses the following mentioned define to set the port length of TX_Symbol and RX_Symbol. The default value is 40 and it is present in `svt_mphy_port_defines.svi` file.

```
`ifndef SVT_MPHY_PORT_SYMBOL_LENGTH
`define SVT_MPHY_PORT_SYMBOL_LENGTH `d40
`endif
```

There are other signals like TX_DataNCtrl, TX_ProtDORDY that depends on TX_Symbol or RX_Symbol port length.

If DUT has less number of pins for TX_Symbol/RX_Symbol bus then to change the width of these buses in VIP, it is recommended to change the value of SVT_MPHY_PORT_SYMBOL_LENGTH define. All the other related port widths will get set by itself.

For example,

If DUT has TX_Symbol pin length as 20, then only one define is recommended to set

```
`define SVT_MPHY_PORT_SYMBOL_LENGTH `d20
```

Other dependent signals width will get set automatically to the following values:

```
TX_DataNCtrl    `d2
TX_ProtDORDY    `d2
```

8.41 Configurable Extra Bits in PREPARE State

MIPI Alliance Specification for MPHY Version 4.1 - 01 December 2016 Table 7 gives the formula for PREPARE period calculation. In this table,unit of PREPARE LENGTH is in SI. So in PREPARE PERIOD symbol boundary is maintained by MTX.

To distort the prepare boundary, the following new field is added in `svt_mphy_configuration` class.

This feature is very useful in checking the MK0 locking by MRX if symbol boundary is distorted. Using this feature in HS-MODE causes the `invalid_sync_symbol_check` to fail.



It is recommended to demote this check from the test case.

Table 8-9 New Field to Distort Prepare Boundary

Variable Name	Type	Description	Constraints
num_of_extra_bits_in_prepare	rand bit [3:0]	This field specifies the number of extra UI transmitted in PREPARE state on SERIAL interface. These extra bit distorts the 10 bit boundary of PREPARE and will test the MK0 locking at MRX in case of distorted symbol boundary.	<p>The valid range is 0-9.</p> <p>There is default constraint to regulate the bit insertion during normal operation.</p> <p>The constraint is as follows:</p> <pre>default_num_of_extra_bits_in_prepare { num_of_extra_bits_in_prepare == 0; }</pre>

8.41.1 Use Case

With the use of this field PREPARE length on SERIAL lines, will not be SYMBOL synchronized.

For example,

If `T_PWM_PREPARE = 1 SI` and `cfg.num_of_extra_bits_in_prepare = 4`, then total bits in PREPARE period will be 14 UI.

8.42 Configure Number of Clock to Enable invalid_cfgrdyn_signal_val_check Signal Check (RMMI Interface)

The M-PHY Verification IP provides the following feature:

- ❖ Configure number of clock to enable `invalid_cfgrdyn_signal_val_check` signal check. This feature provides control to enable `invalid_cfgrdyn_signal_val_check` on RMMI interface.
- ❖ In `svt_mphy_configuration` class `check_cfgrdyn_num_of_clock` field is added to enable `invalid_cfgrdyn_signal_val_check` signal check after configurable number of clocks.
- ❖ Default value for this field is set to one in `svt_mphy_configuration` class.
- ❖ You can configure this field from test case as per the following example:

```
/* Master agent configuration */
svt_mphy_mport_lm_agent_configuration master_cfg;

/* Slave agent configuration */
svt_mphy_mport_lm_agent_configuration slave_cfg;

master_cfg.tx_cfg.check_cfgrdyn_num_of_clock = 3;
slave_cfg.rx_cfg.check_cfgrdyn_num_of_clock = 3;
```

8.43 Check for RX_LCCRdDet Signal

MPHY TX sends LCC read commands when `TX_LCC_Enable` and read bits of `TX_LCC_Sequencer` are set to 1. M-RX receives these commands and asserts `RX_LCCRdDet` signal to indicate PA layer about the reception of LCC read commands.

MPHY VIP LM Monitor generates LCC Indication transaction to verify this feature whether MTX is driving LCC read commands when `TX_LCC_Sequencer` read bits are set to 1 or MRX is asserting `RX_LCCRdDet` after receiving read commands or not.

To enable the generation of LCC Indication from LM Monitor following configuration field is added in `svt_mphy_mport_lm_configuration` class.

Table 8-10 Variable

Variable Name	Variable Type	Description	Constraint
<code>generate_lcc_indication</code>	rand bit	Field to enable the generation of LCC INDICATION from LM MONITOR.	Reasonable constraint on <code>generate_lcc_indication</code> is set for value zero to make sure LCC INDICATION get generated only when this field is set to 1 from test bench.

8.43.1 Use case

Set `generate_lcc_indication` field to one in both master and slave cfg in `mphy_lm_shared_cfg` file.

```
master_cfg.tx_cfg.generate_lcc_indication = 1'b1;
master_cfg.rx_cfg.generate_lcc_indication = 1'b1;
slave_cfg.tx_cfg.generate_lcc_indication = 1'b1;
slave_cfg.rx_cfg.generate_lcc_indication = 1'b1;
```

After setting above configuration fields, LM Monitor generate transaction with `xact.lcc_indication` bit set to one for BURSTS in which LCC commands are driven.

You can write a logic in scoreboard to compare this transaction between master and slave agent LM monitor. If there is mismatch then scoreboard reports an error.



Note This feature requires update in scoreboard.

8.44 Dithering Support

Delaying the start of each HS-BURST with reference to the last BURST, with some random number of UI, accomplishes the desired dithering on SERIAL interface. To enable dithering feature in M-PHY VIP following configuration parameter is added in `svt_mphy_configuration` class. This feature is applicable for lane management topology.

Table 8-11 Variable Details

Variable Name	Variable Type	Description	Constraint
dithering_start_positions	rand bit [3:0]	This field is used to enable dithering and insert random number of UI in STALL state as per Table 65 of the specification.	Valid range is between 0 to 10.

By default value of `dithering_start_positions` is zero (dithering is disabled). Dithering will be enabled, if any value within the range 1 to 10 is assigned to `dithering_start_positions`.

SERIAL Monitor has a check named `invalid_dithering_val_check` for dithering. This check gets enabled when dithering is enabled. This check reports if dithering value on serial lines is not within the range as per Table 65 of the specification.

For example, If `cfg.dithering_start_positions = 2` and number of extra UI on serial lines are more or less than 4-5 UI range then this check is reported.

8.44.1 Use Case for Dithering

When dithering is enabled, VIP includes extra UI's in STALL state during repeated `HS_BURST`. Value used for extra UI's is as per the Table 65 of the specification.

For example,

```
if, cfg.dithering_start_positions = 4;  
then the extra number of UI in STALL state randomly selects a value among 3-4-5-6.
```

Similarly if `cfg.dithering_start_positions = 6` then the extra number of UI in STALL state randomly selects a value among 2-3-4-5-6-7.

8.45 Allow LCC_READ in all PWM gears.

To enable this feature, user needs to set the following configuration parameter `allow_lcc_read_in_all_gear`.

For example:

```
master_cfg.tx_cfg. allow_lcc_read_in_all_gear = 1'b1;  
slave_cfg.tx_cfg. allow_lcc_read_in_all_gear = 1'b1;  
master_cfg.rx_cfg. allow_lcc_read_in_all_gear = 1'b1;  
slave_cfg.rx_cfg. allow_lcc_read_in_all_gear = 1'b1;
```

8.46 Difference Between Prepare State and RX_Burst

While writing sequences should the VIP wait for Prepare state or `RX_Burst` assertion, before transmitting the data?

When the VIP is the controller, it is recommended to wait for Prepare state rather than waiting for `RX_Burst`.

**Note**

The PHY behavior is explained as defined in the Function Specification, when the MRX PHY detects the prepare period, then it asserts `RX_Burst` on interface.

When the VIP is the controller, the `RX_SymbolClk` detects assertion of `RX_Burst` due to clocking block behavior.

If the VIP waits on the `RX_Burst` rather than waiting on Prepare state, then the VIP does a `CfgUdpt` during the time difference between `RX_Burst` and Prepare state that results to a configuration change malfunction leading to simulation failure.

Recommended approach:

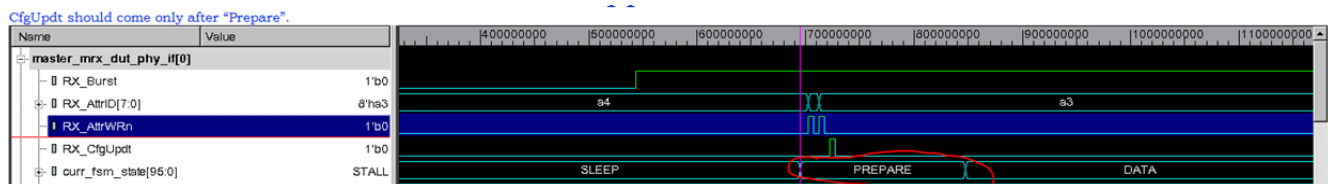
```
wait(env.top_status.master_rx_status.curr_fsm_state == svt_mphy_types::PREPARE);
```

```
//wait(tb_top.master_mrx_dut_phy_if[0].RX_Burst);
```

```
//wait(tb_top.master_mrx_dut_phy_if[1].RX_Burst);
```

Waveform results after the recommended approach:

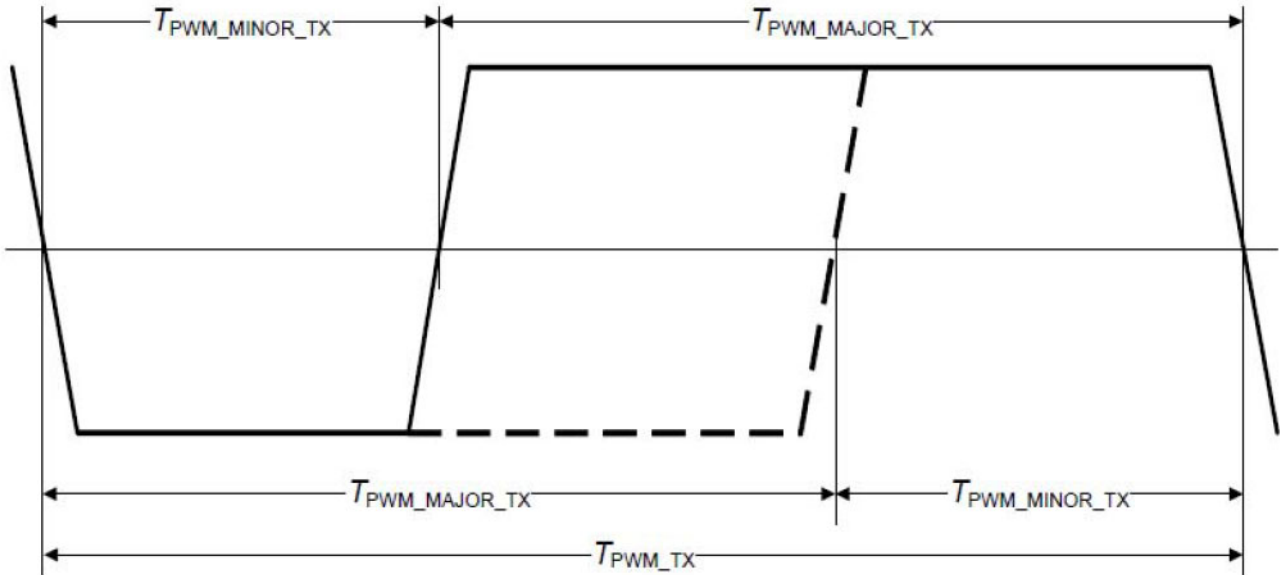
Figure 8-12 Waveform



8.47 PWM Transmit Ratio Usage

The PWM receive bit duration $T_{\text{PWM_RX}}$ is defined as the duration between zero crossings of two consecutive falling edges of a differential signal at the PWM-RX input. $T_{\text{PWM_MINOR_RX}}$, $T_{\text{PWM_MAJOR_RX}}$, and $T_{\text{PWM_RX}}$ are shown in the following figure:

Figure 8-13 PWM Transmit Ratio Waveform



The PWM receive bit duration for all PWM GEARS ($T_{\text{PWM_RX}}$) is the sum of its durations, $T_{\text{PWM_MINOR_RX}}$ and $T_{\text{PWM_MAJOR_RX}}$, as shown in the following equation:

$$T_{\text{PWM_RX}} = T_{\text{PWM_MINOR_RX}} + T_{\text{PWM_MAJOR_RX}}$$

$T_{\text{PWM_MINOR_RX}}$ and $T_{\text{PWM_MAJOR_RX}}$ are determined by $T_{\text{PWM_RX}}$ and the PWM receive ratio $k_{\text{PWM_RX}}$ for PWM-G1 and higher PWM GEARS. $K_{\text{PWM_RX}}$ is defined as the ratio of $T_{\text{PWM_MAJOR_RX}}$ and $T_{\text{PWM_MINOR_RX}}$ of one PWM bit, as shown in following equation:

$$K_{\text{PWM_RX}} = \frac{T_{\text{PWM_MAJOR_RX}}}{T_{\text{PWM_MINOR_RX}}}$$

PWM Transmit Ratio supported features are as follows:

- ❖ MPHY VIP supports PWM transmit ratio (variation in PWM duration).
- ❖ MPHY VIP has `fixed_minor_factor` as configuration parameter, used for fixing minor factor as defined in 0.40 to 0.25 of the functional specification.
- ❖ Major fixed factor, from 0.64 to 0.75 will be owned by VIP internally.
- ❖ The MPHY VIP has the following user replaceable defines, with values 1.7(0.63/0.37) and 2.57(0.72/0.28) respectively:
 - ◆ ``SVT_MPHY_PWM_MIN_MAJOR_TO_MINOR_FACTOR`
 - ◆ ``SVT_MPHY_PWM_MAX_MAJOR_TO_MINOR_FACTOR`
- ❖ Overwrite the above defines in top file to access the entire range from 1.5(0.60/0.40) to 3.00(0.75/0.25).
- ❖ VIP has checker which checks min and max value of PWM transmit ratio `pwm_ratio_violation_check`.
- ❖ The PWM transmit variations are as follows:

- ◆ Overwrite the following defines in the top file to access the entire range from 1.5 to 3.
 - ◇ ``SVT_MPHY_PWM_MIN_MAJOR_TO_MINOR_FACTOR 1.5`
 - ◇ ``SVT_MPHY_PWM_MAX_MAJOR_TO_MINOR_FACTOR 3.0`
- ◆ Set `fixed_minor_value` range from 0.40 to 0.25.
 - ◇ `<mphy_cfg>.fixed_minor_range = <#VALUE>`

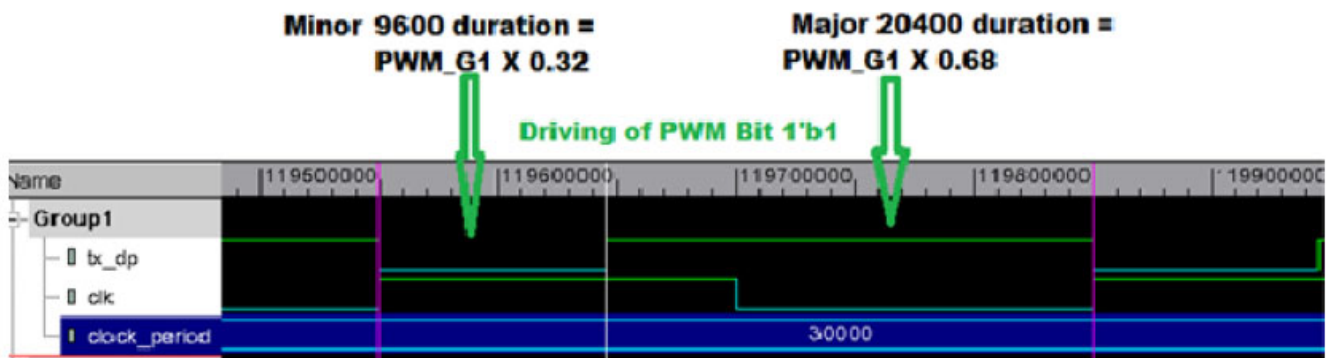


Note If the defines are not overwritten with 1.5 and 3.0, then the check reports an error for getting a value of `fixed_minor_factor` out of 1.7 to 2.57 (0.37 to 0.28) range.

8.47.1 Observation

The following wave snippet with `fixed_minor_factor` 0.32, $k = 0.68/0.32$

Figure 8-14 PWM Transmit Ratio Wave for `fixed_minor_factor`



8.48 Enable External REF Clock (refclk) Support in VIP

MPHY Verification IP is enhanced with VIP RMMI and serial clock from given external `refclk` signal.

User need to do following in-order to enable this feature :

1. Set the `support_ref_clk` configuration parameter to `EXT_REF_CLK_19_20_MHZ` if 19.20MHz or `EXT_REF_CLK_26_MHZ` frequency is used for 26MHz frequency for `refclk` generation.
2. Connect (assign) user generated `refclk` to VIP interface `refclk_ext` in top.

For example,

a. Configuration setting:

```
/** LM agent configuration */
svt_mphy_mport_lm_agent_configuration master_cfg;
svt_mphy_mport_lm_agent_configuration slave_cfg;
/** Enable external refclk support using 26 MHZ. */
/** Set Master TX cfg */
shared_cfg.master_cfg.tx_cfg.supported_ref_clk =
svt_mphy_configuration::EXT_REF_CLK_26_MHZ;
/** Set Master RX cfg */
```

```
shared_cfg.master_cfg.rx_cfg.supported_ref_clk =
svt_mphy_configuration::EXT_REF_CLK_26_MHZ;
/** Set Slave TX cfg */
shared_cfg.slave_cfg.tx_cfg.supported_ref_clk =
svt_mphy_configuration::EXT_REF_CLK_26_MHZ;
/** Set Slave RX cfg */
shared_cfg.slave_cfg.rx_cfg.supported_ref_clk =
svt_mphy_configuration::EXT_REF_CLK_26_MHZ;
```

b. Assign external refclk to VIP `refclk_ext`.

```
/**
 * External refclk to interface, refclk_ext will be used to generate clock if
configuration
 * 'supported_ref_clk' is set to EXT_REF_CLK_26_MHZ or EXT_REF_CLK_19_20_MHZ
 * In place "RefClk" user has to provide user refclk signal.
 */
/** RMMI Interface */
assign master_mtx_dut_phy_if[i].refclk_ext = RefClk;
assign master_mtx_dut_controller_if[i].refclk_ext = RefClk;
/** Serial Interface */
assign master_mtx_serial_if[i].refclk_ext = RefClk;
assign slave_mtx_serial_if[i].refclk_ext = RefClk;
assign master_mrx_serial_if[i].refclk_ext = RefClk;
assign slave_mrx_serial_if[i].refclk_ext = RefClk;
```

8.49 Max Lane Skew

A lane to lane skew checker is added to reports an error whenever the skew between the lanes exceeds the supported maximum skew.



Note

By-default define `SVT_MPHY_MAX_LANE_SKEW` value is 40UI. The define value can also be updated. For example,

```
`define SVT_MPHY_MAX_LANE_SKEW    <any supported value>
```

8.50 TX_DP and TX_DN Lines Driving User Control

Control the driving of TX_DP and TX_DN differential lines transmission using the `<mphy_cfg>.diff_lines_relative_value` configuration variable to TX_DP only, TX_DN only or both lines driving the same value.

For example,

```
<mphy_cfg>.diff_lines_relative_value == svt_mphy_types::DIFF_P_ONLY;    OR
<mphy_cfg>.diff_lines_relative_value == svt_mphy_types::DIFF_N_ONLY;    OR
<mphy_cfg>.diff_lines_relative_value == svt_mphy_types::DIFF_P_N_SAME_VALUE_ONLY;
```



Note

Default value of this variable is set to the default driving behavior for differential lines. For example, both TX_DP and TX_DN relatively opposite of each other.

An additional check is added at the receiver, which is disabled by default as `differential_p_n_lines_check`. However, a WARNING message on any pattern of differential lines especially during BURST or SAVE state is displayed.

8.51 User-Defined Payload Pattern

The following are the user-defined payload patterns:

- ❖ Determines the type of payload at the LM layer with a new transaction variable named `data_pattern_value`.
- ❖ While randomizing the data sequence use `end_of_burst` set and select the data pattern as `ALL_ONES`, `ALL_ZEROS`, `ALL_MARKERS`, `ALL_FILLERS`, or random data, if none of these values are chosen.

For more details on the LM transaction class ENUM variable, see the HTML class reference guide.

8.52 Enabling Debug Port View

Enable debug port view by setting `use_debug_interface` bit, using `uvm_config_db`, in `build_phase()` of test.

```
uvm_config_db#(bit)::set(null, "*", "use_debug_interface", 1'b1);
```

8.53 Corrupt 8b10b ADAPT Pattern

In order to corrupt 8b10b ADAPT pattern perform the following setting:

1. Set error kind : `EN_3B4B_5B6B_ERROR`
2. Value of corruption: `mask_or_value`
3. Position at which error should be injected : `error_position_prbs`
4. Weight of exception : `EN_3B4B_5B6B_ERROR_wt`

The following example is to inject 8b10b error:

```
class lm_transaction_adapt_8b10b_error_exc_error_exception_list extends
svt_mphy_transaction_exception_list;
```

```
svt_mphy_transaction_exception xact_exc = new("xact_exc");
```

```
constraint tx_error_kind {
    foreach (typed_exceptions[i]) {
        //Set Error kind EN_3B4B_5B6B_ERROR
        typed_exceptions[i].error_kind ==
svt_mphy_transaction_exception::EN_3B4B_5B6B_ERROR;
        //Update value to be replace in place of 8b10b value.
        typed_exceptions[i].mask_or_value == 'h333;
    }
}
```

```
function new(string name =
"lm_transaction_adapt_8b10b_error_exc_error_exception_list",
svt_mphy_transaction_exception xact_exc = null);
    super.new(name, xact_exc);
    xact_exc = this.xact_exc;
```

```
xact_exc.set_constraint_weights(0);
//Set 8b10b error weight
xact_exc.EN_3B4B_5B6B_ERROR_wt = 1;
xact_exc.error_position_prbs = 1;
this.randomized_exception = xact_exc;
endfunction : new

constraint test_constraint1 {
    //Set constrain to only inject error in ADAPTSTART transaction.
    if(xact_exc.xact.primitive_name == svt_mphy_transaction::ADAPTSTART) {
        num_exceptions == 1;}
    else {
        num_exceptions == 0;}
}

endclass
```

8.54 Control De-Assertion of TX_ADAPT_REQ

Perform the following steps to de-assert the TX_ADAPT_REQ:

1. Set the configuration parameter `deassert_adapt_req` to select position for de-assertion of TX_ADAPT_REQ before/after TX_ADAPT_ACTIVE goes low

For example,

```
master_cfg.tx_cfg.deassert_adapt_req = svt_mphy_configuration::
BEFORE_ADAPT_ACTIVE;
```

or

```
master_cfg.tx_cfg.deassert_adapt_req = svt_mphy_configuration::
AFTER_ADAPT_ACTIVE;
```

2. Along with the position you can program number of clocks by configuration parameter `number_of_clks_to_deassert_adapt_req`

For example,

```
master_cfg.tx_cfg.number_of_clks_to_deassert_adapt_req = 5;
```

8.55 Corrupt 650bits of ADAPT Pattern Length

In order to resize 650 bits of ADAPT pattern perform the following setting:

1. Set error kind: `RESIZE_ADAPT_PATTERN_ERROR`
2. Resize length by: `mask_or_value[7:0]`
3. Add/subtract length: `maks_or_value[1:0]`
4. Weight of exception: `RESIZE_ADAPT_PATTERN_ERROR_wt`

Following is an example to resize ADAPT 650bits pattern:

```

class lm_transaction_adapt_pattern_resize_error_exc_error_exception_list extends
svt_mphy_transaction_exception_list;

    svt_mphy_transaction_exception xact_exc = new("xact_exc");

    constraint tx_error_kind {
        foreach (typed_exceptions[i]) {
            typed_exceptions[i].error_kind ==
svt_mphy_transaction_exception::RESIZE_ADAPT_PATTERN_ERROR;
            //Resize length +/- 10 bytes
            typed_exceptions[i].mask_or_value[7:0] == 10;
            //Random selection
            //2'b00 : 650 bits - 10bytes
            //2'b01 : 650 bits + 10bytes
            typed_exceptions[i].mask_or_value[9:8] inside {2'b00,2'b01};
        }
    }

    function new(string name =
"lm_transaction_adapt_pattern_resize_error_exc_error_exception_list",
svt_mphy_transaction_exception xact_exc = null);
        super.new(name,xact_exc);
        xact_exc = this.xact_exc;
        xact_exc.set_constraint_weights(0);
        xact_exc.RESIZE_ADAPT_PATTERN_ERROR_wt = 1;
        this.randomized_exception = xact_exc;
    endfunction : new

    constraint test_constraint1 {
        if(xact_exc.xact.primitive_name == svt_mphy_transaction::ADAPTSTART) {
            num_exceptions == 1;
        }
        else {
            num_exceptions == 0;
        }
    }

endclass

```

8.56 80bit Symbol Length

You can verify the DUT with 80bit symbol length.

1. Define SVT_MPHY_PORT_SYMBOL_LENGTH to 80
 - a. define SVT_MPHY_PORT_SYMBOL_LENGTH 80
2. Program mphy configuration "symbol_length" to 80.
 - a. <mphy_configuration>.symbol_length = 80

After this programming, VIP sends the Symbol in terms of 80bit symbol length.

8.57 TX/RX Reset Mode Support

You must perform these steps to exercise RESET mode feature:

1. Set VIP configuration to MPHY 5.0 by:

```
<mphy_configuration>.mphy_protocol_version::MPHY_VERSION_50;
```


2. Connect TX_ResetMode and RX_ResetMode signal to DUT.
3. Update LS and HS line reset duration:

LS:

```
<mphy_configuration>.t_linereset = <value>  
<mphy_configuration>.t_linereset_detect = <value>
```

HS:

```
<mphy_configuration>.t_linereset_hs= <value>  
<mphy_configuration>.t_linereset_detect_hs = <value>
```

8.58 Support for LCC in MPHY5.0

LCC is not supported from MPHY5.0. To enable LCC, you need to set `support_lcc_in_mphy_50` configuration parameter:

```
<mphy_configuration>.support_lcc_in_mphy_50 = 1'b1
```

8.59 Support for All PWM Gear in MPHY5.0

PWMG2 to PWMG7 is not supported from MPHY5.0. To enable all PWM gear, you need to set `support_all_pwm_in_mphy_50` configuration parameter:

```
<mphy_configuration>.support_all_pwm_in_mphy_50 = 1'b1
```




9

Troubleshooting

This chapter provides some useful information that can help you troubleshoot the common issues encountered while using M-PHY VIP.

This chapter discusses the following topics:

- ❖ [Using Trace Files for Debugging](#)
- ❖ [Enabling Tracing](#)
- ❖ [Setting Verbosity Levels](#)
- ❖ [Debug Port](#)

9.1 Using Trace Files for Debugging

Trace files are used to record transaction information in a table format through the simulation. Trace files consist of two sections, a header specifying the fields being recorded and trace section containing data captured during simulation.

For M-PHY VIP, the following two trace files (one each for TX and RX direction) are generated:

- ❖ Data Transaction Trace File
- ❖ Control Transaction Trace File

9.1.1 Data Transaction Trace File

Data transaction trace file captures DATA SAP information as mentioned in Tables 36 and 37 of *MIPI Alliance Specification for MPHY Version 4.1 - 01 December 2016*.

```
<env_instance_name> <agent instance name>.<direction  
identifier_monitor>.data_trans.transaction_trace
```

where,

agent instance name	indicates the instance name of the corresponding VIP Agent
direction identifier_mon	indicates the TX or RX Monitor Trace File

9.1.2 Control Transaction Trace File

Control transaction trace file captures DATA SAP information as mentioned in Tables 47 and 48 in *MIPI Alliance Specification for M-PHY Version 4.1 01 December 2016*.

```
<env_instance_name> <agent instance name>.<direction  
identifier_monitor>.control_trans.transaction_trace
```

where,

agent instance name	indicates the instance name of the corresponding VIP Agent
direction identifier_mon	indicates the TX or RX Monitor Trace File

9.2 Enabling Tracing

Tracing can be enabled or disabled through the `enable_tracing` variable which is defined in the `svt_mphy_agent_configuration` class. The default value of `enable_tracing` variable is 0, which means that tracing is disabled by default.

To enable tracing:

- ❖ Set the value of `enable_tracing` variable to “1” in the derived class that is used in your VIP agent. Assuming that `vip_cfg` is of `svt_mphy_agent_configuration` class, the following code snippet is used to enable tracing:

```
vip_cfg.enable_tracing = 1;
```

9.3 Setting Verbosity Levels

You can set VIP debug verbosity levels either in the testbench, or as an option during run-time.

9.3.1 Setting Verbosity in the Testbench

To set the verbosity level in the testbench, you can use the UVM specified log-levels in the code. The components are extended from `uvm_report_object`.

You can use the following `uvm_report_object` method to change the verbosity for the Agent:

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

The following table lists all the possible verbosity levels `<level>`:

Table 9-1 Agent Verbosity Levels

Verbosity Level	Description
UVM_NONE	Report is always printed. Verbosity level setting cannot disable it.
UVM_LOW	Report is issued if configured verbosity is set to UVM_LOW or above.
UVM_MEDIUM	Report is issued if configured verbosity is set to UVM_MEDIUM or above
UVM_HIGH	Report is issued if configured verbosity is set to UVM_HIGH or above.
UVM_FULL	Report is issued if configured verbosity is set to UVM_FULL or above

9.3.2 Setting Verbosity During Run-Time

One of the following methods can be used to set the verbosity during run-time:

- ❖ **Enabling the specified severity in the VIP, DUT, and Testbench**

Use example: VCS

```
vcs <other run time options> +UVM_VERBOSITY=UVM_MEDIUM
```

- ❖ **Enabling the specified severity to specific sub-classes of VIP**

Instance verbosity is supported by UVM directly,

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

For Example:

```
+uvm_set_verbosity="uvm_test_top.mphy_tx*",_ALL,UVM_HIGH,main
```

- ◆ The following method can also be used to enable component based verbosity:

Add `+vip_verbosity=component_name:verbosity` in `sim_run_options` file.

For Example:

- ❖ Enabling the verbosity of MPHY TX driver

```
+vip_verbosity=svt_mphy_tx:verbose
```

- ❖ Enabling the verbosity of multiple files in single line and with different verbosity levels

```
+vip_verbosity=svt_mphy_tx:debug,svt_mphy_rx:verbose,svt_mphy_monitor:debug
```

9.4 Debug Port

Debug port is an additional interface which is available along with protocol signals while debugging waveforms. It is useful in debugging simulations, when additional information about the model is required for understanding the context of verification.

At present, M-PHY VIP provides information about the FSM state of TX and RX modules as defined in *Table 52 and 56 respectively, of the MIPI Alliance Specification for M-PHY Version 4.1 01 December 2016*.

The following clocking block and its corresponding modport is defined in all of the M-PHY interfaces:

```
//-----
Modport used to connect VIP MPHY M-RX to debug interface signals
//-----

modport mrx_debug_port(clocking mrx_debug_cb);

/**
 *Clocking block that defines MPHY VIP RX's debug signal synchronization and
 *directionality.
 */

clocking mrx_debug_cb @(posedge clk);
    default input #`SVT_MPHY_RX_PORT_SERIAL_SETUP_TIME
        output #`SVT_MPHY_RX_PORT_SERIAL_HOLD_TIME;
        output fsm_state;
endclocking
```

9.4.1 Enabling Debug Port

Debug port is enabled or disabled through the `use_debug_interface` variable which is defined in the `svt_mphy_agent_configuration` class. The default value of this variable is 0, which means that debug port is disabled by default.

To enable the debug port:

- ❖ Set the value of `use_debug_interface` variable to 1 in the derived class that is used in your VIP agent. Assuming that `vip_cfg` is of `svt_mphy_agent_configuration` class, the following code snippet is used to enable tracing:

```
vip_cfg.use_debug_interface = 1;
```

10

Reporting Problems

10.1 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.

10.2 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.fsd* 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*.

10.3 Enabling and Specifying Debug Automation Features

Debug Automation is enabled through the use of a run-time plusarg named *+svt_debug_opts*. This 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 10-1 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 <code>start_time</code> . Two values are accepted in all methodologies: <code>DEBUG</code> and <code>VERBOSE</code> . UVM and OVM users can also supply the verbosity that is native to their respective methodologies (<code>UVM_HIGH/UVM_FULL</code> and <code>OVM_HIGH/OVM_FULL</code>). If this value is not supplied then the verbosity defaults to <code>DEBUG/UVM_HIGH/OVM_HIGH</code> . When this feature is enabled, then all VIP instances that are enabled for debug will have their messages routed to a file named <code>svt_debug.transcript</code> .

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`.

10.4 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.
- ❖ `transaction_trace`: Log files that records all the different transaction activities generated by VIPs.
- ❖ `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).

10.5 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.

10.5.1 VCS

The following compile-time command must be added:

```
-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.

10.5.2 Questa

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

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

10.5.3 Incisive

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

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

10.6 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](#).

10.7 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.

10.8 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.

