DesignWare® Cores PCI Express Controller

User Guide

Version 6.00a
June 2022

Copyright Notice and Proprietary Information

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

Destination Control Statement

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

Disclaimer

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

Trademarks

Synopsys and certain Synopsys product names are trademarks of Synopsys, as set forth at https://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.

Synopsys, Inc.

www.synopsys.com

Contents

Revision History 9

Preface 11
Product Description 11
Related Documentation 11
Reference Documentation 11
Web Resources 11
Synopsys Statement on Inclusivity and Diversity 12
Customer Support 12

Chapter 1
Product Overview 15
1.1 General Product Description 16
1.1.1 Verification Environment Overview 16
1.1.2 coreConsultant Overview 17
1.2 Standards 18
1.3 Applications 19

Chapter 2
Configuring the Controller 21
2.1 Design Flow 23
2.2 Setting up your Environment 24
2.3 Creating a Workspace 25
2.4 Configuring the Controller 27
2.4.1 Prime Profile 29
2.5 Simulating the Controller 31
2.5.1 Running the Simulation 31
2.5.2 Checking Simulation Status and Results 33
2.6 Synthesizing the Controller 34
2.6.1 Synthesizing the Controller for an ASIC 34
2.7 Formal Equivalence Checking using Formality 45
2.8 Inserting Design-For-Test using Design Compiler 46

2.9 Running ATPG using TestMAX™ . . . 47
2.10 Running STA using PrimeTime . . . 49
2.11 Running SpyGlass® . . . 51
2.12 Running VC SpyGlass . . . 53
2.12.1 CDC and RDC Verification . . . 54
2.12.2 VC SpyGlass Folder Files . . . 54
2.12.3 Running VC SpyGlass in Interactive Mode using VC SpyGlass GUI . . . 55
2.13 Running Low Power Static Verification using VCLP . . . 57
2.14 Creating Optional Views and Reports . . . 58
2.14.1 Activity-Specific Reports and Views . . . 58
2.14.2 Format of Activity-Specific Reports . . . 60
2.14.3 Setting Preferences for Report Generation . . . 60
2.14.4 Setting Preferences for IP-XACT . . . 61
2.14.5 Accessing Reports . . . 61
2.14.6 Generating and Accessing Activity-Specific Views . . . 62
2.14.7 Populating IP-XACT File Manually . . . 62
2.15 Exporting a Controller From coreConsultant to Your Chip Design Database . . . 63
2.15.1 Generating a Sample Controller Instantiation . . . 63
2.15.2 Synthesizing to a Device Outside of coreConsultant . . . 64
2.15.3 Exporting Formality, DFT, and ATPG Scripts . . . 65

Chapter 3
Integrating the Controller with the PHY, or Application RTL, or Verification IP . . . 67
3.1 Integrating the Controller with the PHY . . . 68
3.1.1 Overview . . . 68
3.1.2 Connecting the PIPE I/O Interface . . . 69
3.1.3 Registering the PIPE Interface . . . 72
3.1.4 Integrating the PCIe Controller and Synopsys PHYs . . . 79
3.1.5 Integrating the PCIe Controller and Third-party PHYs . . . 81
3.1.6 Bifurcation Guidelines . . . 85
3.2 Integrating with Your Application RTL . . . 92
3.2.1 Overview . . . 92
3.2.2 Instantiating the Controller RTL in Your Design . . . 93
3.2.3 Connecting the Reset I/O Interface . . . 93
3.2.4 Connecting the Clock I/O Interface . . . 95
3.2.5 Connecting the RAM I/O Interface . . . 97
3.2.6 Connecting the Application I/O Interfaces . . . 97
3.2.7 Design Example . . . 97
3.3 Integrating the Controller and VIP using VTB . . . 98
3.3.1 Limitations . . . 98
3.3.2 Setting Up and Running the VTB . . . 99

Chapter 4
UPF Flow and Methodology . . . 101
4.1 Specifying the Power Intent . . . 103
4.2 Simulation . . . 105
4.3 Importing PHY UPF . . . 106

4.4 Synthesis . . . 107
4.5 Clocks . . . 108

Chapter 5
Programming . . . 109

Chapter 6
FPGA-Based Prototyping . . . 111
6.1 FPGA Prototyping Overview . . . 112
6.1.1 Guidance for Successful FPGA-Based Prototyping . . . 112
6.1.2 DWC IP coreConsultant FPGA Versus DW IP Prototyping Kits . . . 112
6.1.3 FPGA Implementation Flow . . . 113
6.1.4 Unlocking coreConsultant GUI FPGA Activity . . . 114
6.2 FPGA Prototyping Using coreConsultant . . . 115
6.2.1 FPGA RTL Configuration Guide . . . 115
6.2.2 Tested Configuration Examples . . . 116
6.2.3 Synthesizing the Controller for an FPGA . . . 118
6.3 FPGA Debug Guide . . . 122

Appendix A
Selecting PCI Express IP for Your Design . . . 123
A.1 PCI Express Overview . . . 124
A.1.1 PCI Express System Example: PC Motherboard Based System . . . 124
A.1.2 PCI Express System Example: Chip-to-Chip System . . . 125
A.1.3 PCI Express Protocol Stack . . . 126
A.1.4 PCI Express Links . . . 127
A.1.5 Digital IP to PHY Interface (PIPE) . . . 128
A.2 Selecting PCI Express IP for Your Application . . . 129
A.2.1 Selecting the Lane Width . . . 129
A.2.2 Selecting Mixed-Signal PCI Express IP (PHY) . . . 129
A.2.3 Selecting Digital PCI Express IP Types . . . 130
A.3 Implementing PCI Express in Your Design . . . 134
A.4 Summary . . . 136

Appendix B
Client Interfaces Configuration Guide (without AXI Bridge) . . . 137

Appendix C
Additional Information (coreConsultant) . . . 147
C.1 Troubleshooting . . . 148
C.1.1 Specify Configuration Activity . . . 148
C.1.2 Create Gate-Level Netlist Activity . . . 148
C.2 Creating a Batch Script . . . 149
C.3 Help Information . . . 150
C.4 Workspace Directory Structure Overview . . . 151
C.5 Dumping Debug Information When Problems Occur . . . 153

Appendix D
Synthesis Troubleshooting Guide . . . 155

D.1 Analyzing Timing Violations in the Synopsys Controller 156
D.1.1 Reading the Timing Report 156
D.1.2 Input and Output Delay 159
D.1.3 Clocks 159
D.1.4 Anatomy of a Timing Path 159
D.1.5 Opening a coreConsultant Generated Netlist in DC Shell 161
D.1.6 Reporting Timing 162
D.2 Resolving Timing Violations in the Synopsys Controller 163
D.2.1 Clocks 163
D.2.2 False Paths 163
D.2.3 Input and Output Delay 164
D.2.4 Levels of Logic 164
D.2.5 Optimization Across Hierarchy Boundaries 164
D.2.6 Multiple Voltage Threshold Libraries 165
D.2.7 Configuration Parameter Settings to Improve Timing 165

Appendix E
Clock Domain Crossing (CDC) Methodology 171
E.1 Synthesis Constraints 172
E.2 Synchronizers Used in PCIe 173

Appendix F
Simulation Troubleshooting Guide 175
F.1 Gen1/Gen2 Link Training 176
F.1.1 Basic Clock, Reset, and Simulation Setup Checks 176
F.1.2 LTSSM in Detect Substates 177
F.1.3 LTSSM in Polling Substates 179
F.1.4 LTSSM in Configuration Substates 181
F.1.5 Smaller Than Expected Link Width 183
F.1.6 Speed Change to Gen2 Data Rate 184
F.1.7 LTSSM in Recovery Substates 184
F.2 Gen3 and Above Equalization 187
F.2.1 Gen3 Equalization Setup Checks 187
F.2.2 Speed Change to Gen3 and Above 188
F.2.3 Equalization Phase0 191
F.2.4 Equalization Phase1 191
F.2.5 Equalization Phase2/Phase3 191
F.3 Gen5 and Above Features 196
F.3.1 Equalization Bypass to Highest Data Rate 196
F.3.2 No Equalization Needed 196
F.3.3 Precoding 197
F.4 Upstream and Downstream Port Enumeration 198
F.4.1 Upstream Port: No Completion Transmitted in Response to Inbound Cfg TLP 198
F.4.2 Downstream Port: Outbound CFG TLP not Transmitted 199
F.5 Memory Transactions 201
F.5.1 Inbound Transaction: MWr/MRd TLP not Seen on Application Interface 201
F.5.2 Outbound Transaction: MWr/MRd TLP not Received by Link Partner 203
F.6 Top Level Reference File Generation 206

Appendix G

Silicon Debug Guide 207
G.1 System Integration 208
G.1.1 System Checks 208
G.1.2 Clocking 210
G.1.3 Reset 212
G.1.4 PHY 213
G.2 Link Training 216
G.2.1 Receiver Detection 216
G.2.2 Broken Lanes 218
G.2.3 Speed Change 219
G.2.4 Link Equalization 220
G.2.5 Example Debug Flow for Link Training Issues 225
G.3 Link Stability 226
G.3.1 Receiver(Rx) Errors 226
G.3.2 Correctable Errors 229
G.3.3 Uncorrectable Errors 231
G.4 Power Management 235
G.4.1 L0s 235
G.4.2 L1 and L1SS 237
G.4.3 L2 238

Appendix H
Application Interface Integration Guide 241
H.1 AXI Interfaces 242
H.1.1 Other Integration Considerations 246
H.2 Native Transmit and Receive Interfaces (XALI0, XALI1, XALI2, Bypass, RTRGT1) 249
H.3 Data Bus Interface (DBI) 250
H.4 External Local Bus Interface (ELBI) 251
H.5 Message Signaled Interface (MSI) 252
H.6 MSI-X Interface 254
H.7 Vendor Message Interface (VMI) 256
H.8 Power Budgeting 257
H.9 System Information Interface (SII) 258
H.9.1 SII: General Core Control Signals 258
H.9.2 SII: Configuration Information Signals 258
H.9.3 SII: Debug Signals 259
H.9.4 SII: Diagnostic Control Signals [Optional] 259
H.9.5 SII: Diagnostic Status Signals [Optional] 260
H.9.6 SII: General Message Reception Signals 267
H.9.7 SII: Interrupt Signals 267
H.9.8 SII: Link Reset/Status Signals 268
H.9.9 SII: LTR Message Generation/Reception Signals 269
H.9.10 SII: OBFF Message Generation/Reception Signals 269
H.9.11 SII: PM Unlock and Error Message Signals 269
H.9.12 SII: Power Management Signals 269
H.9.13 SII: Precision Time Management Signals 270
H.9.14 SII: RAS Data Protection Signals 270
H.9.15 SII: Receive Control/CPL Timeout Signals 271
H.9.16 SII: Transmit Control Signals 271

H.10 FLR 272
H.11 Target Completion LUT 273
H.12 RAS D.E.S Signals 274
H.13 Configuration Intercept Interface (CII) Signals 275
H.14 PHY Register Bus Interface. 276

Revision History

Revision history for PCI Express Controller User Guide:

Version	Date	Description
6.00a	June 2022	A version of this book with change tracking is available at https://www.synopsys.com/dw/doc.php/iip/DWC_pcie/6.00a/doc/DWC_pcie_ctl_user_change.pdf

Preface

This preface contains the following sections:

- "Product Description" on page 11
- "Related Documentation" on page 11
- "Reference Documentation" on page 11
- "Synopsys Statement on Inclusivity and Diversity" on page 12
- "Customer Support" on page 12

Product Description

This document describes the Synopsys DesignWare Cores for PCI Express (PCIe) Controller which provides a solution to implement a PCI Express Root Port, Endpoint, or Switch application. The controllers are compliant with the *PCI Express Base Specification, Revision 6.0, Version 1.0*. For a complete overview of the controller features, see the "Product Overview" chapter of the *Databook*. The DesignWare Cores for PCIe include the following products:

- PCI Express Dual Mode (DM) controller
- PCI Express Root Port (RP) controller
- PCI Express Endpoint (EP) controller
- PCI Express Switch Port (SW) controller

Related Documentation

After installing the controller, the DWC PCI Express documents can be found under: *$DESIGNWARE_HOME/iip/*DWC_pcie_ctl*/latest/doc*.

Reference Documentation

This user guide assumes familiarity with the basic terminology and concepts outlined in the *PCI Express Base Specification, Revision 6.0, Version 1.0* available from http://www.pcisig.com.

Web Resources

- DesignWare IP product information: https://www.synopsys.com/designware-ip.html

- Your custom DesignWare IP page: https://www.synopsys.com/dw/mydesignware.php
- Documentation through SolvNet: https://solvnetplus.synopsys.com(Synopsys password required)
- Synopsys Common Licensing (SCL): http://www.synopsys.com/keys

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.

Customer Support

To obtain support for your product, prepare the required files and contact the support center using one of the methods described:

- Prepare the following debug information, if applicable:
  - For environment set-up problems or failures with configuration, simulation, or synthesis that occur within coreConsultant or coreAssembler, select the following menu:
    - **File > Build Debug Tar-file**

    Check all the boxes in the dialog box that apply to your issue. This option gathers all the Synopsys product data needed to begin debugging an issue and writes it to the <*core tool startup directory*>/debug.tar.gz file.
  - For simulation issues outside of coreConsultant or coreAssembler:
    - Create a waveforms file (such as VPD or VCD).
    - Identify the hierarchy path to the DesignWare instance.
    - Identify the timestamp of any signals or locations in the waveforms that are not understood.
- *For the fastest response*, enter a case through SolvNetPlus:

  a. https://solvnetplus.synopsys.com

Note SolvNetPlus does not support Internet Explorer. Use a supported browser such as Microsoft Edge, Google Chrome, Mozilla Firefox, or Apple Safari.

  b. Click the **Cases** menu and then click **Create a New Case** (below the list of cases).

  c. Complete the mandatory fields that are marked with an asterisk and click **Save**.

  Make sure to include the following:

  - **Product L1:** DesignWare Cores
  - **Product L2:** PCI Express

d. After creating the case, attach any debug files you created.

For more information about general usage information, refer to the following article in SolvNet-Plus:

https://solvnetplus.synopsys.com/s/article/SolvNetPlus-Usage-Help-Resources

- Or, send an e-mail message to support_center@synopsys.com (your email will be queued and then, on a first-come, first-served basis, manually routed to the correct support engineer):
  - Include the Product L1 and Product L2 names, and Version number in your e-mail so it can be routed correctly.
  - For simulation issues, include the timestamp of any signals or locations in waveforms that are not understood
  - Attach any debug files you created.
- Or, 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

1

Product Overview

The DesignWare Cores PCI Express (PCIe) controller provides a solution to implement a PCI Express Root Port, Endpoint, or Switch application. You use coreConsultant to configure your specific design implementation of the PCIe controller. For information on how to download and install the PCIe controller, see the *Installation Guide*.

This chapter provides a general description of the DesignWare Cores PCIe controller. The following topics are covered:

- "General Product Description" on page 16
- "Standards" on page 18
- "Applications" on page 19

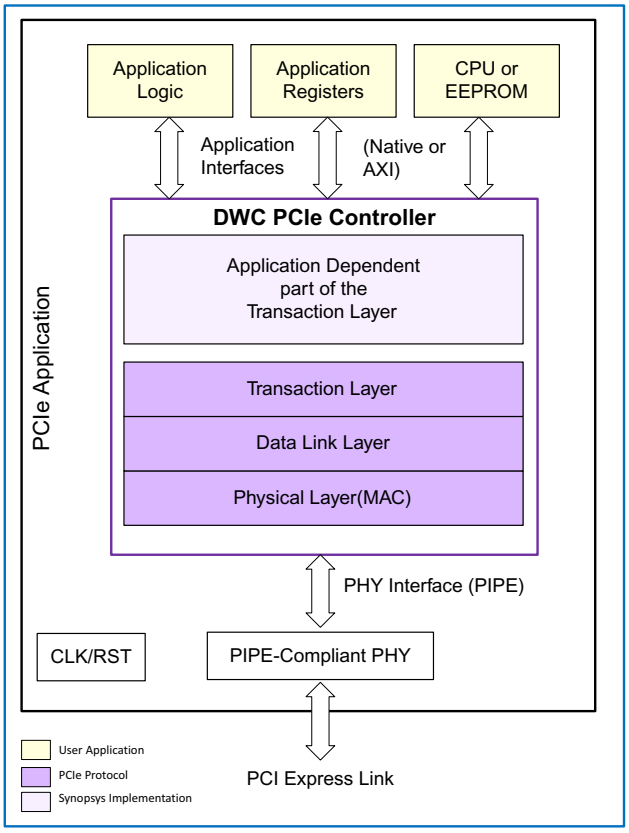
1.1 General Product Description

The DesignWare Cores for PCIe include the following products:

- Dual Mode (DM) controller
- Root Port (RP) controller
- Endpoint (EP) controller
- Switch Port (SW) controller

For a complete overview of the controller features, see the "Product Overview" chapter of the *Databook*.

Figure 1-1 PCI Express Application



1.1.1 Verification Environment Overview

The Verification environment for the controller includes a packaged unit test environment (VTB) and separate Verilog IP (VIP) components. The VTB runs the packaged self-checking regression tests through coreConsultant to validate the configured controller. For more information, see "Simulating the Controller" on page 31. You can also use the Verilog testbench (VTB) as an example when developing your system testbench. For more information, see "Integrating the Controller and VIP using VTB" on page 98.

1.1.2 coreConsultant Overview

The Synopsys coreConsultant tool allows you to configure, simulate, synthesize, and export the controller to your design flow. The final output from coreConsultant is a set of configured RTL files and scripts for integration into your SoC design flow. The coreConsultant features and tasks include:

- A graphical user interface (GUI) to guide you through design flow activities.
- All activities might also be performed through a command line interface. For information, see the coreConsultant User Guide and Help menu in coreConsultant.
- Interactive parameter selection; the coreConsultant tool checks for consistent settings and automatically derives any dependent parameters from your choices.
- Testbench simulation and synthesis configuration consistent with your parameter choices.
- Docbook files documenting your final configured design.

For more information on how to configure your controller and generate RTL files, as well as running synthesis and simulation, see "Configuring the Controller" on page 21.

1.2 Standards

The PCIe controller implements the following standards:

- PCI Express Base Specification, Revision 6.0, Version 1.0

  Access to this specification requires membership in PCI-SIG.

  Download site:

  http://pcisig.com/specifications
- PHY Interface for the PCI Express, SATA, USB 3.2, DisplayPort, and USB4 Architectures (PIPE) Version 4.4.1, Version 5.1.1, Version 6.1.1

  Download site:

  http://www.intel.com/technology/pciexpress/devnet/resources.htm.
- PCI Express Card Electromechanical Specification, Revision 4.0

  Access to this specification requires membership in PCI-SIG.

  Download site:

  https://members.pcisig.com/wg/PCI-SIG/document/download/8250

1.3 Applications

The PCIe controller is targeted for the following applications:

- Motherboard components for desktop and mobile computers
- Graphics devices
- Add-in cards for desktop and mobile computers
- Components and add-in cards in server applications
- Embedded applications
- Data communications equipment
- Telecommunications equipment
- Storage devices
- Wireless devices
- Hyper-transport to PCIe bridge
- SATA to PCIe bridge
- PCI Express switch

Attention The Synopsys PCI Express Port IP does not implement a full Root Port, Endpoint, or Switch. You must add your application logic to the controller to create a complete Root Port, Endpoint, or Switch.

2

Configuring the Controller

This chapter shows you how to use the coreConsultant tool to configure, simulate, synthesize, and export the controller to your own design flow. The final output from the coreConsultant is a set of configured RTL files and scripts for integration into your SoC design flow.

The topics in this chapter are as follows:

- "Design Flow" on page 23
- "Setting up your Environment" on page 24
- "Creating a Workspace" on page 25
- "Configuring the Controller" on page 27
- "Simulating the Controller" on page 31
- "Synthesizing the Controller" on page 34
- "Formal Equivalence Checking using Formality" on page 45
- "Inserting Design-For-Test using Design Compiler" on page 46
- "Running ATPG using TestMAX™" on page 47
- "Running STA using PrimeTime" on page 49
- "Running SpyGlass®" on page 51
- "Running VC SpyGlass" on page 53
- "Running Low Power Static Verification using VCLP" on page 57
- "Creating Optional Views and Reports" on page 58
- "Exporting a Controller From coreConsultant to Your Chip Design Database" on page 63

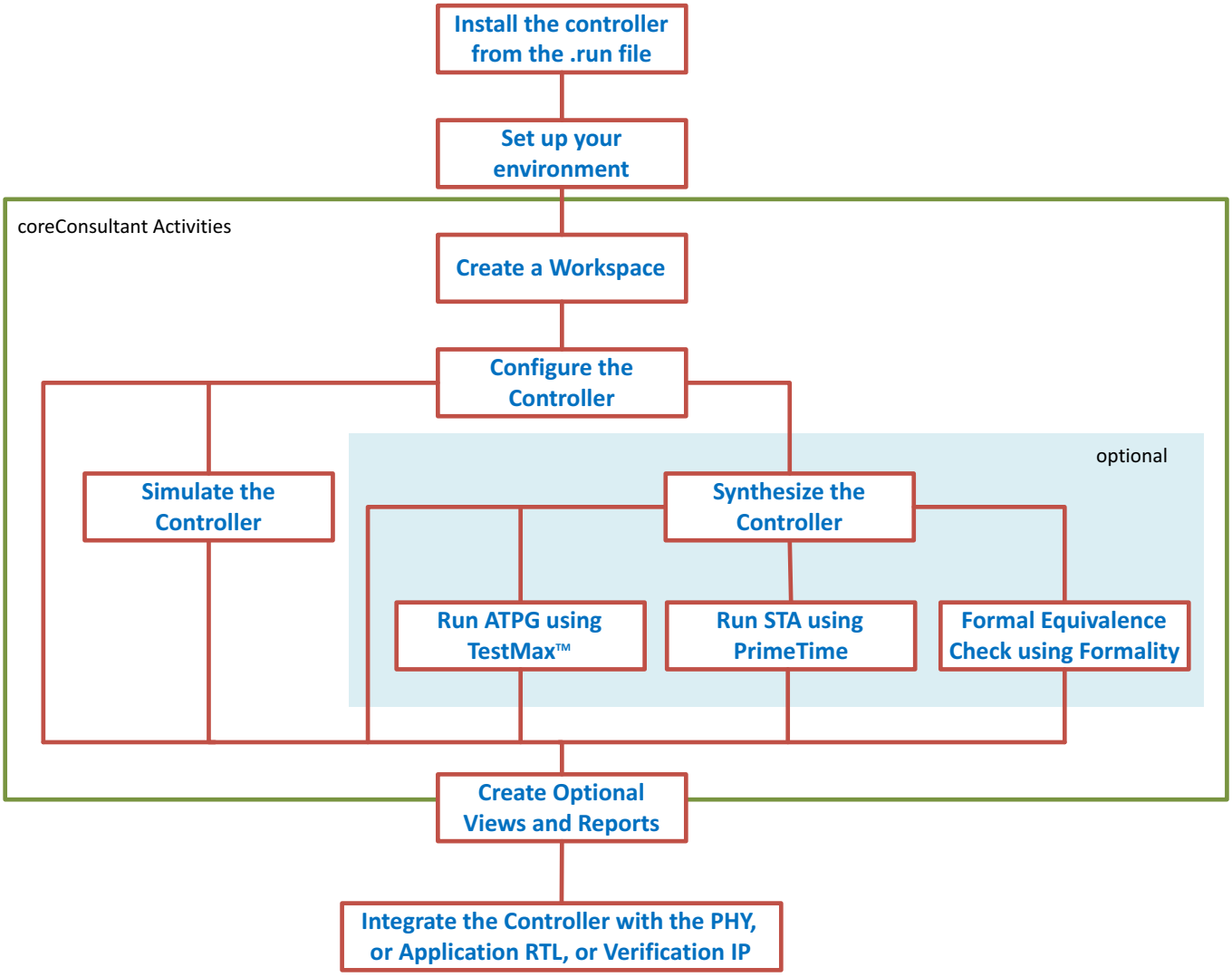
Hint If you are using more than one DesignWare controller, coreAssembler is recommended. The coreAssembler tool includes all the features of coreConsultant and enables you to configure, automatically interconnect, synthesize, and export a subsystem, as opposed to a single component. For more information, see the coreAssembler documents mentioned in "Help Information" on page 150.

Attention	All the coreConsultant GUI screen shots are for illustration purposes only. They might not be an exact representation of the coreConsultant during the controller configuration.

2.1 Design Flow

The coreConsultant GUI guides you through the controller design flow. Most coreConsultant activities are optional as indicated by the various paths shown in Figure 2-1 and need not be completed before exporting the configured RTL to your chip design flow.

Figure 2-1 Design Flow



2.2 Setting up your Environment

Note Correctly setting up your installation directory and environment ensures that you can quickly configure the controller.

To confirm that you have a correct installation directory and environment setup, follow these steps:

- Confirm that you have installed the controller design files and fully set up your environment as described in the *Installation Guide*.
- Confirm that you are using the required versions of coreConsultant and your preferred synthesis and simulation tools as specified in the *Installation Guide*.
- Check that `$DESIGNWARE_HOME` points to your installation base directory. Executing the following command in a Unix terminal shell lists a directory structure similar to that described in the "DESIGNWARE_HOME Tree Overview" section of the *Installation Guide*.

  ```
  % ls $DESIGNWARE_HOME
  ```

- When you are using Synopsys Design Compiler or Fusion compiler, check that `$SYNOPSYS` points to the Synopsys tools tree.
- Check that the PCIe controller licenses are installed correctly on your license server by using the `lmstat` command. For example, enter:

  ```
  % lmstat -a -c $LM_LICENSE_FILE | grep DWC-PCIE
  ```

  For more information, see the "Setting Up Your Licenses" and "Verify License Requirements" sections of the *Installation Guide*.

Attention Do not proceed unless you have completed all of these steps.

2.3 Creating a Workspace

A workspace is a local Unix directory structure containing your configured copy of the PCI Express controller. For more information about this directory structure, see "Workspace Directory Structure Overview" on page 151. You can create several workspaces to experiment with different design alternatives.

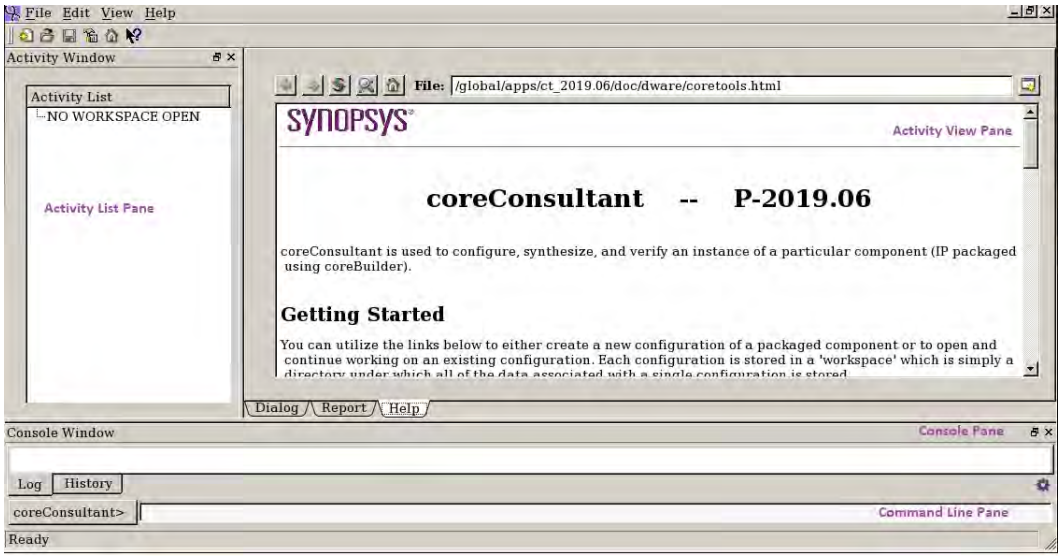
To create a workspace, follow these steps:

1. In a Unix shell, navigate to a directory where you plan to locate your workspace.
2. Start the coreConsultant GUI:

   ```
   % coreConsultant &
   ```

   Figure 2-2 shows the initial coreConsultant screen.

Figure 2-2 Initial coreConsultant Screen



3. Create a workspace:

   From the welcome page (see Figure 2-2) click on the name of the IP controller that you want to configure.

Hint

Here is a common problem that can occur:

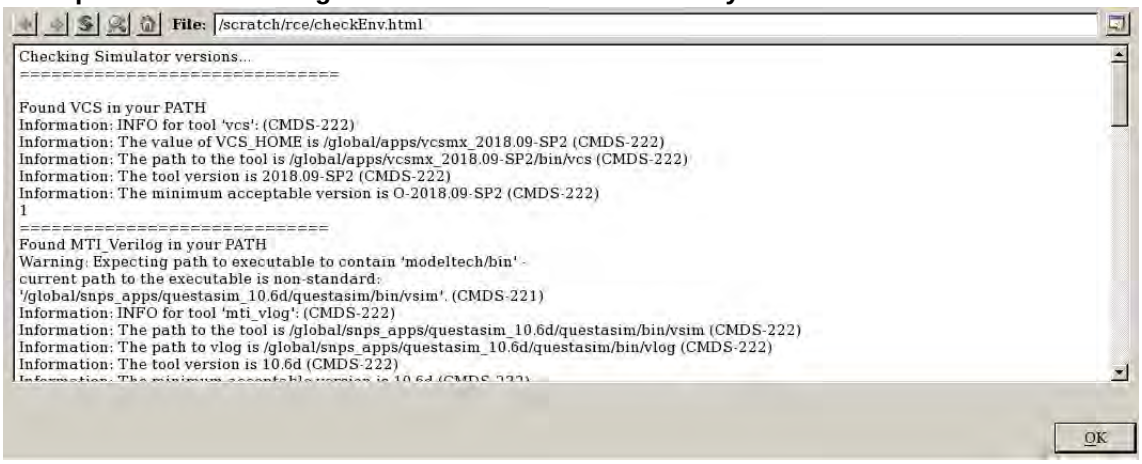
- **The PCIe IP controller is not visible in the initial coreConsultant page**.

  The $DESIGNWARE_HOME environment variable is not set (or is not pointing to your IP installation directory) in the shell from which coreConsultant started. For more information, see "Setting up your Environment" on page 24.

---

4. Validate your installation and your environment by selecting **Help > Check Environment** from the menu bar. A report window (see Figure 2-3) shows the results.

Figure 2-3 Sample Results of Using the Check Environment Activity

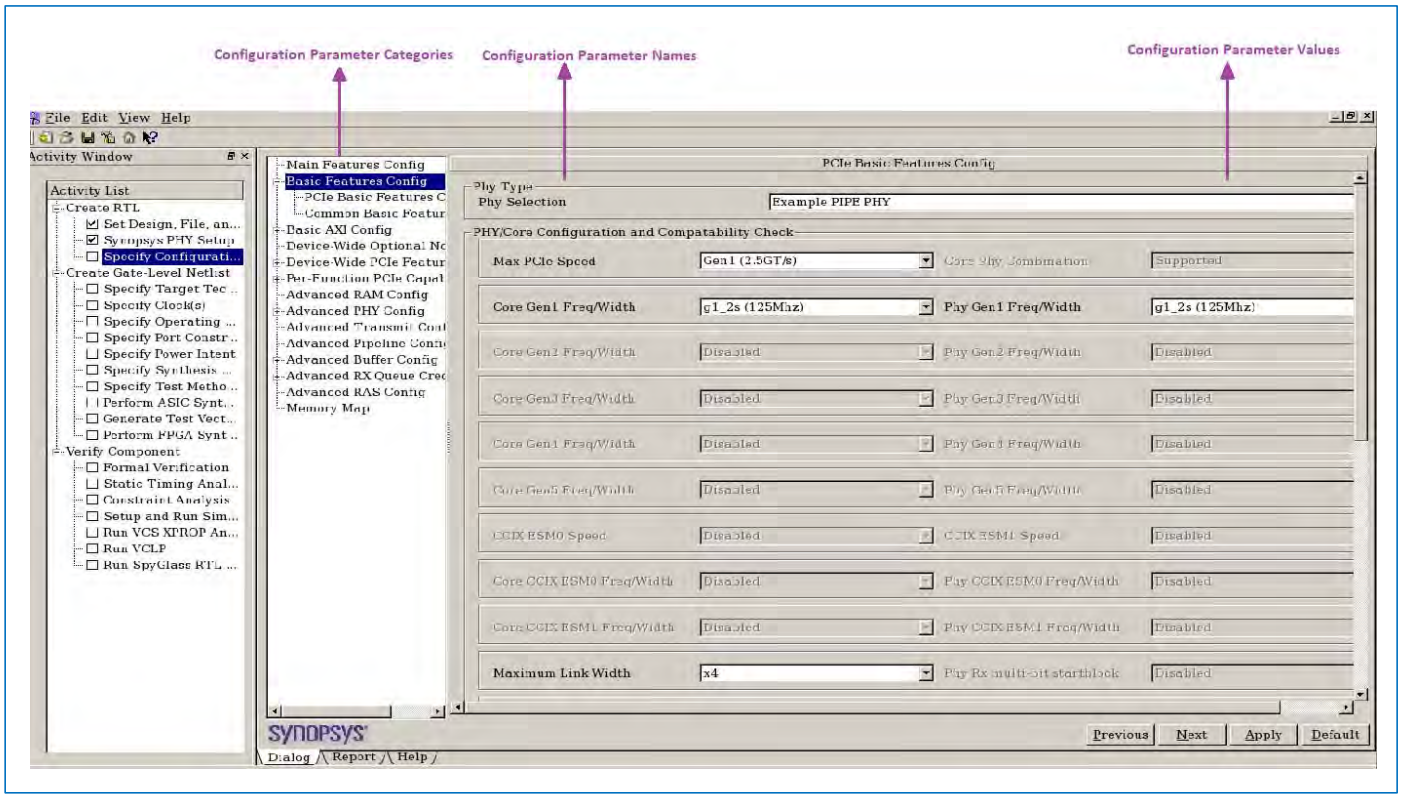


Correct any reported errors by modifying your Unix environment setup (see "Setting up your Environment" on page 24) or through the **Edit > Tool Installation Roots** menu.

2.4 Configuring the Controller

After you have created your workspace, you configure the controller and create the RTL using the **Specify Configuration** activity dialog boxes as shown in the following figure.

Figure 2-4 Specify Configuration Activity



1. Specify your configuration

   Select options to enable or disable features, set memory sizes and default values for configuration registers as appropriate for your design.

   - You can use the defaults for an initial simulation and synthesis trial. However, you must select or enter specific values required to implement your design.
   - Make sure that you understand the definition of each parameter and change the default only when it is not suitable for your application.

     You can access detailed information about each parameter by right-clicking on the parameter label and selecting *What's This* or by selecting the Help tab.
   - The coreConsultant tool enforces the parameter interdependencies interactively. For example, when you select a single-function configuration, coreConsultant disables the parameter selection for functions 1–7.

     For more information about the configuration parameters, see the "Parameters" chapter of the *Reference Manual* and also the "Client Interfaces Configuration Guide (without AXI Bridge)" on page 137".

Note Use the **Set Design and File Prefix** activity when you plan to instantiate the controller more than once in your design. Note that this activity is already checked. It is used to create a unique name for each design in your component. It ensures that each instance (including all sub-blocks) has a unique name to prevent files from the different configurations getting mixed up.

To specify configuration of the controller using prime profiles, see "Prime Profile" on page 29.

2. Generate RTL

   Click **APPLY** button to generate configured RTL code for the controller. The coreConsultant tool then checks your parameter values and generates the configured RTL code in the `<workspace>/src/` directory.

   You can return to the **Specify Configuration** activity to reconfigure controller (create new RTL) at any time.

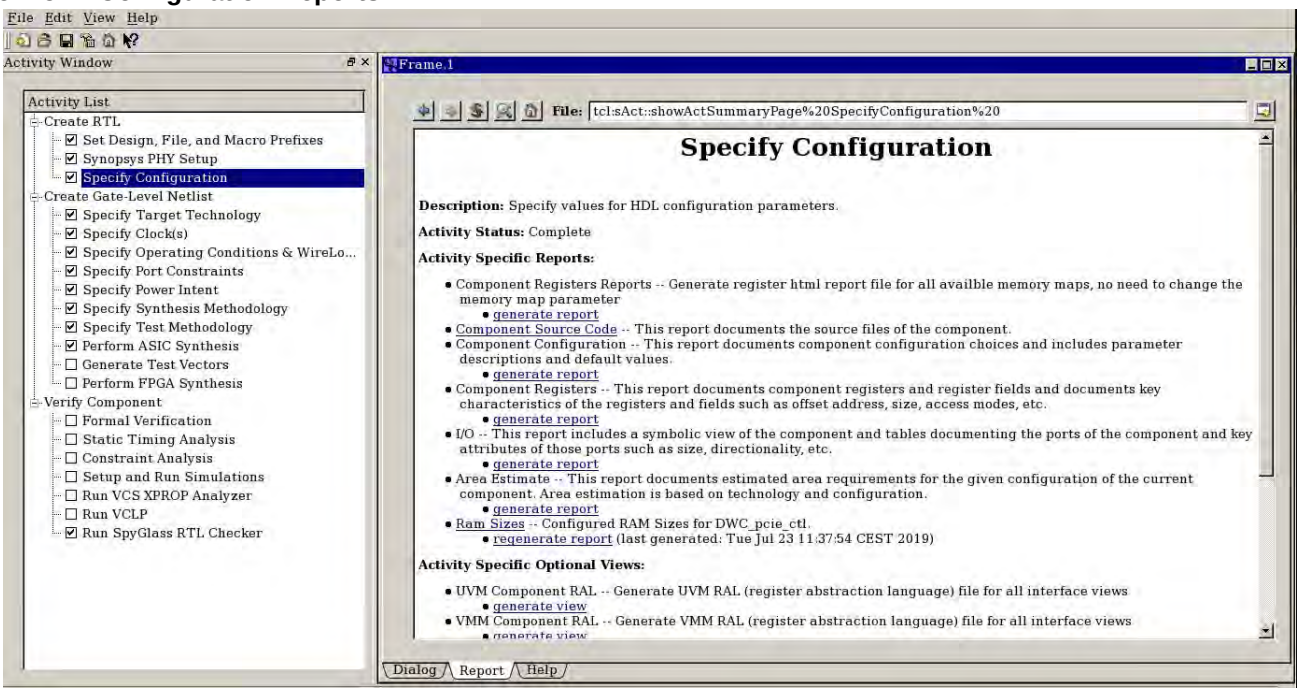
Hint Create a batch script so that you can recreate your exact configuration at a later stage in case you delete or overwrite your current workspace. For more information, see "Creating a Batch Script" on page 149.

3. View Reports and Generate Optional Views and Reports

   After you have configured the controller, coreConsultant generates several configuration reports. A RAM size report is also generated. You can access these from the Report tab (shown in Figure 2-5).

   You can also optionally generate **Activity Specific Reports** and **Activity Specific Optional Views**. For more information, see "Creating Optional Views and Reports" on page 58. For example, to generate an example component instantiation, click the **generate view** link as indicated in Figure 2-5.

Figure 2-5 Configuration Reports



Note

- In case of issues during or after the **Specify Configuration** activity, see "Troubleshooting" on page 148.
- At this point you can also generate a gate-level netlist ("Synthesizing the Controller" on page 34).

2.4.1 Prime Profile

A prime profile is an application-focused configuration of the controller. All prime profile configurations are pre-qualified by Synopsys.

2.4.1.1 Selecting a Prime Profile

To select a prime profile

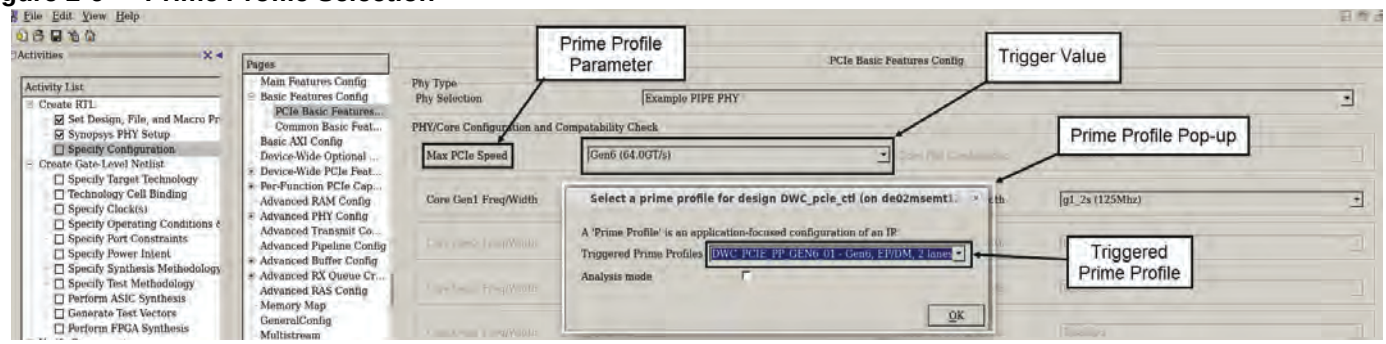
- Set the prime profile parameter to a trigger value specified in Table 2-1 during the **Specify Configuration** activity.

  A prime profile pop-up appears in coreConsultant, as shown in Figure 2-6.
- Select one of the triggered prime profiles and click **OK**.

Table 2-1 Prime Profile Configurations

Prime Profile Parameter	Trigger Value	Triggered Prime Profiles	Description
CX_MAX_PCIE_SPEED	6	DWC_PCIE_PP_GEN6_01 - Gen6, EP/DM, 2 lanes, AXI3, IDE, 1-8PFs, 16/32/64VFs, ATU, ATS, PRS, TPH, PASID, AER, RAS.DP, RAS.DES, PTM, MSI/MSIX, DMWr, ATOMICS	Gen6 x2 lane controller Dual Mode or End Point. AXI3 256-bit application interface. Supports 1-8 Physical functions and 16, 32 or 64 Virtual functions.
		DWC_PCIE_PP_GEN6_02 - Gen6, EP/DM, 4 lanes, AXI3, IDE, 1-129PFs, 16/32/64VFs, ATU, ATS, PRS, TPH, PASID, AER, RAS.DP, RAS.DES, PTM, MSI/MSIX, DMWr, ATOMICS	Gen6 x4 lane controller Dual Mode or End Point. AXI3 512-bit application interface. Supports 1-129 Physical functions and 16, 32 or 64 Virtual functions.
		DWC_PCIE_PP_GEN6_03 - Gen6, EP/DM, 4 lanes, AXI3, 1-8PFs, 16/32/64VFs, ATU, ATS, PRS, TPH, PASID, AER, RAS.DES, PTM, MSI/MSIX, DMWr, MPEM	Gen6 x4 lane controller Dual Mode or End Point. AXI3 256-bit application interface. Supports 1-8 Physical functions and 16, 32 or 64 Virtual functions.
		DWC_PCIE_PP_GEN6_04 - Gen6, EP/DM, 8 lanes, Native, 1-8PFs, 16/32/64VFs, ATS, PRS, TPH, PASID, AER, RAS.DES, PTM, MSIX, DMWr, MPEM	Gen6 x8 lane controller Dual Mode or End Point. Native 512-bit application interface. Supports 1-8 Physical functions and 16, 32 or 64 Virtual functions.

Figure 2-6 Prime Profile Selection



Many design parameters default to pre-qualified values for prime profiles. coreConsultant sets these parameters as ReadOnly. You can edit non-ReadOnly parameters and click **Apply** in the **Specify Configuration** activity window to generate the RTL code.

Note It is not possible to exit prime profile mode after enabling it.

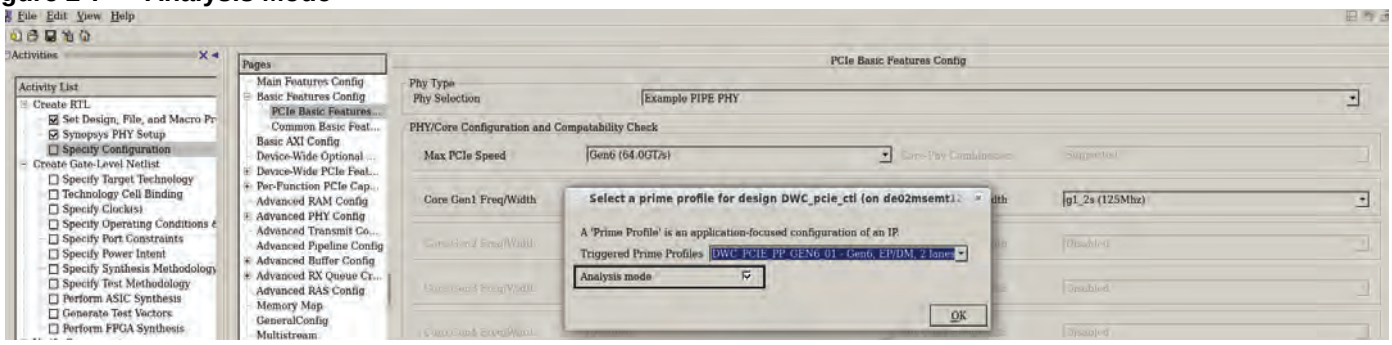
2.4.1.2 Analysis Mode

If the existing prime profiles do not meet your requirements, modify the default ReadOnly values of parameters by clicking **Analysis mode**, as shown in Figure 2-7. Create a batch script by following the instructions in "Creating a Batch Script" on page 149.

Share this batch script with Synopsys so that Synopsys can evaluate the modifications and provide new prime profile configurations along with installation instructions. You need not switch to a new controller release to use the new prime profile configuration of the controller shared by Synopsys.

Note It is not possible to complete the **Specify Configuration** activity in Analysis mode. So, the RTL cannot be generated in Analysis mode.

Figure 2-7 Analysis Mode



2.5 Simulating the Controller

This section shows you how to simulate the PCI Express controller in the coreConsultant environment. You can simulate the controller RTL in the supplied Verilog Testbench (VTB). The VTB allows you to re-execute the packaged regression tests on the selected configuration of the controller for the purpose of verifying your configuration. This environment includes a set of self-checking tests, and automated test suite execution through coreConsultant.

The VTB also serves as a reference for integrating the controller, PHY, and Synopsys VIP into your system-level test environment. The VTB shows basic transactions for configuring the environment, link establishment, and running basic transactions. For more information, see "Setting Up and Running the VTB" on page 99".

Note You can use the VTB as a basis for a system test bench.

The steps involved in simulating the controller are:

- "Running the Simulation" on page 31
- "Checking Simulation Status and Results" on page 33

Attention If you have selected the "Custom" PHY option, then you need to manually integrate your PHY simulation model before proceeding with the **Simulate** activity. For more information, see "Integrating the Controller with the PHY, or Application RTL, or Verification IP" on page 67. Otherwise, you can proceed immediately to the **Simulate** activity.

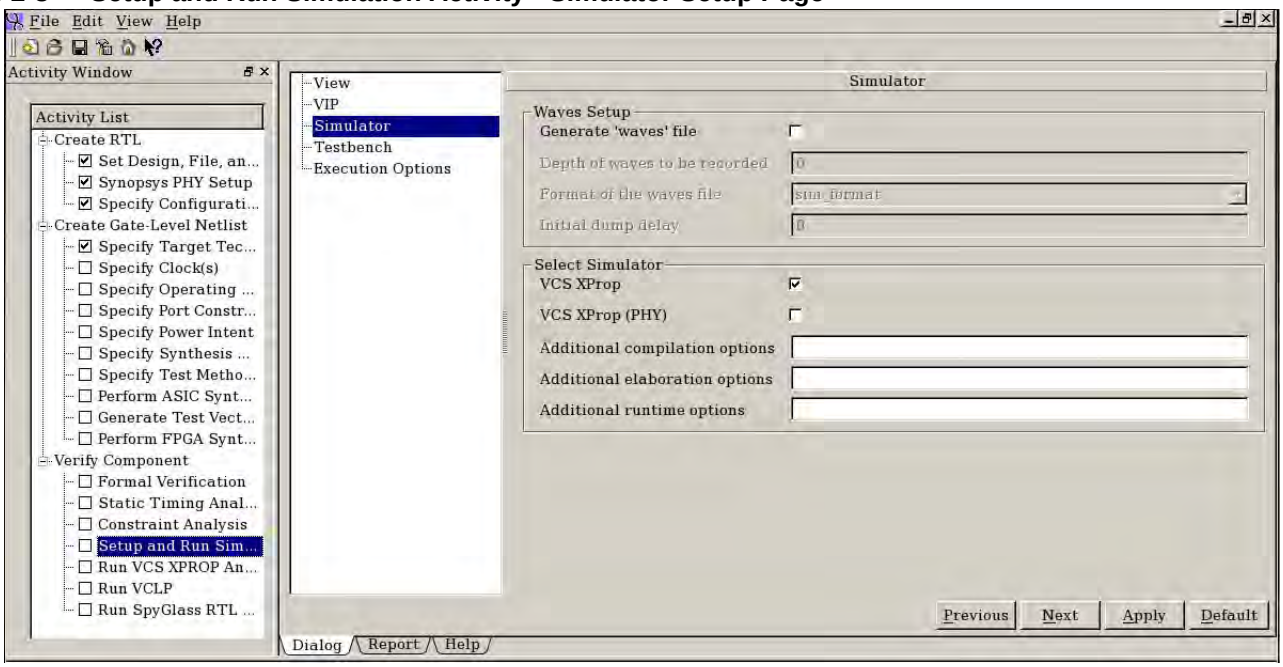
2.5.1 Running the Simulation

The **Simulate** activity (Figure 2-8) is where you select your simulation options and run the simulation. When the simulation is complete, the results are available in the Report tab of the Simulate dialog.

Before You Start

Check that you have the latest tool versions installed and your environment variables are set correctly (see "Setting up your Environment" on page 24). To check your tools, click the **Help > Check Environment** menu item. Correct any reported errors by modifying your Unix environment setup or through the **Edit > Tool Installation Roots** menu.

Figure 2-8 Setup and Run Simulation Activity - Simulator Setup Page

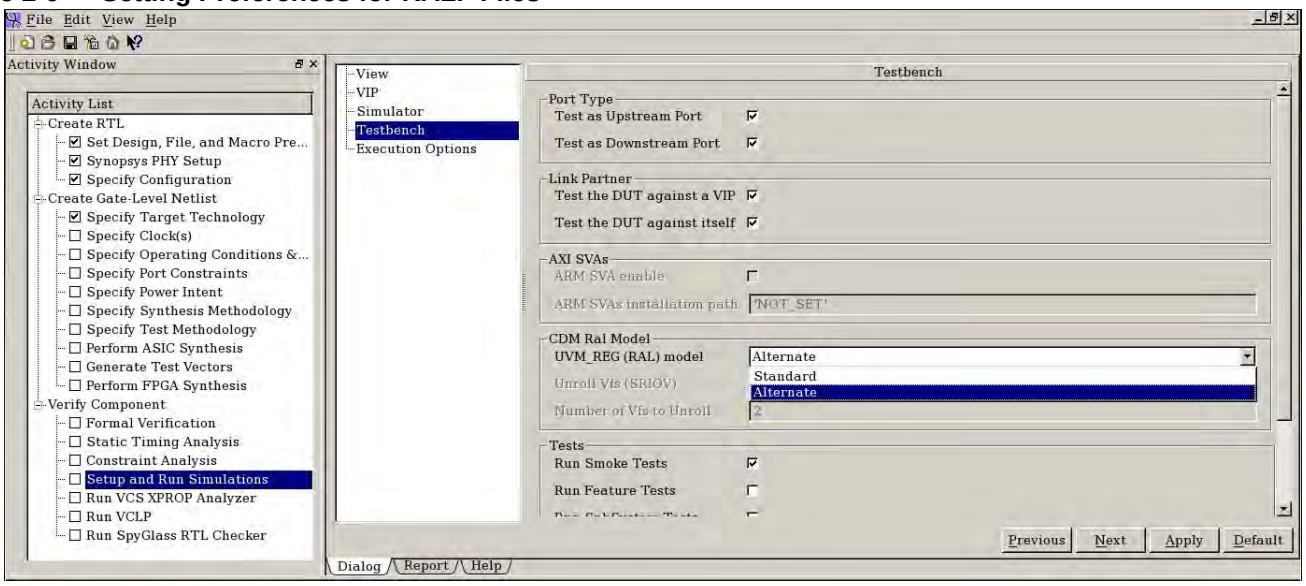


Setting Preferences for RALF Files

When you choose the **Alternate** ral model in the **Simulator** setup page, ralf files for all the memory map views described in "Generating Register Map" in the "Register Module, LBC, and DBI" section of the "Controller Operations" chapter of the *Databook* are created, when you run the `vtb_ral` test. The generated ralf files are located at `<workspace>/sim/testbench/ral`.

Note: In AMBA bridge configuration, a single ralf files is created for all the memory blocks including iATU, DMA, and shadow registers, in the DBI view.

Figure 2-9 Setting Preferences for RALF Files

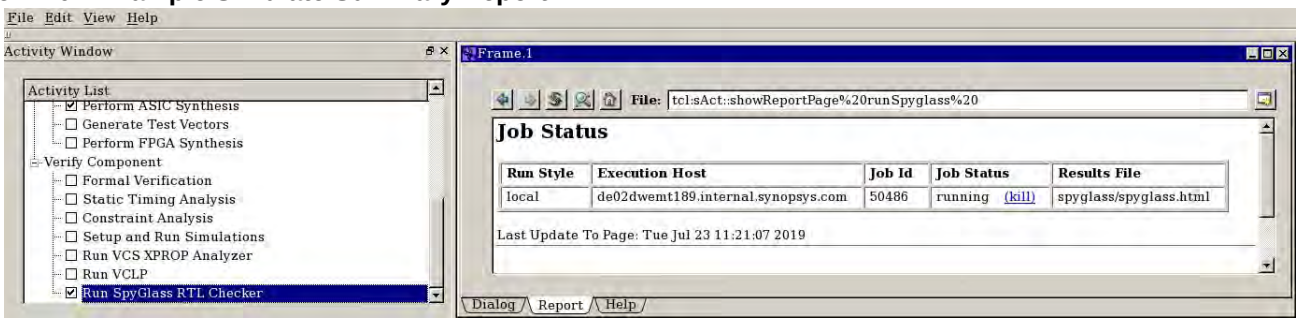


2.5.2 Checking Simulation Status and Results

To check simulation status and results:

- Click the **Report** tab
- Click the **Simulate Summary** link

Figure 2-10 Example Simulate Summary Report



Note When you select the “LSF/GRD” option for the Run Style in the “Execution Options” of the Simulate Dialog in Figure 2-8, the status of the simulation jobs (running or complete) is incorrect. After all the simulation jobs are submitted to the LSF/GRD queue, the status indicates complete. You should use “bjobs/qstatus” to check if all of the jobs are completed.

2.6 Synthesizing the Controller

The coreConsultant tool is your interface to Synopsys synthesis tools for ASIC synthesis of the controller. If you want to access the synthesis scripts for exporting to your chip database, or if you are using non-Synopsys synthesis tools, then see also "Creating Optional Views and Reports" on page 58. The topics in this section are as follows:

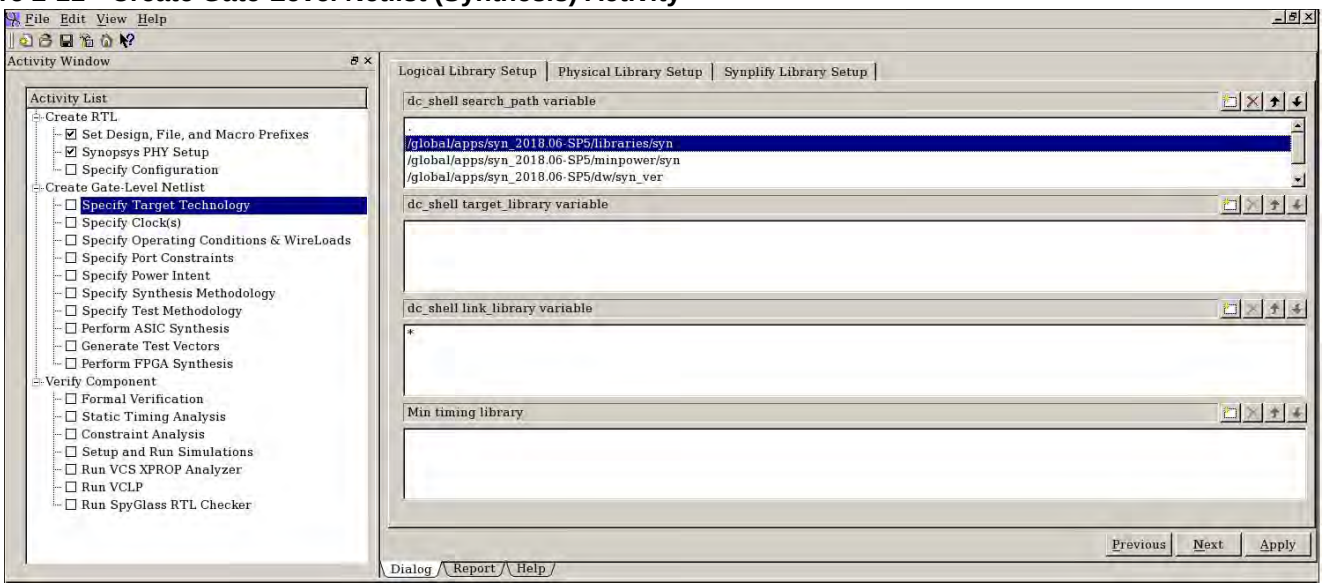
- "Synthesizing the Controller for an ASIC" on page 34
- "Formal Equivalence Checking using Formality" on page 45
- "Synthesizing the Controller for an FPGA" on page 118

Before You Start

Check that you have the latest tool versions installed and your environment variables set up correctly (see "Setting up your Environment" on page 24). To check your tools, click the **Help > Check Environment** menu item. Correct any reported errors by modifying your Unix environment setup or through the **Edit > Tool Installation Roots** menu. You can set global synthesis tool options through the **Edit -> Preferences -> Synthesis** menu.

To synthesize your controller, select **Create Gate-Level Netlist** from the **Activity List** in the coreConsultant GUI shown in Figure 2-11.

Figure 2-11 Create Gate-Level Netlist (Synthesis) Activity



2.6.1 Synthesizing the Controller for an ASIC

If you want to map and synthesize the controller to an ASIC using the Synopsys Design Compiler or Fusion Compiler tool within coreConsultant, then follow the process outlined in this section. Otherwise, you must export the synthesis scripts from coreConsultant as outlined in "Synthesizing to a Device Outside of coreConsultant" on page 64 so that you can synthesize the controller in your own design flow.

The topics in this section are as follows:

- "Performing ASIC Synthesis" on page 35
- "ASIC Synthesis Constraints" on page 43

- "Checking ASIC Synthesis Results" on page 43

2.6.1.1 Performing ASIC Synthesis

You must configure the controller (see "Configuring the Controller" on page 27) before starting this task. The following key features are strongly recommended and can be enabled in the synthesis flow:

- Physical Synthesis (using Milkyway)
  - Use Design Compiler Ultra in Topographical mode to do physical synthesis. Doing physical synthesis instead of using wire load models is strongly recommended because it predicts path delays more accurately and eliminates the need to over-constrain the design.
- Clock Gating
  - Enabling the clock gating can reduce power dissipation, area, and congestion, by removing feedback nets and multiplexers for synchronous load-enable registers and gate the clock for one or more cycle.
- Scan Insertion
  - Enable scan chain insertion even though it increases area and congestions to provide test capabilities required to take a chip to production.

Note Some configurations do not meet timing as easily as others. To find out more about which parameters affect area and timing the most, see "Configuration Parameter Settings to Improve Timing" on page 165.

To synthesize your controller, complete the following steps in the **Create Gate-Level Netlist** activity (see Figure 2-11):

1. **Specify Target Technology**

   The **Specify Target Technology** activity includes setting up of the Logical Library and Physical Library.

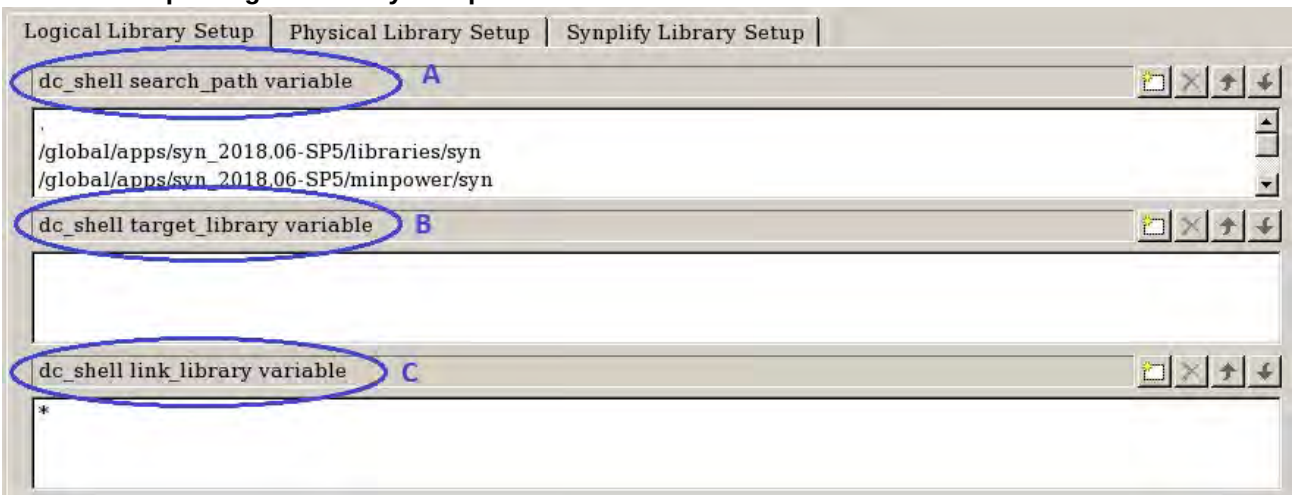
   - **Logical Library Setup**

     To complete logical library setup:

     - Specify the search path, that is, the path to directory where your library .db file is located (A in Figure 2-12).
     - Specify the target technology library, that is, the .db library file to be used (B in Figure 2-12). For ASIC synthesis, use a target technology of 0.18-micron or less. By default, Synopsys DW Logic Libraries are used, but any library in the form of *.db file can be used.
     - Specify the link library, that is, the link library corresponding to the target technology specified or any other library required to link your design (C in Figure 2-12).

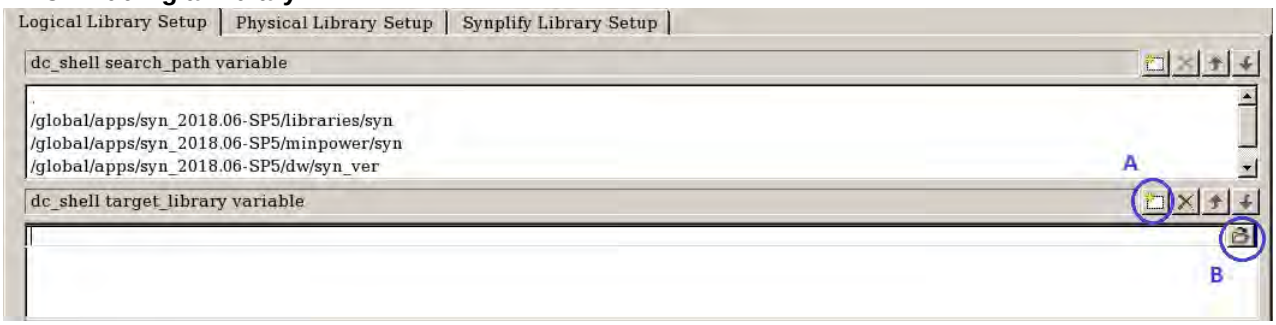
     Figure 2-12 shows and example of completed Logical Library Setup.

Figure 2-12 Example Logical Library Setup



To add the target library, click the folder icon (A in Figure 2-12) and use the navigation tool (B in the Figure 2-12). Search path and link library can also be added in a similar manner.

Figure 2-13 Adding a Library



- **Physical Library Setup**

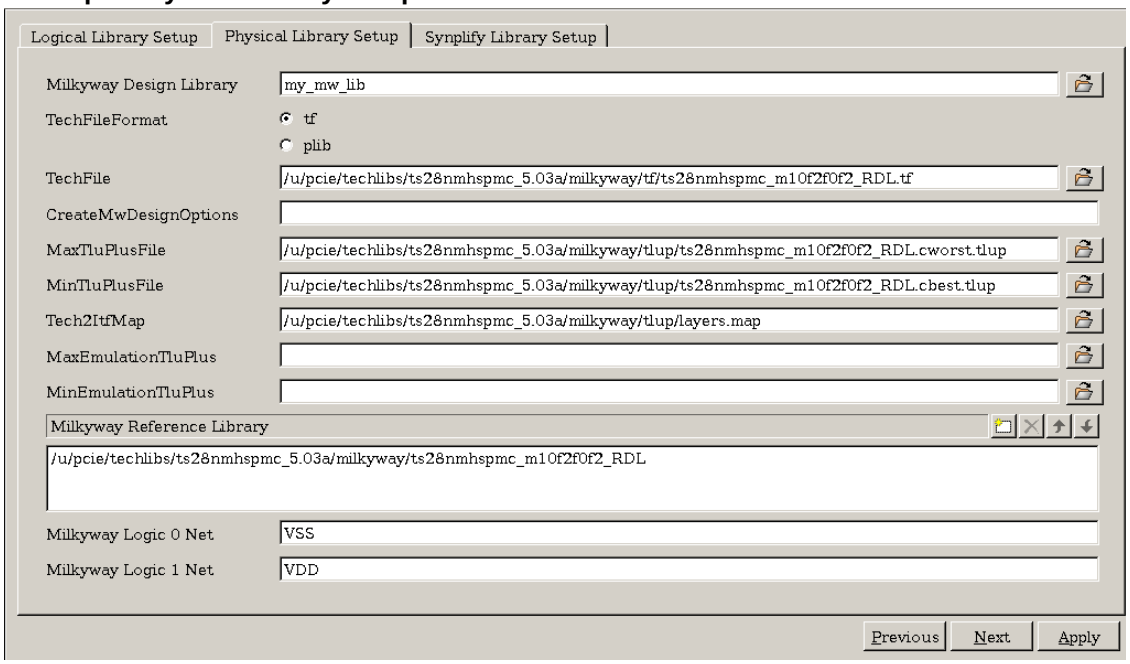
  This setup activity is used when you do physical synthesis. You need to specify the following parameters:

  - **Milkyway Design Library**: Specify the name of the Milkyway Design Library to be created. DC uses its library to store a representation of the physical design, that can later be accessed by other implementation tools.
  - **TechFileFormat**: Select whether to use tech files (.tf) or physical library source files (.plib) to create the Milkyway Design Library. Milkyway technology files contain technology-specific information required to route a design. In this example .tf files are used.
  - **TechFile**: Path to the physical library technology file to be used.
  - **MaxTluPlusFile/MinTluPlusFile**: Path to the MaxTlu+ and MinTlu+ files to be used. Using Tlu+ files is optional, it improves correlation with back-end results by using them for RC delay estimation. If you use Tlu+ file, you must specify the MaxTLUP+ file and the map file.
  - **Note:** The TLU+ files are not provided in the library download, you must generate it from the technology specific Interconnect Technology Format (ITF) file using the STAR RCTM tool.
  - **Map File**: Maps layer names between the Milkyway technology file and the process Interconnect Technology Format (ITF) file.

- **Milkyway Reference Library**: Path to the Milkyway Reference Library. This library contains the physical representation of standard cells and macros.

Figure 2-14 shows an example of completed Physical Library Setup.

Figure 2-14 Example Physical Library Setup



2. **Specify Clocks** (in the **Specify Clocks** activity)

   After you have completed the **Specify Target Technology** activity, you can set up the clocks to be used in synthesis.

   The default clocks are:

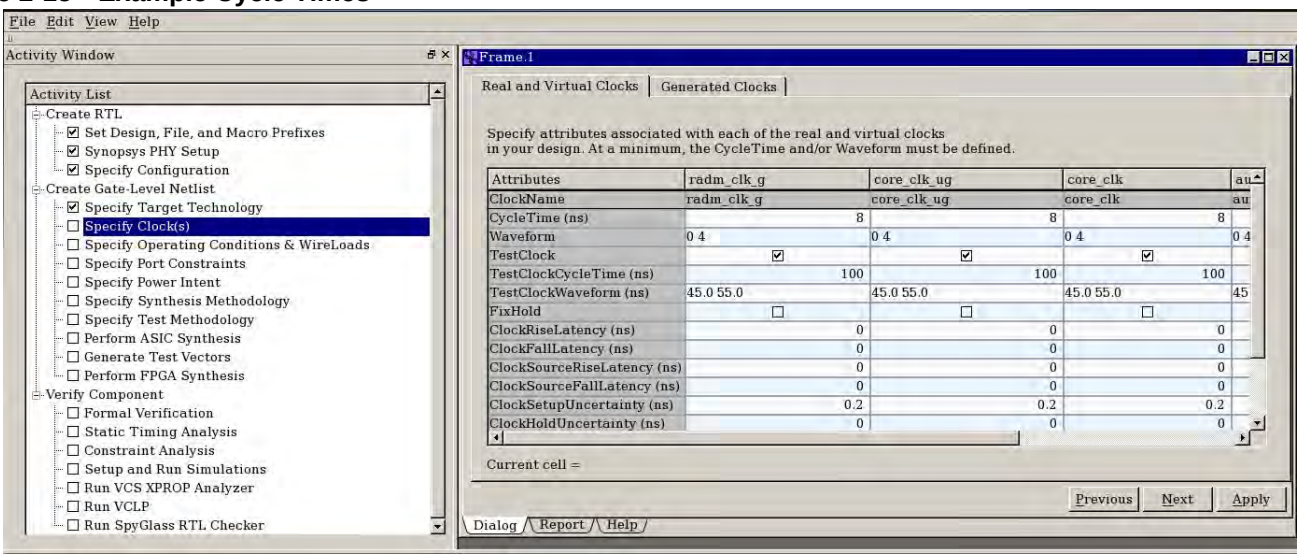
   - `core_clk/pipe_clk/aux_clk`: The default values for these clocks are based on the supported speed modes and base frequencies. These values comply with the PCIe specification and need not be modified.
   - AXI (if selected): 250 MHz

     In the **Specify Clocks** activity, the cycle time for each clock is listed (in ns or ps depending on the time scale in the library). Synopsys adds a clock margin of 10% on all clocks in our reference flow (the clock period is reduced). To duplicate this in coreConsultant you can either change this manually in the GUI or source the following script on the coreConsultant command line:

```
foreach_in_collection i [get_clocks] {
set clk_period [expr [get_clock_attribute $i CycleTime] * 0.9]
set_clock_attribute $i CycleTime $clk_period
}
```

     Figure 2-15 shows an example of cycle times after the script has been sourced.

Figure 2-15 Example Cycle Times

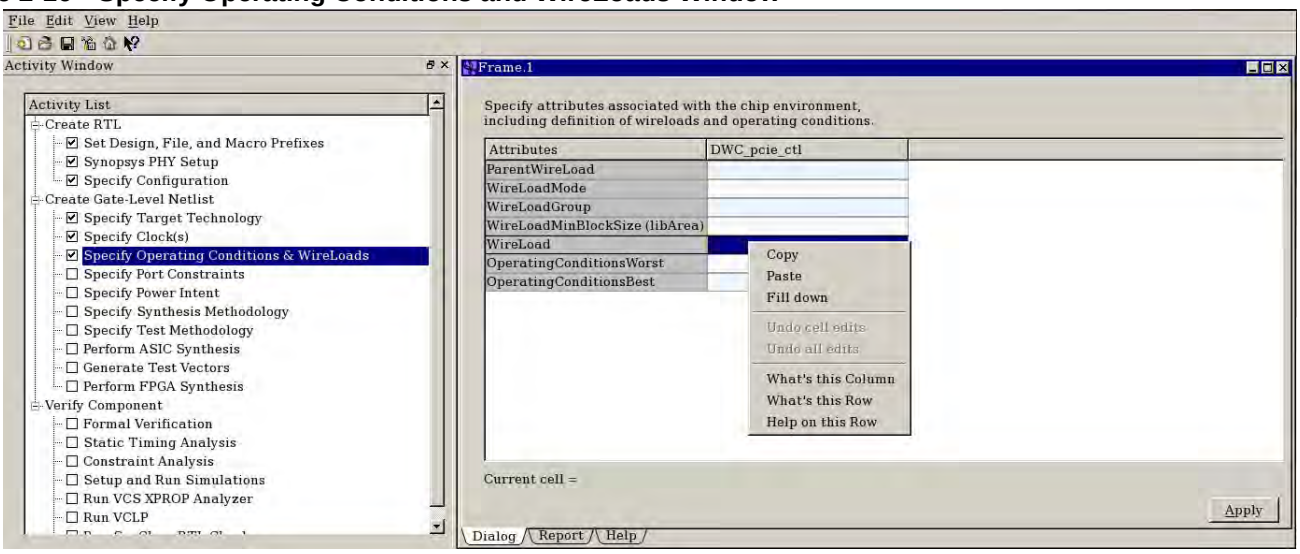


3. **Specify Operating Conditions and WireLoads**

   When you do not see a value beside “Operating Conditions Worst” (Figure 2-16), select an appropriate value from the drop-down list. If there is no value for this attribute, you might get an error message. You should also set the various WireLoad attributes, unless you are using Physical Synthesis.

   You should also set the various “WireLoad” attributes, unless you are using Physical Synthesis. For detailed help on any of the options, right click and select “Help on this Row”. Click **APPLY** button and look at the report, which gives the operating conditions and wire load information.

Figure 2-16 Specify Operating Conditions and WireLoads Window



4. **Specify Port Constraints**

   Port input and output delay constraints are specified relative to a virtual clock (`clk_ideal`), which has the same CycleTime as the real clocks `core_clk` and `aux_clk`.

   The default input and output delay constraint values are specified as percentages of the `clk_ideal` CycleTime and are configuration-dependent. The default setting of these attributes provides acceptable synthesis results in most libraries.

5. **Specify Synthesis Methodology**

   In the **Specify Synthesis Methodology** activity, review the synthesis strategy attributes. These attributes are set by the controller developer and can be optionally modified. The default setting of these attributes provides acceptable synthesis results in most libraries.

6. **Specify Test Methodology**

   These attributes are set by the controller developer and can be optionally modified. For more information, see "Inserting Design-For-Test using Design Compiler" on page 46.

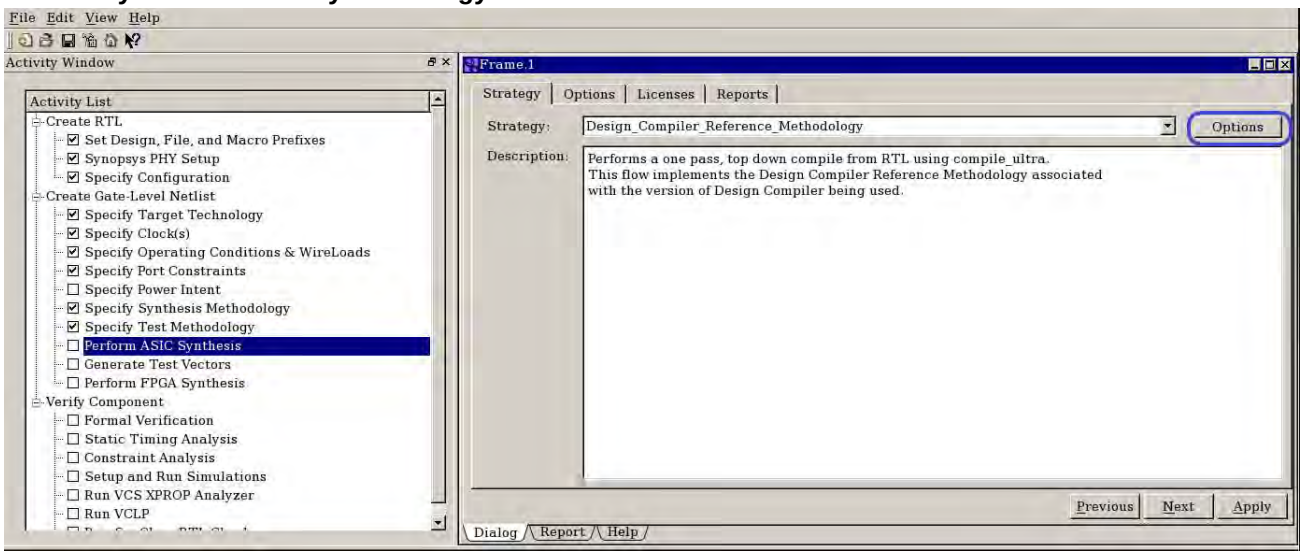
7. **Perform ASIC Synthesis**

   You can select the synthesis strategy on the Strategy tab shown in Figure 2-17. A detailed explanation of each strategy is given in the GUI.

Hint

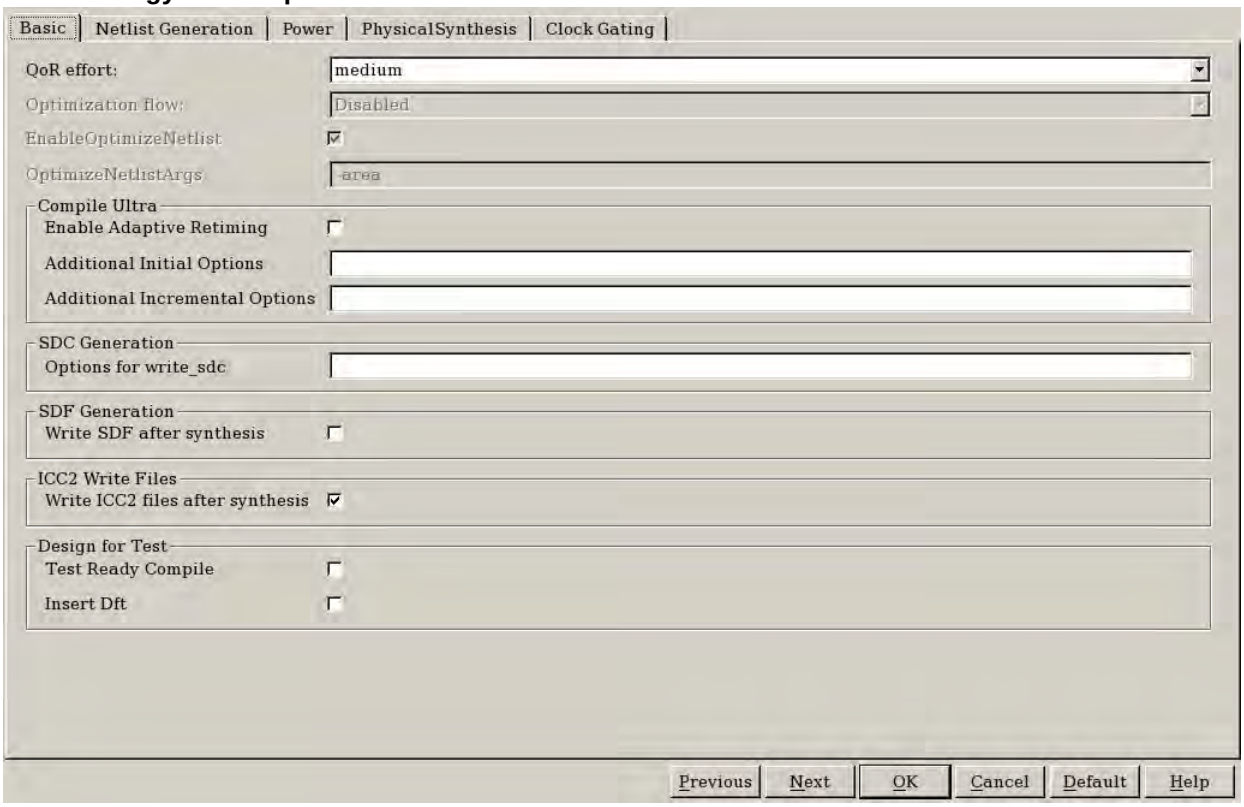
One strategy that is of interest is the DCTCL_one_pass_compile_ultra, which uses a simple Tcl script based on the Synopsys Design Compiler Reference Methodology at https://solvnetplus.synopsys.com/s/article/Design-Compiler-Reference-Methodology-1577133837658 This synthesis strategy is quicker and does not generates intermediate stages. Therefore, it is much easier to trace the steps in the flow.

Figure 2-17 Synthesize Activity -> Strategy Tab



Clicking on the Options button (highlighted in Figure 2-17) lists a series of other options (as shown in Figure 2-17) for that strategy. More information on these options is available in the *coreConsultant User Guide* (see "Help Information" on page 150) or by right clicking on any dialog text.

Figure 2-18 Strategy Tab-> Options Button



These are the most important options.

- **Design for Test** in Basic tab

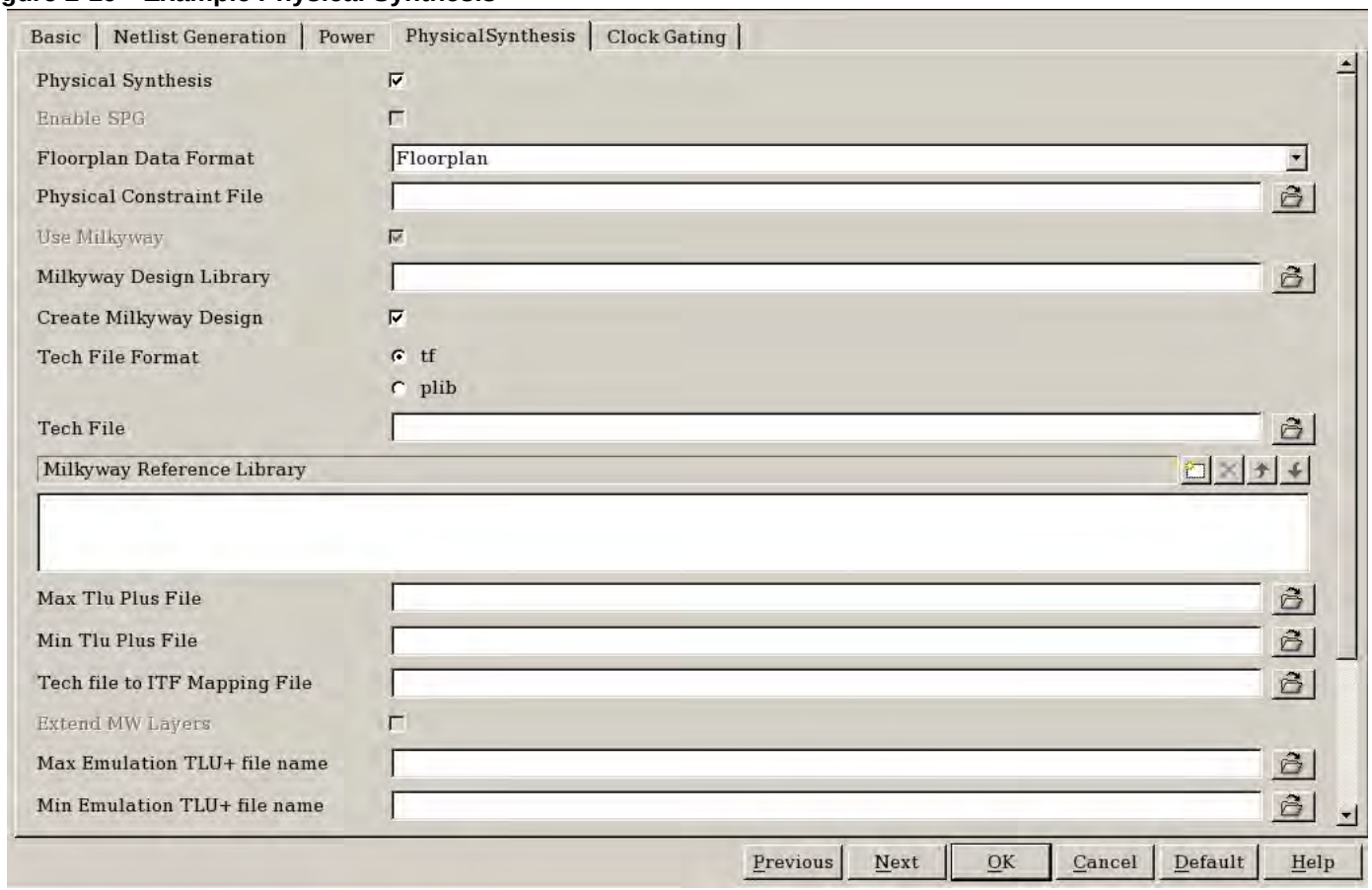
  Here you specify whether to add the -scan option to the initial compile call (Test Ready Compile) and/or insert DFT circuitry (Insert Dft). For more information, see "Inserting Design-For-Test using Design Compiler" on page 46.

- **Physical Synthesis**

  If you are running Physical Synthesis, then tick the Physical Synthesis box (all other values on this tab are populated with information from the **Physical Library Setup** page).

  Setting this parameter causes the physical synthesis placement engine to be used during optimization. Net loads and delays are estimated during placement. The flow can be used with a physical library in the .ddc format or Milkyway reference library. Note that no placement data is saved with the ddc or Milkyway database.

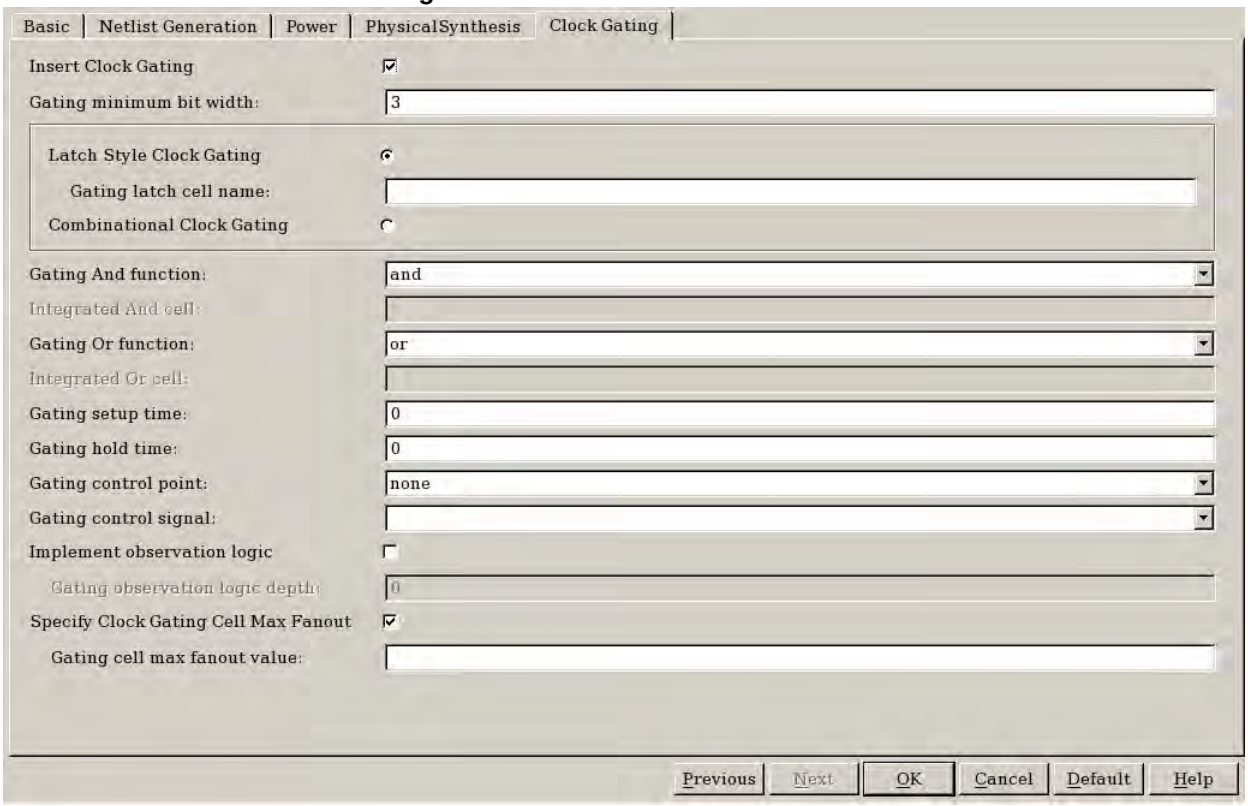
Figure 2-19 Example Physical Synthesis



- **Clock Gating**

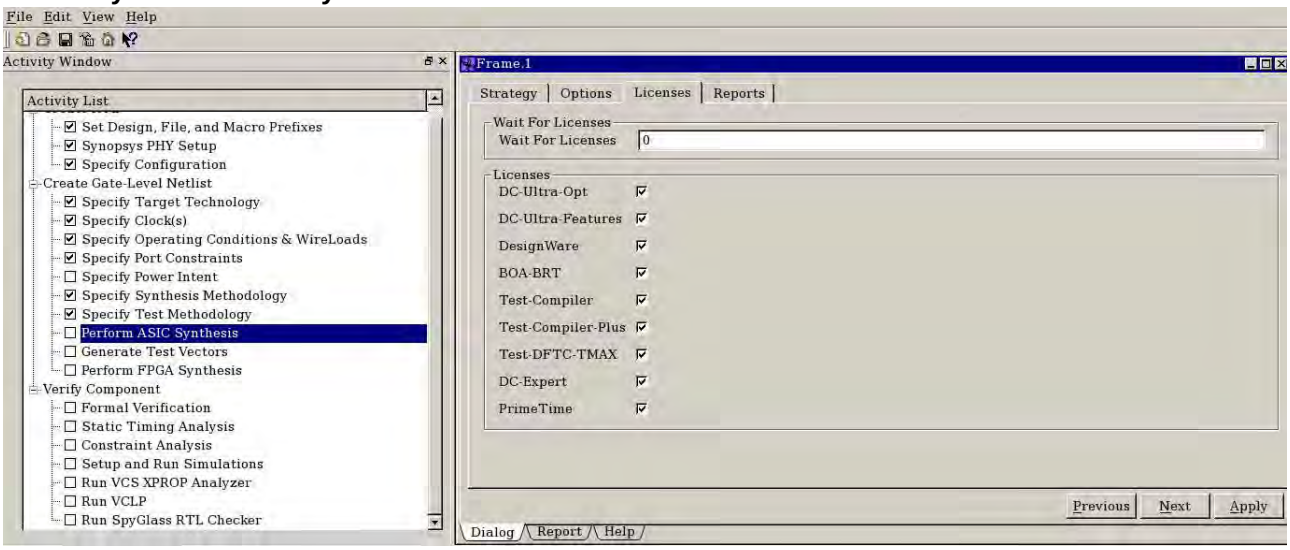
  To inset clock gating tick the Insert Clock Gating box. Figure 2-20 shows the recommended values for Clock Gating parameters.

Figure 2-20 Recommended Clock Gating Parameter Values



To enable the physical synthesis as well as other advanced optimization features, you must enable DC-Ultra-Opt and DC-Ultra-Features in the **Licenses** tab as shown in Figure 2-21.

Figure 2-21 Synthesize Activity -> Licenses Tab



8. When you have finished specifying the remaining synthesis options, click **APPLY** button.

Note If any problems occur during synthesis, see “Troubleshooting” on page 148 and “Synthesis Troubleshooting Guide” on page 155.

2.6.1.2 ASIC Synthesis Constraints

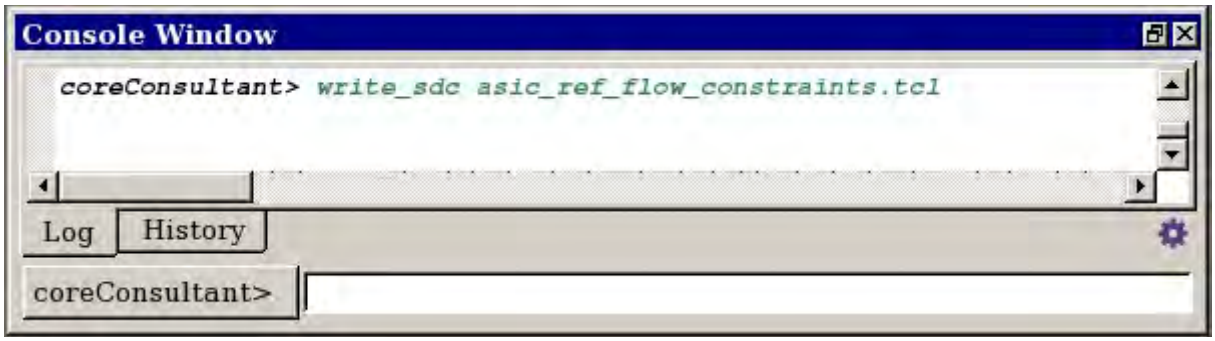
The constraints used in the synthesis flow can be found at the following location in your workspace:

```
<workspace>/syn/constrain/script/DWC_pcie_ctl.sdc
```

You can also write out the constraints from the coreConsultant GUI command line with the following command:

```
write_sdc <file name>
```

Figure 2-22 coreConsultant Console Window



CDC Constraints

Synopsys recommends using `set_max_delay` constraint instead of `set_false_path` between the clock domains to avoid scenic routing. The max delay should match the maximum allowed skew between two related signals by design and the maximum delays used in the re-convergence simulation. 1-clock destination clock period is used as max delay.

For example, when using a 300 MHz AXI clock:

```
set_max_delay \
-from [list [get_clocks aux_clk_g] [get_clocks aux_clk]] \
-to [list [get_clocks mstr_aclk] [get_clocks mstr_aclk_ug]] 3333
```

2.6.1.3 Checking ASIC Synthesis Results

After you have started synthesis, coreConsultant generates several reports. You can access these by clicking the Report tab in Figure 2-11.

All the synthesis results and log files are created under the syn directory in your workspace. The “final” symbolic link points to the current synthesis stage directory. The final log file is `<workspace>/syn/run.log`.

Excluding `DCTCL_one_pass_compile_ultra strategy`, your final netlist and report directories (`initial`, `incr1` or `incr2`) depend on the QoR effort you have chosen for your synthesis (default is Medium) or whether you chose to insert DFT or not (see “Inserting Design-For-Test using Design Compiler” on page 46):

- QoR Effort: “Low effort” – `<workspace>/syn/initial`
- QoR Effort: “Medium effort” – `<workspace>/syn/incr1`
- QoR Effort: “High effort” – `<workspace>/syn/incr2`

The QoR effort is selected through the **Synthesis -> Options**[1] **-> Basic** tab. There are also options here (amongst many) to:

- Generate reports for all stages[2].
- Generate netlist for all stages.

Table 2-2 includes the other files of interest that are generated.

Table 2-2 Synthesis Output Files

Files	Purpose
`./syn/final/db/DWC_pcie_ctl.ddc`	Synopsys database files (gate level) that can be read into dc_shell for further synthesis, if desired.
`./syn/final/db/DWC_pcie_ctl.v`	Gate-level netlist that is mapped to technology libraries that you specify.
`./syn/final/report/*.*`	Synthesis report files.

If you are running Physical Synthesis, then the final `.ddc` file is stored in the `<workspace>/syn/final/db` directory.

Caution

- In the <workspace>/syn directory, run.scr. Makefile and final are symbolic links to the current synthesis stage files.
- These links are also used and set to different locations during ATPG and Formal Verification.

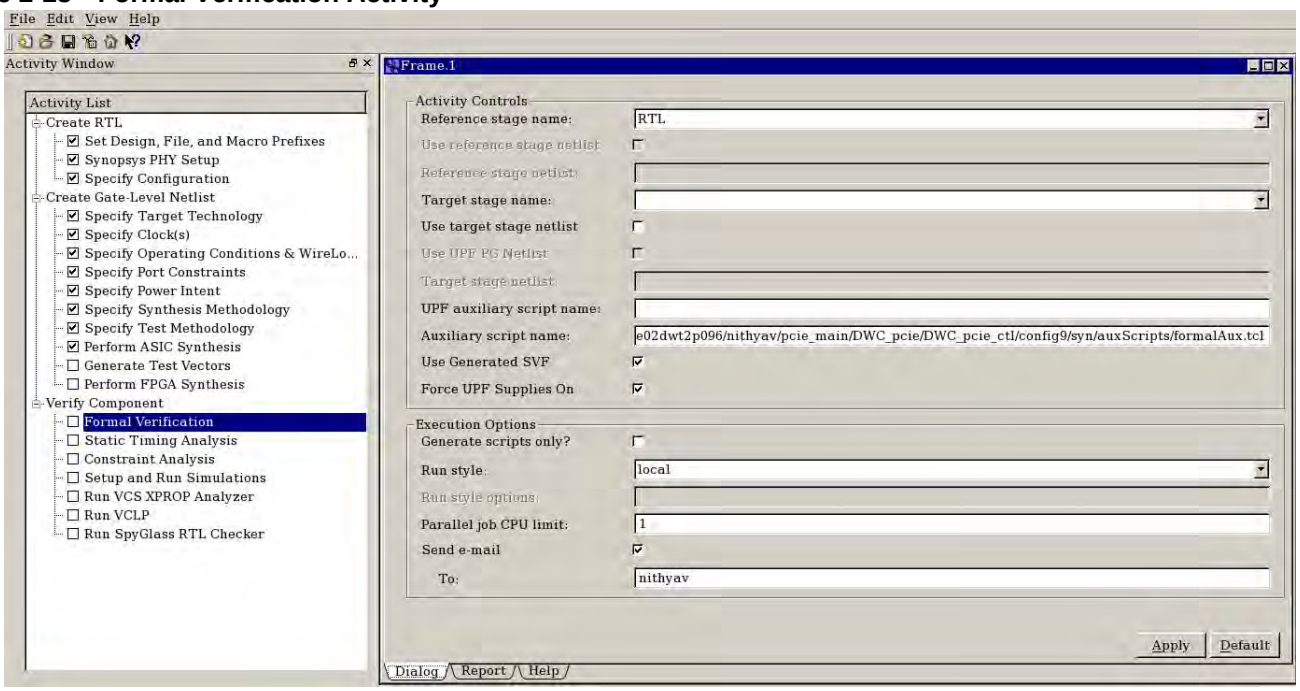
1. BUTTON as circled in Figure 2-17 and not TAB.
2. The synthesis process runs in stages or steps which are normally initial, incr1, or incr2, depending on the value of the QoR Effort synthesis strategy parameter.

2.7 Formal Equivalence Checking using Formality

You use the **Formal Verification** activity (in Figure 2-23) for RTL-to-gate comparisons in coreConsultant.

Attention Formal equivalence checking is an activity that takes the output of synthesis as input. You must synthesize your controller before a meaningful setup exists in the Formality window.

Figure 2-23 Formal Verification Activity



Choose **Formal Verification** menu and fill in the following fields:

- **Reference stage name**: RTL
- **Target stage name**: specify the last synthesis stage from the drop-down menu.
- **Output stage**: set to any name that you want to use identify the output files that are created in your workspace directory. For example, *atpg*.
- **Use Generated SVF**: enabled.

The Formality results are saved in the `<workspace>/syn/<Output stage>` directory. Look at `<workspace>/syn/<Target stage name>.tcl` to understand the Formal Verification flow.

2.8 Inserting Design-For-Test using Design Compiler

You perform Design-For-Test (DFT) insertion during synthesis. To insert DFT:

1. Check that the controller input clocks are selected as “Test Clock” in the “Specify Clocks” window. This is normally selected by default.
2. In the “Specify Test Methodology” window, modify the options according to your chip DFT scheme. For detailed help on any of the options, right click and select “Help on this Row”.
3. DFT insertion is controlled in the Design-For-Test tab that is displayed by clicking the Options button (NOT the Options tab) in the Synthesis window (Figure 2-18). You must select one of these two options:
   - “Test Ready Compile”: Design Compiler uses scan flops in synthesis, but it does not insert and route the chains. the scan option is added to the initial compile call.
   - “Insert DFT”: Design Compiler completes DFT insertion, that is, it performs synthesis with scan flops and scan chain insertion.

The DFT results are saved in the `<workspace>/syn/dft` directory.

2.9 Running ATPG using TestMAX™

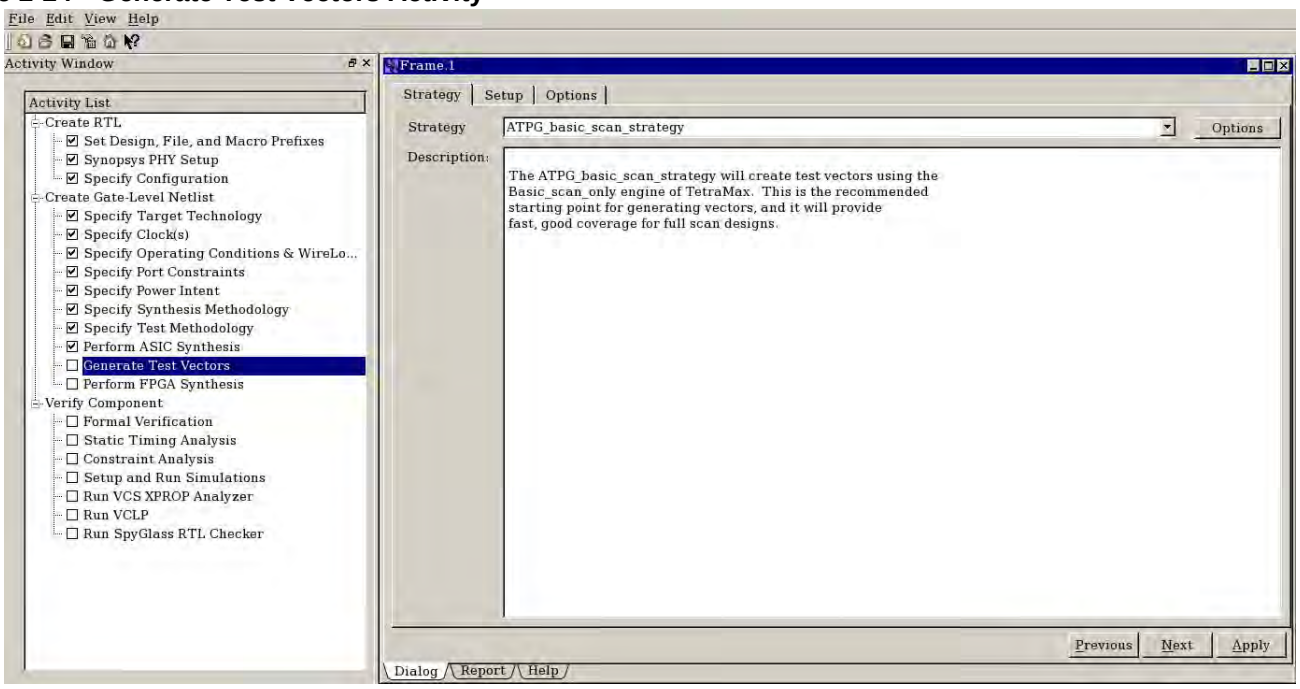
You can perform automatic test pattern generation (ATPG) by using the Synopsys TestMAX tool.

Attention

- ATPG is an activity which takes the output of synthesis as input.
- Synthesis must be completed (with DFT as described in "Inserting Design-For-Test using Design Compiler" on page 46) before a meaningful setup can exist in the ATPG window.

You use the **Generate Test Vectors** activity (in Figure 2-24) for ATPG in coreConsultant.

Figure 2-24 Generate Test Vectors Activity



After you have synthesized the controller with the Insert Dft option (see "Inserting Design-For-Test using Design Compiler" on page 46), enter information in the following fields in the **Generate Test Vectors -> Setup** menu.

- **Input stage**: specify the last synthesis stage from the drop-down menu.
- **Output stage**: set to any name that you want to use to identify the output files that are created in your workspace directory. For example, <atpg>.
- **Test Libraries**: specify the library Verilog file names.

  These test libraries are the Verilog simulation models for the target libraries used during synthesis. They provide the description of the logical functionality of all the cells in the standard cell library. They are needed for the ATPG tool to comprehend the logic function implemented by each standard cell instantiated.
- **Test Pattern Format**: specify your test format.

The ATPG results are saved in the `<workspace>/syn/<Output stage>` directory.

Check the following files in `<workspace>/syn/<Output stage>` to help you understand the ATPG flow.

- script/DWC_pcie_ctl.tcl
- Makefile

The `<workspace>/syn/auxScripts/snps_phy_atpg_aux.tcl` file is used during ATPG and defines the Synopsys PHY as a black box.

Hint For more information about running ATPG, see also the coreConsultant User Guide (see“Help Information” on page 150).

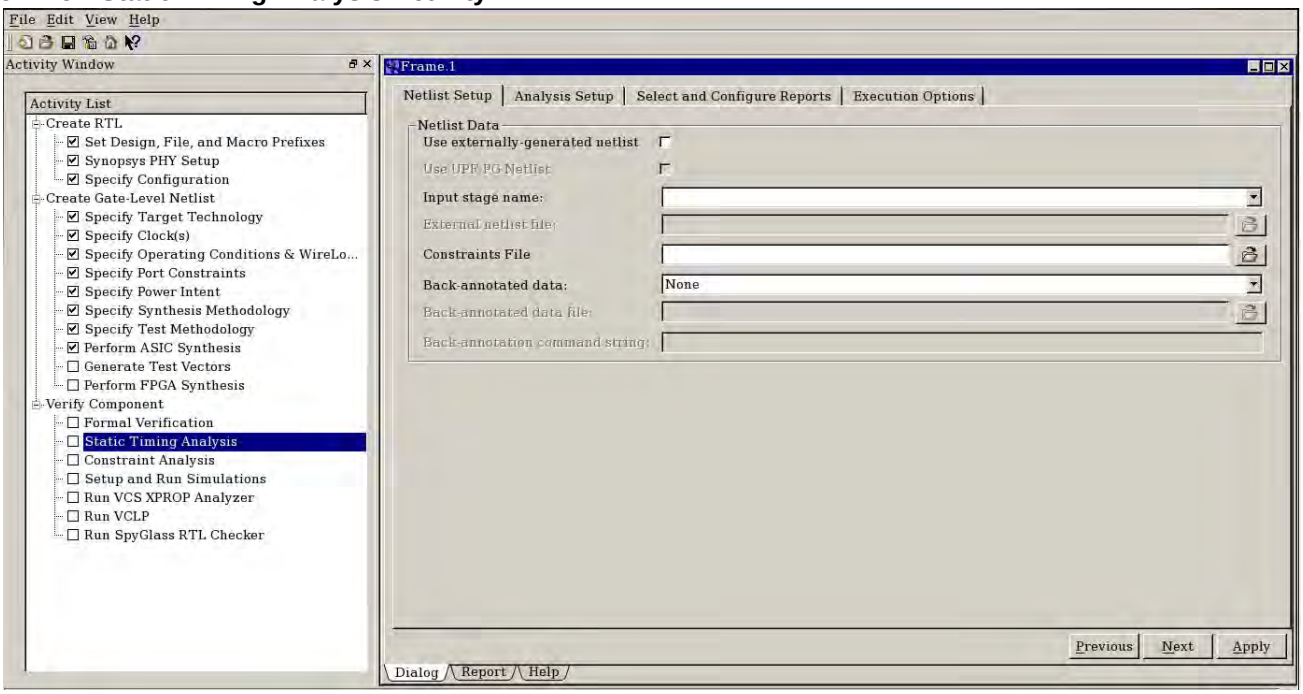
2.10 Running STA using PrimeTime

You perform **Static Timing Analysis** (STA) by using the Synopsys PrimeTime tool.

The STA activity allows you to run PrimeTime static timing analysis, using constraints generated from coreConsultant. This can be run either on the netlist generated from synthesis, or on an externally generated netlist (for example from a layout tool). An externally generated SPEF or SDF file can also be read in for back-annotation.

You must have completed the "Performing ASIC Synthesis" on page 35, or have an externally generated netlist available before starting this task. To perform STA, select the **Static Timing Analysis** activity.

Figure 2-25 Static Timing Analysis Activity



1. **Netlist Setup**

   Click the Netlist Setup tab and fill in the following fields:

   - **Input stage name**: This field should be automatically filled in by coreConsultant if you have successfully completed ASIC Synthesis. Otherwise, specify the last synthesis stage from the drop-down menu. Typically, it is `Last<initial>`.
   - **Constraints file**: This field should be automatically filled in by coreConsultant if you have successfully completed ASIC Synthesis.

2. **Analysis Setup**

   Next, click on the Analysis Setup tab, which allows you to select various STA setup options. You can accept most of the default settings here.

3. **Select and Configure Reports**

   In this tab, you can choose what reports you would like PrimeTime to produce. You can accept most of the default settings here.

   To generate a Cross Domain Clocking paths report:

   - Check "Report Cross-Domain Paths"

- Change “Number of path to report” to 100,000
- The generated report resides in: <*workspace*>/syn/final/report/sta/DWC_pcie_ctl_cross_-clock_domain_paths.rpt
- Search for all *sample_meta* and *sample_sync* registers in the DWC_pcie_ctl_cross_clock_domain_-paths.rpt report

When you have chosen all relevant options for your Static Timing Analysis run, click APPLY button.

The STA results are saved in the `<workspace>/syn/final/report/sta` directory.

2.11 Running SpyGlass®

You perform Clock Domain Crossing (CDC) analysis, Linting, Reset Domain Crossing (RDC), and DFT analysis using the SpyGlass tool as shown in Figure 2-26.

Figure 2-26 SpyGlass Options in coreConsultant

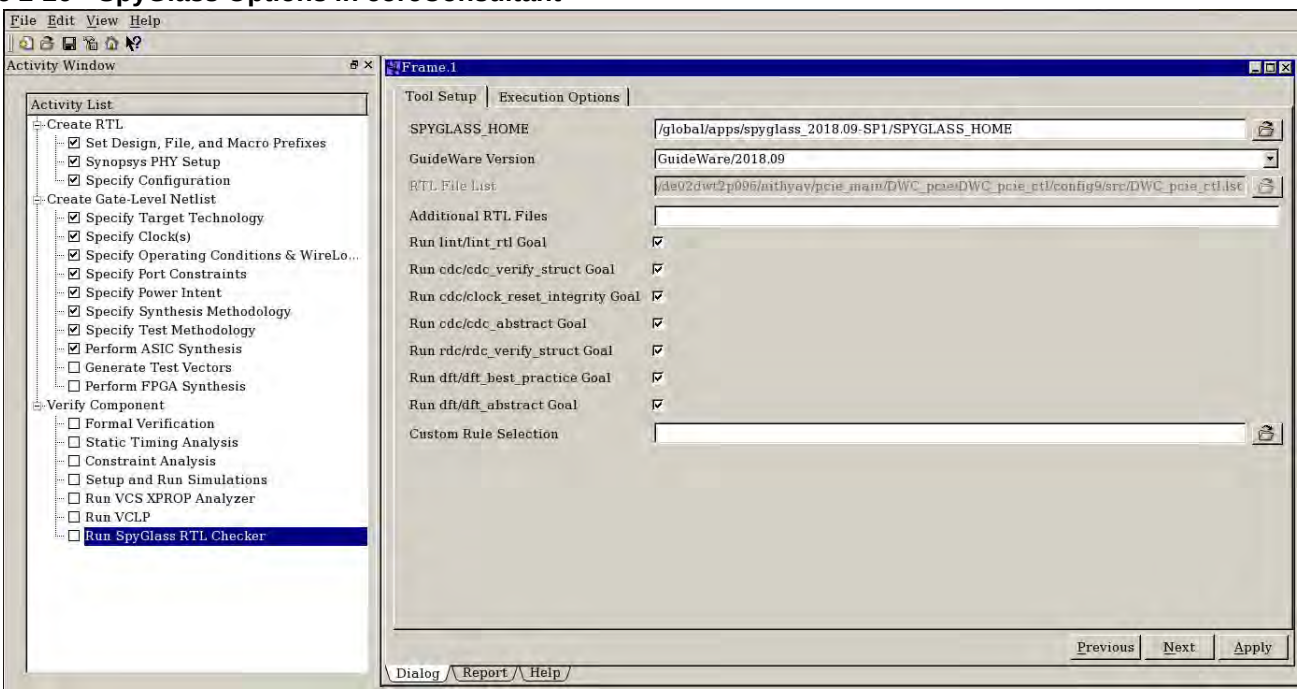


Table 2-3 Waiver Files in <workspace>/spyglass

File Name	Description
`spyglass_design_specific_waivers.swl`	These are DWC_pcie_ctl design-specific rule waivers. The reason for each of the waivers is included as comments in the file.
`spyglass_engineering_council_waivers.swl`	These are rules that Synopsys waives for all its digital IPs.

Note For more information on CDC, see the “Clock and Data Crossing (CDC)” appendix in the Databook. To create a CDC path report, see “Running STA using PrimeTime” on page 49.

Path Debug Using the SpyGlass GUI

For debugging a CDC path or a Lint failure, run SpyGlass in interactive mode as follows:

1. `cd <workspace>/spyglass`
2. `./sh.spyglass`
3. In the SpyGlass GUI, the Tool Setup window opens by default.
   - To run only CDC, check the `cdc_verify_struct` option and uncheck the `lint_rtl` option.
   - To run only Lint, check the `lint_rtl` option and uncheck the `cdc_verify_struct` option.

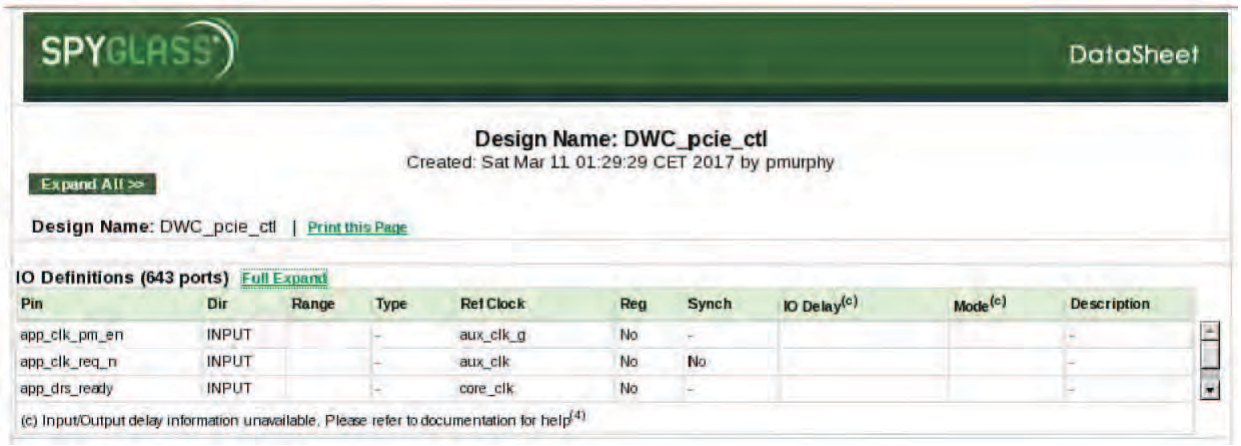
4. After the Spyglass run is complete, the Analyze Results window displays the results.
5. Navigate to and select the relevant errors to open a schematic for analysis.

Determining SynchronousTo Attributes Using the SpyGlass Datasheet

To create a configuration-specific report detailing the SynchronousTo attributes (see "Synchronous To Attribute" in the "General Clock Relationships" section in the "Architecture" chapter of the *Databook*) for your design, run SpyGlass in interactive mode as follows:

1. `cd <workspace>/spyglass`
2. `./sh.spyglass`
3. In the SpyGlass GUI, select the "`lint/design_audit`" goal to run. There are two ways to do this:
   - In the pulldown menu beside the RunGoal button, select `lint/design_audit`
   - In the "Shell" at bottom of the GUI, type `current_goal lint/design_audit -top DWC_pcie_ctl`
4. Run the goal by pressing the "RunGoal" button or typing `run_goal` in the Shell.
5. After the Spyglass run is complete, select the "Reports -> DataSheet" menu item.
6. The report is also in `<workspace>/spyglass/work/core/html_reports/datasheet.html`

Figure 2-27 Spyglass Datasheet

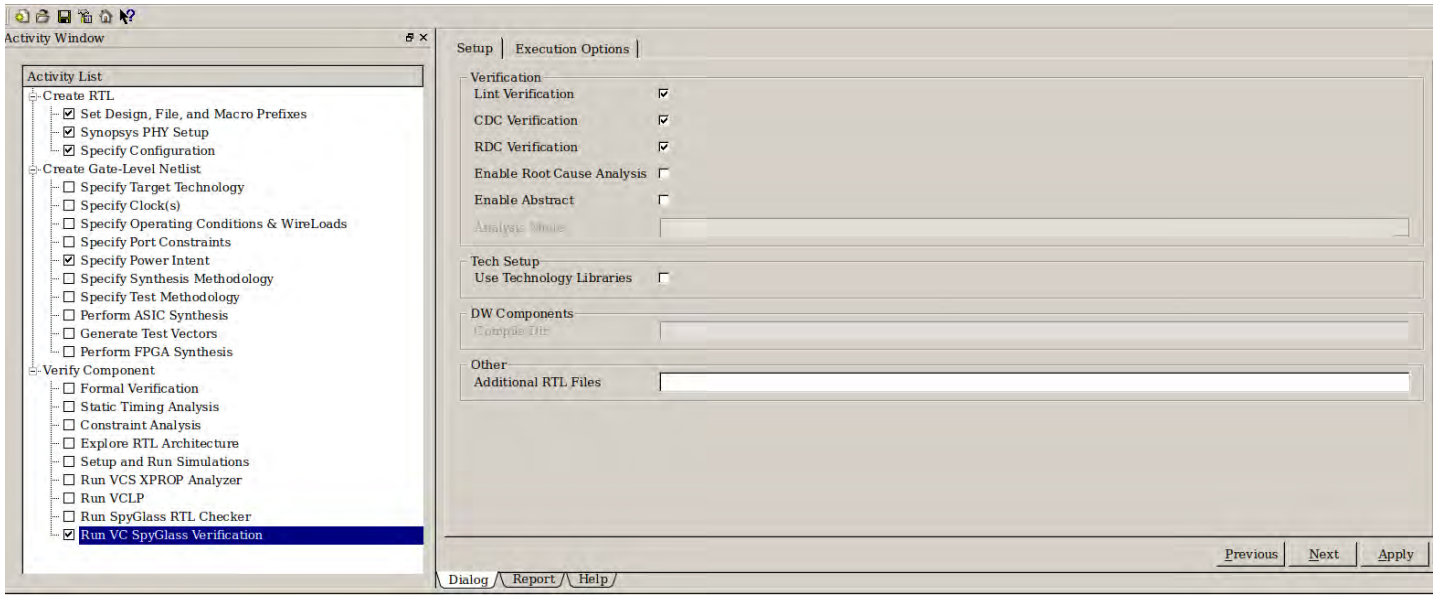


2.12 Running VC SpyGlass

This section describes the procedure to run VC SpyGlass activity.

Figure 2-28 shows the coreConsultant GUI for VC SpyGlass activity.

Figure 2-28 VC Spyglass Activity in coreConsultant



Verification

The VC SpyGlass flow in coreConsultant runs the following verifications:

- Lint
- CDC
- RDC

Also, it is possible to enable the Root Cause Analysis, and Abstract mode.

Root Cause Analysis: This facilitates analysis and debug.

Abstract: Provides an abstraction model for SoC usage.

Analysis Mode (if enabled): This facilitates the analysis of the operation mode.

TechSetup

This propagates the TechSetup SearchPath, and LinkLibrary information to the VC SpyGlass.

DW Component

This dialog box lists the DesignWare (DW) components, if any, used by the controller.

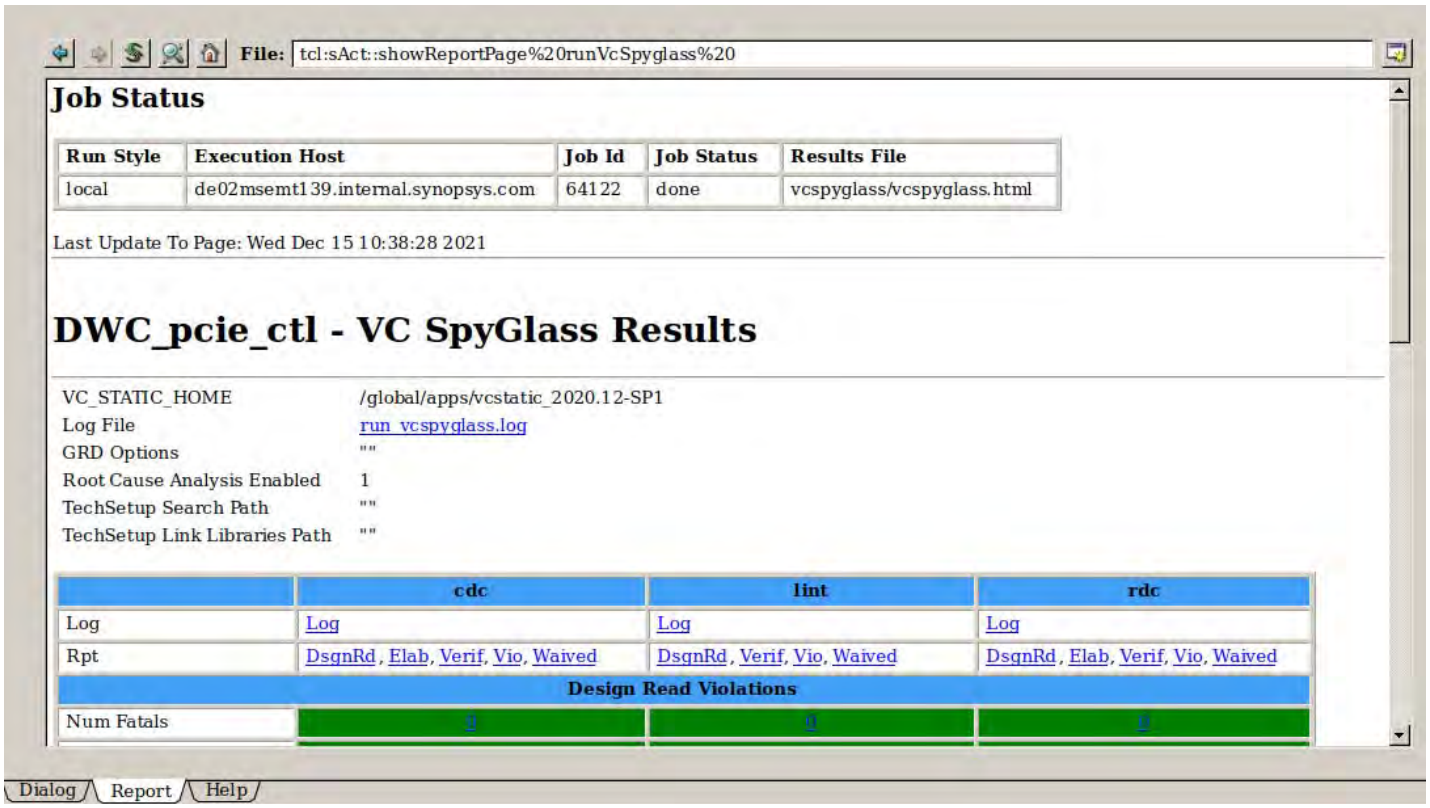
Other

This text box lists the additional RTL files that are used.

Results

After the run, the report tab displays the results. Errors (if any) are displayed in a red colored cell and warnings (if any) are displayed in a yellow colored cell, as shown in Figure 2-29.

Figure 2-29 coreConsultant VC SpyGlass Report Summary



2.12.1 CDC and RDC Verification

To define SpyGlass CDC constraints, it is important to understand the clock and reset logic used in the controller. For more information about clock and reset structure, refer "Clock Domain Crossing (CDC) Methodology" on page 171.

2.12.2 VC SpyGlass Folder Files

Table 2-4 lists the files for VC SpyGlass activity in coreConsultant.

Table 2-4 VC Spyglass Folder Files

File Name	Description
<dir>/vcspyglass/run.scr	coreTools run script for VCSpyGlass
<dir>/vcspyglass/run_vcspyglass.bash	Main run script
<dir>/vcspyglass/run_vcspyglass.log	Main run script log file
<dir>/vcspyglass/vcspyglass.done	Indicates the activity has completed

File Name	Description
<dir>/vcspyglass/vcspyglass.html	Summary HTML report of all enabled verification modes
<dir>/vcspyglass/vcspyglass.errors	Errors detected during verification
<dir>/vcspyglass/vcspyglass.warnings	Warnings detected during verification
<dir>/vcspyglass/cdc|rdc|lint/	Folder where the VCSpyGlass runs to analyze CDC, RDC, and Lint respectively
<dir>/vcspyglass/cdc|rdc|lint/design_read.rpt	Design Read phase report
<dir>/vcspyglass/cdc|rdc|lint/elab_summary.rpt	Elaboration phase report
<dir>/vcspyglass/cdc|rdc|lint/bcm_qualifiers.rpt	Automatically constrained BCMs
<dir>/vcspyglass/cdc|rdc|lint/verification.rpt	CDC/RDC/Lint verification report
<dir>/vcspyglass/cdc|rdc|lint/verification.violated.rpt	CDC/RDC/Lint errors and warnings verification report
<dir>/vcspyglass/cdc|rdc|lint/verification.only_waived.rpt	CDC/RDC/Lint waivers verification report
<dir>/vcspyglass/cdc|rdc|lint/vcst_session.log	VCSpyGlass run log file
<dir>/vcspyglass/cdc|rdc|lint/inferred_clocks.tcl	VCSpyGlass incrementally inferred clocks
<dir>/vcspyglass/cdc|rdc|lint/inferred_resets.tcl	VCSpyGlass incrementally inferred resets
<dir>/vcspyglass/cdc|rdc|lint/inferred_qualifiers.tcl	VCSpyGlass incrementally inferred qualifiers
<dir>/vcspyglass/constraints/<Design Name>.sdc	coreConsultant generated SDC constraints
<dir>/vcspyglass/constraints/constraints.tcl	Design constraints
<dir>/vcspyglass/constraints/resets.tcl	Design top level resets
<dir>/vcspyglass/constraints/waivers.tcl	Design waivers
<dir>/vcspyglass/engineering_council/vcspyglass_engineering_council_rules.tcl	Ruleset for Synopsys IP

Note The inferred_*.tcl files are for informative purpose only and is not used by VC SpyGlass.

2.12.3 Running VC SpyGlass in Interactive Mode using VC SpyGlass GUI

To run VC SpyGlass in interactive mode, use the run_vcspyglass.bash script. Table 2-5 lists the arguments supported in interactive mode.

Table 2-5 Arguments Supported in Interactive mode

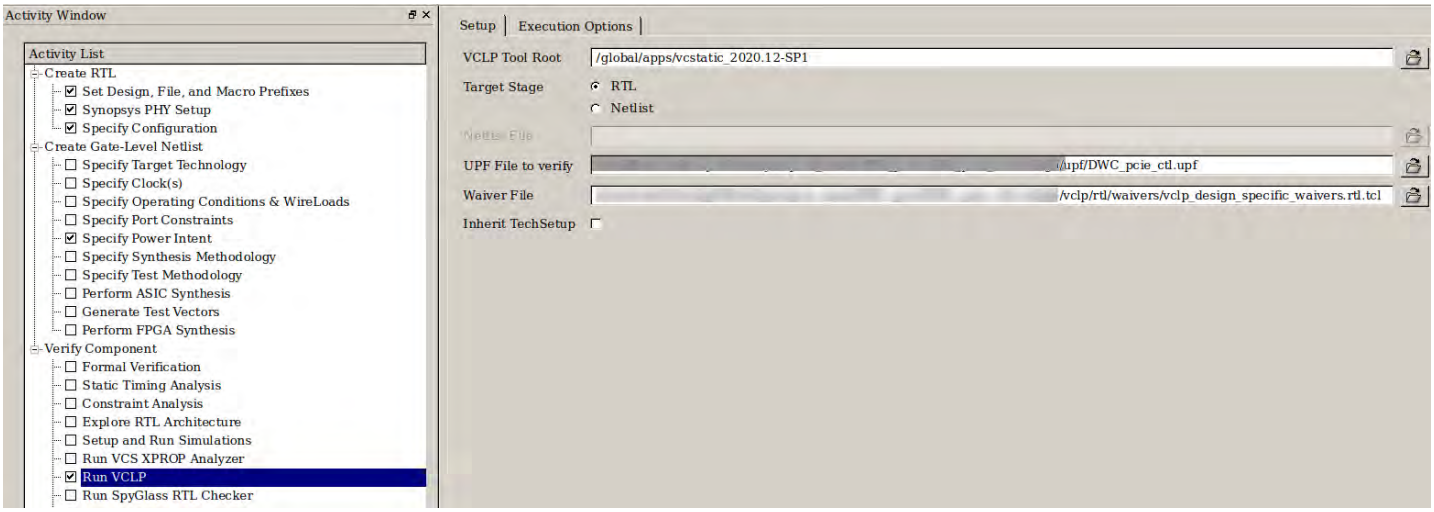
Argument	Description
-cdc	Run CDC verification
-rdc	Run RDC verification
-lint	Run Lint verification
-shell	Stop on shell mode after verification
-gui	Open GUI mode and run verification
-clean	Removes previous results from selected verification folder. All verification folders are removed if no verification mode is selected (-cdc, -rdc, -lint)
-rca	Enables Root Cause Analysis (RCA), which helps debugging
-abstract	Generates abstraction model
-checkpoint	Generates checkpoints during run

2.13 Running Low Power Static Verification using VCLP

You can perform low-power static verification using Synopsys VC Low-Power (VCLP) tool.

As shown in Figure 2-30 you can run VCLP in the coreConsultant GUI.

Figure 2-30 VCLP in the coreConsultant GUI



The VC LP flow in the coreConsultant runs the following low-power checks either on the RTL or on Netlist:

- Power Intent Consistency Checks
- Signal Corruption Checks
- Structural Checks
- Power and Ground (PG) Checks
- Functional Checks

For more information on each of the above checks, refer to the *VC Low Power User Guide*, which is available at the location: https://solvnet.synopsys.com/dow_retrieve/latest/ni/vc_static.html

To run VC LP, complete the following steps:

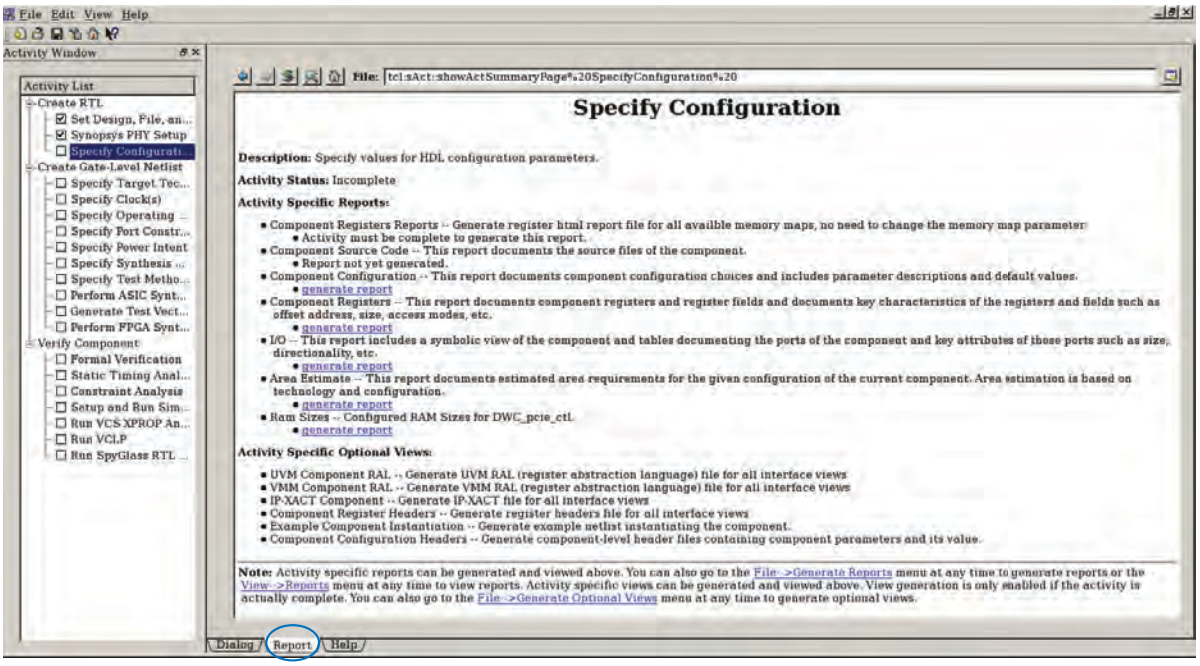
1. Select **Verify Component** > Run **VCLP activity**.
2. In the **Setup** tab, complete the following fields:
   - **VCLP Tool Root**: Specify the complete path for the VC LP tool.
   - **Target Stage**: Select either RTL or Netlist stage.
     - For **RTL** stage, specify the location of the *.upf file.
     - For **Netlist** stage, specify the location of the netlist and the upf prime file (UPF output of Design Compiler).
3. Click on **Apply**.

On completing the above steps, the results are available in the **Report** tab. Go to the report summary, and check for any errors or warnings. For more information on debugging the warnings or errors, refer to the *VC Low Power User Guide*.

2.14 Creating Optional Views and Reports

After configuring the controller, you can generate several configuration reports. The coreConsultant tool displays a list of activity-specific views you can optionally generate. You can access this list by clicking the **Report** tab, shown in Figure 2-31, in the **Specify Configuration** activity.

Figure 2-31 Generating Activity-Specific Reports and Views



2.14.1 Activity-Specific Reports and Views

You can generate reports that document the I/O signals, parameters, and register descriptions. describes the activity-specific optional reports.

Table 2-6 Activity-Specific Reports

Report	File Names (in <workspace>/report)	Description
Component Configuration	ComponentConfiguration.html ComponentConfiguration.xml	Documents component configuration choices and includes parameter descriptions and default values
Component Registers	ComponentRegisters.html ComponentRegisters.xml	Documents component registers and register fields; documents key characteristics of the registers and fields such as offset address, size, and access mode
I/O	IO.html IO.xml	Includes a symbolic view of the component and tables documenting the ports of the component and key attributes of those ports such as size, direction

You can also generate other reports and views such as area estimates, IP-XACT (to use IP-XACT XML format), component level header files, example component instantiation, and RAL files. These reports can be viewed through the coreConsultant **Reports** tab, and they are saved in the `<workspace>/export` directory.Table 2-7 describes the activity-specific optional views. Table 2-8 describes the UVM/VMM RALF and IP-XACT memory map views generated.

Table 2-7 Activity-Specific Optional Views

Optional Views	File Name (in <workspace>/export)	Description
UVM/VMM Component RAL	DWC_pcie_<memory map view>_cpcie_usp.ralf DWC_pcie_<memory map view>_cpcie_dsp.ralf	Generates UVM/VMM RAL (register abstraction language) files for all interface views
IP-XACT Component	DWC_pcie_<memory map view>_cpcie_usp.xml DWC_pcie_<memory map view>_cpcie_dsp.xml	Generates IP-XACT files for all interface views
Component Register Headers	./headers/*	Generates register header files for all interface views
Example Component Instantiation	DWC_pcie_ctl_inst.sv	Generates an example netlist that instantiates the component
Component Configuration Headers	DWC_pcie_ctl.h	Generates component-level header files containing component parameters and its values

Table 2-8 UVM/VMM RALF and IP-XACT Memory Map Views

View Name[a]	Memory Blocks Visible Native DBI Controller	 AMBA Bridge Controller
dbi	All the memory blocks excluding iATU, DMA and shadow registers	When AMBA Bridge is enabled, the dbi view includes the iATU, DMA, and Shadow registers. The starting address of iATU, DMA, and Shadow registers is dependent on the configuration and is decided by the position of CS2 and CDM/ELBI bit. For more information on accessing iATU, DMA, and Shadow registers through DBI when AMBA bridge is enabled, see “AXI Bridge DBI Access Details” section in the *Databook*.
unroll_dbi	Only iATU and DMA register memory blocks	This view is not generated when AMBA Bridge is enabled
wire	All the memory blocks excluding iATU, DMA and shadow registers	All the memory blocks excluding iATU, DMA and shadow registers
unroll_wire	Only iATU and DMA register memory blocks	Only iATU and DMA register memory blocks
dbi2	Only the shadow register memory blocks	This view is not generated when AMBA Bridge is enabled

a. Starting Address: For all the views the starting address is '0h except for dbi view in AMBA Bridge configuration.

Note To automatically generate these reports or views, select appropriate check boxes in the File > Generate Reports or File > Generate Optional Views dialog box.

2.14.2 Format of Activity-Specific Reports

The reports are generated in XML using the DocBook schema (an open source XML-based language). These reports are subsequently rendered in HTML by applying an XSLT style sheet.

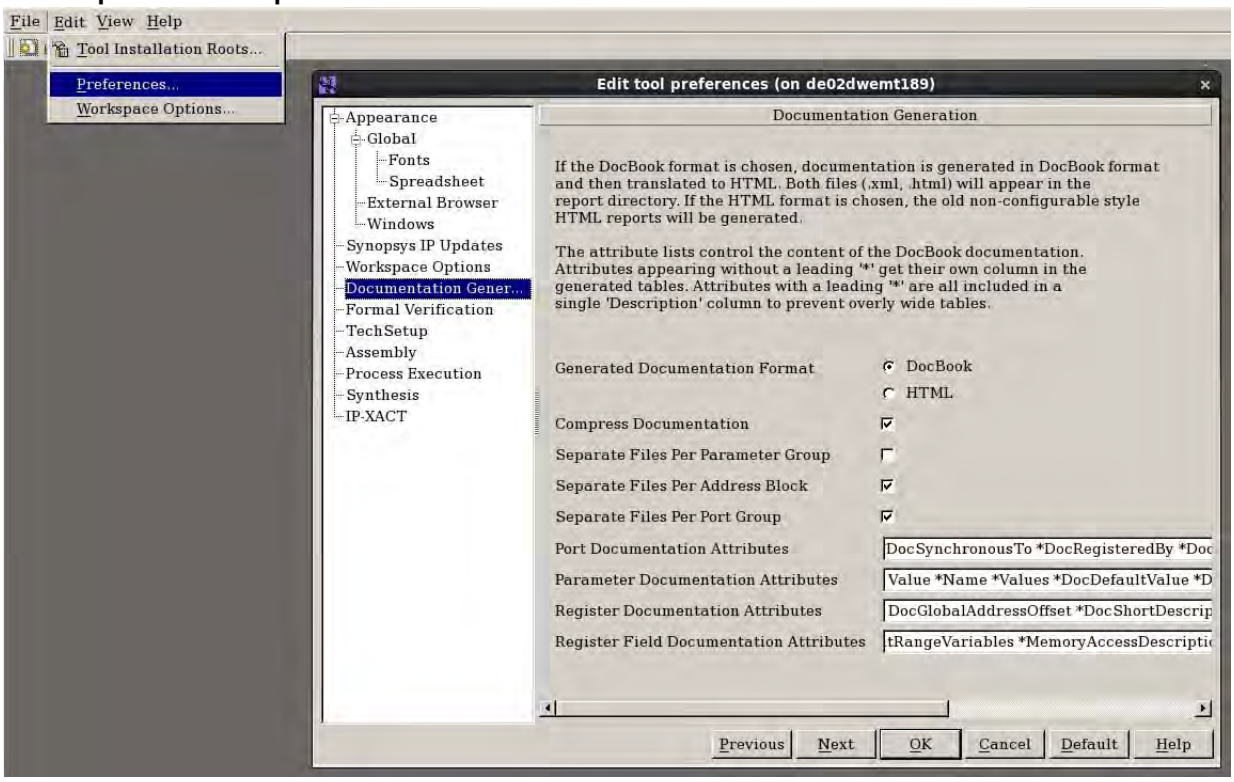
2.14.3 Setting Preferences for Report Generation

It is possible to generate reports as a single file or multiple files based on specific groups of signals and registers. To generate reports contained in a single file for easier navigation, set the preference as follows:

1. In the coreConsultant GUI, click **Edit > Preferences**.
2. In the **Edit tool preferences** window, select **Document Generation**.
3. Uncheck the "Separate Files Per..." check boxes, as shown in Figure 2-32.

You have to do this only once because these settings are saved in your `~/.synopsys_rt_prefs.tcl` file. You must do this before you create the reports. If you have previously created reports with old settings, delete the workspace and start again.

Figure 2-32 Separate Files Options

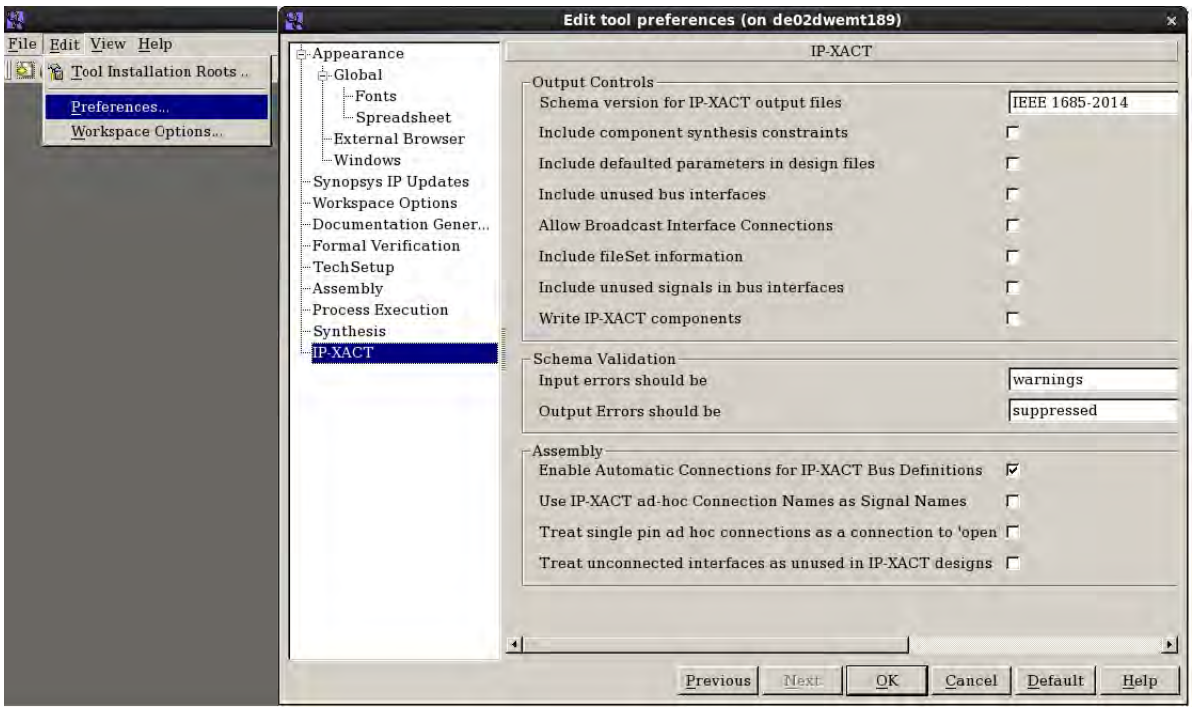


2.14.4 Setting Preferences for IP-XACT

It is possible to configure the output of IP-XACT to the required spec. To choose the schema version for IP-XACT output files set the preference as follows:

1. In the coreConsultant GUI, click **Edit > Preferences**.
2. In the Edit tool preferences window, select IP-XACT
3. Select the schema version for IP-XACT output files, from the drop-down list as shown in Figure 2-33.

Figure 2-33 IP-XACT Preferences



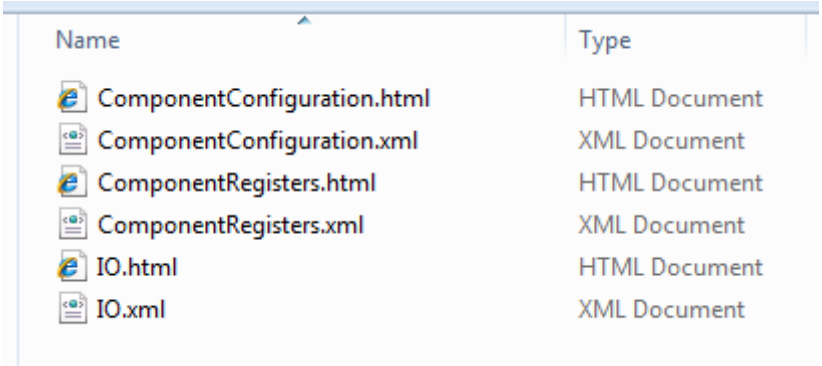
2.14.5 Accessing Reports

To view the report, click **Reload Page** after clicking **generate report** as shown in Figure 2-34.

Figure 2-34 Viewing Reports

You can also access these report files from the `<workspace>/report` directory. The content of this directory is like that shown in Figure 2-35.

Figure 2-35 Contents of the report Directory

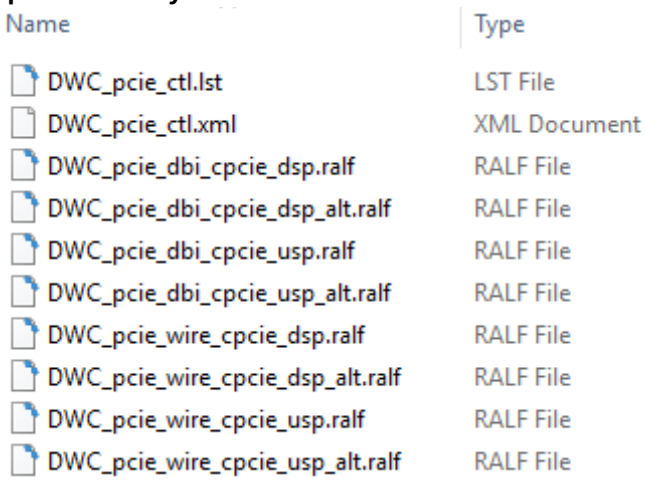


2.14.6 Generating and Accessing Activity-Specific Views

To generate the required view, click the respective **generate view** in the **Report** tab.

You can also access the generated views from the `<workspace>/export` directory. The content of this directory is like that shown in Figure 2-36.

Figure 2-36 Contents of the export Directory



2.14.7 Populating IP-XACT File Manually

For SR-IOV configurations with more than 256 VFs (`CX_SRIOV_ENABLE =1, CX_NVFUNC > 256`), the generated IP-XACT file only defines the first VF in each PF. This is a coreConsultant tool limitation. However, you can post-process the IP-XACT file to add the other VF's as follows:

```
cd <workspace>
```

```
/export/unroll_vfs_registers.pl -cc_file src/DWC_pcie_ctl_cc_constants.v
-ipxact_file export/DWC_pcie_ctl.xml -map_vf_ctrl 1
```

For more information on writing batch file, see "Creating a Batch Script" on page 149.

2.15 Exporting a Controller From coreConsultant to Your Chip Design Database

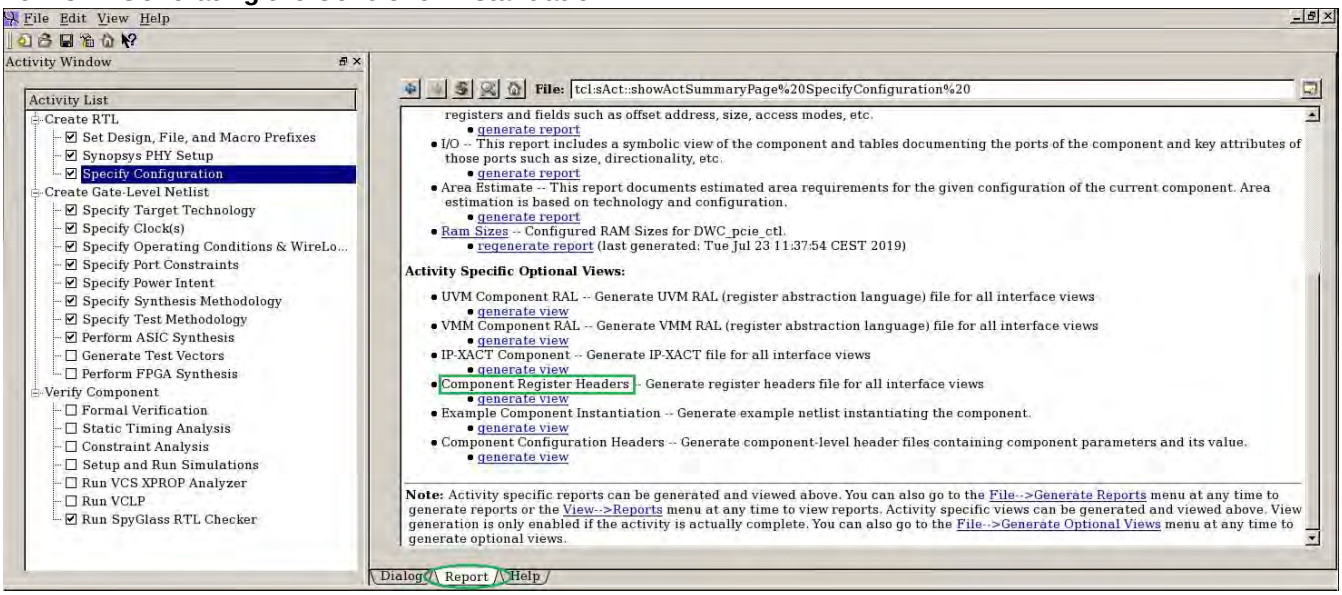
At a certain point you might want to transfer (export) a configured controller into your own custom design flow. The coreConsultant tool has several features to accomplish this. This section shows you how to access the relevant files, scripts, and information for your configured controller. You can then transfer these to your chip design database. The topics in this section are as follows:

- "Generating a Sample Controller Instantiation"
- "Synthesizing to a Device Outside of coreConsultant" on page 64
- "Exporting Formality, DFT, and ATPG Scripts" on page 65
- "Exporting Views and Reports" on page 66

2.15.1 Generating a Sample Controller Instantiation

After you have configured your PCIe controller, you can generate a simple component instantiation.

Figure 2-37 Generating the Controller Instantiation



To instantiate the controller, follow these steps:

1. Select the Report tab under the **Specify Configuration** activity.
2. Click the "generate view" link as indicated in Figure 2-37.

The controller instantiation is now available in `export/DWC_pcie_ctl_inst.v`. If you want coreConsultant to always generate this 'Component Instantiation', you must set the relevant option as described in "Creating Optional Views and Reports" on page 58.

Hint

- There is also an instantiation in <workspace>/examples/pcie_iip_device.v
- It has connections to a "Clock & Reset" block, external memories (where appropriate), and to other useful interface modules.
- For more information, see "Design Example" on page 97.

2.15.2 Synthesizing to a Device Outside of coreConsultant

If you want to map and synthesize the controller to a device using the Synopsys synthesis tools within coreConsultant, then follow the process as outlined in "Synthesizing the Controller" on page 34. Otherwise, you must export the synthesis scripts from coreConsultant as outlined here so that you can synthesize the controller in your own design flow. This section shows you how to access the relevant synthesis scripts and information for your configured controller. You can then transfer these to your chip design database.

There are two methods used to export the synthesis scripts. If you have access to Synopsys Design Compiler, then you can use either method. If you do not have access to Synopsys Design Compiler, you must use the "write_sdc" method.

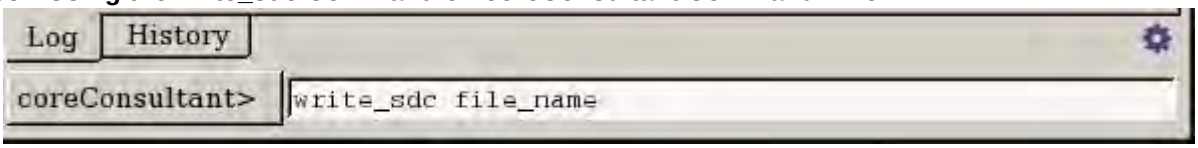
Hint To get a detailed description of how to integrate the IP at chip level. enter man Synthesis_API on the coreConsultant command line.

Method 1: Use the write_sdc Command

You use the `write_sdc` command to write out a script in a Synopsys Design Constraints (SDC) format. The SDC files are TCL scripts that use a subset of the commands supported by PrimeTime and Design Compiler. This file generation process does not require a Design Compiler license. You invoke this command on the coreConsultant command line as follows:

```
write_sdc file_name
```

Figure 2-38 Using the write_sdc Command on coreConsultant Command Line



This file resides in the directory where you invoked coreConsultant.

Attention

- When a technology library is not loaded in the **Specify Target Technology -> dc_shell target _library variable**, the generated SDC file contains constraints relative to a generic technology library. You need to modify these constraints to match your target library cell names.
- If a technology library is not loaded, use the [-force] option to force the writing of the SDC file even if the **Specify Target Technology** activity has not been completed: write_sdc -force file_name

Method 2: Access Synopsys Design Compiler Scripts

You must first complete ALL the setup steps as outlined in "Performing ASIC Synthesis" on page 35, and then select the "**Synthesize -> Options -> Execution Options -> Generate scripts only?**" option before clicking APPLY button.

Table 2-9 ASIC Synthesis Flow Files in <workspace>/syn

Type	File Name	Description
ASIC	constrain/script/DWC_pcie_*.cscr	Primary Constraints
	run.scr	Runs synthesis
	Makefile	Creates synthesis scripts
	auxScripts/set_ahfs_zero.tcl	Reduces high fanout nets to prevent Formality failures

2.15.3 Exporting Formality, DFT, and ATPG Scripts

If you want to run Formality and TestMAX (for ATPG) using the Synopsys synthesis tools within coreConsultant, then follow the process as outlined in "Formal Equivalence Checking using Formality" on page 45 or "Running ATPG using TestMAX™" on page 47.

Otherwise, you must export the relevant scripts from coreConsultant as outlined in this section and transfer these scripts to your chip design database where you can run Formality or TestMAX. Note, you must have access to Synopsys TestMAX and Formality tools to generate and export the relevant scripts.

The topics covered in this section are as follows:

- "Exporting Formality Scripts"
- "Exporting DFT Scripts" on page 65
- "Exporting ATPG Scripts" on page 65
- "Exporting Views and Reports" on page 66

2.15.3.1 Exporting Formality Scripts

You must first complete the setup steps as outlined in "Formal Equivalence Checking using Formality" on page 45, and then select the "**Execution Options -> Generate scripts only?"** option before clicking APPLY button. The generated Formality scripts in `<workspace>/syn` do not contain any controller-specific directives. They are only useful to keep track of the `.svf` files (for all synthesis stages) and pass them all to Formality. You must use the Svf flow for RTL-to-gate comparison. To disable non-applicable warning messages in Formality, ensure that your `synopsys_fm.setup` file contains the following line:

```
set hdlin_warn_on_mismatch_message {FMR_ELAB-146 FMR_ELAB-147 FMR_VLOG-116}
```

2.15.3.2 Exporting DFT Scripts

For more information, see "Inserting Design-For-Test using Design Compiler" on page 46.

2.15.3.3 Exporting ATPG Scripts

You must first complete the setup steps as outlined in "Running ATPG using TestMAX™" on page 47, and then select the "**Execution Options -> Generate scripts only?"** option before clicking APPLY button. The generated TestMAX scripts in `<workspace>/syn` do not contain any controller-specific directives. They are only useful to keep track of the following and to pass them to TestMAX:

- `.spf` file (from the synthesis DFT stage)
- Technology library netlist
- Design netlist

2.15.3.4 Exporting Views and Reports

As part of the export process, you might want to generate documentation for your configured controller, so that it can be a part of the documentation for the entire chip. The report generation process is DocBook-based. This allows you to generate documentation for I/O definition, parameter definition, and memory maps from the DocBook XML using your in-house DocBook XML conversion tools. You can also extract documentation from the IP-XACT file.

For more information see "Creating Optional Views and Reports" on page 58. All reports can be accessed in the `<workspace>/report` directory. The optional views are in the `<workspace>/export` directory.

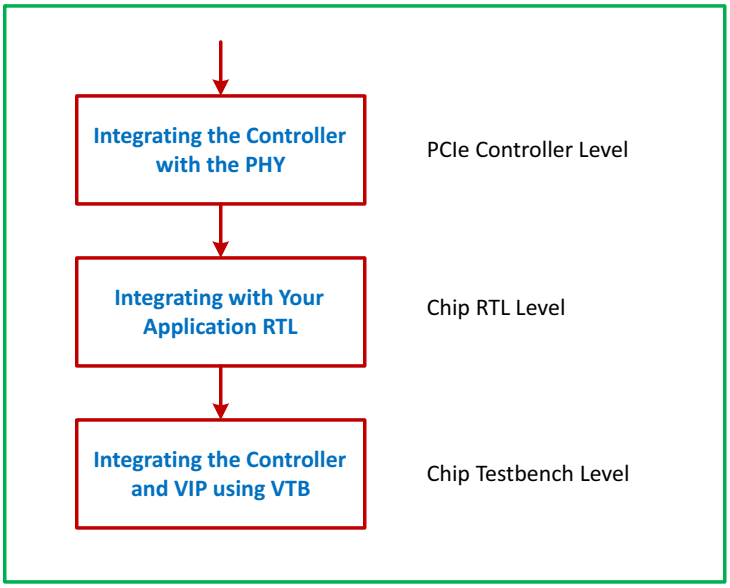
3

Integrating the Controller with the PHY, or Application RTL, or Verification IP

This chapter provides an overview of the step-by-step process you use to integrate the PCIe controller with your PHY and chip-level RTL and Verification IP. The following topics are covered:

- "Integrating the Controller with the PHY" on page 68
- "Integrating with Your Application RTL" on page 92
- "Integrating the Controller and VIP using VTB" on page 98

Figure 3-1 Integration Flow



3.1 Integrating the Controller with the PHY

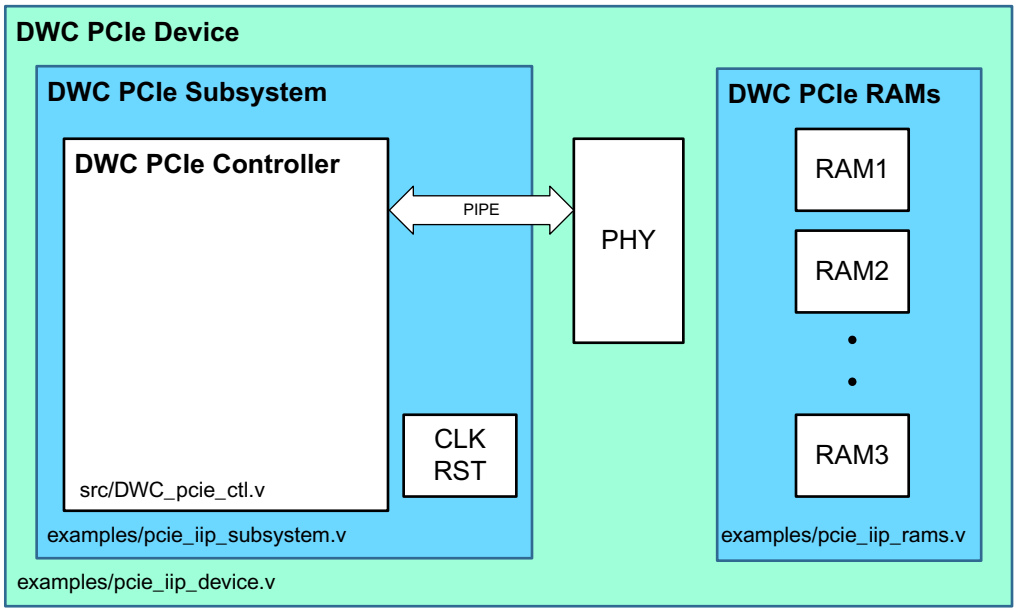
You must configure the controller to match the requirements of the PHY you are using. This section covers the following topics:

- "Overview" on page 68
- "Connecting the PIPE I/O Interface" on page 69
- "Integrating the PCIe Controller and Synopsys PHYs" on page 79
- "Integrating the PCIe Controller and Third-party PHYs" on page 81
- "Bifurcation Guidelines" on page 85

3.1.1 Overview

PHYs are instantiated outside the controller, interfacing through the standard PIPE interface. A custom instantiation file is provided for all Synopsys PHYs, and some third party PHYs. For more information, see "Integrating a Custom PIPE PHY" on page 82.

Figure 3-2 Integrating Controller with PHY



Example PHY

Synopsys also provides an example digital PIPE PHY simulation module (Physical Coding Sublayer [PCS], glue logic RTL and generic digital SERDES) that connects to hi-speed I/O pads in your design. This block is provided as an example for generic simulation purposes only. You might temporarily use this for simulation purposes until you receive a simulation model from your PHY vendor. For more information, see "Integrating an Example PIPE PHY" on page 82. This PHY is not for implementation/synthesis in ASIC.

3.1.2 Connecting the PIPE I/O Interface

- The controller is compliant to the *PHY Interface for the PCI Express, SATA, USB 3.2, DisplayPort, and USB4 Architectures (PIPE) Version 4.4.1, Version 5.1.1, Version 6.1.1* by default. Depending upon the configuration, the controller is configured to *PIPE Specification for PCI Express, Version 4.3* or *4.2*.
- For functional information of the signals in this interface, see "PIPE Interface" section in the "Signal Interfaces" chapter of the *Databook*.
- An example subsystem describing the connections between the controller and the PHY can be found at `<workspace>/examples/pcie_iip_subsystem.v`.

Figure 3-3 to Figure 3-8 provides guidance for connection of PIPE interface signals of the PCIe controller with your PHY.

Figure 3-3 Guidelines for Connecting Common PIPE Signals

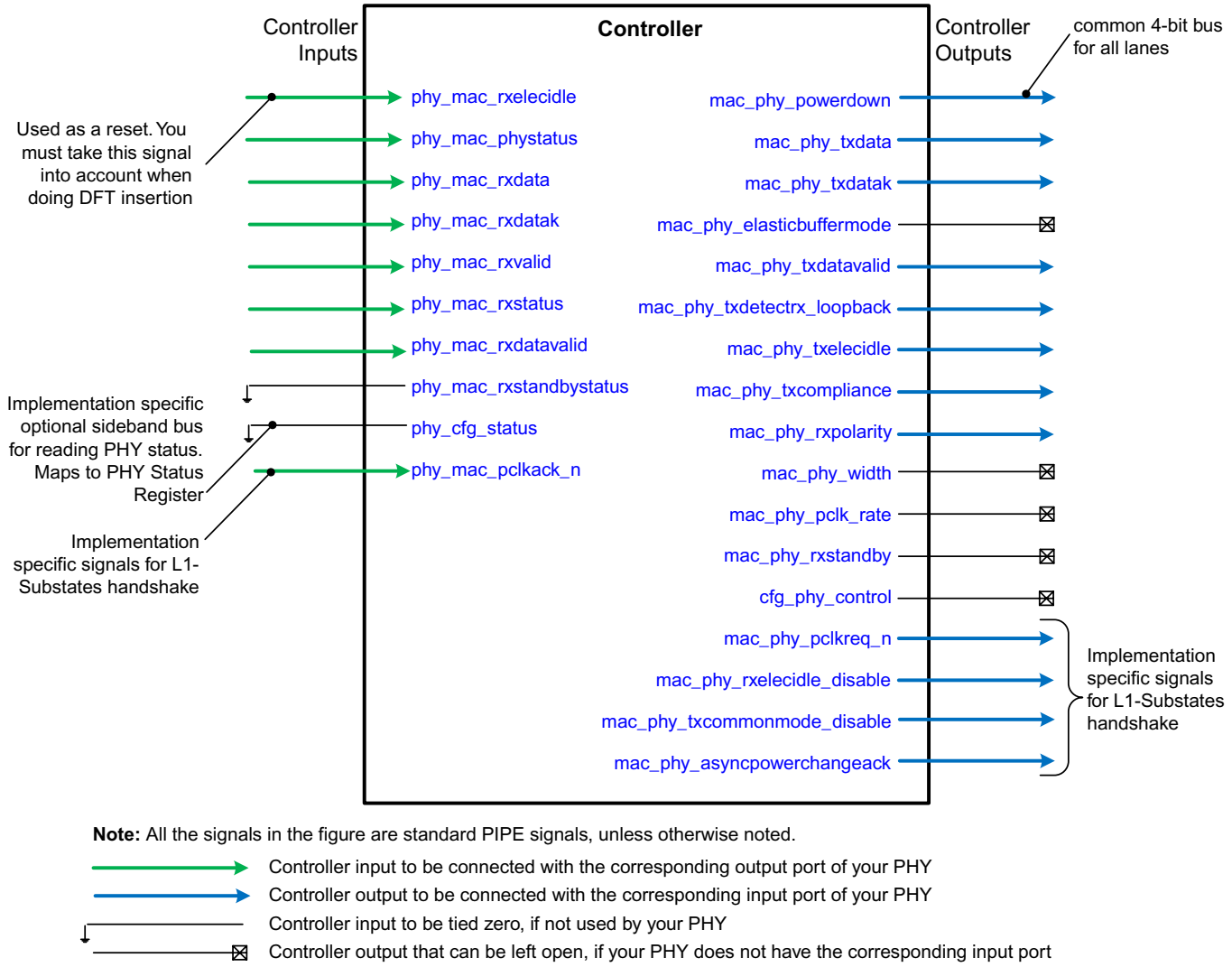


Figure 3-4 Guidelines for Connecting Gen2/Gen3 PIPE Signals

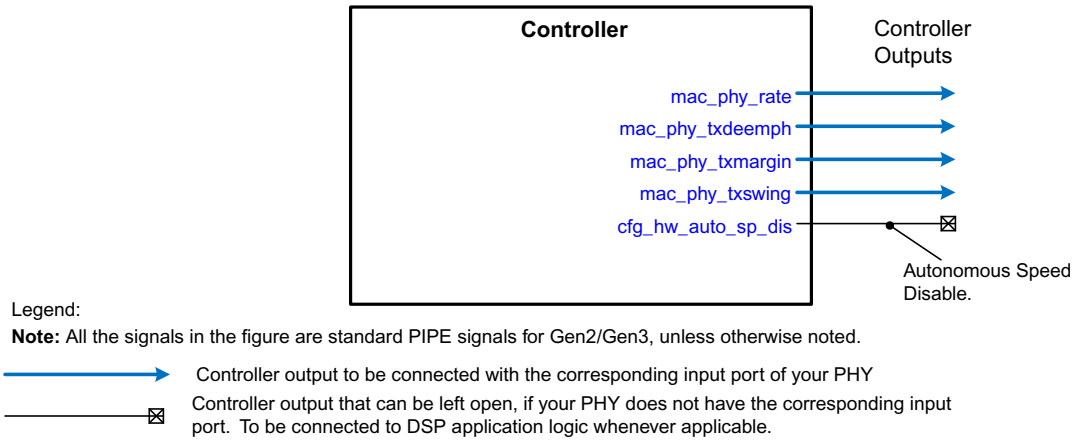


Figure 3-5 Guidelines for Connecting Gen3-Only PIPE Signals

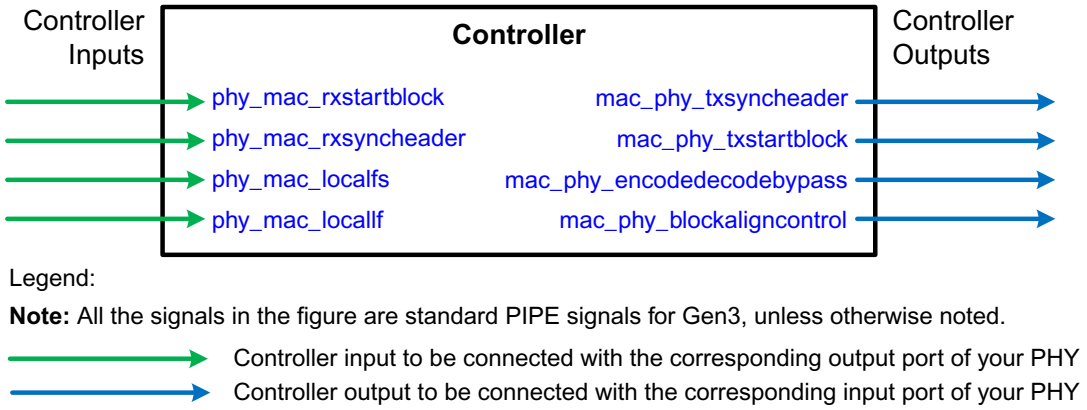


Figure 3-6 Guidelines for Connecting Gen3-Only Equalization PIPE Signals

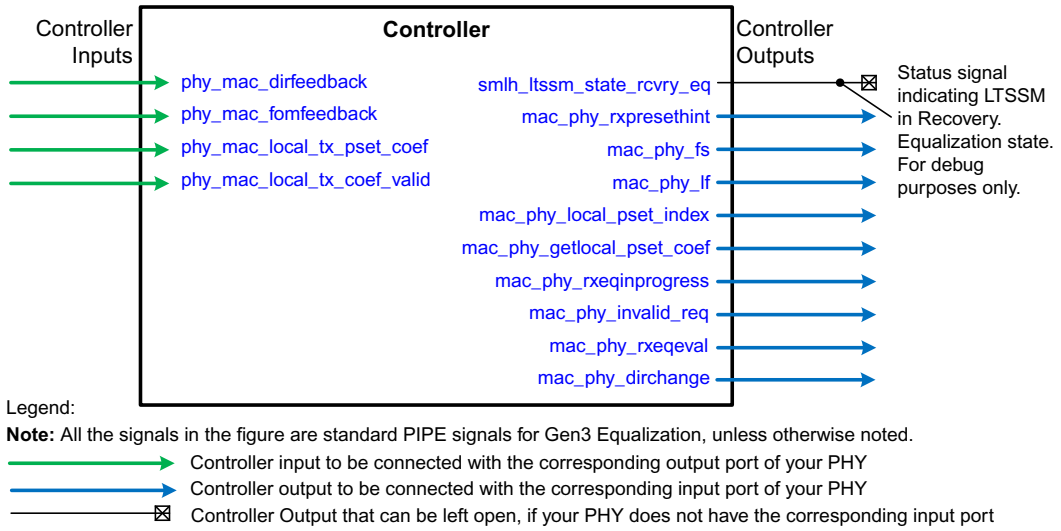


Figure 3-7 Guidelines to Connect Gen4-Only PIPE Signals

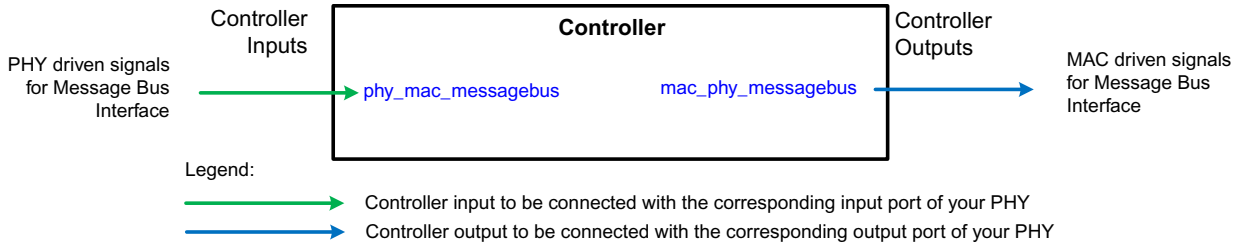
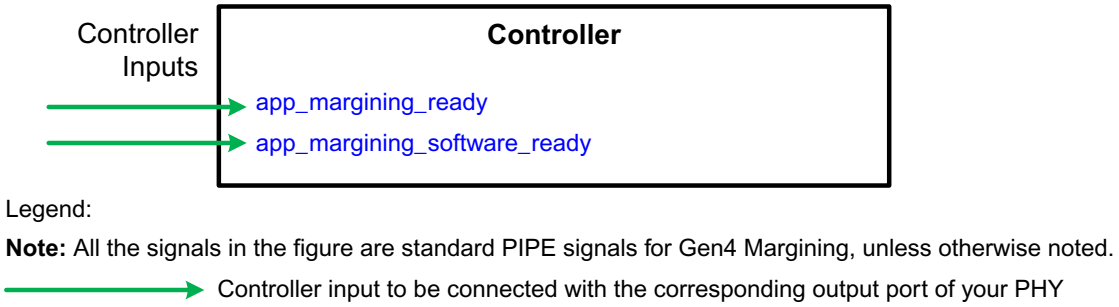


Figure 3-8 Guidelines to Connect Gen4-Only Margining Signals



3.1.2.1 Other Integration Considerations

Bifurcation and Aggregation

For more information on bifurcation and aggregation, see "Bifurcation Guidelines" on page 85.

3.1.3 Registering the PIPE Interface

3.1.3.1 Enabling Pipeline Stages Inside the Controller

You can insert pipelining on the PIPE Tx and Rx paths, if you enable the following parameter in the controller from coreConsultant:

- `CX_CUSTOM_PIPELINING`

The following two parameters can be set to register the lane_flip module top level PIPE ports:

- `CX_LANEFLIP_RX_REGIN`
- `CX_LANEFLIP_TX_REGOUT`

If you have not enabled `CX_LANE_FLIP_CTRL_EN`, the following parameters can be used to register the `rmlh_pipe` and `xmlh_pipe` top level PIPE ports

- `CX_RMLH_PIPE_REGOUT`
- `CX_XMLH_PIPE_REGOUT`

For more information on parameters, see "Advanced Pipeline Config Parameters" section in "Parameters Description" chapter of the *Reference Manual*.

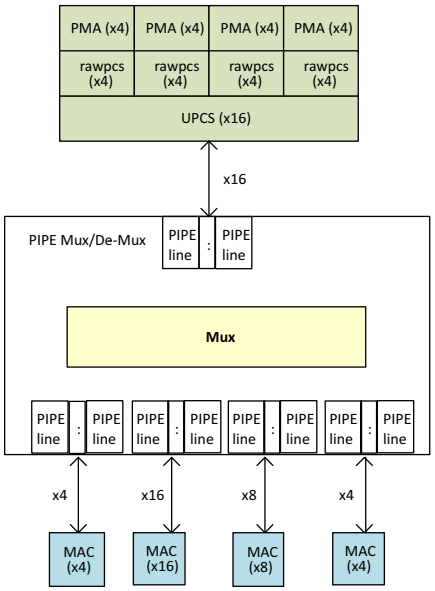
3.1.3.2 Inserting Pipeline Stages Outside the Controller

If you have very long paths between the controller and the PHY, for example due to controller/PHY muxing logic, you may need to insert pipeline stages outside the controller. Consider the example logic in Figure 3-9. Multiple controller instances are connected to a single PHY instance through muxing logic. Pipeline stages have been inserted on both sides of the mux, in identical per-lane pipeline modules.

Figure 3-11, Figure 3-12 describes the suggested pipeline logic required for each of the PIPE signals with PCLK or RxCLK as PHY input or output.

Note You can inspect the sample module at `<workspace>/examples/DWC_pcie_pipe_retiming.sv`

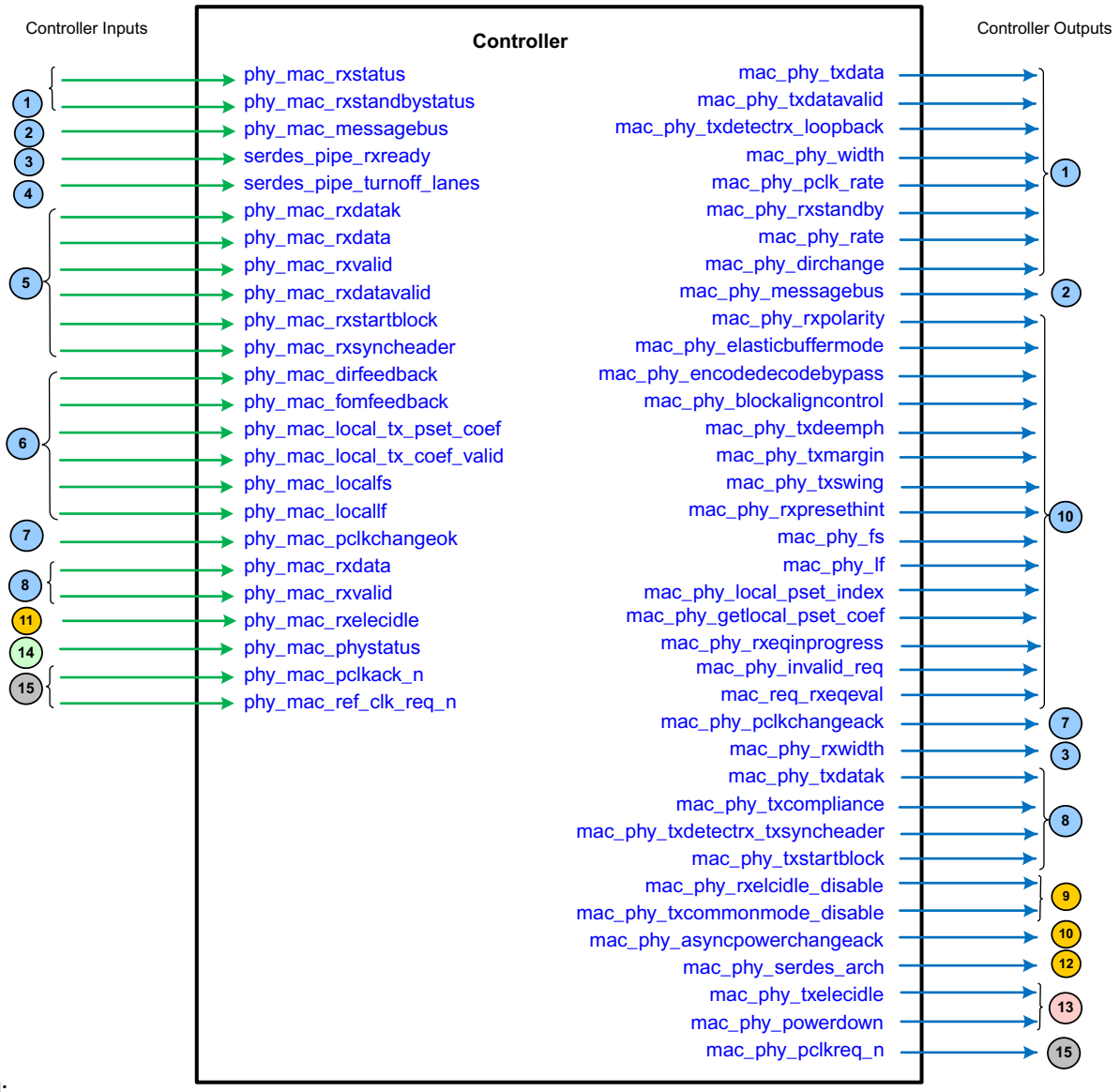
Figure 3-9 Example to Insert Pipeline Stages Outside the Controller



3.1.3.3 Signal Connections and Logic

Figure 3-10 shows the signal connections and groups.

Figure 3-10 Signal Connections



Legend:

Signal-flop-registered signals with PCLK

- 1 when CX_L1_SUBSTATES_ENABLE = 1
- 2 when CX_PIPE_VER > 2
- 3 when CX_PIPE_SERDES_ARCH = 1
- 4 when CX_PIPE_SERDES_ARCH_LANES_TURNOFF = 1
- 5 when CX_PIPE_SERDES_ARCH = 0
- 6 when CX_PIPE_LOW_PIN_COUNT=0 (CX_PIPE_VER < 3)
- 7 when CX_PIPE_PCLK_MODE = 1
- 8 when CX_PIPE_SERDES_ARCH = 0

Non registered asynchronous signal

- 9 when CX_L1_SUBSTATES_ENABLE = 1
- 10 when CX_PIPE43_ASYNC_HS_BYPASS =0
- 11

Non registered quasi-static signal

- 12

Signals that are asynchronous in P1.CPM, P1.1(2) or P2, otherwise they are registered

- 13

Signal that is synchronously asserted but asynchronously de-asserted in P1.CPM, P1.1(2) or P2, otherwise registered. (PCLK as PHY output) Signal that is asynchronous when transitioning from P1.CPM, P1.1(2) or P2 to any of these states, otherwise registered. (PCLK as PHY input)

- 14

Asynchronous sideband signals that are not part of the PIPE interface. No need to add pipeline.

- 15

3.1.3.4 Other Requirements

Figure 3-11 Pipeline Logic for Each PIPE Signal with Pclk as PHY output (CX_PIPE_PCLK_MODE = 0)

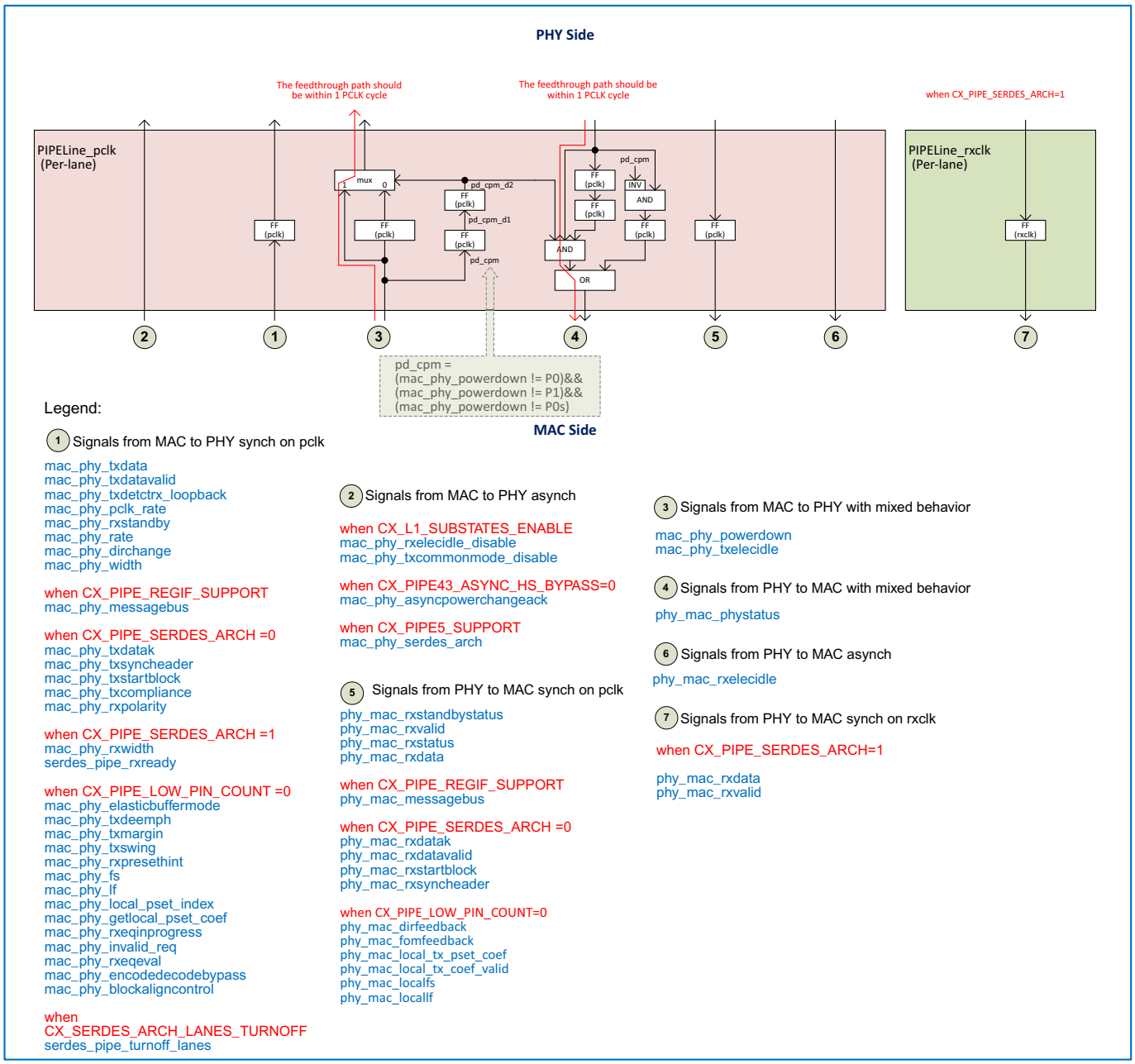
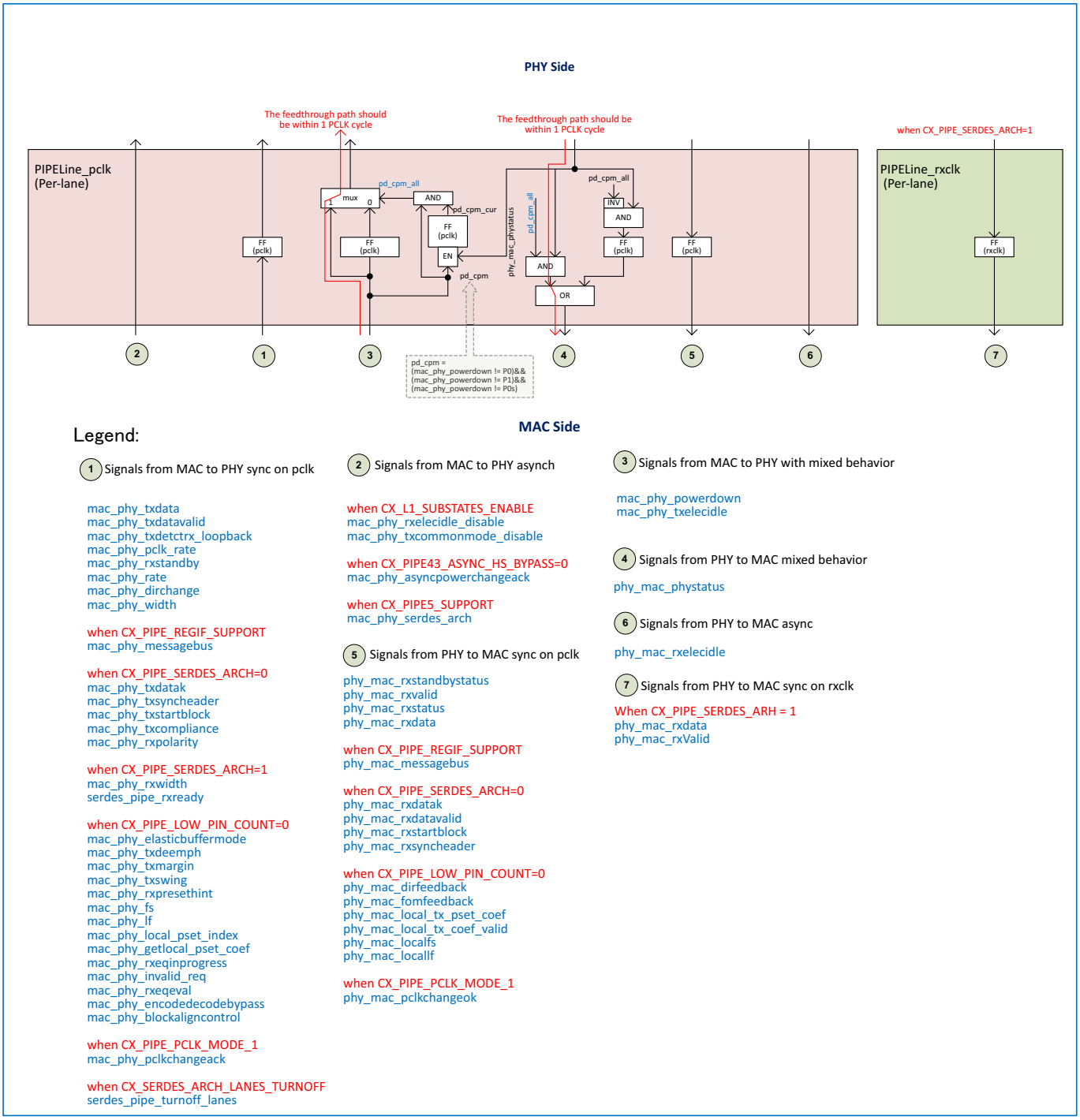


Figure 3-12 Pipeline Logic for Each PIPE Signal with PCLK as PHY input (CX_PIPE_PCLK_MODE! =0)



In PCLK as PHY Output, the `phy_mac_phystatus` width during P2/P1.CPM entry must be at least 3 PCLK cycles if one pipeline module is instantiated and at least 6 PCLK cycles when two pipeline modules are instantiated.

Signals that are common between lanes can be fanned out to the per-lane pipeline module for simplification. However, the trade-off is that it costs extra gates.

3.1.3.5 SDC Guidance

The following feedthrough paths can be set false path for all configurations:

- `set_false_path -through [get_pins mac*/phy_mac_rxelecidle*]`
- `set_false_path -through [get_pins u_phy/pipe_tx*_disable]`
- `set_false_path -through [get_pins u_phy/pipe_rx*_disable]`
- `set_false_path -through [get_pins u_phy/pipe_lane*_asyncpowerchangeack]`

The following feedthrough paths can be set multi-cycle path (indicated by red arrows in Figure 3-11).

- When one `DWC_pcie_pipe_retiming.sv` is instantiated between Controller and PHY, following multi-cycle can be applied.
  - `set_multicycle_path 2 -setup -through [get_pins mac*_pipe_retiming*_pipe-lane*/i_m_txelecidle] -through [get_pins mac*_pipe_retiming*_pipe-lane*/o_p_txelecidle]`
  - `set_multicycle_path 2 -setup -through [get_pins mac*_pipe_retiming*_pipe-lane*/i_m_powerdown] -through [get_pins mac*_pipe_retiming*_pipe-lane*/o_p_powerdown]`
  - `set_multicycle_path 2 -setup -through [get_pins mac*_pipe_retiming*_pipe-lane*/i_p_phystatus] -through [get_pins mac*_pipe_retiming*_pipelane*/o_m_-phystatus]`
  - `set_multicycle_path 1 -hold -through [get_pins mac*_pipe_retiming*_pipe-lane*/i_m_txelecidle] -through [get_pins mac*_pipe_retiming*_pipe-lane*/o_p_txelecidle]`
  - `set_multicycle_path 1 -hold -through [get_pins mac*_pipe_retiming*_pipe-lane*/i_m_powerdown] -through [get_pins mac*_pipe_retiming*_pipe-lane*/o_p_powerdown]`
  - `set_multicycle_path 1 -hold -through [get_pins mac*_pipe_retiming*_pipe-lane*/i_p_phystatus] -through [get_pins mac*_pipe_retiming*_pipelane*/o_m_-phystatus]`
- When two `DWC_pcie_pipe_retiming.sv` are instantiated between Controller and PHY, following multi-cycle can be applied.

  "`mac*_pipe_retiming*_pipelane_PHY_SIDE_INSTANCE`" must be updated to correct instance name which is used by the customer.
  - `set_multicycle_path 3 -setup -through [get_pins mac*_pipe_retiming*_pipe-lane*/i_m_txelecidle] -through [get_pins mac*_pipe_retiming*_pipe-lane*/o_p_txelecidle]`
  - `set_multicycle_path 3 -setup -through [get_pins mac*_pipe_retiming*_pipe-lane*/i_m_powerdown] -through [get_pins mac*_pipe_retiming*_pipe-lane*/o_p_powerdown]`
  - `set_multicycle_path 3 -setup -through [get_pins mac*_pipe_retiming*_pipe-lane*/i_p_phystatus] -through [get_pins mac*_pipe_retiming*_pipelane*/o_m_-phystatus]`
  - `set_multicycle_path 2 -hold -through [get_pins mac*_pipe_retiming*_pipe-lane*/i_m_txelecidle] -through [get_pins mac*_pipe_retiming*_pipe-lane*/o_p_txelecidle]`

- `set_multicycle_path 2 -hold -through [get_pins mac*_pipe_retiming*_pipe-lane*/i_m_powerdown] -through [get_pins mac*_pipe_retiming*_pipe-lane*/o_p_powerdown]`
- `set_multicycle_path 2 -hold -through [get_pins mac*_pipe_retiming*_pipe-lane*/i_p_phystatus] -through [get_pins mac*_pipe_retiming*_pipelane*/o_m_-phystatus]`
- `set_multicycle_path 2 -setup -from [get_pins mac*_pipe_retiming*_pipelane_-PHY_SIDE_INSTANCE/phy_mac_phystatus_nocpm/CK] -through [get_pins mac*_pipe_retiming*_pipelane*/i_p_phystatus] -through [get_pins mac*_pipe_retiming*_pipelane*/o_m_phystatus]`
- `set_multicycle_path 1 -hold -from [get_pins mac*_pipe_retiming*_pipelane_-PHY_SIDE_INSTANCE/phy_mac_phystatus_nocpm/CK] -through [get_pins mac*_pipe_retiming*_pipelane*/i_p_phystatus] -through [get_pins mac*_pipe_retiming*_pipelane*/o_m_phystatus]`

When the controller and PHY are working in Pclk as PHY Input (`CX_PIPE_PCLK_MODE >0`) the following additional constraints are required:

- When one DWC_pcie_pipe_retiming.sv is instantiated between Controller and PHY, following multi-cycle can be applied.
  - `set_multicycle_path 2 -setup -through [get_pins mac*_pipe_retiming*_pipe-lane*/i_m_powerdown] -through [get_pins mac*_pipe_retiming*_pipe-lane*/o_p_txelecidle]`
  - `set_multicycle_path 1 -hold -through [get_pins mac*_pipe_retiming*_pipe-lane*/i_m_powerdown] -through [get_pins mac*_pipe_retiming*_pipe-lane*/o_p_txelecidle]`
  - `set_multicycle_path 2 -setup -through [get_pins mac*_pipe_retiming*_pipe-lane*/i_m_powerdown] -through [get_pins mac*_pipe_retiming*_pipelane*/o_m_-phystatus]`
  - `set_multicycle_path 1 -hold -through [get_pins mac*_pipe_retiming*_pipe-lane*/i_m_powerdown] -through [get_pins mac*_pipe_retiming*_pipelane*/o_m_-phystatus]`
- When two DWC_pcie_pipe_retiming.sv are instantiated between Controller and PHY, following multi-cycle can be applied.
  - `set_multicycle_path 3 -setup -through [get_pins mac*_pipe_retiming*_pipe-lane*/i_m_powerdown] -through [get_pins mac*_pipe_retiming*_pipe-lane*/o_p_txelecidle]`
  - `set_multicycle_path 2 -hold -through [get_pins mac*_pipe_retiming*_pipe-lane*/i_m_powerdown] -through [get_pins mac*_pipe_retiming*_pipe-lane*/o_p_txelecidle]`
  - `set_multicycle_path 3 -setup -through [get_pins mac*_pipe_retiming*_pipe-lane*/i_m_powerdown] -through [get_pins mac*_pipe_retiming*_pipelane*/o_m_-phystatus]`
  - `set_multicycle_path 2 -hold -through [get_pins mac*_pipe_retiming*_pipe-lane*/i_m_powerdown] -through [get_pins mac*_pipe_retiming*_pipelane*/o_m_-phystatus]`

3.1.4 Integrating the PCIe Controller and Synopsys PHYs

This section covers the following scenarios for integrating the Synopsys PHYs:

- "Integrating Synopsys PHYs"
- "Troubleshooting the Integration of Synopsys PHYs" on page 80

3.1.4.1 Integrating Synopsys PHYs

Caution

- To ensure that the controller works correctly with the Synopsys PHY package in use, you must follow the guidelines given in this section, even if you are not running the Subsystem test suite provided in the coreKit.
- If you are importing a previously generated configuration batch file, you must source the batch file before completing the Synopsys PHY Setup activity. This creates a baseline configuration, on top of which you can add the required settings. This also ensures that the latest PHY settings from the PHY installation area are applied.
- If you want to override the settings extracted from the PHY installation area, the batch file should be sourced after completing the Synopsys PHY Setup activity,

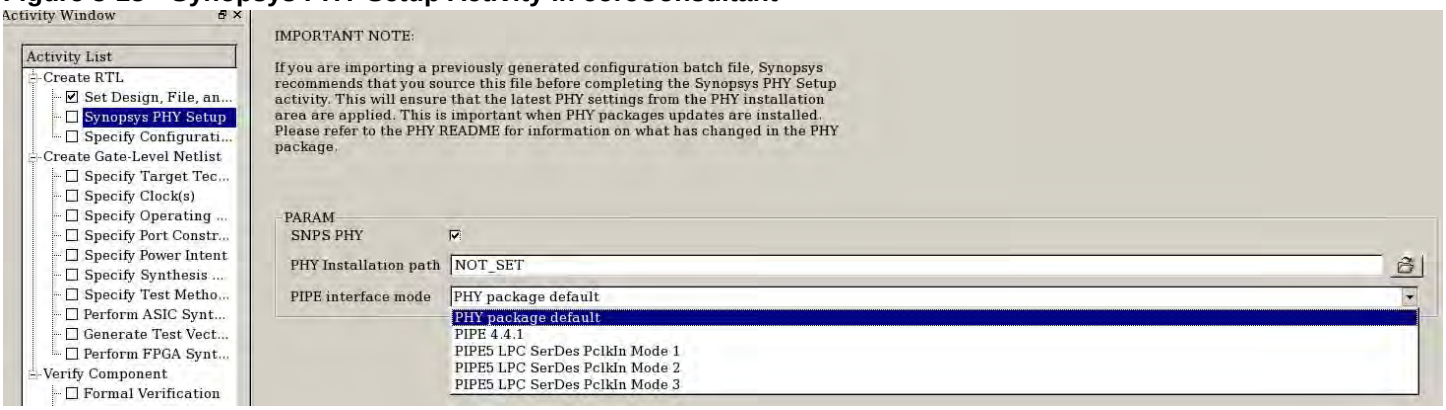
The Synopsys PHY should be configured through the coreConsultant **Synopsys PHY Setup** activity. Click SNPS PHY checkbox and provide the PHY installation path to complete this activity.

You can use the PIPE interface mode parameter to configure the PIPE interface for Synopsys PHY packages that support a selectable PIPE 4.4.1/ PIPE 5.1.1. This information is advertised in the Tcl fragment of the PHY package at `<PHY_INSTALL>/phy/ipxact/*_iip.config`. This file is for reference purposes only, you must not modify it.

Note

Synopsys recommends using the default value of the PIPE interface mode parameter. If you set the PIPE interface mode parameter to any value other than the default value, you must ensure that you configure the PHY package for these values according to the guidelines provided in the "Device Configuration" section of the *uPCS Databook* available at `<PHY_INSTALL>/doc/*_pcs_databook.pdf`.

Figure 3-13 Synopsys PHY Setup Activity in coreConsultant



Note The path to the PHY installation area depends on the specific PHY package version and/or technology you are using. There are two possible directory structures within a PHY package, the following are examples of the two possibilities:

- `<installation_path>/dwc_pcie2phy_smic40ll_x1_2.2a/9m2t`
- `<installation_path>/synopsys/dwc_pcie2phy-tsmc28hpm-x1ns/2.7a`

The **Synopsys PHY Setup activity** scans the PHY installation area and automatically configure the controller, and simulation environment, to use the appropriate PHY instantiation file.

The following message is displayed in the coreConsultant log window when you are configuring the controller:

```
Information: PHY installation area =
/path/to/phy/installation/synopsys/dwc_12g_phy_tsmc28hpcp_x1ns/1.00a
```

```
Information: validating configuration fragment = /path/to/phy/installation
/synopsys/dwc_12g_phy_tsmc28hpcp_x1ns/1.00a/doc/dwc_ent12_mp_phy_tsmc28hpcp_x1_ns_
snps_iip.tcl
```

Table 3-1 describes the location of each Synopsys instantiation file. If you decide to integrate the pcie_iip_device reference in your system, you must export or tie-off the JTAG and other PHY-specific configuration ports.

Table 3-1 PHY Specific Integration Details

Synopsys PHY	Location of PHY Instance File
Consumer 8G	`<workspace>/src/Phy/pcs_c8/phy_instance.vh`
Consumer 10G	`<workspace>/src/Phy/pcs_c10/phy_instance.vh`
Enterprise 12G	`<workspace>/src/Phy/pcs_e12/phy_instance.vh`
Enterprise 16G	`<workspace>/src/Phy/pcs_e16/phy_instance.vh`
Enterprise 25G	`<workspace>/src/Phy/pcs_e25/phy_instance.vh`
Enterprise 32G	`<workspace>/src/Phy/pcs_e32/phy_instance.vh`
SSP Gen1	`<workspace>/src/Phy/pcs_ssp/phy_instance.vh`
SSP Gen2	`<workspace>/src/Phy/pcs_ssp/phy_instance.vh`
Combo 6GMP	`<workspace>/src/Phy/pcs_6gmp/phy_instance.vh`

3.1.4.2 Troubleshooting the Integration of Synopsys PHYs

This section provides troubleshooting tips if you encounter simulation failures in the process of integrating Synopsys PHYs.

Compilation Failing due to an Unresolved Module

- Inspect the file `<workspace>/src/Phy/<architecture>/compile.f`.
- Review the installation path provided to the **Synopsys PHY Setup** activity. For more information, see "Integrating Synopsys PHYs" on page 79.
- If the macro pointing to the top-level is 'null', contact Synopsys support. This might indicate that you are using a PHY that is not supported by your controller release. This can happen if the PHY package is released after your controller release. Synopsys support guides you through the configuration and provides updated integration files if needed.

Simulations Failing due to Link up Errors

- Inspect the content of `<workspace>/src/Phy`.
- If 'custom' appears as a folder, it implies that correct integration files are not used. In this case review your integration steps.
- If the correct integration folder appears, review the controller configuration to ensure that the selected PIPE mode and Link Width (`CX_NL`) are compatible with the PHY package used. For more information on controller-PHY compatibility, see "Frequency, Speed, and Width Support" section in "Product Overview" chapter of the *Databook*. Also consult the PHY databook for a list of the supported PIPE modes.

Contacting Synopsys Support

Open a support ticket and provide the following information:

- PHY package in use (For example, `dwc_pcie2phy_tsmc28hpc_x2ns/2.7c`).
- A copy of`<workspace>/src/Phy/<architecture>/compile.f.`
- The batch file used to create the workspace.
- A copy of the parameters files located at `<workspace>/src/DWC_pcie_ctl_cc_constants.v`
- Log and wave file of the failing test.

Note If multiple tests are failing, check the status of the `tl_traffic*` tests. If these are failing, it is recommended to submit the data for these specific tests as they are faster to debug and root cause.

3.1.5 Integrating the PCIe Controller and Third-party PHYs

This section covers the integration of the following third-party PHYs:

- "Integrating an Example PIPE PHY" on page 82
- "Integrating a Custom PIPE PHY" on page 82
- "Troubleshooting the Integration of Third-party PHYs" on page 83

The PIPE signal descriptions are in the "Signal Descriptions" chapter of the *Reference Manual*. Synopsys supplies the wrapper files to complete the interconnection between your third-party PHY and the controller.

3.1.5.1 Integrating an Example PIPE PHY

Set `PHY_TYPE = Example PIPE PHY (8)`. You do not need to perform any integration tasks before proceeding with the simulation activity in coreConsultant. The PHY wrapper is already automatically connected *externally* to the controller through the PIPE interface. The example PHY is located at `<workspace>/src/Phy/custom/phy_generic.v`. This PHY is not for implementation/synthesis in ASIC.

3.1.5.2 Integrating a Custom PIPE PHY

Note Use this flow when you are integrating a third-party PHY that is not in the list of supported PHYs.

Set `PHY_TYPE = Custom PHY (7)`. You must manually instantiate your PHY wrapper into a Synopsys-supplied template at `<workspace>/src/Phy/custom/phy_instance.vh`, before proceeding with the simulation activity in coreConsultant. These are the steps required:

1. Run coreConsultant and select Custom PHY.
2. Set the PHY options on the "Advanced PHY Configuration" page and configure the other controller options (including L1 Exit latencies) and click **APPLY** button to generate the RTL.
3. Write out your configuration batch script (`temp.config`).

   Use the coreConsultant **File->Write Batch Script** menu to specify the file name temp.config and select the "Write default parameter values" option.
4. From the temp.config file, extract the parameters that are indicated in `<workspace>/src/Phy/custom/custom.phyconfig`
5. Write these parameters into a file (`custom.config`) for later reference.
6. Copy the files from the `<workspace>/src/Phy/custom` directory to a safe location outside the workspace area so that these files can be referenced in later workspaces as needed.

```
% cd <workspace>/src/Phy
% cp -r custom <safe_area_not_in_workspace>
% \rm -rf custom
% ln -s <safe_area_not_in_workspace>/custom custom
% cd custom
```

Caution When you change the configuration (by clicking the APPLY button) of the PCIe controller in coreConsultant, the complete workspace is recreated and any files that you have created manually in the workspace are deleted.

7. Instantiate the top-level of your PHY model according to the template provided in the `custom/phy_instance.vh`wrapper file.
8. Edit the `compile.f` file, which is used to compile your PHY model into the coreConsultant simulation environment.

- Add the Verilog compile options for compiling your PHY.

9. If your configuration is Gen3 or higher, you must setup a behavioral model for the RX EQ slice of the PHY that you are simulating. Failing to do so results in equalization tests failures when running the packaged test suite. For more information, see `<workspace>/src/Phy/custom/phy_tb_back-door.v`
10. [Optional] If you have enabled the Advanced Power Management and UPF features `(CX_EN-HANCED_PM_EN =1)`, follow the instructions provided in section "UPF Flow and Methodology" on page 101.

    For more information, see "Advanced Power Management and Power Domain Gating" section in the "Controller Operations" chapter of the *Databook*.
11. At this point, you are ready to simulate with your PHY model. Return to coreConsultant and proceed to the Simulation tab.
12. If you have compile problems, navigate the `<workspace>/sim/<testname>` directory of the failing sim. The compile.log and test.log file contains the errors and warnings. Make the necessary changes to your PHY `compile.f`script and PHY model to resolve the compile problems and rerun the simulation using by typing `./test.startsim` in the test directory.
13. If you have simulation failures, see the VTB documentation available at`<work-space>/doc/vtb/testenv` to enable wave dumping and control verbosity level of debug messages.
14. On subsequent coreConsultant sessions, the following sequence can be used.

    a. Use the `custom.config file` created in the preceding section for subsequent coreConsultant sessions. This can be accomplished by entering the following command at the coreConsultant prompt:

    ```
    source <full path to custom.config file>
    ```

    b. Configure all other controller settings and click the APPLY button to write the configuration into the workspace.

    c. Link your customized "Custom Phy" files with the workspace:

    ```
    % cd <workspace>/src/Phy
    % \rm -rf custom
    % ln -s <safe_area_not_in_workspace>/custom custom
    ```

    d. Return to coreConsultant and continue with the simulation task.

3.1.5.3 Troubleshooting the Integration of Third-party PHYs

This section provides troubleshooting tips if you encounter simulation failures in the process of integrating third-party PHYs.

Fast Link Simulation Mode

Fast simulation mode tuning for Synopsys PHYs is performed automatically by coreConsultant. Fast Link Mode sets all internal timers to Fast Mode for speeding up simulation purposes. It forces the LTSSM training (link initialization) to use shorter timeouts and to link up faster. Fast Link Mode can be activated by setting the `FAST_LINK_MODE` field in the `PORT_LINK_CTRL_OFF` register, or by asserting the `diag_ctrl_bus` input to 3'b100.

PHY models implement a fast simulation mode which might not meet Synopsys IIP and VIP requirements. Large mismatches can cause test failures (including link up timeouts). These failures can be resolved by

appropriately configuring the IIP fast mode and by tweaking VIP timeouts in the PHY `compile.f` file. Synopsys recommends you to follow these steps, in the order in which they are presented, to address such issues.

- **Fast mode scaling factor:** By default, the fast simulation mode is enabled for all tests. In the default fast mode, all LTSSM timeouts are scaled by a factor of 1024 (so that 24 ms -> 24us) and the number of required TSs to exit Polling.Active is reduced from 1024 to 16. The default scaling factor can be changed by using the `DEFAULT_FAST_LINK_SCALING_FACTOR` parameter, or through the `FAST_LINK_SCALING_FACTOR` field of the `TIMER_CTRL_MAX_FUNC_NUM_OFF` register. A value of '1' or higher should be set to completely Dynamic Link Width Change successfully. A value of '2' (corresponding to a scaling factor of 64) works well for most PHY models. When proper scaling factor is selected, all tests running in vDUT mode complete successfully.

If the tests running in VIP mode fail to linkup, even after setting the fast mode scaling factor appropriately, you need to adjust VIP timeouts to match the behavior of the PHY model. It is strongly recommended to fix the timeouts in the following order; "boot up sequence" followed by "rate change".

- **Boot up sequence:** (PCLK generation, common mode detection, receiver detection): If the PHY takes more than ~28us to boot, then the VIP LTSSM might timeout and move from Polling.Active into Polling.Compliance. From that point onwards, the LTSSM in the controller and the LTSSM in the VIP get out of sync, resulting in either a simulation timeout, or an excessively long link up sequence. To fix this issue, increase the VIP timeout in Detect.Quiet to be around the same time as required by your PHY model to detect common mode (first de-assertion of `phy_mac_phystatus`). Add the following to the PHY `compile.f` file:

  ```
  +define+TB_DETECT_QUIET_TIMEOUT=XYZ //(expressed in us)
  ```

  If the VIP times-out in Polling.Active and moves to Polling.Compliance (in any test except compliance tests), then increase the timeout in Polling.Active to be larger than the PHY receiver detection time, as measured from `mac_phy_txdetectrx_compliance` assertion to `phy_mac_phystatus` assertion. It is not possible to directly control the VIP timeout in DetectActive. Add the following to the PHY `compile.f` file:

  ```
  +define+TB_POLL_ACTIVE_TIMEOUT=XYZ //(expressed in us)
  ```

- **Rate change sequence:** If the PHY takes more than 1 us to acknowledge a rate change and assert `phy_mac_phystatus`, then the VIP could timeout in either *Recovert.RcvrLock* or *Recovery.Idle*. To make the two sides of the link behave more symmetrically, extend the amount of time the VIP spends in electrical idle during a rate change. Set this value to be +/-1 us the amount of time required by the PHY PLL to lock.

  ```
  +define+TB_RCVRY_SPEED_EIDLE=XXX //(expressed in us)
  ```

  If this is not enough, it is also possible to modify the timeouts on `Recovery.RcvrLock` and/or `Recovery.Idle` in these states using the following defines:

  ```
  +define+TB_RCVRY_LOCK_TIMEOUT=YYY //(expressed in us)
  +define+TB_RCVRY_IDLE_TIMEOUT=ZZZ //(expressed in us)
  ```

- **Overall linkup sequence:** When tweaking VIP timeouts, it is possible that the overall linkup sequence might exceed the default timeout of 300us. This can be changed through the PHY `compile.f` file using the define:

  ```
  +define+LINKUP_TIMEOUT_VAL= 300000 //(expressed in us)
  ```

Failure of pm*l0s*tests

`*l0s*` tests fail when the PHY is not able to recover symbol lock/block alignment within the window of Fast Training Sequences (FTSes) reception, causing the LTSSM to move to the Recovery state. This leads to

extended exit latency, and sub-optimal performance. Check the following PHY characteristics to prevent the `*l0s*` tests from failing:

- **RxStandby in L0s:** Some PHYs require `mac_phy_rxstandby` to be asserted during Rx.L0s, however some have no such requirement; for such PHYs, the assertion of `mac_phy_rxstandby` increases the lock time and degrades the performance. You can control the behavior of `mac_phy_rxstandby` by using the `CX_RXSTANDBY_CONTROL` configuration parameter.
- **Number of FTSes:** This is the number of fast training sequences required by the PHY to re-establish symbol lock/block alignment after transition to Rx.L0s. The higher the number, the wider is the time window. The controller advertises this number to the link partner during link training. You can change the number of FTSes advertised by the controller using the parameters `CX_NFTS, CX_COMM_NFTS`, and `DEFAULT_GEN2_N_FTS`.

Contacting Synopsys Support

Open a support ticket and provide the following information:

- A copy of`<workspace>/src/Phy/<architecture>/compile.f`or a list of macros defined to support fast simulation mode.
- The batch file used to create the workspace.
- A copy of the parameters files located at `<workspace>/src/DWC_pcie_ctl_cc_constants.v`
- Log and wave file of the failing test.

Note If multiple tests are failing and `tl_traffic*` is among them, it is recommended to submit data for this specific scenario. `tl_traffic`is a basic scenario with no special handshake or error injection sequences, so it is easier to troubleshoot when fundamental issues occur.

3.1.6 Bifurcation Guidelines

This section discusses port bifurcation using your PHY, and the controller. There is no configuration option in the controller that enables bifurcation. Therefore, bifurcation in this discussion is about (1) how the PHY and PCS are connected to the controller, and (2) how they are controlled by your reset logic. Your system architecture, and choice of IP determines if you can construct a bifurcated design or not. For more information on Link width negotiation, see the “Lane Reversal and Broken Lanes” Appendix in the *Databook*. The following topics are discussed:

- “Bifurcation Overview”
- “Bifurcation Implementation Requirements” on page 87
- “Examples of Connecting a PCS to the Controller for Bifurcation” on page 88
- “Bifurcation FAQ” on page 91

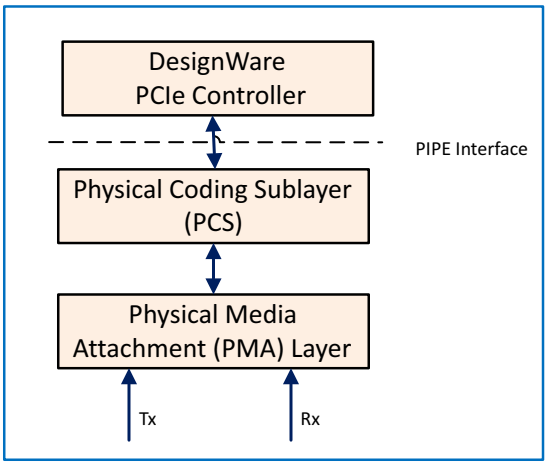
Caution

- This section only gives general guidelines, and pointers to aspects of bifurcation that you must be aware of.
- Synopsys accepts no responsibility for any PCIe IP controller and PHY bifurcation decisions made by you as a consequence of accessing this document.
- Bifurcation is not verified as part of the controller or PHY IP offering. The bifurcation example in this document is not simulated or verified by Synopsys.
- The Clock and Reset module at `<workspace>/examples/DWC_pcie_clkrst.v`, does not support bifurcation. You must modify it to support bifurcation.

3.1.6.1 Bifurcation Overview

The Physical Coding Sublayer (PCS) is the lowest level of the "logical sub-block" of the physical layer (as defined in the PCI Express base specification). The Physical Media Attachment (PMA) is the analog hard macro. Figure 3-14shows the location of the PCS and PMA in relation to the PIPE Interface and the controller.

Figure 3-14 Location of PCS and PMA in Relation to PIPE Interface and the Controller



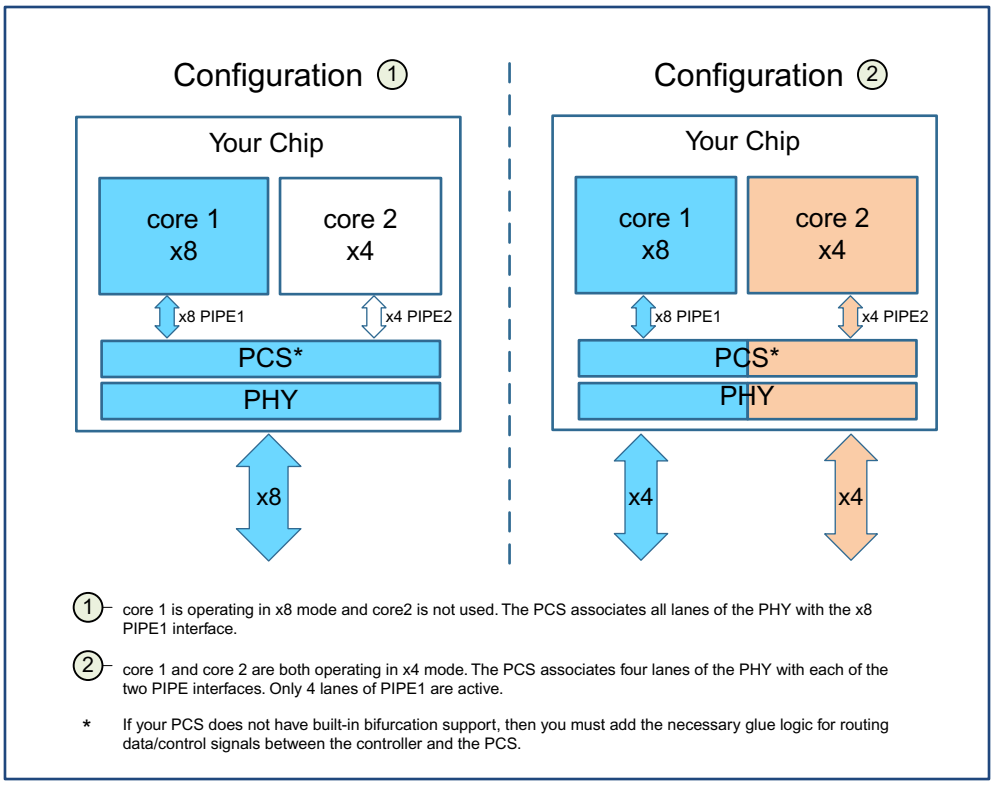
Note In this section, unless specified otherwise, the term PHY is used to indicate the PMA.

Consider a chip that you want to use in more than one system configuration, as in Figure 3-15, where your chip supports:

- One single x8 link connected to a single device, or
- Two x4 links connected to two separate devices

You use bifurcation to share the PHY and PCS.

Figure 3-15 Bifurcation Example



3.1.6.2 Bifurcation Implementation Requirements

To implement bifurcation, you need multiple cores, and a PHY+PCS[1] infrastructure that supports bifurcation. Some of the requirements of a PHY+PCS infrastructure that supports bifurcation are:

- PCS has a Per-Link Lane Assignment, Powerdown, and Reset architecture.

  For the example in Figure 3-15, the PCS must be designed to be able to connect to the two PIPE interfaces (one per bifurcated link), say, PIPE1 and PIPE2. When the operating mode is “Configuration 1”, it must assign all eight PCS lanes to PIPE1, and no lanes to PIPE2. When the operating mode is “Configuration 2”, it must assign four PCS lanes to PIPE1, and four lanes to PIPE2.

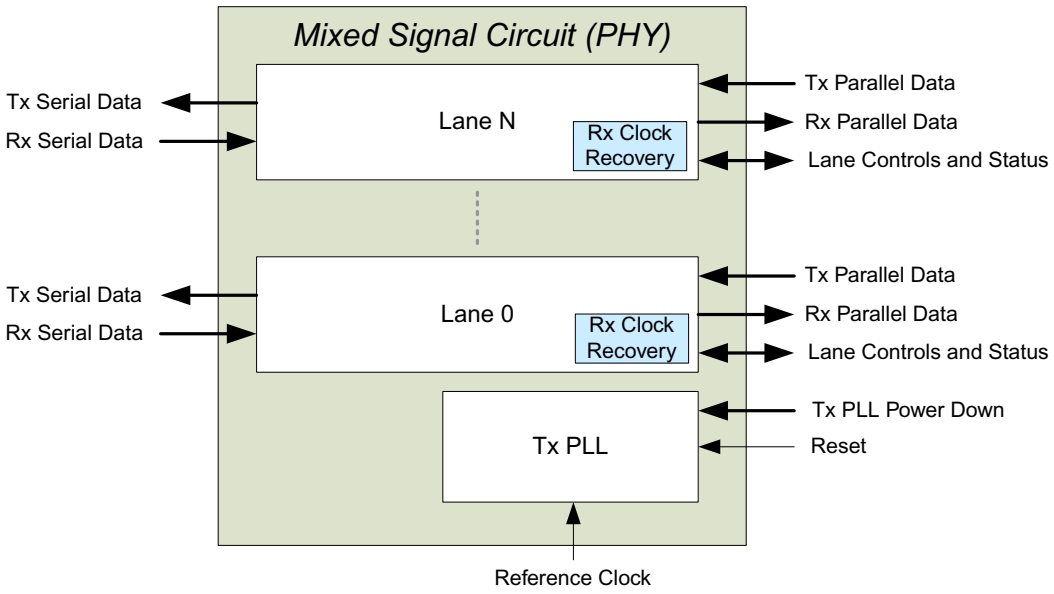
  In addition, the PCS must have independent reset and power control over each link. This allows the cores to independently power down (or reset) each link. For example, if any of the links experiences a fatal error, and goes down, you need to reset the corresponding PCS link section without compromising the PCS section assigned to the other link that is still working.

- PHY has a Per-Link Lane Assignment, Powerdown, and Reset architecture.

  Many PHYs have an architecture similar to that in Figure 3-16. There is a per-lane RX clock recovery block[2], but only a single TX multiplying PLL for all lanes.

1. For the remainder of this section, PCS refers to a combination of PCSs and glue logic.
2. PLL or DLL.

Figure 3-16 Block Diagram of Typical Multilane PLL



If one of the PHY links experiences a fatal error, goes down[1], and if you need to reset it without compromising the other good links that are still working, then this type of architecture presents a challenge in the design of your reset logic. This is because you need to keep the Tx PLL active when there is at least one link working. A possible approach for recovery is:

1. Place the PCS and PHY lanes, of the bifurcated link that has gone down, into the P1 power state. Use the powerdown signals of the corresponding PCS PIPE interface.
2. Wait until that PCS and PHY lanes are in P1. Monitor the status signals of the corresponding PCS PIPE interface.
3. Reset the relevant controller (by asserting and releasing its `core_rst_n` input).
4. Do not reset the PCS or PHY.

If this method does not work in your application, then you must:

- Use multiple[2] smaller PHYs if possible (Each PHY is completely independent). For the example in Figure 3-15, use two 4x PHYs, or
- Reset the complete PHY (after placing all bifurcated links into the P1 power state), or
- Use a PLL with a true per-lane architecture.

3.1.6.3 Examples of Connecting a PCS to the Controller for Bifurcation

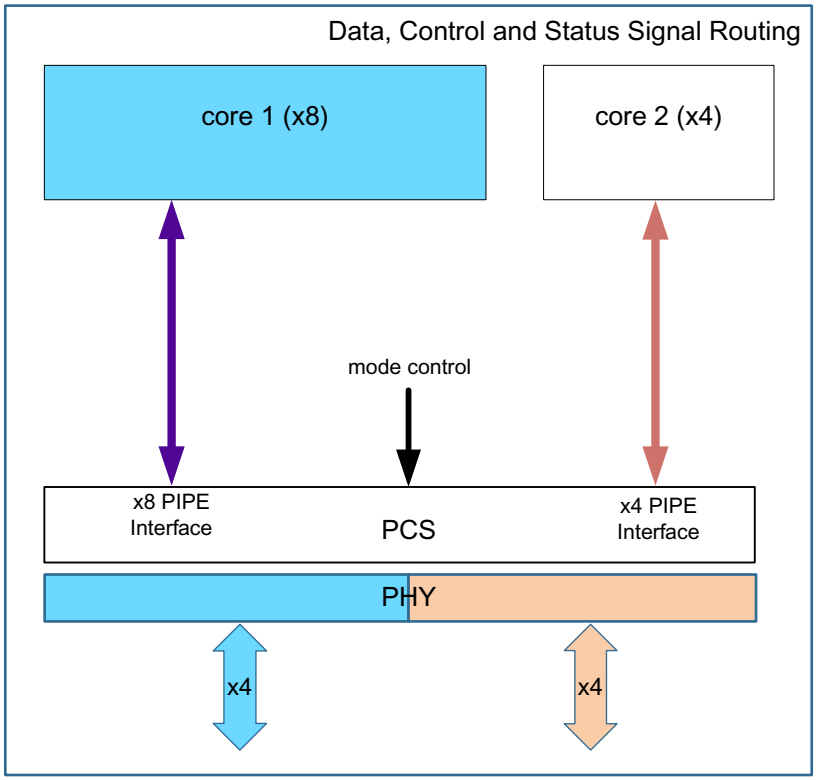
Consider how to configure a generic PCS for the example in Figure 3-15. There are at least four examples, depending on the PCS architecture. You must consult with your PCS vendor, and carefully read their documentation, so that you can select and configure the correct PCS for your bifurcated design.

1. The LTSSM in the controller asserts the link_req_rst_not output
2. This is called aggregation.

Example 1

A single PCS that supports bifurcation and that has all the necessary bifurcation glue logic and control inside it. It would have two multilane PIPE interfaces, and a mode control signal to switch between supporting a single x8 link, and two x4 links. For more information, see Figure 3-17.

Figure 3-17 Suggested PCS <-> Controller Signal Routing for Bifurcation Example 1



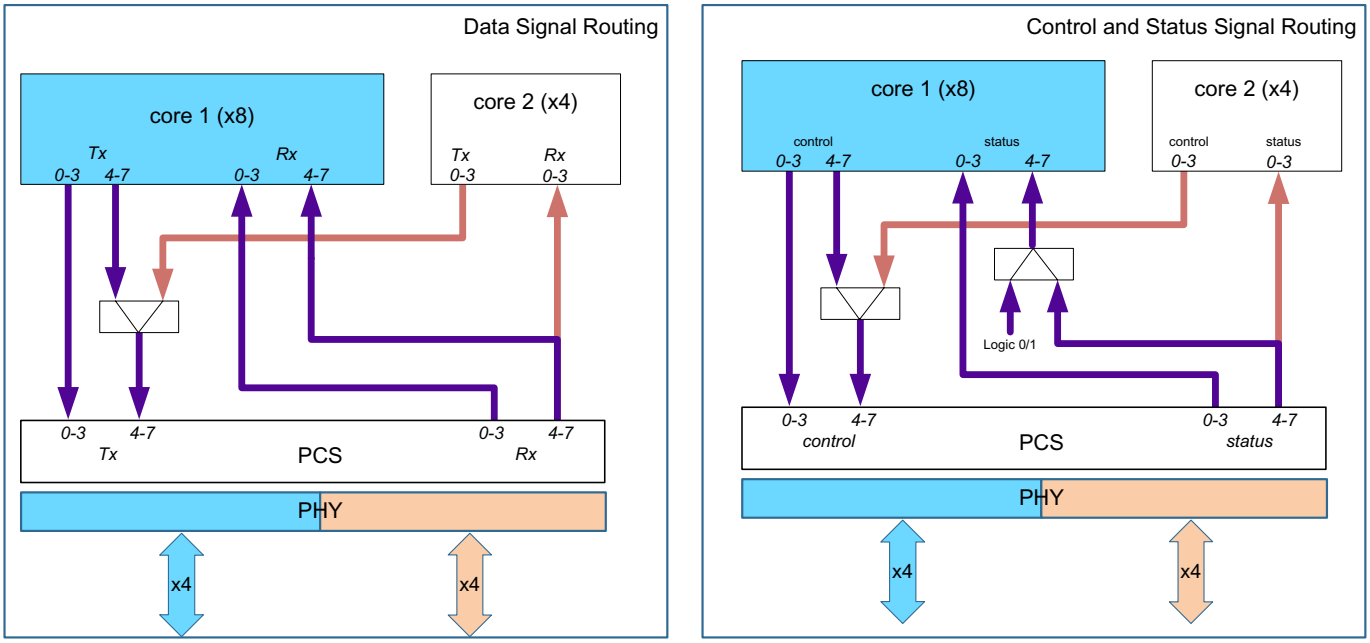
For examples 2, 3, and 4, you must add the necessary glue logic for routing data/control signals between the cores and the PCS. Attempting to control multiple PHY instances with a single PCS or attempting to connect multiple PCS instances to a single PHY is not supported without additional glue logic that must be developed for the specific application. Synopsys does not provide such glue logic.

The controller PIPE interface concatenates the per-lane signals to form a bus, with the low-order bits corresponding to lane0. Your glue logic must correctly split and splice the correct lane signals. Majority of the PIPE interface signals are per-lane. The notable exceptions are the reset and powerdown controls.

Examples 2 and 3

A single PCS that does not support bifurcation, but that has two fully independent four-lane PIPE interfaces, or that has eight fully independent single-lane PIPE interfaces. For more information, see Figure 3-18.

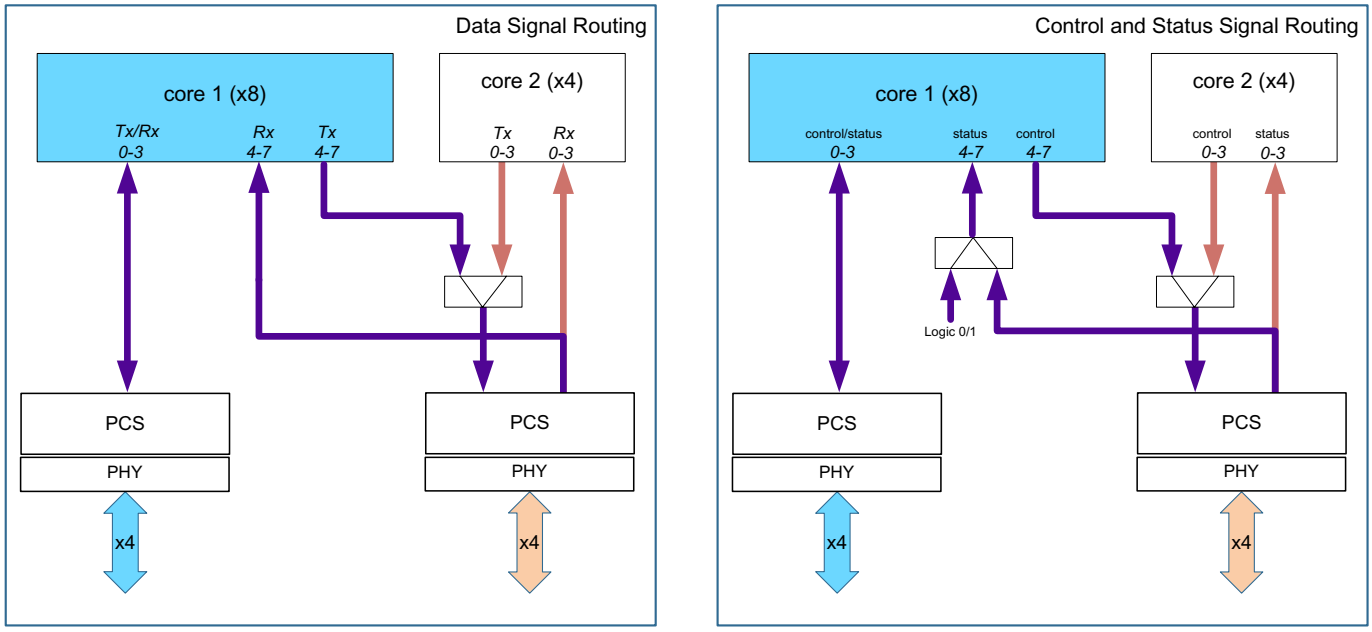
Figure 3-18 Suggested PCS <-> Controller Signal Routing for Bifurcation Examples 2 and 3



Example 4

An individual four-lane PCS for each link. For more information, see Figure 3-19.

Figure 3-19 Suggested PCS<->Controller Signal Routing for Bifurcation Example 4



3.1.6.4 Bifurcation FAQ

Q: *Does Synopsys have a reference design of a bifurcated PCI Express port?*

A: No, bifurcation is not verified as part of the controller or PHY IP offering.

Q: *Apart from link down, are there other scenarios in a bifurcated link where I need to be aware of the state of other links?*

A: Yes, when the links are operating at different speeds, when a link is in loopback mode, or when the links are in different power states.

Q: *Can I directly wire control signals between the PCIe controller and the PHY?*

A: No, you must design a clock, reset, and powerdown controller block in your application logic. Typically, your block monitors mac_phy_powerdown, along with other states and clocks. For more information, see the "Clock Requirements" and "Reset Requirements" sections in the "Architecture" chapter of the *Databook*.

Q: *Can I use the clk_rst module to generate the individual resets for a bifurcated PHY?*

A: No, the clk_rst module is designed for one link only. However, depending on your PHY reset architecture, you might be able to use multiple instances of a clk_rst module to generate the individual per-link resets to the PHY (PCS). You then need to address the issue of having multiple power-on resets targeting the PHY.

Q: *What happens in configuration 2 (Figure 3-15) when one of the controller is entering L1SS (sub-states) while the other controller is in L0?*

A: The handshake from PHY to the controller must stay operational as per the L1SS handshake. The common Tx PLL in the PHY cannot be shut down and the reference clock should not be gated by your clock control circuitry until all the links on the PHY go to L1.1/L1.2. This is similar to the topology given in the L1SS ECN for "Multiple downstream ports with a shared PLL."

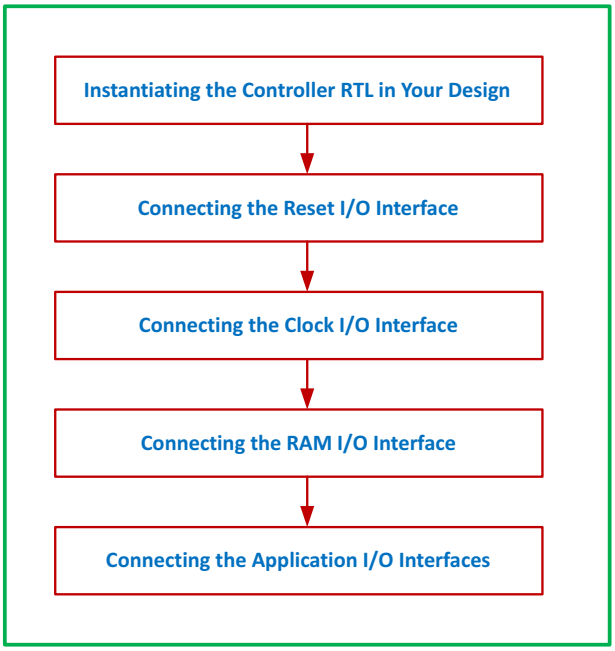
The same guidelines apply for power gating; the PIPE interface timing stays accurate, but the power can only be gated if no other logic needs it. In terms of what glue logic might be applicable (depending on your actual system configuration); an external AND is needed to generate a common CLKREQ#, and an external OR is needed for power gating control of the PHY as documented in the *PHY Databook*.

3.2 Integrating with Your Application RTL

This section describes how to integrate the controller with your SoC or application logic and contains the following topics.

- "Overview"
- "Instantiating the Controller RTL in Your Design" on page 93
- "Connecting the Reset I/O Interface" on page 93
- "Connecting the Clock I/O Interface" on page 95
- "Connecting the RAM I/O Interface" on page 97
- "Connecting the Application I/O Interfaces" on page 97
- "Design Example" on page 97

Figure 3-20 Application RTL Integration Flow

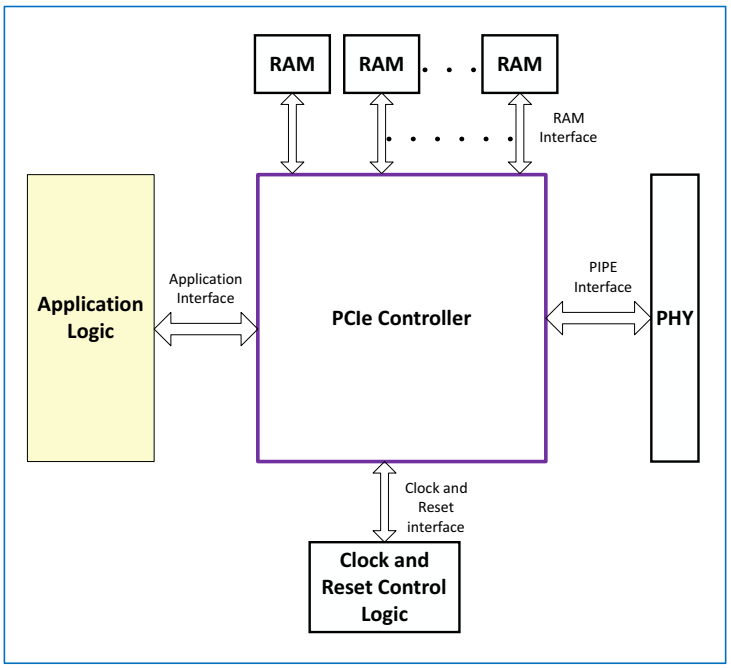


3.2.1 Overview

Input/Output (I/O) interfaces of the controller can be broadly classified into the following categories, based on the components to which they connect to in your design.

- PIPE Interface
- Reset Interfaces
- Clock Interfaces
- RAM Interfaces
- Application Interfaces

Figure 3-21 PCIe Controller Top-Level IO Overview



3.2.2 Instantiating the Controller RTL in Your Design

Follow the procedure outlined in "Configuring the Controller" on page 21 to configure the controller, and the procedure in "Generating a Sample Controller Instantiation" on page 63 to export (extract) the RTL files from the workspace directory.

3.2.3 Connecting the Reset I/O Interface

- You must implement the reset generation logic required to drive all reset inputs of the controller using `DWC_pcie_clkrst_cpcie.v` as a reference.
- Review "Reset Requirements" section in the "Architecture" chapter of the *Databook* to correctly connect and drive all reset related ports of the controller.

Figure 3-22 and Figure 3-23 describe the reset related ports of the controller and their connections with your application.

Figure 3-22 Guidelines for Connecting Reset Signals

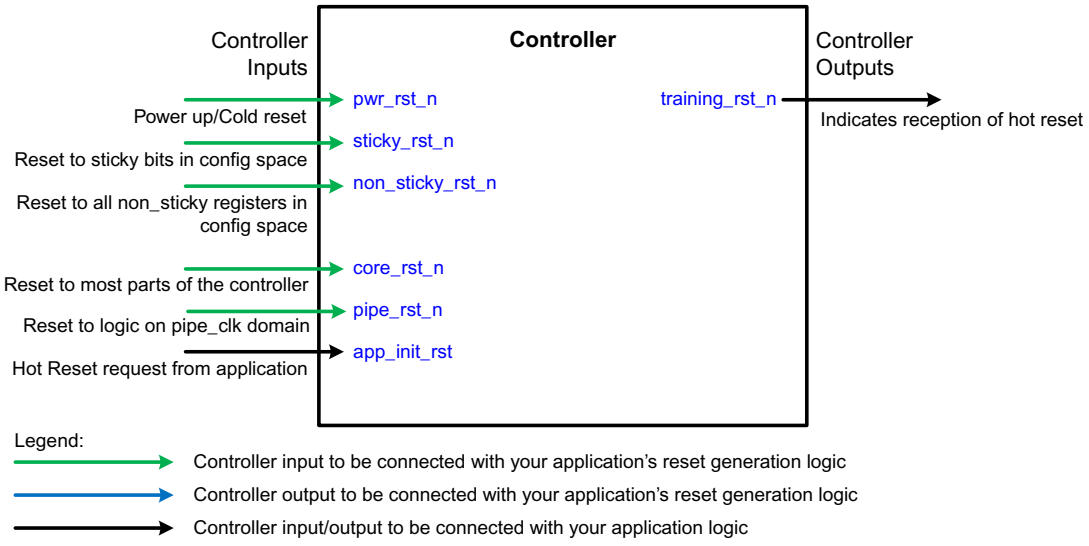
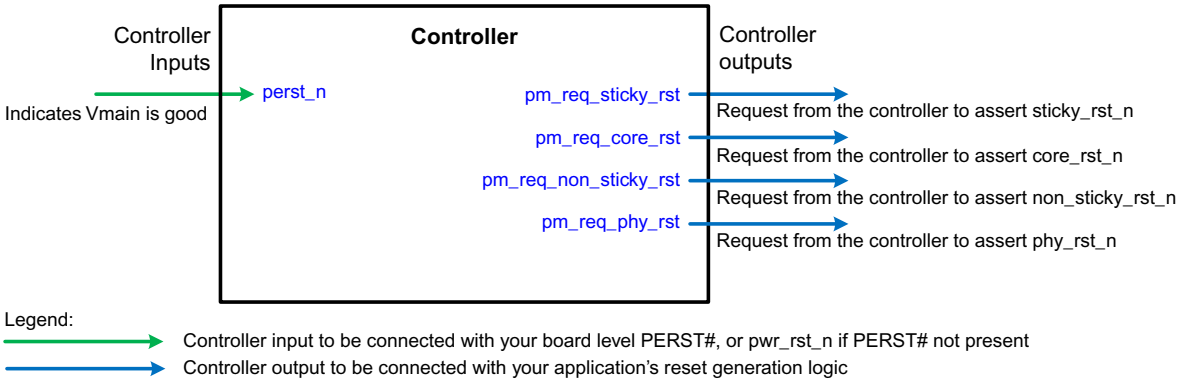


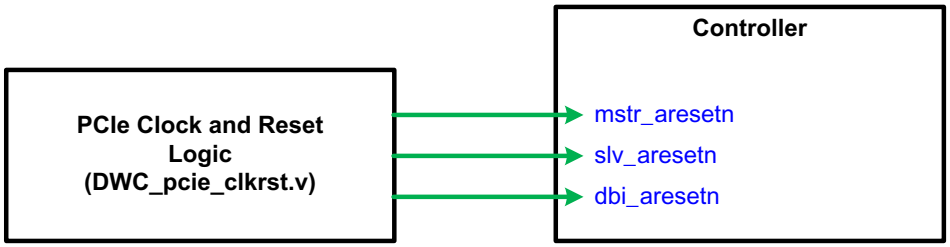
Figure 3-23 Guidelines for Connecting Reset APM Signals



AXI Resets

The resets of the AXI interfaces are driven from the Clock and Reset generator in your SoC. An example of this block is provided in `DWC_pcie_clkrst.v`. AXI resets must be asserted whenever the `core_rst_n` is asserted. For more information, see the example design `DWC_pcie_clkrst.v`.

Figure 3-24 AXI Resets



3.2.4 Connecting the Clock I/O Interface

- You must implement the clock generation logic required to drive all clock inputs of the controller using `DWC_pcie_clkrst_cpcie.v` as a reference.
- Review "Clock Requirements" section in the "Architecture" chapter of the *Databook* to correctly connect and drive all clock related ports of the controller.

Figure 3-25 and Figure 3-26 list the clock related ports of the controller and their connections.

Figure 3-25 Guidelines for Connecting Clock Signals

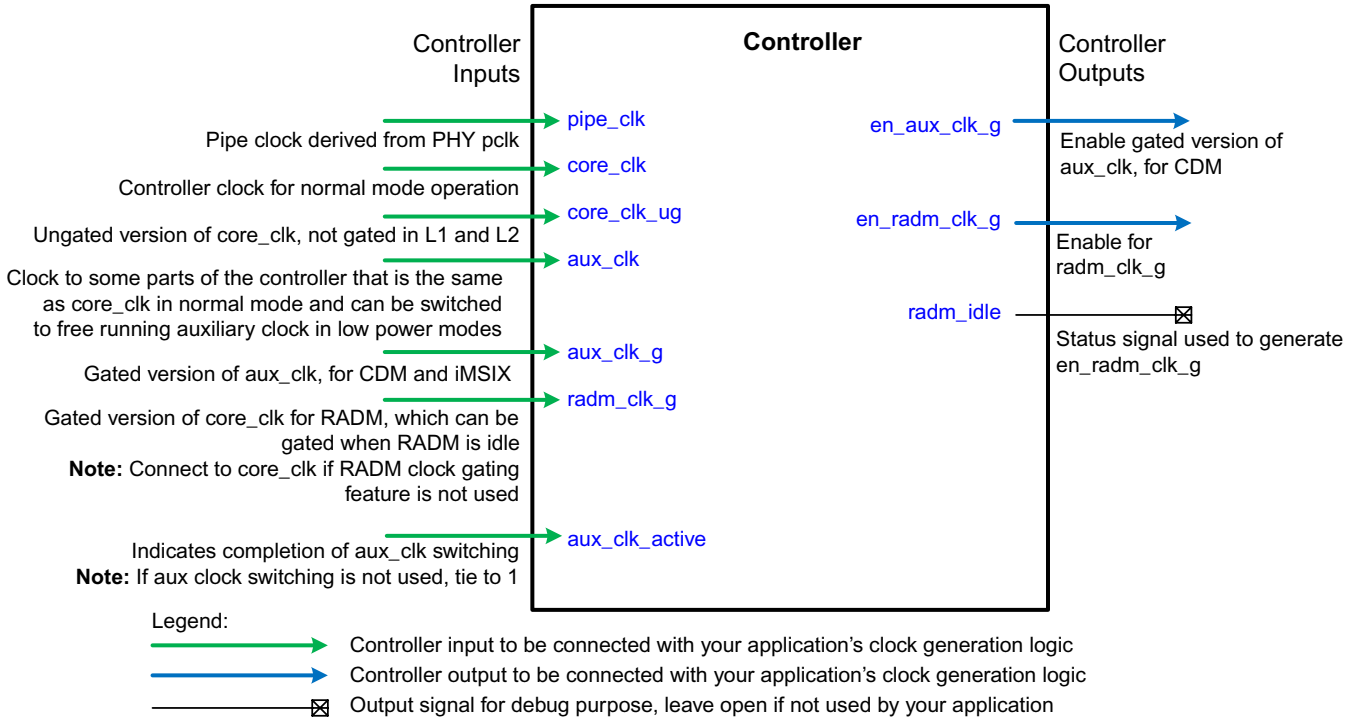
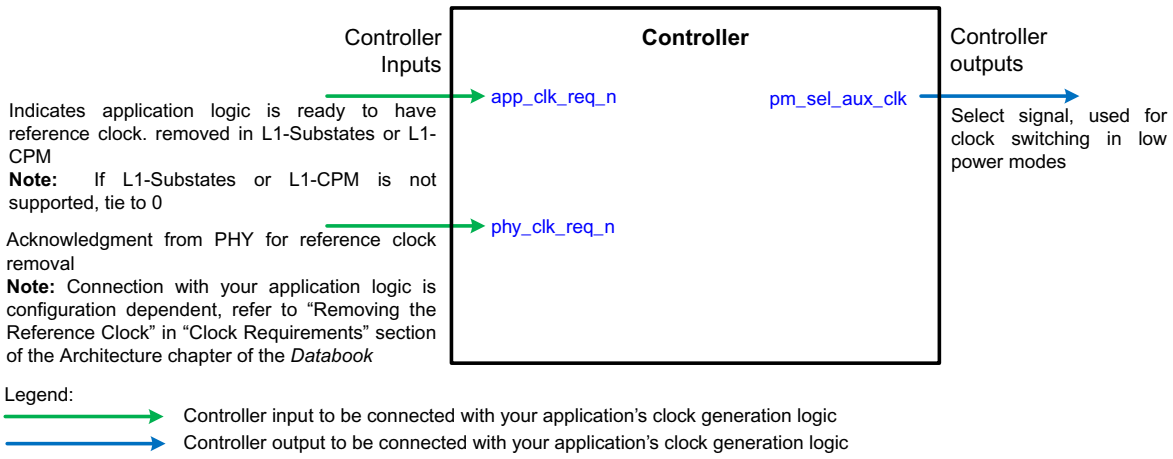


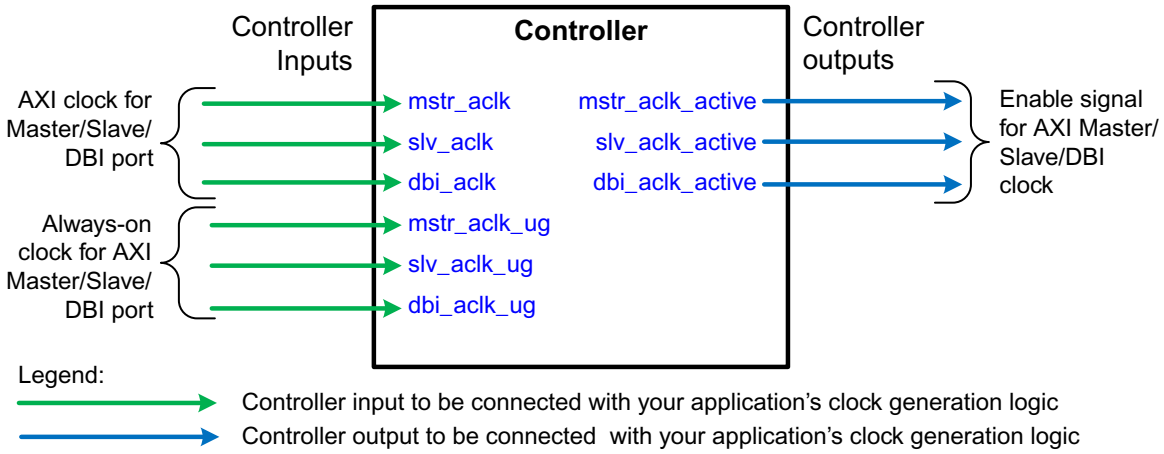
Figure 3-26 Guidelines for Connecting Clock APM Signals



AXI Clocks

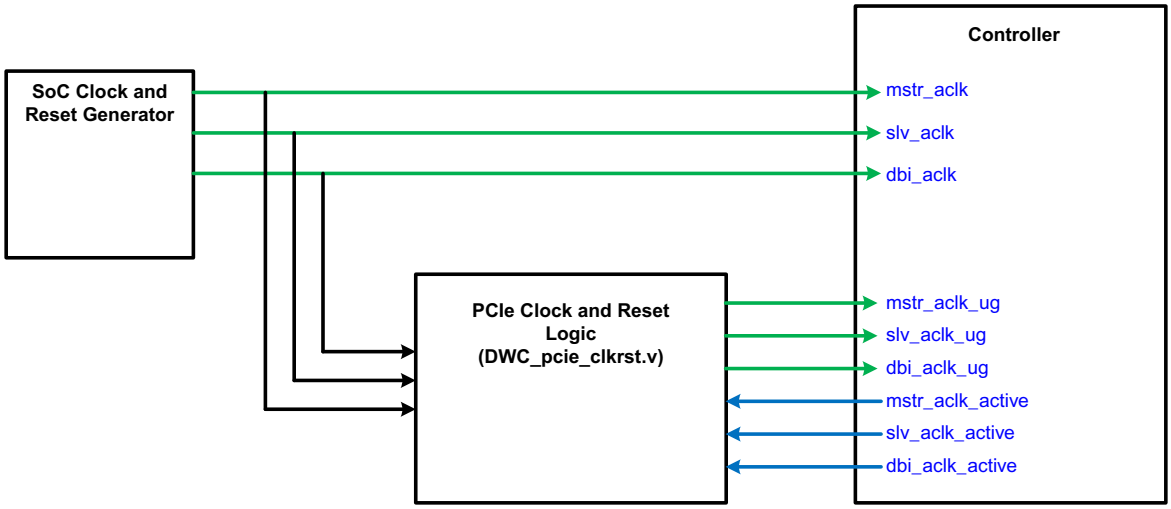
Clocks of the AXI interfaces are driven from the Clock and Reset generator in your SoC. An example of this block is provided in `DWC_pcie_clkrst.v`.

Figure 3-27 Guidelines for Connecting AXI Clock Signals



AXI clocks can be gated during PCIe low power modes, the clock gating signals generated by the controller are connected to the Clock and Reset generator as shown in Figure 3-28.

Figure 3-28 AXI Clock Gating



3.2.5 Connecting the RAM I/O Interface

- Controller uses two-port and one-port RAMs.
- For more information on connecting the PCIe controller RAM signals, see "RAM Requirements" section in the "Architecture" chapter of the *Databook*.
- There is a RAM instantiation template in `<workspace>/examples/pcie_iip_device.sv`.

3.2.6 Connecting the Application I/O Interfaces

For more information, see "Application Interface Integration Guide" on page 241".

3.2.7 Design Example

A structural chip-level design example is provided to facilitate building your PCIe device. For more information, see "Integrating the Controller and VIP using VTB" on page 98.

3.3 Integrating the Controller and VIP using VTB

The controller includes a testbench known as the VTB (Verilog Testbench), which can serve as a reference only for integrating the controller and Synopsys VIP into your system level test environment. The VTB integrates the controller and the VIP, and demonstrates configuration of the environment, link establishment, and running basic transactions for most features. The VTB utilizes VIP to drive stimulus and response to and from the controller. You can access the VTB and Doxygen HTML documentation at `<workspace>/doc/html/vtb/testenv/index.html.`

The VTB contains a set of interface protocol checkers implemented as SystemVerilog Assertions (SVAs). The SVA files are located at `<workspace>/sim/testbench/common/customer_sva.` The protocol checkers can be ported to a SoC level testbench to verify correct integration of the controller in a SoC application. Only checkers that apply to your configuration is included and compiled. More information about VTB SVAs can be found in the "Interface Protocol Checkers (SVA)" section of VTB documentation as shown in Figure 3-29.

Figure 3-29 VTB SVAs

3.3.1 Limitations

The VTB is intended only as an example testbench for this version of Controller it is packaged with. The VTB is not a fully functional verification environment. The following limitations apply to the current release of the VTB:

- Not backward compatible with previous releases
- Not UVM compliant
- No support for big-endian (`AHB_ENDIANNESS`) configurations

Caution The VTB is not designed to be a portable verification environment for SoC level testing. Porting the VTB APIs to use as a verification component in SoC level testing or updating it to use with other controller versions requires significant VTB changes and deep understanding of the VTB and the controller.

3.3.2 Setting Up and Running the VTB

When you have configured your controller in coreConsultant and have initialized the **Simulate** activity, then the VTB files in Table 3-2 are automatically generated in `<workspace>/sim/testbench/vtb`.

Table 3-2 Primary VTB Files

Description	File name	Locations
Testbench top level	vtb.v	sim/testbench/vtb/src/verilog
IIP (DUT, interface drivers, SystemVerilog API hooks)	pcie_iip_chip.v	sim/testbench/vtb/src/verilog
VIP	pcie_vip_chip.v	sim/testbench/vtb/src/verilog
DUT (PCIe controller, RAMs, PHY, Clock and Reset)	pcie_iip_device.sv	<workspace>/examples
Clock & reset design example	DWC_pcie_clkrst.v	<workspace>/examples

To run the VTB, do the following:

1. Setup the VIP. The VTB does not work unless you properly install the correct versions of VIP. For more information, see the "Download and Install Verification IP (VIP)" section in the *Installation Guide*.
2. To launch VTB tests through coreConsultant see "Simulating the Controller" on page 31".

To rerun a test from the Linux command line:

```
% cd <workspace>/sim/up_pl_link_speeds_vtb [for example]
% ./test.startsim
```

4

UPF Flow and Methodology

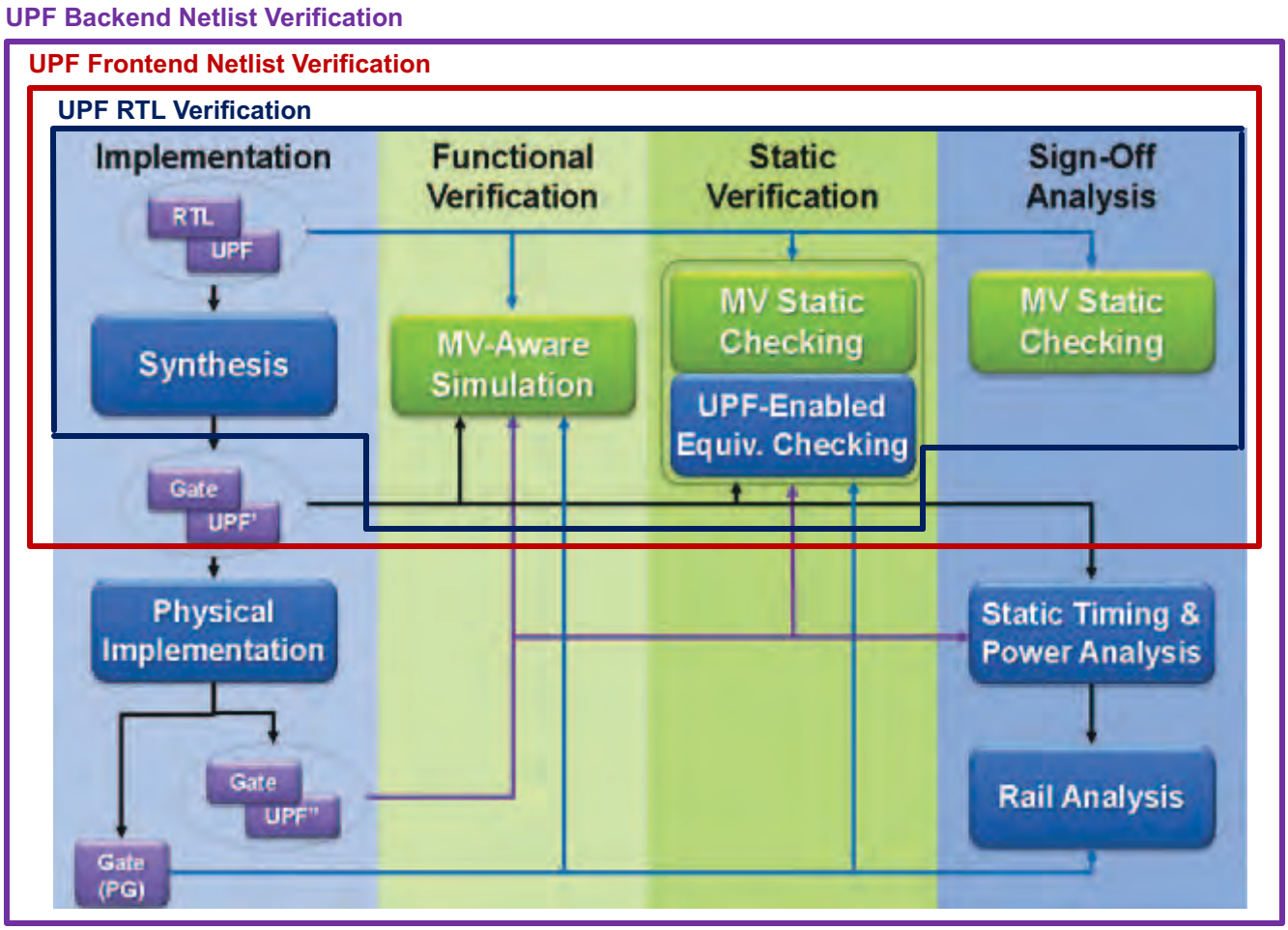
This chapter describes how to use the IEEE 1801 2009 Unified Power Format (UPF) 2.0[1] with DWC_pcie_ctl when 'Advanced Power Management' and 'UPF' features have been enabled. It describes how to setup the flow for simulation and synthesis and provides guidelines for integrating a PHY UPF. The following topics are discussed:

- "Specifying the Power Intent" on page 103
- "Simulation" on page 105
- "Importing PHY UPF" on page 106
- "Synthesis" on page 107
- "Clocks" on page 108

Figure 4-1 shows the Synopsys low-power implementation, functional verification, static verification, and sign-off analysis flow.

1. UPF 2.0 is not backward compatible with UPF 1.0.

Figure 4-1 UPF Flow



4.1 Specifying the Power Intent

The coreConsultant **Specify Power Intent** activity generates configuration specific UPF files. These UPF files are supported by VTB simulations and synthesis.

Figure 4-2 Specify Power Intent Activity: Cell Specification

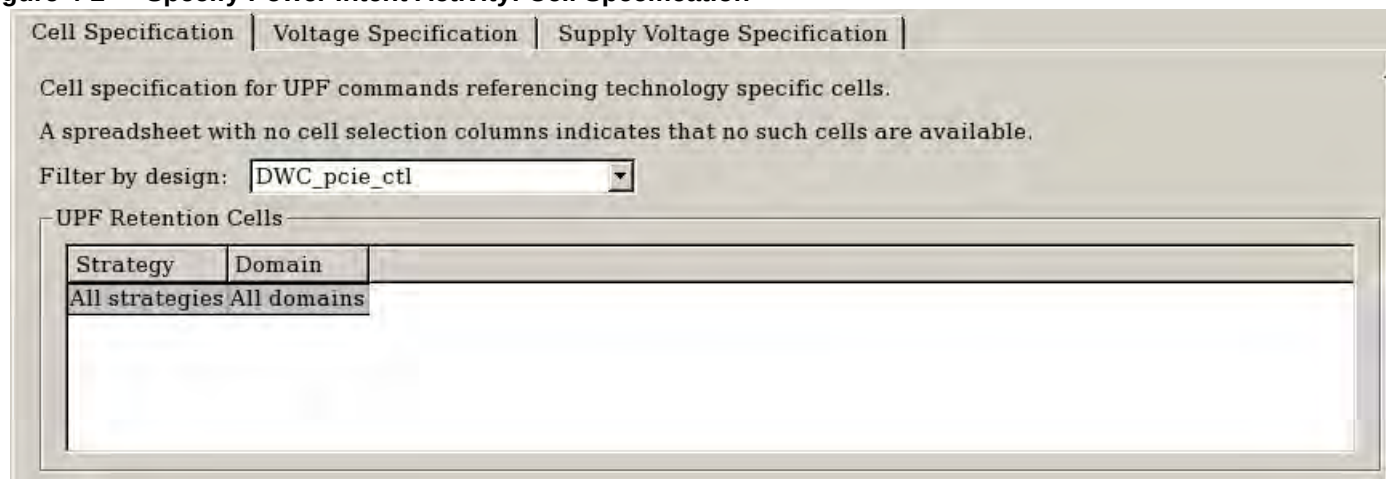
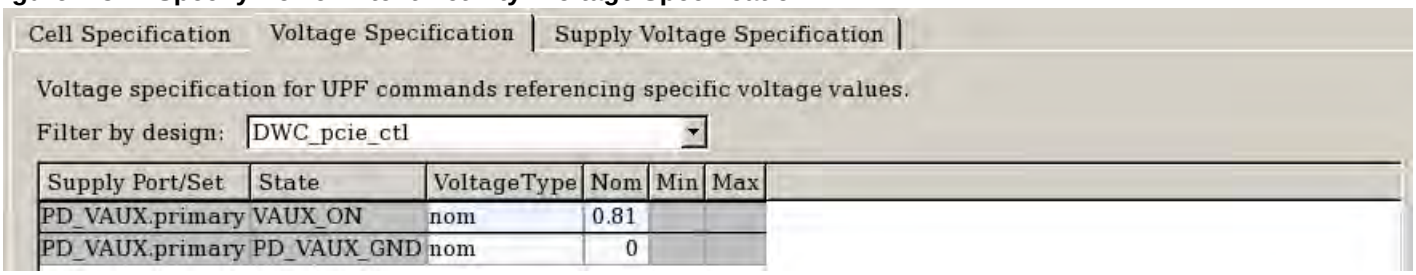


Figure 4-3 Specify Power Intent Activity: Voltage Specification



Specify Power Intent activity is dependent upon **Specify Target Technology** activity. If the **Specify Target Technology** activity is completed prior to **Specify Power Intent** activity, both **Cell Specification** and **Voltage Specification** tabs of **Specify Power Intent** activity are accessible. This enables the customization of the UPF with technology specific aspects such as the cells being used for retention and isolation, the voltages used by each power domain, and so on.

If the **Specify Target Technology** activity is not completed prior to entering **Specify Power Intent** activity, only **Voltage Specification** tab is accessible; this can affect the content of UPF. This flow should be selected if you want to run only simulations.

On completion of the **Specify Power Intent** activity (with or without the technology library setup), the output files are created under `<workspace>/upf`. Figure 4-4 shows the hierarchy of the generated UPF files and Table 4-1 describes the UPF files.

Figure 4-4 UPF File Structure

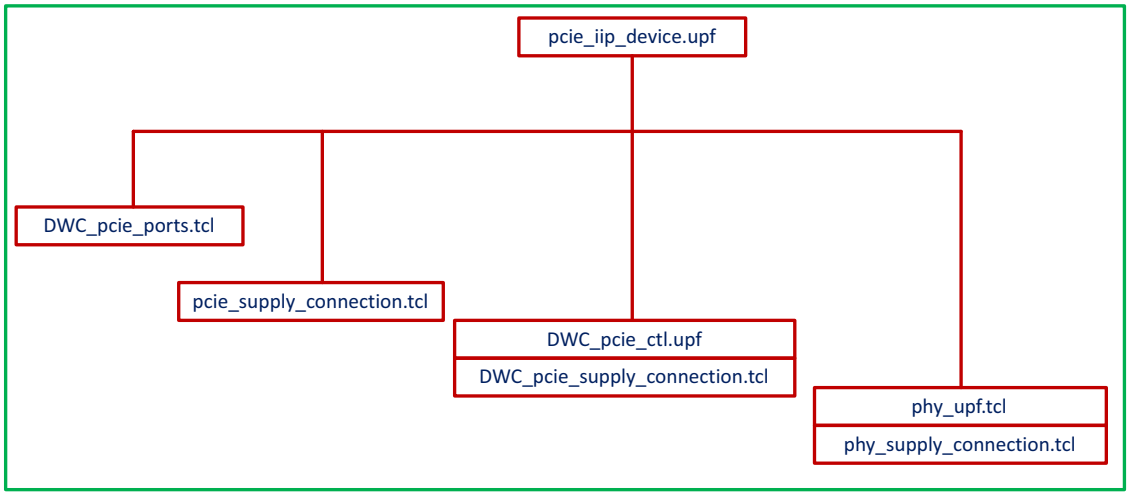


Table 4-1 UPF Flow Files

File	Description
pcie_iip_device.upf	Hierarchical UPF file that loads the controller and PHY UPF files. You can use this as an example/reference for integration of the PCIe IIP into your system level/SoC design.
DWC_pcie_ports.tcl	Configuration-specific UPF port list file. This file is used as a part of the port attribute definition for the ports on the PCIe controller.
pcie_supply_connection.tcl	Contains definitions of top-level supply sets, isolation rules, and top-level power state table.
DWC_pcie_ctl.upf	Configuration-specific UPF file used for simulation and synthesis flow.
DWC_pcie_supply_connection.tcl	Contains supply set associations for the DWC_pcie UPF supplies.
set_voltage.tcl	TCL file to set the voltage values. This file is required for synthesis.
phy_upf.tcl	This is the entry point for loading the PHY UPF. The location of this file is PHY dependent. For more information, see “Importing PHY UPF” on page 106.
phy_supply_connection.tcl	This file is used to connect the power supplies defined in the PHY UPF. The location of this file is PHY dependent. For more information, see “Importing PHY UPF” on page 106.
vcs_power_config.tcl	VCS power aware flow configuration file.

4.2 Simulation

When advanced power management is enabled, and **Specify Power Intent** activity is completed, the VTB simulations automatically load the UPF files and insert virtual isolation cells. On simulation of power gating, the powered-off logic is corrupted by injecting Xs.

Caution Power aware simulations are supported with Synopsys VCS simulator only.

As a part of the simulation, the PHY UPF is also loaded and the corresponding logic is corrupted. For more information, see "Importing PHY UPF" on page 106.

VTB `pm_basic` tests with l2 and l1p2 suffixes demonstrate power gating when the corresponding low power state (L2 or L1.2) is entered. For more information on the VTB Power Management test cases, see VTB documentation at `<workspace>/doc/vtb/testenv`.

4.3 Importing PHY UPF

The simulation flow loads the `pcie_iip_device.upf` for the complete PCIe subsystem, which includes the PCIe IIP controller, a PCIe PHY, the clock/reset block and the required external RAMs.

The simulation flow provides the flexibility to load PHY UPF if the PHY vendor provides it. The entry point for loading the PHY UPF into the PCIe subsystem is `phy_upf.tcl` file. This file is located at `<workspace>/src/Phy/<phy used>/phy_upf.tcl`. For Synopsys PHYs supporting power gating, this file is pre-generated and can be used as is.

For integration of third party PHYs the file `<workspace>/src/Phy/custom/phy_upf.tcl` can be used as reference. The `load_upf` command in the file `phy_upf.tcl` brings the PHY UPF within the `pcie_iip_device` scope. If required environment variables corresponding to PHY UPF can be set in this file.

Note: Review the file `<workspace>/src/Phy/<phy used>/phy_supply_connection.tcl` when loading PHY UPF. For Synopsys PHYs supporting power gating, this file is pre-generated and can be used as is.

For third party PHYs, the file `<workspace>/src/Phy/custom/phy_supply_connection.tcl` can be used as a reference to map the PHY's power supply connections to the supply nets used at the subsystem level (`pcie_iip_device.upf`).

Caution

The `phy_supply_connection.tcl` is for simulation purposes only. If you intend to use this file for implementation purposes, you need to review the connectivity in this file to match the packaging requirements of the specific PHY package in use and the power requirements of your SoC.

4.4 Synthesis

To enable the synthesis flow using UPF, perform the following steps in coreConsultant:

- In the **Perform ASIC Synthesis** activity, select Strategy: Design_Compiler_Reference_Methodology.
- In the Power tab, select Enable UPF Flow.

  The following UPF file is loaded by default: `<workspace>/upf/DWC_pcie_ctl.upf`
- In the power constraints file field, set the voltage values by loading the power constraint file `<workspace>/upf/set_voltage.tcl`

Table 4-2 describes the files that are generated for synthesis flow using UPF.

Table 4-2 Synthesis Flow Files

File	Location	Description
DWC_pcie_ctl.upf	<config>/syn/final/db	UPF file generated by Design Compiler. This UPF must accompany the netlist in any downstream tool which consumes UPF.
DWC_pcie_ctl_power_domains.rpt	<config>/syn/final/report	Contains a list of the power domains.
DWC_pcie_ctl_isolation_cell.rpt	<config>/syn/final/report	Contains a list of signals having isolation cells.
DWC_pcie_ctl_retention_cell.rpt	<config>/syn/final/report	Contains a list of retention cells in the design.

4.5 Clocks

During low-power mode, power management controller (PMC) module clock `aux_clk` should remain active. The reset input to PMC module `pwr_rst_n` must be always on and inactive during low-power modes.

A clock-gated version of the `aux_clk` called `aux_clk_g` clocks the CDM and LBC modules. This clock does not need to be always on. It is gated off during power down and enabled only when access to the CDM is required.

When the `aux_clk` clock comes from an off domain to always-on domain, the always-on clock buffers should buffer the clock network. You can do this by using the following methods:

- When the always-on clock buffers are not available in the library, you can use normal buffers. These buffers should be placed in such a way that their power signal is always on
- When the always-on clock buffers are available in the library, you can place it as normal cells. The always-on clock buffers have the following dual-power signals:
  - A signal connected to always-on power supply.
  - A signal connected to the power rail of the off domain.

  You need to create few power straps of always-on power supply in the off domain to connect to the dual-power clock buffers present in the off domain.

  Alternatively, you can place top-level clock port in the always-on domain so that the always-on buffers are not required.

5

Programming

The following sections provide information on programming examples for specific features of the controller.

- iATU "Outbound Programming Example" section in the "Controller Operations" chapter of the *Databook*.
- iATU "Inbound Programming Examples" section in the "Controller Operations" chapter of the *Databook*.
- "Programming Examples" section in the "DMA" chapter of the *Databook*.
- "SR-IOV Usage and Programming Guide" appendix of the *Databook*.
- "Usage Example" section in the "Gen3 Equalization Details and Examples" appendix of the *Databook*.

The following sections provide information on controller initialization and register access.

- "Initialization" section in the "Controller Operations" chapter of the *Databook*.
- "Register Default Values" section in the "Registers" chapter of the *Reference Manual*.
- "Writing to Read-Only Registers" section in the "Advanced LBC and DBI Usage" chapter of the *Databook*.
- "Port Logic (Synopsys Specific) Registers" section in the "Registers" chapter of the *Reference Manual*.

6

FPGA-Based Prototyping

This chapter highlights the challenges when prototyping the DesignWare Cores IP controllers on an FPGA and the solutions to achieve successful FPGA-Based prototyping. The following topics are discussed:

- "FPGA Prototyping Overview" on page 112
- "FPGA Prototyping Using coreConsultant" on page 115
- "FPGA Debug Guide" on page 122

6.1 FPGA Prototyping Overview

The following topics are discussed:

- "Guidance for Successful FPGA-Based Prototyping"
- "DWC IP coreConsultant FPGA Versus DW IP Prototyping Kits" on page 112
- "FPGA Implementation Flow" on page 113
- "Unlocking coreConsultant GUI FPGA Activity" on page 114

6.1.1 Guidance for Successful FPGA-Based Prototyping

The DesignWare Cores (DWC) IP controllers are optimized for ASIC applications, with power and area as a primary goal, which requires expertise for integration, bring up, and software development. Thus, be aware that FPGA prototyping is NOT a straightforward activity and has many challenges presented in Table 6-1. Use this table to check if you have the necessary resources from Synopsys or from your company to be successful.

Table 6-1 Prototyping Challenges

Type	Description
DWC IP Configuration and FPGA Timing Closure	■ The DWC IP controllers are optimized for ASIC applications. The ASIC technologies allow reaching much higher operating frequencies compared to FPGAs. Some DWC PCIe IP features may have to be disabled in the IP configuration, and some performance targets may not be achieved. These decisions require protocol-specific knowledge, IP knowledge, and expertise with using synthesis and place and route tools to get the best performance out of the FPGA. ■ Some DWC IP configuration examples are provided to help you get started, but an exact ASIC configuration may not be possible to achieve or may require great effort to reach optimal FPGA performance. ■ If you need help with the IP configuration and reaching FPGA timing closure, contact Synopsys Customer Support.
Integration in FPGA SoC Design	■ FPGAs have dedicated resources like PLLs, RAMs, and PHYs which require some design effort when integrating DesignWare IP controllers on an FPGA platform. ■ The DWC IP controller does not include any FPGA reference design as it is specific to the target prototyping platform and FPGA. ■ FPGA reference design is provided as part of the DesignWare IP Prototyping Kits.
Prototype Bring-Up	■ The bring-up phase is usually critical, time consuming, and requires expertise in the protocols involved. The cause of an interface is not from sending or receiving data, but can originate many places, such as software, clocks, resets, PHY interface, the DesignWare IP integration with the system bus, the DesignWare IP configuration, and so on. ■ An FPGA Prototype Debug guidelines are provided to help you with some basic checks, but the effort can be quite considerable.
Drivers and Tools	■ Most of the DWC IP controllers have Linux Kernel drivers available in the Linux Kernel (https://git.kernel.org). They are a starting point, but do not include all the features the DesignWare IP supports. You need to add the necessary features required for your application. ■ You can download the PCIe Software sample at https://www.synopsys.com/dw/ipdir.php?c=dwc_pcie_controller_sw_samples.

6.1.2 DWC IP coreConsultant FPGA Versus DW IP Prototyping Kits

Synopsys offers two different approaches to the DWC IP controller FPGA prototyping:

- FPGA Prototyping activity available in the DesignWare IP Controller coreConsultant configuration tool, which is part of the DesignWare IP Controller itself.
- DesignWare IP Prototyping Kit product.

  This is an additional product, NOT included in the DesignWare IP Controller deliverables.

Table 6-2 shows the differences of the two approaches. More information about the DesignWare IP Prototyping Kits can be found at the following location:
https://www.synopsys.com/designware-ip/ip-accelerated/ip-prototyping-kits.html

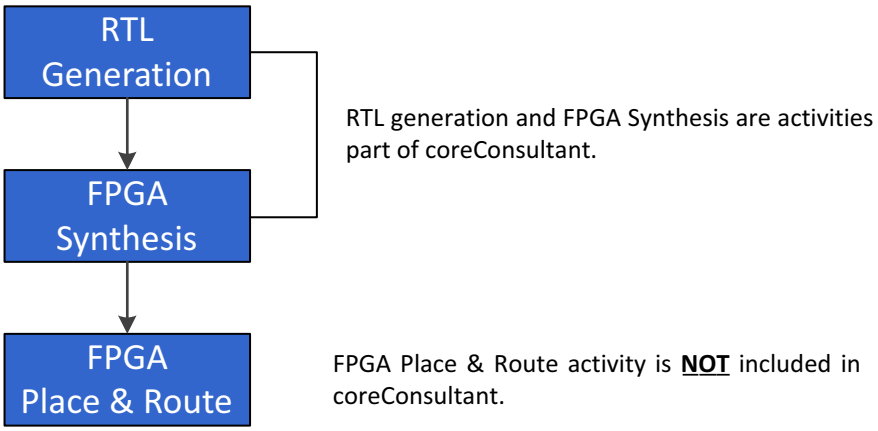
Table 6-2 DWC IP coreConsultant FPGA Versus DW IP Prototyping Kits

Type	DWC IP coreConsultant FPGA Activity	DW IP Prototyping Kit
Reference Design with PHY and Application Logic Example	No	Yes
DWC IP Controller Configuration	User-defined	Pre-defined to the configurations tested on the target prototyping platform
Synthesis Scripts	Yes (Synopsys Synplify)	Yes (Synopsys ProtoCompiler)
Place & Route Scripts	No	Yes (Xilinx VIVADO)
Target Platform	None (Any FPGA device can be selected)	Synopsys HAPS-80 and HAPS-DX7 platforms
Software	No	Yes (Linux Kernel drivers, application examples and optionally real-time monitoring/observation tool)
Out-of-the-Box Working Hardware Setup	No	Yes (only for HAPS-DX7 based products)

6.1.3 FPGA Implementation Flow

The coreConsultant tool generates the RTL source of the DWC IP controller, as well as performs synthesis using the Synopsys Synplify tool, however it does not include Place & Route for the target FPGA. The FPGA Place & Route is an additional step that needs to be performed outside of coreConsultant with a third-party EDA tool.

Figure 6-1 FPGA Implementation Flow



6.1.4 Unlocking coreConsultant GUI FPGA Activity

The FPGA activity in the coreConsultant GUI is locked by default. To unlock this activity, contact Synopsys Customer Support. The Customer support provides you a key to unlock the FPGA activity. The key is only valid for one controller version. A new key must be requested for the next version of the DWC IP controller.

The motivation behind this step is for Synopsys Customer Support to understand your FPGA prototyping goals to recommend the best prototyping approach and which DWC IP controller configuration to implement.

The limitations change over time as new releases of the DWC IP controller EDA tools, and FPGA platforms become available.

6.2 FPGA Prototyping Using coreConsultant

This section provides an FPGA RTL configuration guide for the DWC PCI Express Controller (DM, EP, and RP). It also provides configuration examples that have already been tested and also provides details of how to run the FPGA Synthesis activity in coreConsultant.

The following topics are discussed in this chapter:

- "FPGA RTL Configuration Guide"
- "Tested Configuration Examples" on page 116
- "Synthesizing the Controller for an FPGA" on page 118

6.2.1 FPGA RTL Configuration Guide

Note It is recommended to contact Synopsys Customer Support as soon as your prototyping goals are defined, so that the unlock key is provided before you start your prototyping activity.

The configuration parameter settings and feature restrictions in Table 6-3 apply for an FPGA. You must configure the controller according to these restrictions before starting the synthesis task at "Formal Equivalence Checking using Formality" on page 45. You could use a PLL/DPLL instead of the clock divider in the `DWC_pcie_clkrst.v` module (see the "Clock Requirements" section of the "Architecture" chapter of the *Databook*). This eases balancing of the `core_clk` and `pipe_clk` clock trees as the PLL/DPLL can eliminate the divider insertion delay. For more information on configuration parameter settings to improve timing, see "Configuration Parameter Settings to Improve Timing" on page 165. For more information on resource usage, see the "Area and Power Estimates" appendix in the *Databook*.

Table 6-3 FPGA Related Parameter Settings and Restrictions

Configuration Parameter	Required Values	Note
FPGA	1	
PHY_TYPE	7 (Custom PHY)	
CX_FREQ	<= 125 MHz	
CX_TECHNOLOGY	0 (SLOW)	This is used to enable additional pipeline stages internal to the controller to trade off latency and gates for ease of timing closure.
CX_RAM_TYPE	2 (FPGA RAM)	FPGA synthesis tool should infer the appropriate block RAMs
FREQ_STEP_CORE_CLK_ENABLE[a]	1	This setting is optional. Setting it moves most of the logic in the freq_step block from the pipe_clk domain to the core_clk domain at the expense of logic duplication. This facilitates meeting timing with FPGA.
When You Are Using the Xilinx GTX/GTH Transceiver		

Configuration Parameter	Required Values	Note
PHY_TYPE	7	You can use either Synopsys PHY daughter cards or the FPGA embedded Xilinx PHY. If using FPGA embedded Xilinx PHY, the following features are not supported: PCIe beacon signal Power management Tx Electrical Idle Note: Xilinx Ultrascale PHY supports Gen1/2/3 (8Gbps max) Xilinx Ultrascale+ PHY supports Gen1/2/3/4 (16Gbps max) Synopsys DesignWare PHY Testchip boards support Gen1/2/3/4 and CCIX (25Gbps max)
CX_MAC_SMODE_GEN1	g1_2s (125Mhz)	Controller Gen1 Freq/Width
CX_MAC_SMODE_GEN2	g2_4s (125Mhz)	Controller Gen2 Freq/Width
CX_MAC_SMODE_GEN3	3_8s (125Mhz)	Controller Gen3 Freq/Width
CX_PHY_SMODE_GEN1	g1_2s (125Mhz)	PHY Gen1 Freq/Width
CX_PHY_SMODE_GEN2	g2_2s (250Mhz)	PHY Gen2 Freq/Width
CX_PHY_SMODE_GEN3	g3_4s (250Mhz)	PHY Gen3 Freq/Width

a. This parameter is not visible in the coreConsultant GUI but can be set through the coreConsultant command line (set_configuration_parameter FREQ_STEP_CORE_CLK_ENABLE 1), or in your configuration batch script file.

6.2.2 Tested Configuration Examples

Table 6-4 lists the tested configurations on a hardware platform. Not all the configurations meet timing, but all the configurations pass the functional tests in nominal operating conditions (room temperature, typical FPGA supply voltage). All the HAPS platforms mentioned in the table are speed grade (-2), the fastest speed grade available.

Table 6-4 Tested Configurations Examples

Mode	Gen	Lanes	PHY	HAPS	DMA	DMA Channels	MSI	MSI-X	iATU	L1SS	RAS-DES	AXI I/F	Native I/F	DPA	LTR	PTM
EP	3	1	PCIe 3.0 PHY (C10) Board	DX7	No	-	Yes	No	No	Yes	Yes	Yes	No	Yes	Yes	No
EP	3	1	PCIe 4.0 PHY (E16) Board	DX7	No	-	Yes	No	No	No	Yes	Yes	No	No	No	No
EP	4	1	PCIe 4.0 PHY (E16) Board	DX7	No	-	Yes	No	No	No	Yes	Yes	No	No	No	No
EP	1	1	Xilinx GTH	DX7	No	-	Yes	No	No	No	Yes	Yes	No	No	No	No
EP	1	4	Xilinx GTH	DX7	No	-	Yes	No	No	No	Yes	Yes	No	No	No	No
EP	1	8	Xilinx GTH	DX7	No	-	Yes	No	No	No	Yes	Yes	No	No	No	No
EP	1	8	Xilinx GTH	DX7	Yes	8RD/8WR	Yes	No	No	No	Yes	Yes	No	No	No	No
EP	2	1	Xilinx GTH	DX7	No	-	Yes	No	No	No	Yes	Yes	No	No	No	No
EP	2	4	Xilinx GTH	DX7	No	-	Yes	No	No	No	Yes	Yes	No	No	No	No
EP	2	8	Xilinx GTH	DX7	No	-	Yes	No	No	No	Yes	Yes	No	No	No	No
EP	3	1	Xilinx GTH	DX7	No	-	Yes	No	No	No	Yes	Yes	No	No	No	No
EP	3	2	Xilinx GTH	DX7	No	-	Yes	No	No	No	Yes	Yes	No	No	No	No
EP	3	4	Xilinx GTH	DX7	No	-	Yes	No	No	No	Yes	Yes	No	No	No	No
RP	3	1	PCIe 3.0 PHY (C10) Board	DX7	No	-	Yes	No	Yes	Yes	Yes	Yes	No	No	Yes	No

Mode	Gen	Lanes	PHY	HAPS	DMA	DMA Channels	MSI	MSI-X	iATU	L1SS	RAS-DES	AXI I/F	Native I/F	DPA	LTR	PTM
RP	3	1	PCIe 4.0 PHY (E16) Board	DX7	No	-	Yes	Yes	Yes	No	Yes	Yes	No	No	No	No
RP	4	1	PCIe 4.0 PHY (E16) Board	DX7	No	-	Yes	Yes	Yes	No	Yes	Yes	No	No	No	No
RP	3	1	Xilinx GTH	DX7	No	-	Yes	No	Yes	No	Yes	Yes	No	No	No	No
EP/RP	1	1/2/4	Xilinx GTH	80	Yes	8RD/8WR	Yes	No	No	No	Yes	Yes	No	No	No	No
EP/RP	2	1/2/4	Xilinx GTH	80	Yes	8RD/8WR	Yes	No	No	No	Yes	Yes	No	No	No	No
EP/RP	3	1/2/4	Xilinx GTH	80	Yes	8RD/8WR	Yes	No	Yes	No	Yes	Yes	No	No	No	No

6.2.3 Synthesizing the Controller for an FPGA

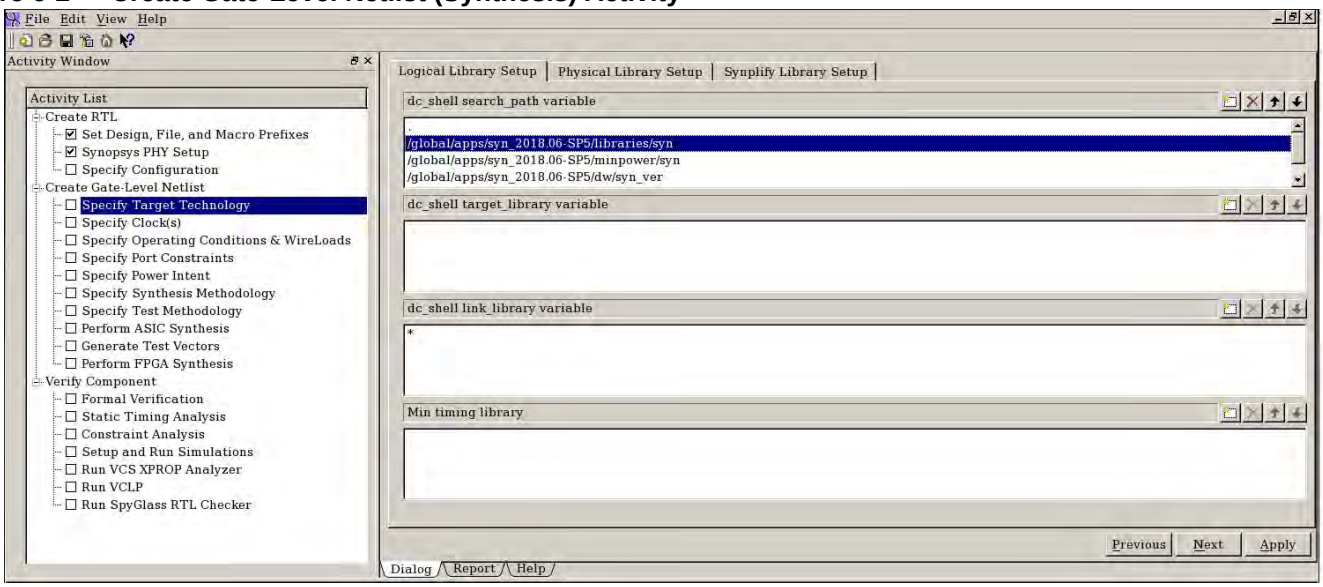
If you want to map and synthesize the controller to an FPGA using the Synopsys Synplify tool within coreConsultant, then follow the process outlined in this section. Otherwise, you must export the synthesis scripts from coreConsultant as outlined in "Synthesizing the Controller for an FPGA" on page 118" so that you can synthesize the controller in your own design flow. The topics in this section are as follows:

- "Performing FPGA Synthesis" on page 119
- "Checking FPGA Synthesis Results" on page 121

Certain configuration parameter settings and feature restrictions apply when your target device is an FPGA. You must configure the controller (see Figure 6-2) according to these restrictions before starting the synthesis task.

To synthesize your controller, select **Create Gate-Level Netlist** from the **Activity List**in the coreConsultant as shown in Figure 6-2.

Figure 6-2 Create Gate-Level Netlist (Synthesis) Activity



6.2.3.1 Performing FPGA Synthesis

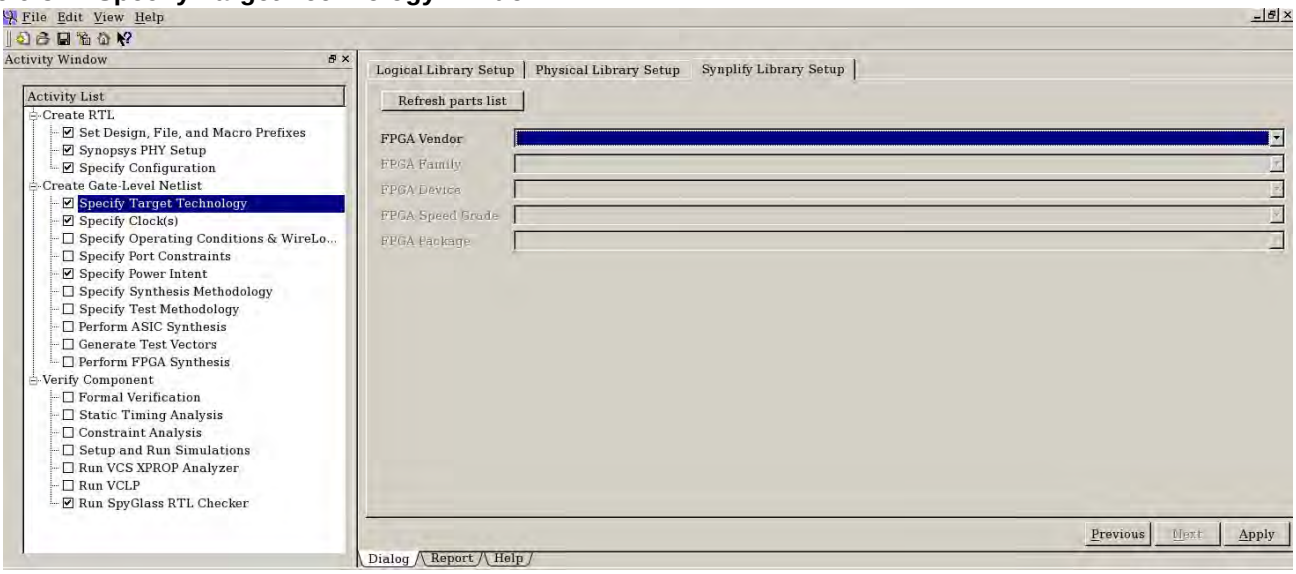
Attention You must first define the location of your Synplify tool installation (**Edit -> Tool Installation roots -> Synplify FPGA menu**) or you cannot proceed with the following steps.

You must configure the controller before starting this task. To synthesize your controller, complete the following steps in the **Create Gate-Level Netlist** activity:

1. **Specify Target Technology**

   Specify your FPGA device in the window under the Synplify Library Setup tab as shown in Figure 6-3.

Figure 6-3 Specify Target Technology Window



Hint For more information about running synthesis for an FPGA device, see also the *coreConsultant User Guide* (see "Help Information" on page 150for more information on how to access this document).

2. **Specify Clocks (in the Specify Clocks activity)**

   The cycle time for the clocks is set to match the selection made on the PCIe Basic Features Config page in the Specify Configuration activity.If the controller and PHY frequencies are not the same, a frequency step module is inserted. Inserting a frequency step module would allow you to run the controller at a lower frequency than the PIPE interface and make it easier to meet timing. The default set of the other clock-related synthesis attributes provide acceptable synthesis results for most typical devices.

3. **Specify Operating Conditions and WireLoads**

   This step is not required for FPGA synthesis.

4. **Specify Port Constraints**

   The default input and output delay constraint values are configuration-dependent. The default set of these synthesis attributes provide acceptable synthesis results for most typical devices.

5. **Specify Synthesis Methodology**

   In the **Specify Synthesis Methodology** activity, review the synthesis strategy attributes. These attributes are set by the controller developer and can be optionally modified.

Hint You can access global synthesis options through **Edit -> Preferences -> Synthesis** menu.

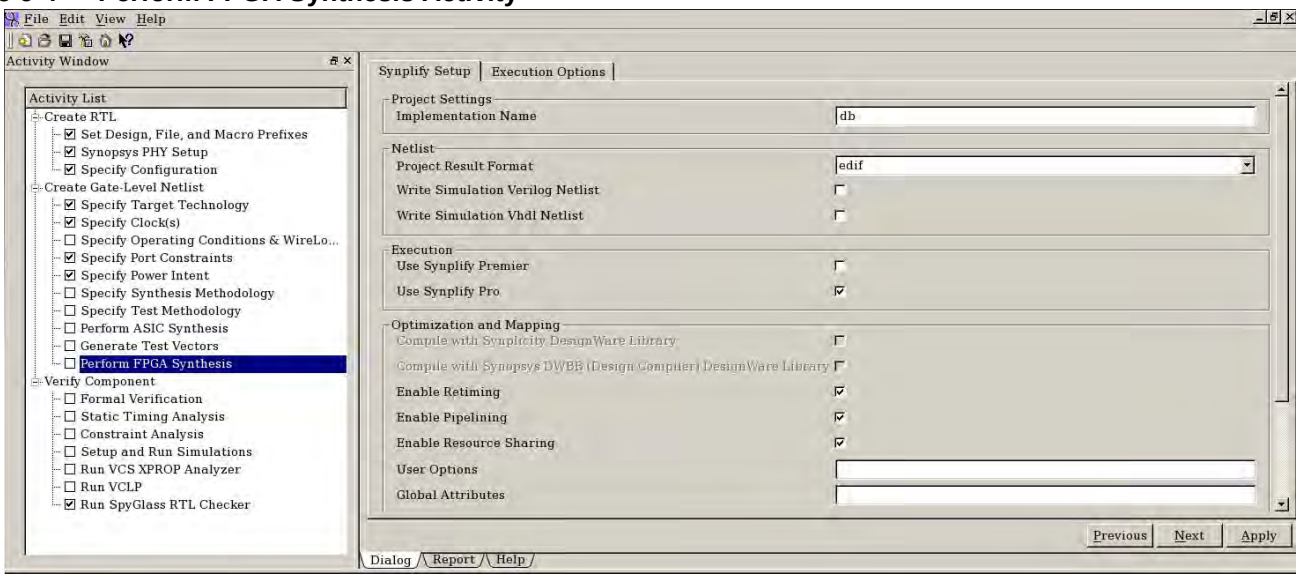
6. **Specify Test Methodology**

   This step is not required for FPGA synthesis.

7. **Perform FPGA Synthesize**

   Click **APPLY** button in Figure 6-4. Specify User Option `-disable_io_insertion 01`, since the FPGA may not have enough IOs for PCIe controller and it might calculate very long delays.

Figure 6-4 Perform FPGA Synthesis Activity



For more information on these options, see the*coreConsultant User Guide* (see "Help Information" on page 150).

6.2.3.2 Checking FPGA Synthesis Results

After you have started synthesis, coreConsultant generates several reports. You can access these by clicking the Report tab in Figure 6-2.

All the synthesis results and log files are created under the `syn/synplify` directory in your workspace. The `final` log file is `workspace/syn/run.log`.

Table 6-5 includes the other files of interest that are generated.

Table 6-5 Synthesis Output Files

Files	Purpose
`./syn/synplify/db/DWC_pcie_ctl.*`	Synopsys database files (gate level) that can be read into Synplify for further synthesis, if desired.
`./syn/synplify/report/*.*`	Synthesis report files.

Table 6-6 FPGA Synthesis Flow Files in Workspace/syn

Type	File Name	Description
FPGA	synplify/script/DWC_pcie_*.sdc	Primary constraints
	synplify.tcl	Creates synthesis scripts
	synplify.scr	Runs synthesis

6.3 FPGA Debug Guide

For more information on FPGA debug options, see "Silicon Debug Guide" on page 207.

A

Selecting PCI Express IP for Your Design

PCI Express, the next generation of the PCI bus, is being widely adopted in today's high-performance PCs, servers and embedded applications. This high bandwidth protocol keeps the same software interface and many of the key features of PCI but has a number of differences and new features. The biggest changes with PCI Express are the use of serial data transfers and gigahertz clock speeds, which make the protocol more complex, but provide significant improvement in data throughput.

This chapter provides a very brief introduction to the PCI Express protocol and explains how selecting the right digital and mixed-signal IP can accelerate the implementation of PCI Express functionality in your designs.

The following topics are discussed:

- "PCI Express System Example: PC Motherboard Based System" on page 124
- "PCI Express System Example: Chip-to-Chip System" on page 125
- "PCI Express Protocol Stack" on page 126
- "PCI Express Links" on page 127
- "Digital IP to PHY Interface (PIPE)" on page 128
- "Selecting the Lane Width" on page 129
- "Selecting Mixed-Signal PCI Express IP (PHY)" on page 129
- "Selecting Digital PCI Express IP Types" on page 130

A.1 PCI Express Overview

PCI Express provides low pin count, high reliability, and high-speed data transfer at rates of 2.5 Gbps and up, for serial links on backplanes and printed wiring boards.

A.1.1 PCI Express System Example: PC Motherboard Based System

In the example PCI Express system shown in Figure A-1, the dashed lines represent PCI Express Links, the purple boxes indicate plug-in cards, and the other boxes are components found on a system card.

The blue and yellow boxes represent PCI Express IP, which can be digital (blue):

- **Root Port (RP):** Initializes and manages the PCI Express fabric
- **Switch Port (SW):** Routes data between multiple PCI Express Links
- **Endpoint (EP):** Associated with I/O devices and terminate a PCI express hierarchy

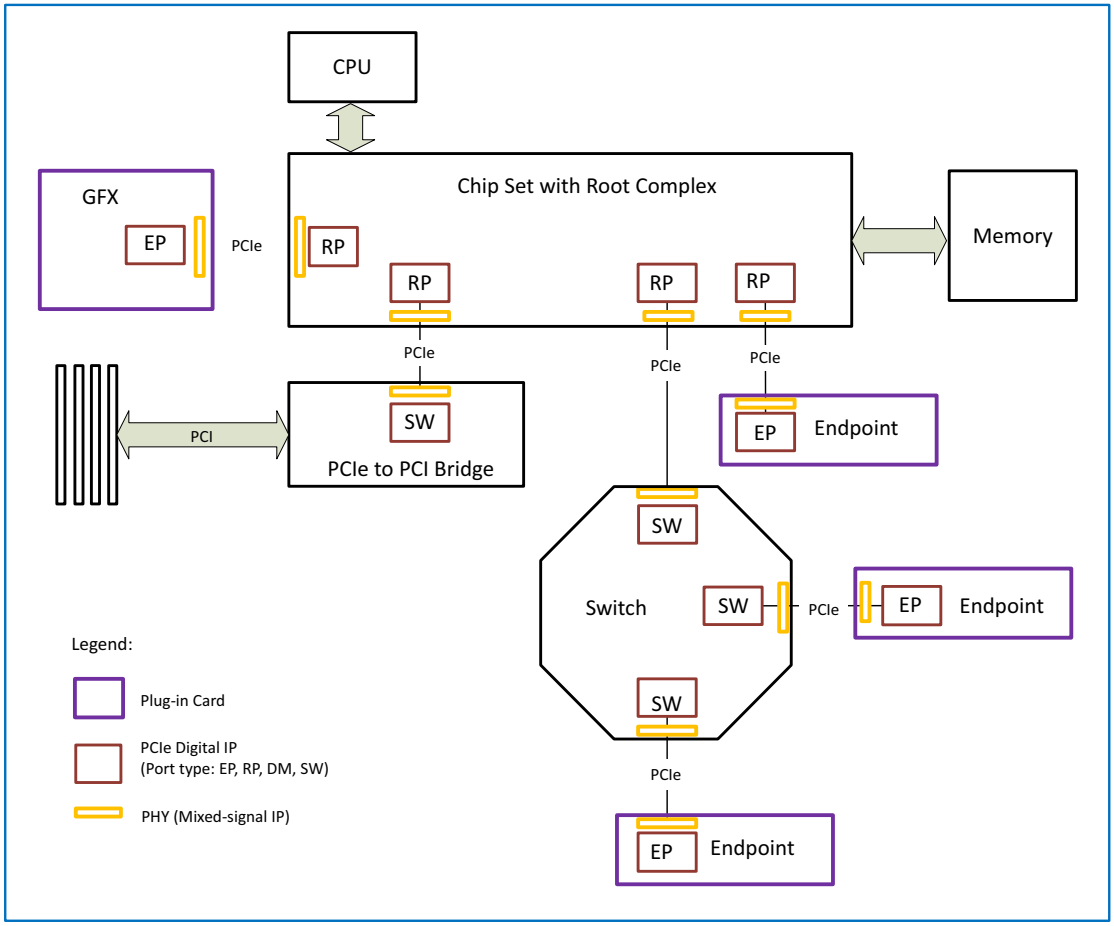
or mixed-signal (yellow):

- **PHY:** Performs analog/digital conversion and serialization/deserialization

You can choose the throughput of each PCI Express Link by selecting the number of Lanes.

The PCI Express IP handles the protocol-related functions such as Link initialization, error recovery, power management, and data buffering.

Figure A-1 PC Motherboard Based System

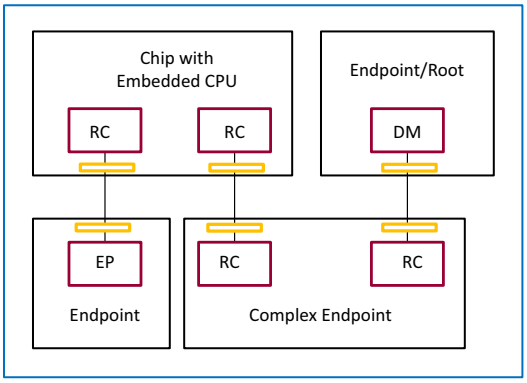


A.1.2 PCI Express System Example: Chip-to-Chip System

In the example shown in Figure A-2, all circuits are on a single printed wiring board.

The Complex Endpoint chip includes two independent PCI Express Links. The Endpoint/Root chip includes an additional type of digital PCI Express IP, Dual Mode (**DM**), which can operate as a Root Port, as shown in Figure A-2. The DM digital IP can also operate as an Endpoint; for example, when plugged into a PC motherboard slot.

Figure A-2 Chip-to-Chip System

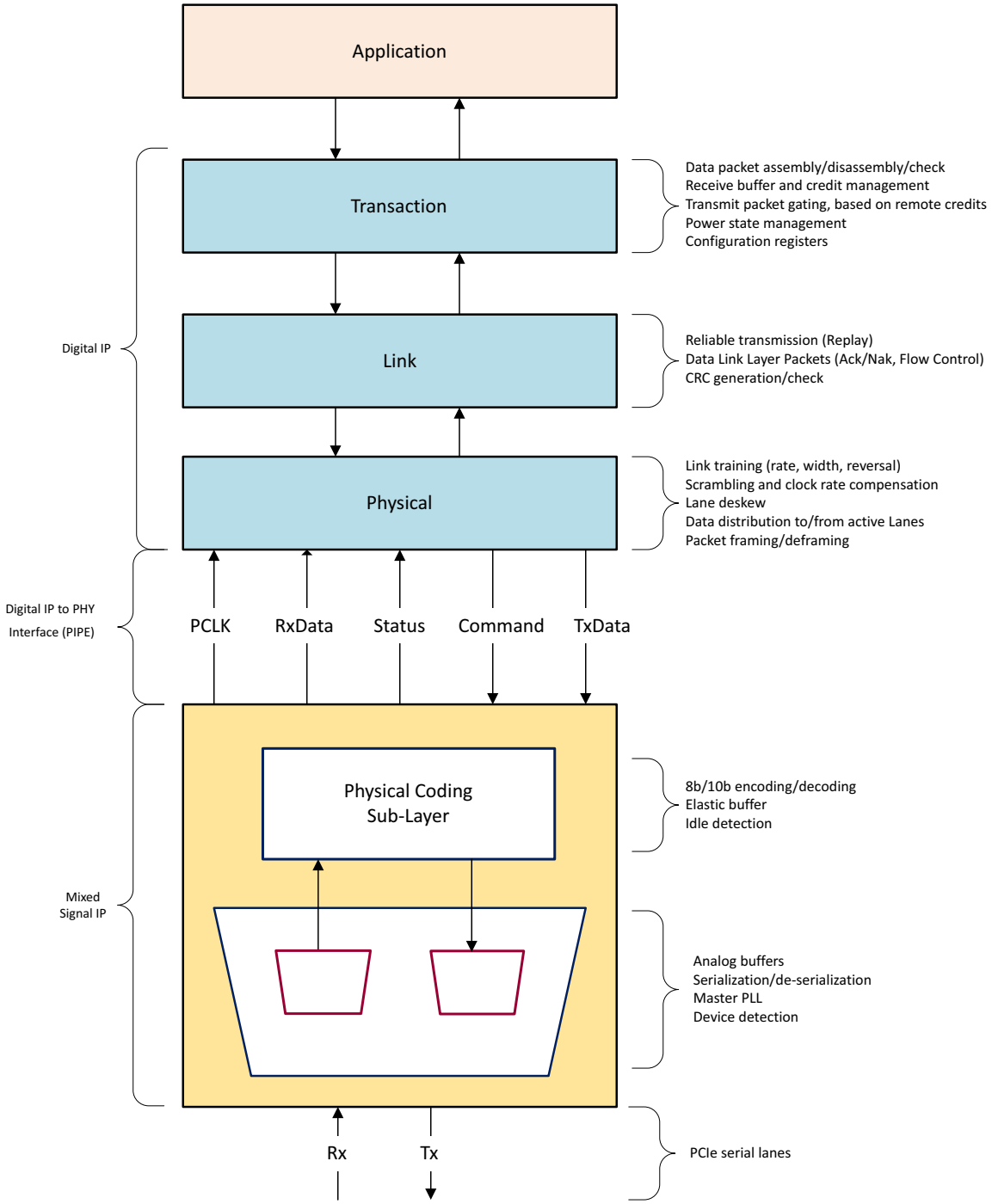


A.1.3 PCI Express Protocol Stack

The base PCI Express protocol stack is common to all the IP shown in the examples.

PCI Express IP hides most of the complexity of the protocol from your application logic, as shown in Figure A-3.

Figure A-3 PCI Express Protocol Stack



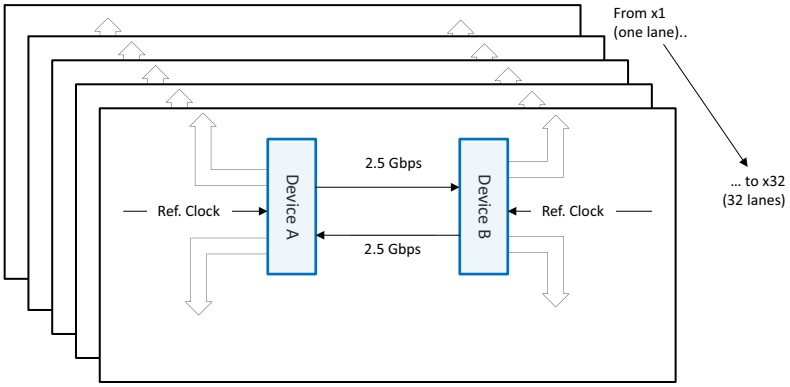
A.1.4 PCI Express Links

As shown in Figure A-4, each PCI Express Link:

- Is dual unidirectional with no sideband signals

- Is serial, with differential signaling
- Includes embedded clocking
- Operates at a scalable frequency (2.5 Gbps initially)
- Can be implemented in low-voltage silicon at 0.25u and smaller

Figure A-4 PCI Express Link



Data is transferred in packets, which include an address and a variable size data payload.

Reliability is provided by character checks, format checks, CRC code, automatic retransmission in case of error, and exchange of buffer space information, in the form of credits.

A handshake protocol to power down an idle Link is included, as well as Messages to handle interrupts, error reporting, and Hot-Plug events.

Configuration registers are available to customize Link behavior.

Wide Ports automatically configure to narrower Ports, as required.

Quality of service[1] and isochronous traffic are supported through optional virtual channels (VCs).

A.1.5 Digital IP to PHY Interface (PIPE)

Transmit and receive data, as well as status and control, are transferred between the digital and mixed-signal IP on the PHY Interface for PCI Express (PIPE).

There are two standard options for transferring data across the PIPE:

- One byte per clock cycle per PCI Express Lane. In this case, the interface and the digital IP operate at 250 MHz.
- Two bytes per clock cycle per PCI Express Lane. In this case, the interface and the digital IP operate at 125 MHz.

1. The AXI bridge does support QOS.

A.2 Selecting PCI Express IP for Your Application

To add a PCI Express port to your chip you must:

- Select the Lane width
- Select mixed-signal IP including PIPE frequency and optional features
- Select digital IP type (EP, RP, DM, or SW) and optional features

A.2.1 Selecting the Lane Width

An N Lane PCI Express Port provides N x 2.5 Gbps of raw throughput in each direction. Because of 8b10b encoding, packet payload size, and Link overhead, the actual throughput varies. Table A-1 shows some example throughput values based on Lane width and packet payload size.

Note The values in Table A-1 are a selection of reasonable examples.

Table A-1 PCI Express Link Throughput Examples

	Real Throughput Examples (Gbps)			
	Packet Payload Size in Bytes			
Lane Width	**16**	**128**	**256**	**4096**
x1	0.5	1.7	1.7	2.0
x4	2.0	6.8	7.0	7.8
x8	4.0	13.7	14.0	15.7
x16	7.9	27.4	28.0	31.4

A.2.2 Selecting Mixed-Signal PCI Express IP (PHY)

Mixed-signal IP is generally sold as a “hard macro”, which is tailored to your chip manufacturer’s process.

Both the PCI Express and PIPE interfaces are standard. You can usually choose:

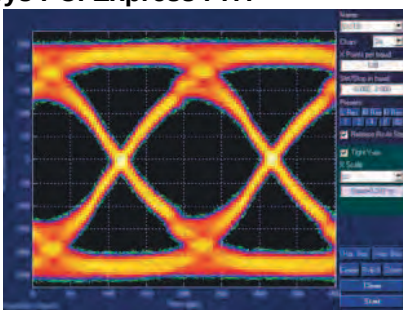
- The number of PCI Express Lanes
- A one-byte or two-byte PIPE. As explained previously, you determine the PIPE configuration and digital IP operating frequency when you make this selection.

In addition to these standard features, your PHY vendor might offer additional features, such as:

- Lower cost ICs with smaller die size (some PHYs are 50% smaller than others)
- Better performance margin (some have twice the sensitivity and jitter margin)
- Yield and reliability
- Built-in diagnostics and system test

For example, the Synopsys PCI Express PHY includes unique built-in diagnostics, which provide on-chip visibility into the actual performance of your 2.5-Gbps Links. Figure A-5 shows actual scope data from the Synopsys PHY.

Figure A-5 Signal Display from Synopsys PCI Express PHY



A.2.3 Selecting Digital PCI Express IP Types

A.2.3.1 Digital IP for Basic Applications

Based on your application, operating frequency, and Lane width, you can select PCI Express digital IP optimized for your application. lists several examples of the range of Synopsys Endpoint (EP), Root Port (RP), Dual Mode (DM), and Switch Port (SW) digital IP available to you.

Table A-2 Synopsys Digital Implementation IP Examples

IP Type Example (EP, RP, DM, SW)	Explanation	Comments
32-bit optimized x1	32-bit datapath, one Lane (x1). One Lane only operation allows extra multiLane logic to be removed for lower gate count and power.	Supports a single Lane (x1) with: ■ 8-bit PHY interface at 250 MHz, or ■ 16-bit PHY interface at 125 MHz.
32-bit	32-bit datapath, one Lane (x1) to four Lanes (x4)	■ x1 to x4 with 8-bit PHY interface at 250 MHz, or ■ x1 to x2 with 16-bit PHY interface at 125 MHz
64-bit	64-bit datapath, one Lane (x1) to eight Lanes (x8)	■ x1 to x8 with 8-bit PHY interface at 250 MHz, or ■ x1 to x2 with 16-bit PHY interface at 125 MHz
128-8	128-bit datapath, one Lane (x1) to 8 Lanes (x8)	Supports 16-bit PHY interface at 125 MHz
128-16	128-bit datapath, one Lane (x1) to 16 Lanes (x16)	Supports 8-bit PHY interface at 250 MHz

You can trade off gate count, operating frequency, and power for throughput.

A.2.3.2 Digital IP for Advanced Applications: Overview

It is relatively simple to select between Root, Switch, Endpoint, and Dual Mode Ports if your application fits one of the examples shown at the beginning of this chapter.

However, if you have an advanced application, you might want to know more about the differences. The following sections summarize the differences between the types of digital IP; see also the DesignWare IP product user/reference manuals for complete information.

A.2.3.3 Digital IP: Upstream vs. Downstream Differences

A PCI Express hierarchy contains one or more PCI Express Links, and includes a Root Port, optional Switch Ports, and Endpoints.

Each Link in the hierarchy must include exactly one "downstream" facing Port and one "upstream" facing Port.

- Root Ports are always Downstream Ports.
- Endpoints are always Upstream Ports.
- Switch Ports can be configured to be upstream or downstream. Figure A-1 shows a PC system example. In a Switch device, the Switch Port closest to the Root is always an Upstream Port; all the other Switch Ports are Downstream Ports.

Why does this matter?

- When the digital IP initializes the PCI Express Link, the Upstream and Downstream Ports use slightly different protocols to automatically configure the Link for width, and for Lane and data reversal.
- During idle times, the digital IP can autonomously transition the Link into a deep low-power state. This transition is requested by the Downstream Port and approved by the Upstream Port.

A.2.3.4 Digital IP: Configuration Registers Differences

Each instance of PCI Express IP contains a set of configuration registers:

- Root and Switch Ports contain "Type 1" configuration registers.
- Endpoints contain "Type 0" configuration registers.

Type 0 configuration registers:

- Indicate to system software that this device is the "end" of a PCI Express hierarchy.
- Contain a full set of Base Address Registers (BARs) that help you filter and address map received packets.

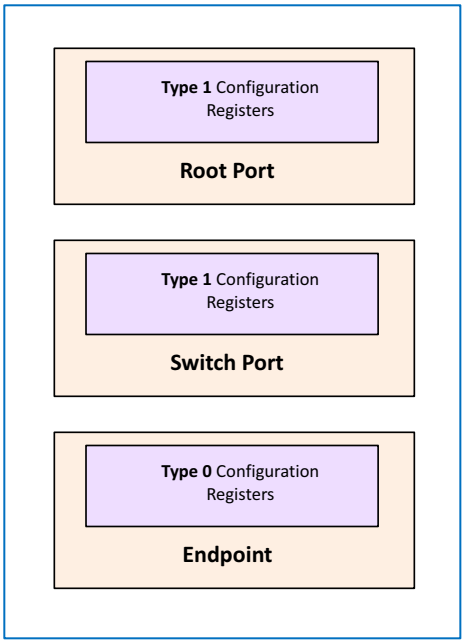
Type 1 configuration registers:

- Indicate to system software that there are more devices to discover beyond this device.
- Contain limit registers to assist in packet routing to other devices.
- Contain only minimal BARs for packets mapped to the Type 1 device itself.

Some other configuration register differences:

- Endpoint devices can contain multiple copies of the configuration registers. This feature is used to implement multifunction devices.
- Root Ports include extra registers to summarize error status for a PCI Express hierarchy.
- Root and Switch Ports contain registers to manage Hot-Plug events.

Figure A-6 Configuration Space Types



A.2.3.5 Digital IP: Configuration Transaction Differences

Configuration transactions can only be initiated by Root Ports and can only be responded to by Endpoints and Upstream ports.

Configuration transactions are used to:

- Determine the topology of a PCI Express hierarchy.
- Initialize the configuration registers after a PCI Express Link is initialized. Many values can be initialized in hardware; for example, by configuring the IP using the Synopsys coreConsultant tool. However, other values, such as memory space enable and base addresses, must be initialized through configuration transactions.
- Change the power state of a device.
- Read error report registers.

A.2.3.6 Digital IP: Interrupt and Error Message Differences

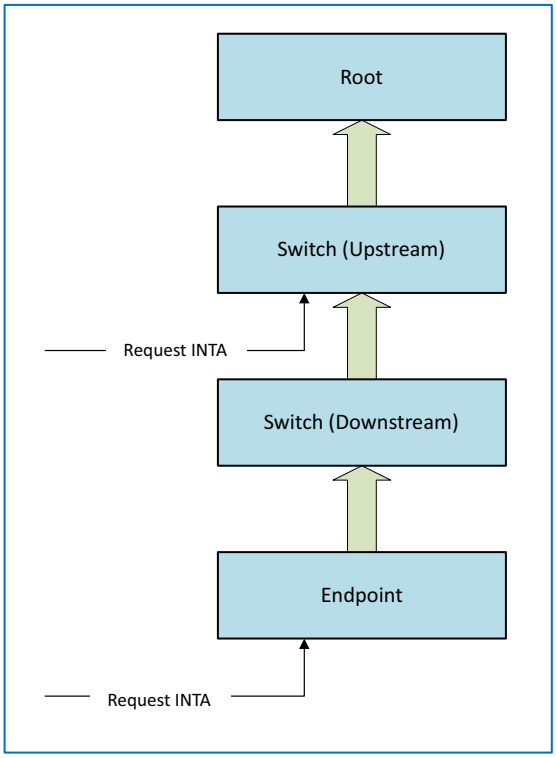
The PCI Express devices emulate PCI interrupt wires (INTA, INTB, INTC, and INTD) by sending PCI Express Messages toward the Root Port, as shown in Figure A-7. These legacy PCI interrupt Messages have the following characteristics:

- Endpoints and upstream facing Switch Ports can initiate legacy interrupt Messages.
- Downstream and upstream facing Switch Ports pass legacy interrupt Messages through to Switch controller logic.
- Root Ports can receive legacy interrupt Messages.

Other types of interrupt Messages (MSI, MSI-X) do not have these restrictions.

Error Messages are sent by PCI Express devices in response to Link errors. Endpoints initiate error Messages, Switch Ports initiate and pass along error Messages, and Root Ports receive error Messages.

Figure A-7 Interrupt Message Transfer



A.3 Implementing PCI Express in Your Design

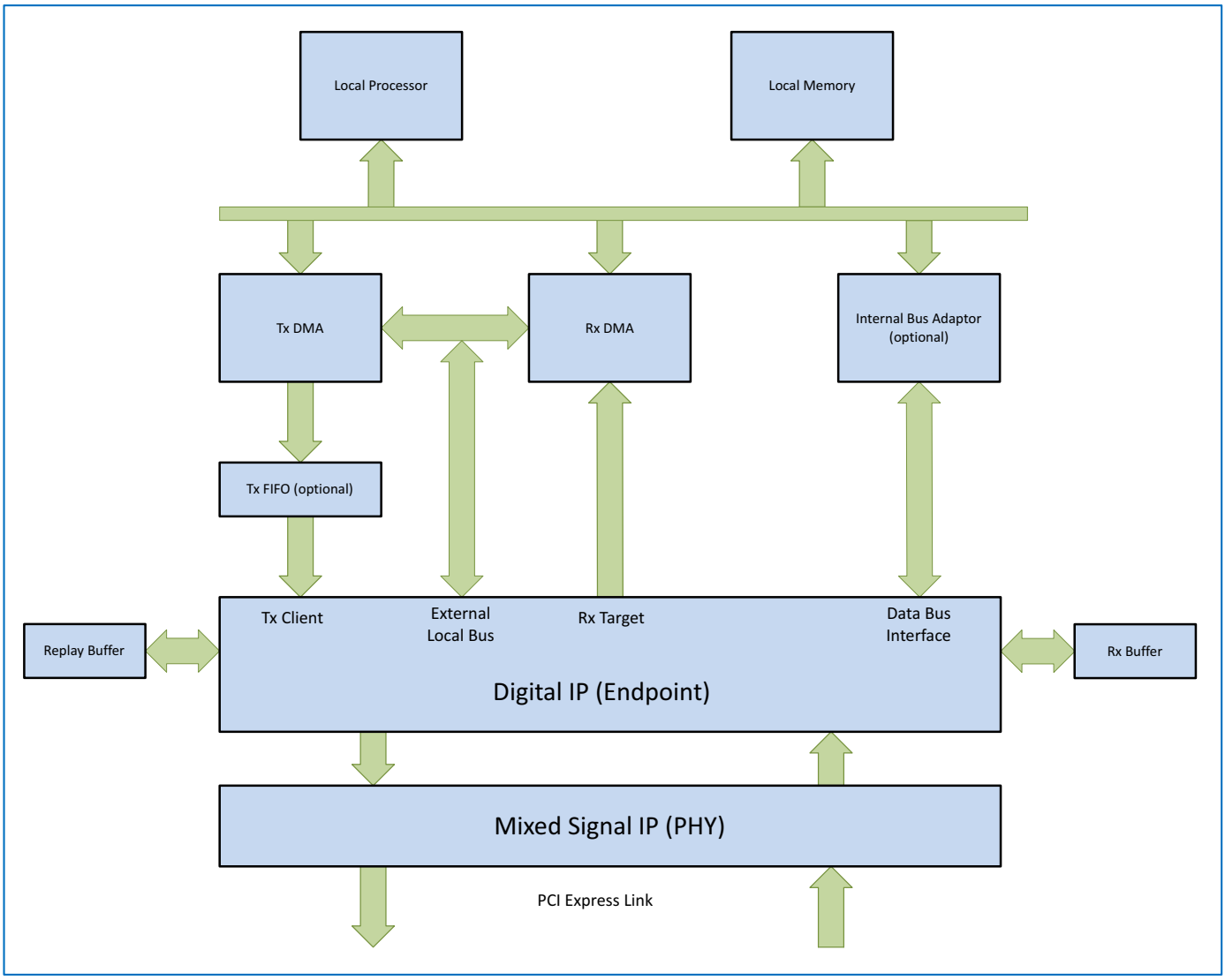
Figure A-8 shows the major features of a simple Endpoint design. For more information, see the DesignWare PCI Express IP user/reference manuals.

The replay buffer is single-port RAM; the receive (Rx) buffer is a dual-port RAM. All logic to manage these buffers is included in the Endpoint IP. For the receive buffer, you can choose store-and-forward, cut-through, or bypass (no buffer) packet storage.

The transmit FIFO is optional—if you transmit DMA can continuously supply the data for an entire packet, the FIFO is not necessary.

The Internal Bus Adapter is optional. It is only required if you want to update the "read-only" fields in the IP configuration registers before Link communication begins. As an alternative, all these fields can be configured at synthesis time with the Synopsys coreConsultant tool.

Figure A-8 Endpoint Block Diagram



The Digital IP interfaces shown in Figure A-8 can be configured to fit your application:

- The transmit (Tx) client builds packets for you from data, address, and other attributes presented by your logic. It also gates your packets according to the PCI Express rules for buffer space (credits) at the other end of the Link. You can configure the IP to have up to three transmit clients.
- The receive (Rx) target interface disassembles validated packets and presents them to you as data, address, and other attributes.
- Use of the External Local Bus Interface (ELBI) is optional. The ELBI provides a convenient way for the processor at the other end of the Link to read and write your local application control registers. No additional application hardware is required in this case. The Endpoint IP can be configured to map these register read/writes to the ELBI.
- The data bus interface (DBI) provides local access to the Endpoint configuration registers. Use of the DBI is also optional, as discussed previously with respect to the Internal Bus Adaptor.

A.4 Summary

PCI Express is a robust interface and selecting the right IP can help solve the complexities of implementing the PCI Express protocol in your designs and accelerate your development process. The DesignWare IP for PCI Express is silicon-proven in customer designs and is the industry standard, powering the PCI-SIG protocol test card, and is the first PCI Express IP to pass the compliance test.

The DesignWare IP has gone through extensive interoperability testing with third party PHYs, verification IP, and hardware. By providing a complete solution for PCI Express including digital controllers, verification IP, and mixed-signal PHY IP, Synopsys helps lower your integration risk and overall deployment costs, while saving you significant time and effort.

B

Client Interfaces Configuration Guide (without AXI Bridge)

This appendix helps you choose the appropriate transmit client architecture for your application. To familiarize yourself with the architecture of the PCIe and associated terminology, you must first read the "Architecture" chapter of the *Databook*. For more information on the client interfaces, see the "Transmit Client Interfaces" section of the "Signals Descriptions" chapter of the *Reference Manual*. XALI0, XALI1, and XALI2 are three identical client interfaces connected to three clients at your application side. Each client can be a Requester or a Completer. Requests on XALI0, XALI1, and XALI2 can be asserted in the same clock cycle. The controller automatically arbitrates requests received from the XALI0, XALI1, and XALI2 interfaces. In the following sections, reference is to made to LBC, DBI, and ELBI.

Note

For controller configured as a downstream port (DM controller in RP mode or SW controller configured as a Downstream Port)

- Your application can access CDM registers or ELBI through the DBI.
- PCIe wire access (through RTRGT0) to the CDM registers or ELBI is NOT possible.

RP Controller:

- The ELBI interface is not present.
- Your application can access CDM registers directly through the DBI. There is no LBC.
- PCIe wire access (through RTRGT0) to the CDM registers is NOT possible.

If you configure the controller with an AXI bridge then the client interfaces are no longer visible. They are internally connected between the native PCIe controller and the AXI bridge. The client architecture in the AXI bridge (if selected) is fixed and is described in the "AMBA" chapter of the *Databook*.

For an overview of PCIe and the different types of PCIe devices and controller, see "Selecting PCI Express IP for Your Design" on page 123.

Overview

Table B-1 explains client interface usage models.

Table B-1 Usage Models for the Transmit Client Interfaces

Usage Models for the Transmit Client Interfaces[a]	Modes of the Controller			
	DM	EP	RP	SW
■ A Master is connected to each client interface. In some cases, a Requester might consider implementing some ordering rules in your application logic, for example, holding off a Memory Read transaction until the Memory Write transaction is completed.	Yes (EP mode only)	Yes	No	No
■ A target completer is connected to XALI0. ■ A Master is connected to XALI1. ■ A second Master is optionally connected to XALI2.	Yes	Yes	Yes	No
■ A Master for P traffic is connected to XALI0. ■ A Master for Non-P traffic is connected to XALI1. ■ A target completer is connected to XALI2. (This is a model with one type of TLP per client (Posted, Non-Posted, Completion)).	Yes	Yes	Yes	Yes
■ XALI0 for traffic passing through the Switch. ■ XALI1 for internally-generated traffic. ■ A Single client interface is used and the arbitration is performed external to the XADM.	No	No	No	Yes

a. The XADM arbitrates among each client interface. There is no guarantee that the order is preserved among client interfaces.

One Outbound Requester

Figure B-1 shows an example where only Requester0 is used. Requester0 sends memory read (MRd) or memory write (MWr) requests to the XADM through the XALI0 interface and accepts Completions from the RCPL interface. After granted, data must be presented to the XALI0 interface as a continuous stream.

Figure B-2 shows a similar example except the data from Requester0 is slower than the line rate on the Link side. In such cases, your application should implement a FIFO to buffer the transmit data.

The examples shown in Figures B-1 and B-2 are practical for EP mode-only applications because RTRGT1 is not connected. All incoming Requests that pass filtering are routed to RTRGT0. The LBC accepts received MRd and MWr Requests from the RTRGT0 interface and converts them to local bus cycles to access application registers through the ELBI. The LBC also generates Completions for received NP requests.

In RP mode, the ELBI is not used. Incoming Requests that must be routed to your application must be routed to your application on RTRGT1.

Figure B-1 Example: One Requester—Requester with Streaming Data (Practical for EP Mode Only)

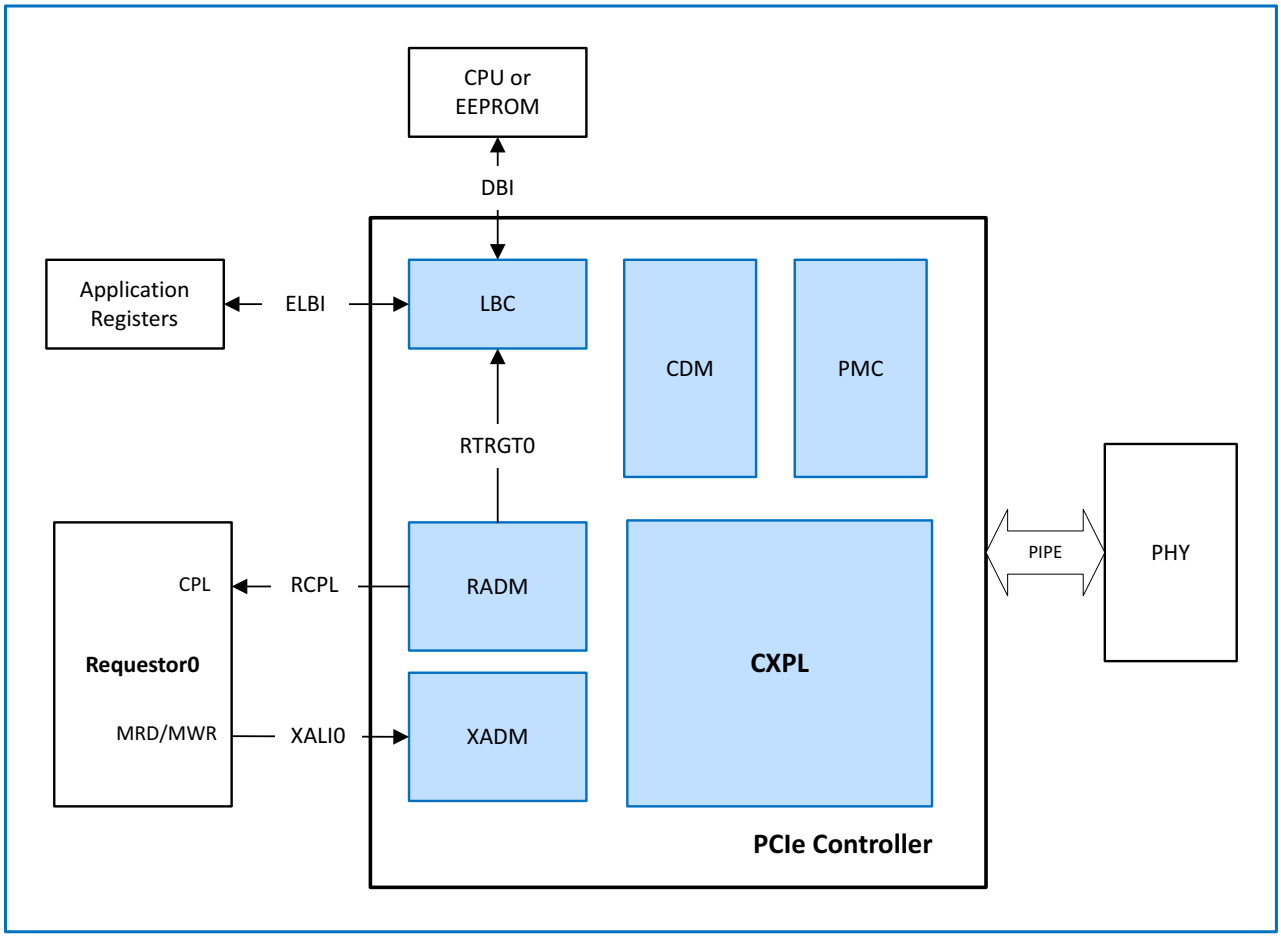
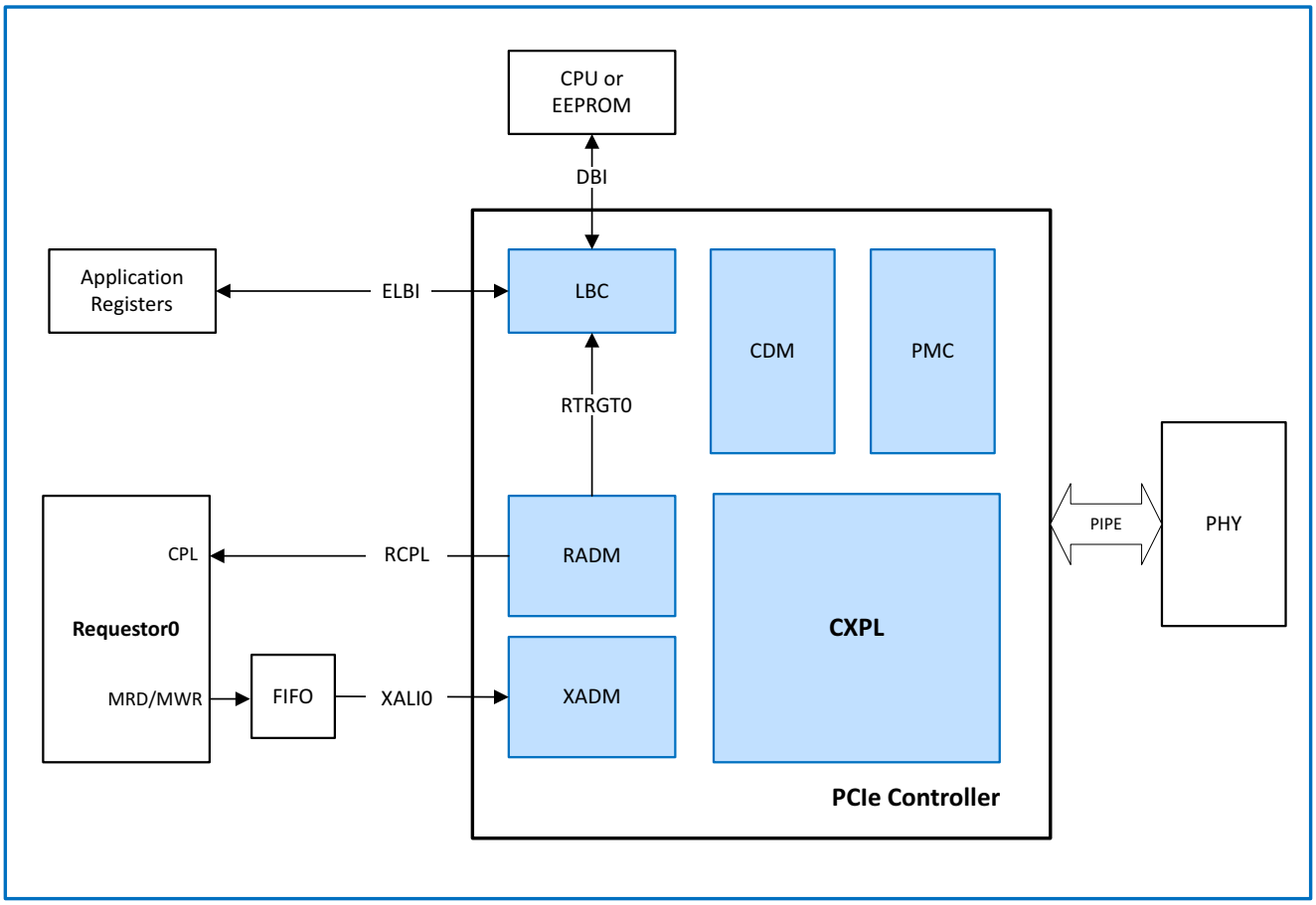


Figure B-2 Example: One Requester—Requester with Slower Data (Practical for EP Mode Only)



Two Requesters

Figure B-3 shows a two-requester example where the MRd and MWr requests share one XALI0 interface. Your application has the advantage of controlling the priority for MRd and MWr in the external arbiter. Requester0 sends MRd requests to the XADM through the XALI0 interface and accepts Completions from the RADM through the RCPL interface. It is assumed that the data is a continuous stream from Requester0. Requester1 sends MWr requests to the XADM through the XALI0 interface.

The example shown in Figure B-3 is practical for EP mode-only applications because RTRGT1 is not connected. In EP mode, the LBC accepts received MRd and MWr requests that are routed to RTRGT0 and converts them to local bus cycles to access application registers through the ELBI. The LBC also generates Completions for received NP requests.

In RP mode, the ELBI is not used. The controller routes the incoming requests to your application on RTRGT1.

Figure B-3 Example: Two Requesters – Requester0 Accepting Completions (Practical for EP Mode Only)

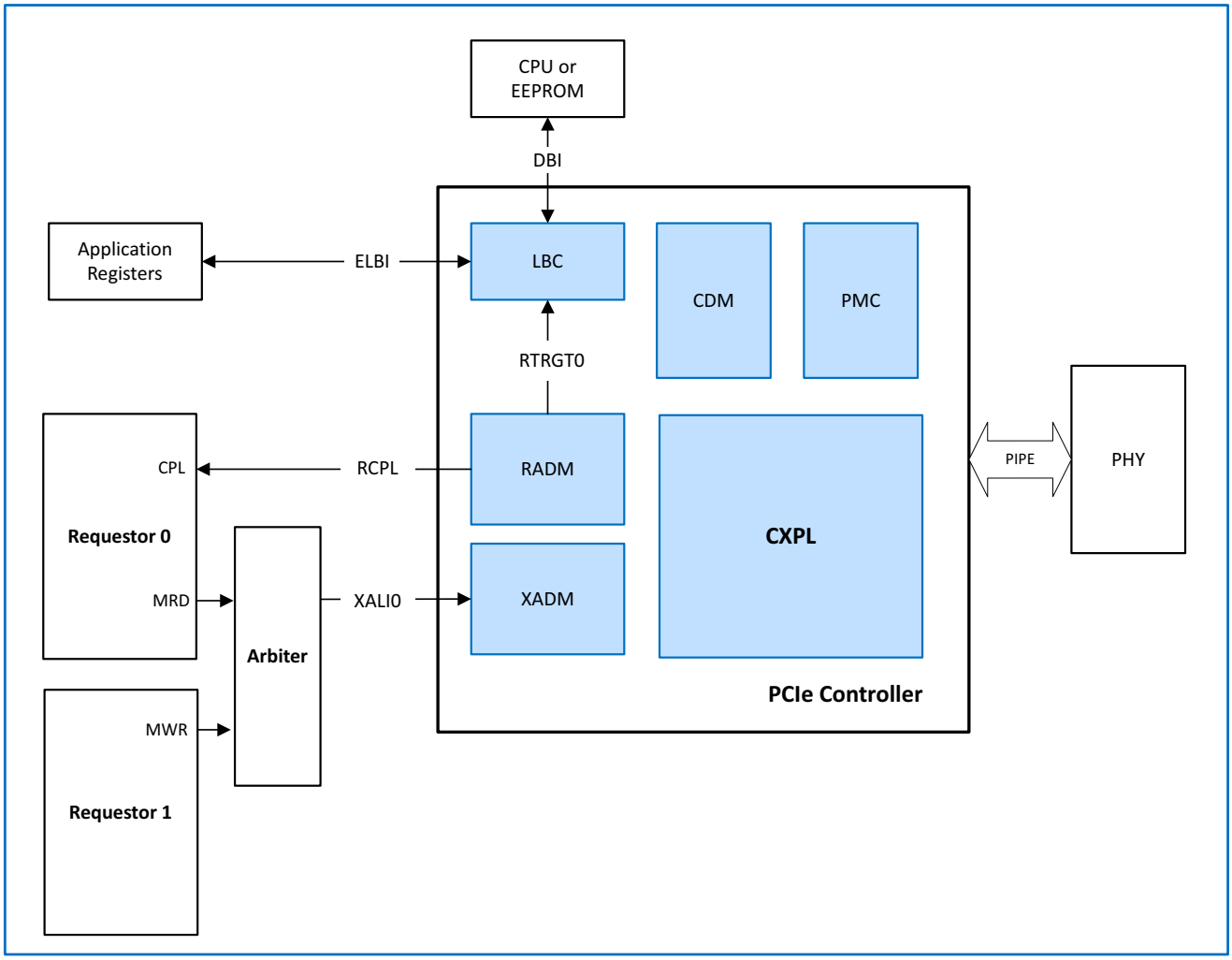


Figure B-4 shows a two-requester example where Requester0 and Requester1 are connected to XALI0 and XALI1. Requester0 sends MRd/MWr requests to the XADM through the XALI0 interface; Requester1 sends MRd/MWr requests to the XADM through the XALI1 interface. The internal arbiter inside the XADM arbitrates the Requests according to the selected arbitration method. Completions are sent to Requester0 and Requester1 from the RADM through the RCPL interface.

The example shown in Figure B-4 is practical for EP mode-only applications because RTRGT1 is not connected. In EP mode, the LBC accepts received MRd and MWr requests that are routed to RTRGT0 and converts them to local bus cycles to access application registers through the ELBI. The LBC also generates Completions for received NP requests.

In RP mode, the ELBI is not used. Incoming Requests that must be routed to your application must be routed to your application on RTRGT1.

Figure B-4 Example: Two Requesters – Both Accepting Completions (Practical for EP Mode Only)

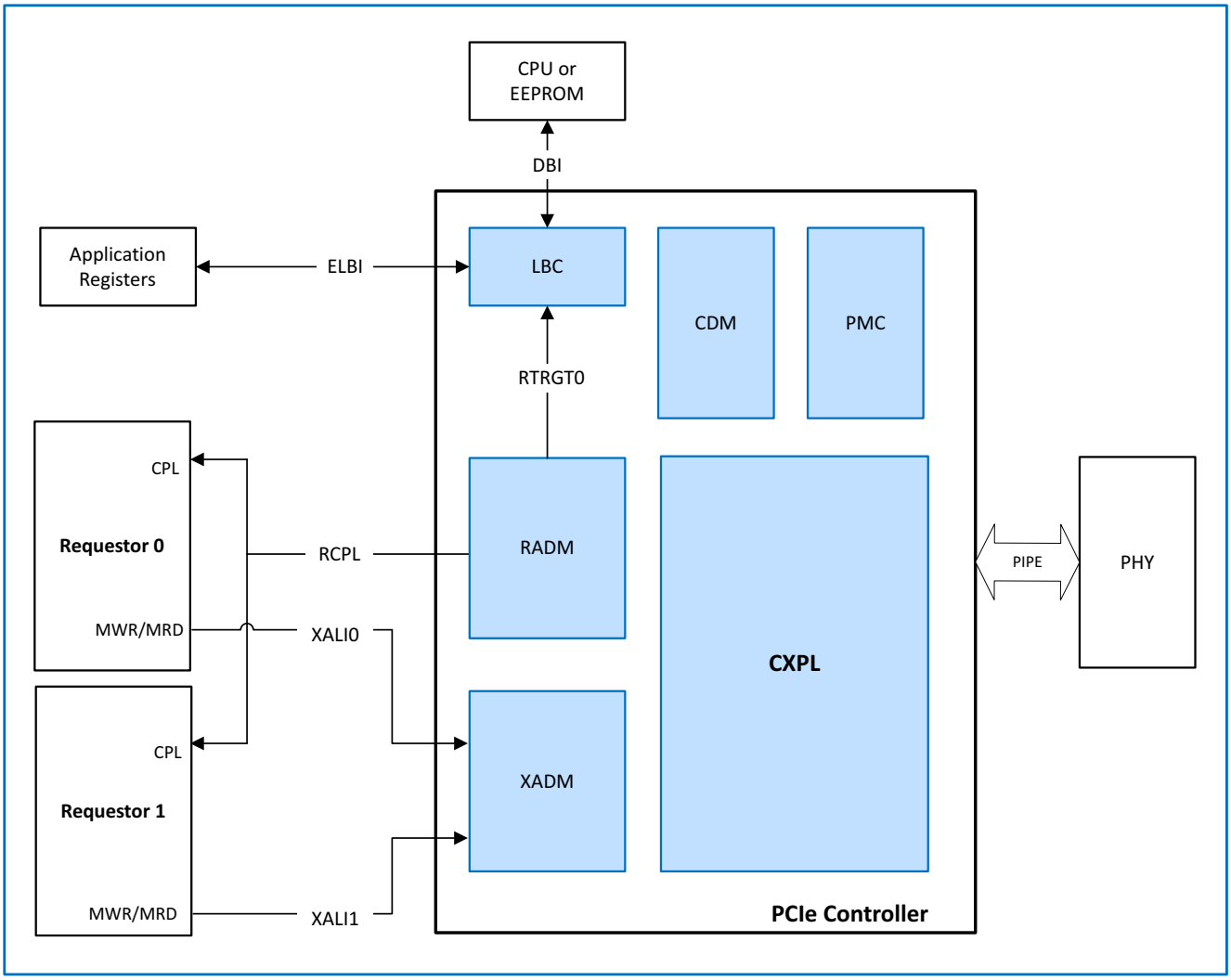
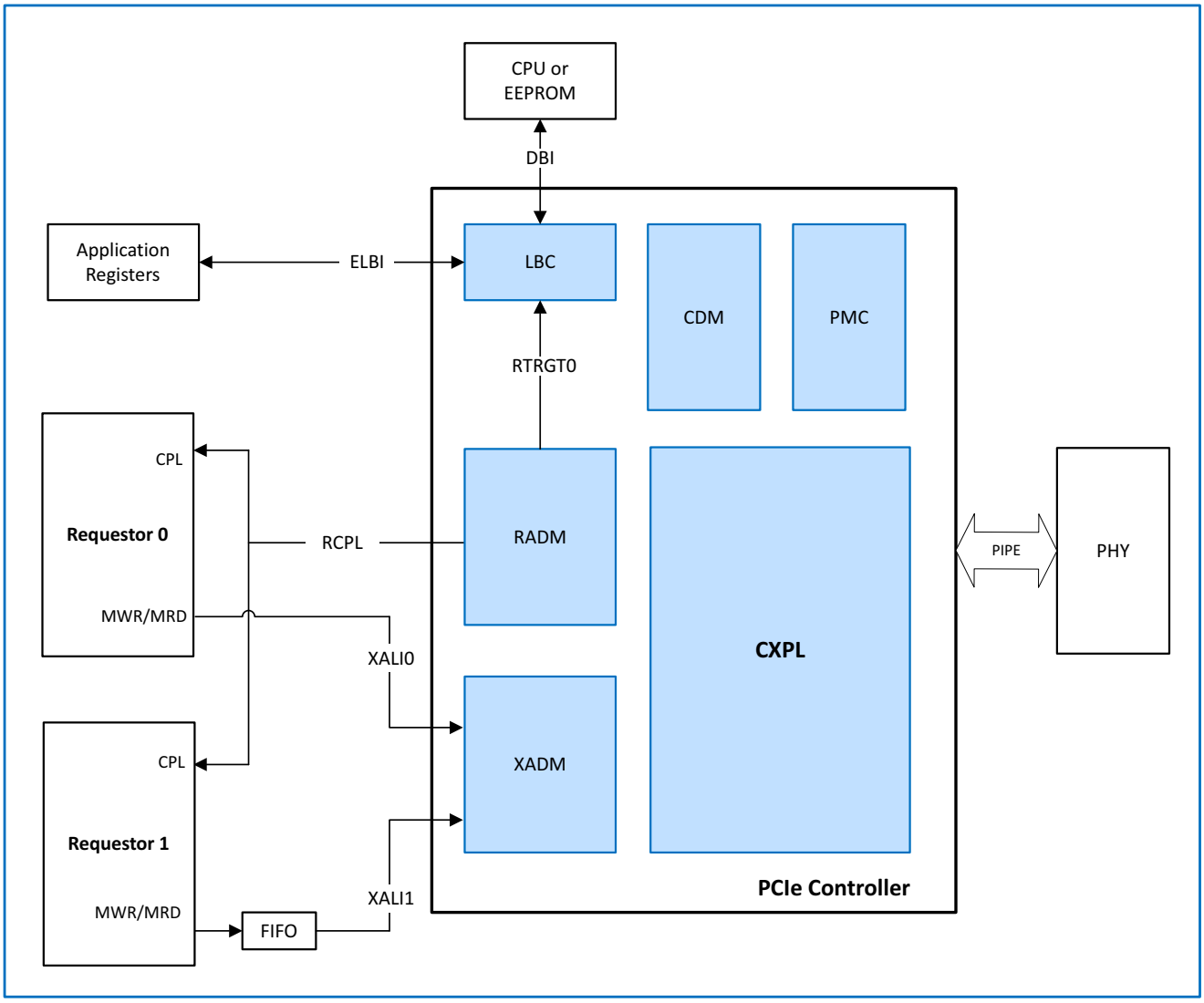


Figure B-5 shows an example where Requester0 can be used to send stream data while Requester1 accumulates data in a FIFO to catch up to the Lane rate. Data can also be a continuous stream from Requester0 and Requester1.

The example shown in Figure B-5 is practical for EP mode-only applications because RTRGT1 is not connected. In EP mode, the LBC accepts received MRd and MWr Requests that are routed to RTRGT0 and converts them to local bus cycles to access application registers through the ELBI. The LBC also generates Completions for received NP requests.

In RP mode, the ELBI is not used. Incoming Requests must be routed to your application on RTRGT1.

Figure B-5 Example: Two Requesters - Requester0 with Stream Data and Client1 with Slower Data (Practical for EP Mode Only)

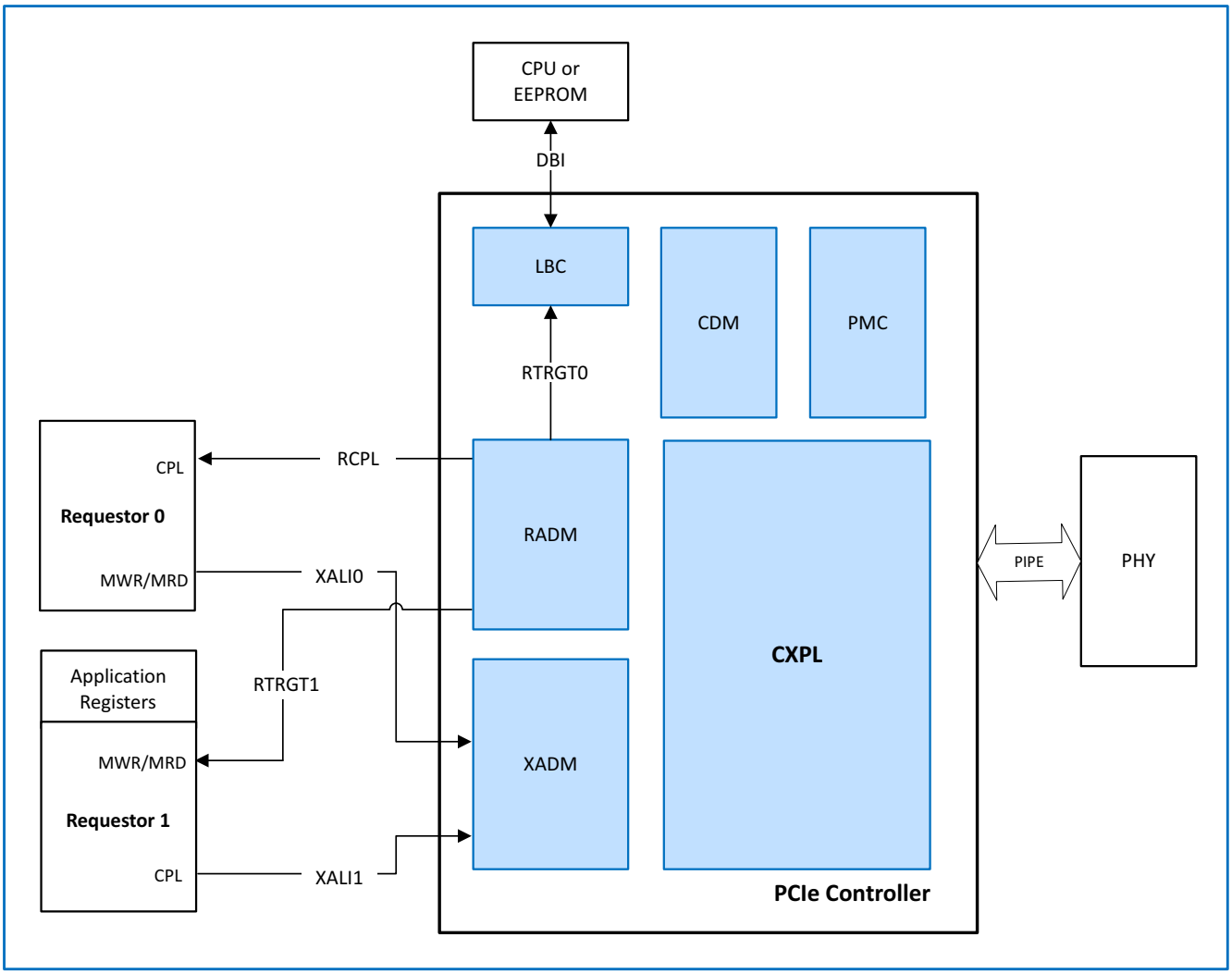


One Requester and One Completer

Figure B-6 shows an example where your application has one master requester and one target completer. The master requester sends MRd/MWr through the XALI0 interface and accepts Completions from the RCPL interface. On the target side, the target completer accepts requests from the RTRGT1 interface and sends Completions through the XALI1 interface.

The example shown in Figure B-6 supports both EP mode and RP mode.

Figure B-6 Example: One Requester and One Completer (Practical for Both EP and RP Modes)



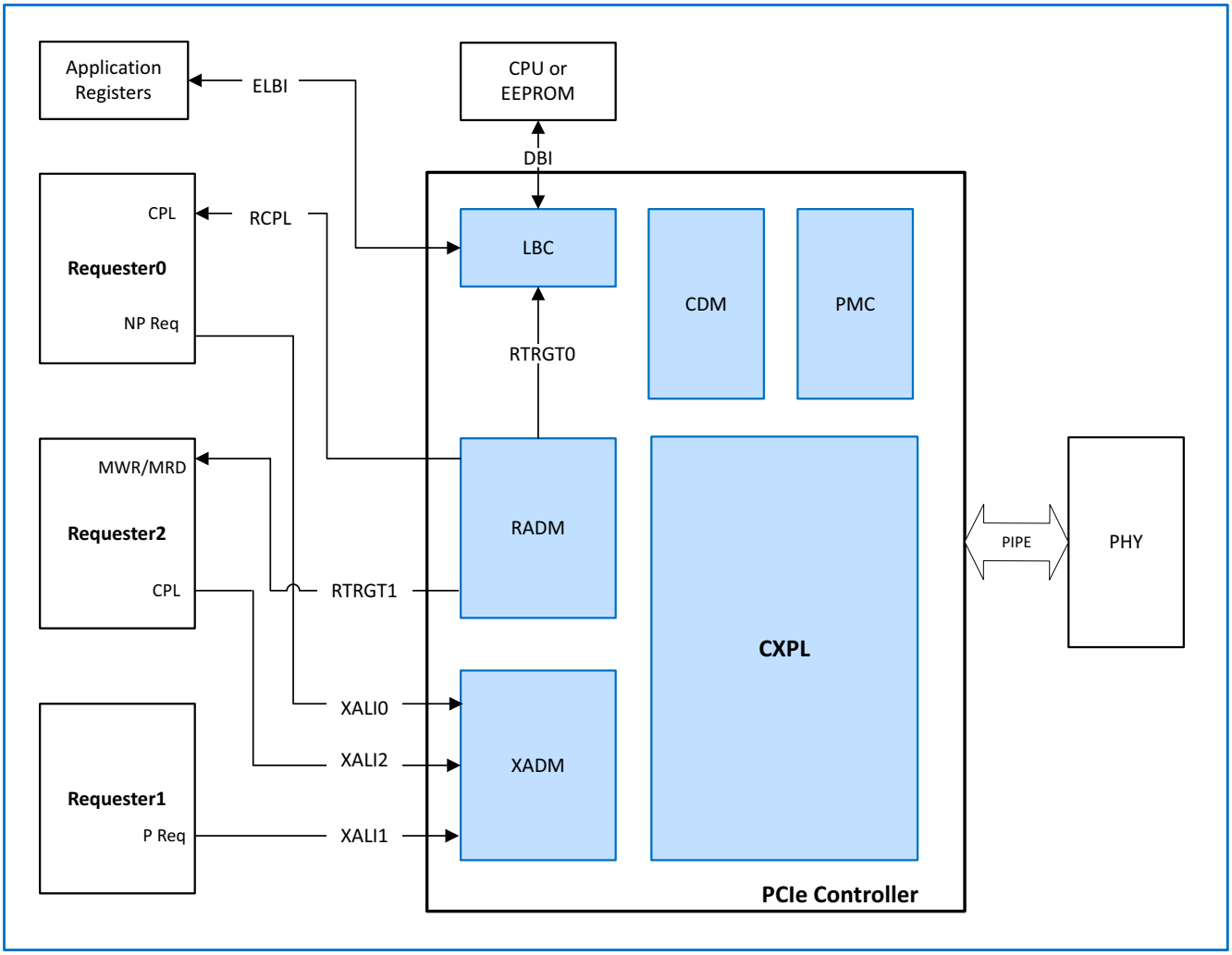
Two Requesters and One Completer

Figure B-7 shows an example where the three transmit client interfaces are used as follows:

- Requester sends Non-Posted Requests through the XALI0 interface and accepts Completions from the RCPL interface.
- Requester accepts Requests from the RTRGT1 interface and sends Completions through the XALI2 interface.
- Requester sends Posted Requests through the XALI1 interface.
- Requests that are mapped to RTRGT0 through the map-by-BAR options go to the EBLI for application register access.

The example shown in Figure B-7 supports both EP mode and RP mode.

Figure B-7 Example: Two Requesters (Requester 0 and 1) and One Completer (Requester 2) (practical for both EP and RP modes)



C

Additional Information (coreConsultant)

The topics in this appendix are as follows:

- "Troubleshooting" on page 148
- "Creating a Batch Script" on page 149
- "Help Information" on page 150
- "Workspace Directory Structure Overview" on page 151
- "Dumping Debug Information When Problems Occur" on page 153

C.1 Troubleshooting

This section discusses some common problems you might encounter when performing activities in coreConsultant. For more troubleshooting help, see the "Troubleshooting and Support" appendix in the *DesignWare Cores PCI Express Installation Guide*.

C.1.1 Specify Configuration Activity

Here are some common problems that can occur after you generate RTL for your configured controller:

- Encrypted source code is generated. Typically, this problem occurs when a Project ID in not provided when installing the controller. To resolve this issue, you must re-install the controller using the Project ID number.
- The coreConsultant tool cannot complete configuration due to a PHY selection issue. When you choose a Synopsys PHY, then you must ensure that the $PHY_PATH environment variable points to an installed Synopsys PHY.
- The coreConsultant tool issues a message that it cannot write to certain files. Check your available storage; you might not have enough on your server.

C.1.2 Create Gate-Level Netlist Activity

Here are some common problems that can occur after you perform synthesis (ASIC) for your configured controller:

- coreConsultant cannot invoke the synthesis tool after a crash. The activity has detected the presence of a synthesis '*guard*' file from the previous run. Usually this indicates that a synthesis run is in progress in this workspace. The guard file is removed when the synthesis run is completed. This activity cannot continue because files) written by this activity can adversely affect the outcome of the synthesis run. Check the Console Pane for any messages that are called out for the guard file name.
- A user-defined strategy file does not exist. Go the Console Pane and scroll to the message about the file not being created. Create that file.
- The specified cell name could not be found in the currently loaded technology libraries. Review the technology and link libraries being used and select a cell that exists within one of the libraries.
- When you synthesize the DWC PCIe controller with a PHY without the Library Compiler license, then you get the following error:

  ```
  Error: This site is not licensed for 'Library-Compiler'. (SEC-51)
  ```

  This error message is received because Synopsys PHY products provide lib format only, and Design Compiler tries to read db format. Even with the error message, Design Compiler still can read the timing information and synthesize the PCIe controller correctly. Therefore, the error message can be ignored.
- If you cannot achieve timing closure with the default settings then, set the Technology Speed configuration parameter (CX_TECHNOLOGY) to SLOW. This inserts pipeline registers into the controller to facilitate timing closure. You can also use a "fine grain" approach and select individual pipeline stages to be enabled through the coreConsultant GUI. For more information, see the "Advanced Pipeline Configuration Parameters" section of the *Reference Manual*.

C.2 Creating a Batch Script

Batch mode allows you to execute a series of coreConsultant commands from a batch file. You create a batch script after you configure the controller, so that you can recreate your exact configuration at a later stage, in case you delete or overwrite your current workspace. To create a batch file, choose the **File > Write Batch Script**menu item and enter a name for the file.

Figure C-1 Creating a Batch Script

You can review and edit the batch file by looking at the file in an ASCII editor.

You can use the batch script to reproduce the workspace using any of the following methods:

- To run in non-GUI batch mode.
  - `% coreConsultant -shell -f <batch_file_name>`
- To run in GUI mode:
  - `% coreConsultant`
  - Source the batch file from the coreConsultant command line by typing
    `source <batch_file_name>`

C.3 Help Information

Several types of online help are available through coreConsultant:

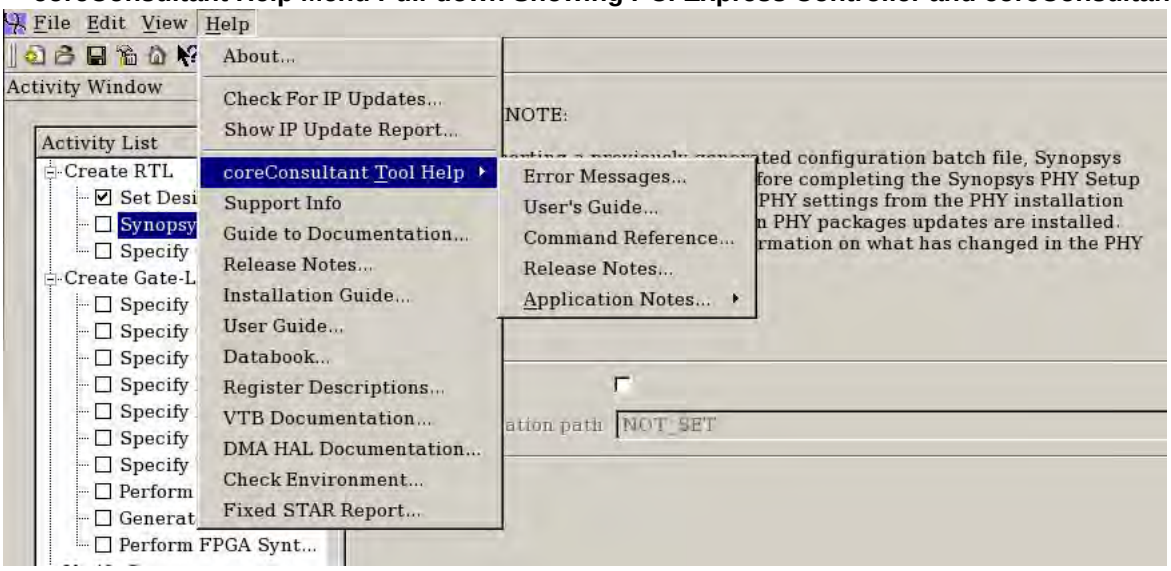
- coreConsultant and coreAssembler User Guides

  These are available through the Help menu (see Figure C-2) and are also available at `<cc_tool_root>/../doc/dware` where `<cc_tool_root>` is the path to your coreConsultant executable as returned by typing the following command in a Unix shell:

  ```
  % which coreConsultant
  ```

- **"What's This?" Quick Help**When you position your mouse pointer over a GUI item and right-click, coreConsultant displays a menu with the "What's This?" option. Clicking "What's This?" displays a brief help message for the selected item. "What's This?" can also be accessed by left-clicking the Question Pointer on the toolbar and then left-clicking on a GUI item.
- **The Toolbar Help Menu** Click the toolbar Help button to show a list of help topics. When you click a topic, the corresponding help information appears. Figure C-2 shows the Help pull-down with the available manuals for both the DesignWare Library PCI Express Controller and the coreConsultant tool.

Figure C-2 coreConsultant Help Menu Pull-down Showing PCI Express Controller and coreConsultant Manuals



- **Activity View Help Tab**The 'Activity View' pane features a Help tab that displays a detailed, context-specific help page, with additional links to the online command reference and other appropriate references.
- **Help on coreConsultant Commands** The Synopsys coreTools Online Command Reference Index is available through the Help menu. It contains a collection of man pages for all coreConsultant commands, attributes, variables, and item types.

  You can also access coreConsultant command help by entering one of the following commands at the coreConsultant prompt in the Console pane:

  ```
  coreConsultant> help [cmd] [-verbose]
  coreConsultant> cmd -help
  coreConsultant> man [cmd]
  ```

C.4 Workspace Directory Structure Overview

Figure C-3 illustrates some general directories and files in a coreConsultant workspace.

Figure C-3 coreConsultant Workspace Directory

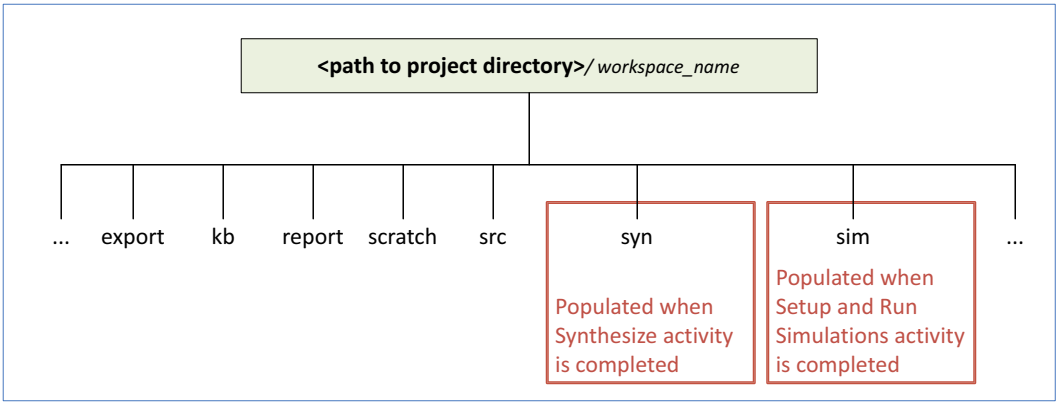


Table C-1 provides a description of the workspace directory and subdirectories.

Table C-1 Workspace Directory Contents

Directory/Subdirectory	Description
auxiliary	Scripts and text files used by coreConsultant. Generated upon first creating workspace.
doc	Contains local copies of PCIe Cores documentation (.pdf files). Generated upon first creating workspace.
examples	Contains usage and validation examples.
export	Contains files used to integrate results from the completed source configuration and synthesis activities into your design (outside coreConsultant). Generated upon first creating workspace; populated during the Specify Configuration and other activities.
kb	Contains knowledge base information used by coreConsultant. These are binary files containing the state of the design. Generated upon first creating workspace; populated and updated throughout activities.
report	Contains all of the reports created by coreConsultant during build, configuration, test and synthesis phases. An index.html file in this directory links to many of these generated reports. Generated upon first creating workspace; populated and updated throughout activities.
scratch	Contains temp files used during the coreConsultant processes. Generated upon first creating workspace; populated and updated throughout activities.

Directory/Subdirectory	Description
sim	Contains test stimulus and output files. Generated upon first creating workspace; updated during the **Simulate** activity.
src	Includes the top-level RTL file, `design_name.v`. Generated upon first creating workspace; populated during the **Specify Configuration** activity.
syn	Contains synthesis files for the PCIe Cores. Generated upon first creating workspace; updated during **Create Gate-Level Netlist** activity and **Formal Verification** activity.
tcl	Contains synthesis intent scripts. Generated upon first creating workspace.

C.5 Dumping Debug Information When Problems Occur

The menu entry, **File > Build Debug Tar-file**, is used to capture debug information for the Synopsys Support Center. This menu item creates the file <working_dir>/debug.tar.gz. This debug file includes:

- Batch script for recreating the workspace.
- Output from the existing debug_info command.
- Synthesis and simulation log files (if available).
- Results of an environment check (if available).

D

Synthesis Troubleshooting Guide

The PCIe controller is designed to meet timing out of the box for most configurations. However, in some scenarios you may encounter difficulties in meeting timing. This appendix describes best synthesis debugging practices, and guidance on parameters that affects timing and area to help you resolve synthesis issues. The following topics are discussed:

- "Analyzing Timing Violations in the Synopsys Controller" on page 156
- "Resolving Timing Violations in the Synopsys Controller" on page 163

D.1 Analyzing Timing Violations in the Synopsys Controller

This section gives you guidance on how to analyze timing violations in the controller and possible ways to resolve them.

The following questions help you determine what the next step is.

- Did you synthesize the Synopsys controller at SoC level or stand-alone?
  - Stand-alone: For more information, see "Resolving Timing Violations in the Synopsys Controller" on page 163 to analyze and resolve timing violations in the Synopsys controller.
  - Sub-system: Continue to next Question.
- Is the **worst** violating timing path located in the Synopsys controller or in external logic?
  - Synopsys controller: For more information, see "Resolving Timing Violations in the Synopsys Controller" on page 163 to analyze and resolve timing violations in the Synopsys controller.
  - In external logic:
    - Resolve the external timing issues before analyzing the timing failures in the Synopsys controller. Synthesis tools might not resolve smaller timing violations until the biggest timing violation has been resolved.
    - Synthesize the Synopsys controller stand-alone. See "Analyzing Timing Violations in the Synopsys Controller" on page 156 to generate a base-line synthesis run in coreConsultant.

D.1.1 Reading the Timing Report

This section guides you on how to analyze the timing report and decide what the next steps should be.

Figure D-1 shows a report with timing violations. This report is used as reference in this section to explain the essential elements in the report and analyze the timing violations.

Figure D-1 Report with Timing Violations

```
****************************************
Report :  timing
          -path full
          -delay max
          -nets
          -max_paths 1
Design :  DWC_pcie_ctl
Version:  P-2019.03-SP3
Date   :  Sat Oct 12 08:05:19 2019
****************************************

 * Some/all delay information is back-annotated.

 # A fanout number of 1000 was used for high fanout net computations.

Operating Conditions: SS0P81VN40C   Library: ts28nmhspmc_ss0p81vn40c
Wire Load Model Mode: Inactive.

  Startpoint: app_init_rst
            (input port clocked by core_clk_virt)
  Endpoint: cxpl_debug_info[51]
          (output port clocked by core_clk_virt)
  Path Group: FEEDTHROUGH
  Path Type: max

  Point                          Fanout    Incr     Path
  ---------------------------------------------------------------
  clock core_clk_virt (rise edge)          0.00     0.00
  clock network delay (ideal)              0.00     0.00
  input external delay                     6.00     6.00 f
  app_init_rst (in)                        0.02     6.02 f
  app_init_rst (net)             2         0.00     6.02 f
  U18729/X (SEP_OR2_1)                     0.32 *   6.34 f
  cxpl_debug_info[51] (net)      3         0.00     6.34 f
  cxpl_debug_info[51] (out)                0.01 *   6.35 f
  data arrival time                                 6.35

  clock core_clk_virt (rise edge)          2.00     2.00
  clock network delay (ideal)              0.00     2.00
  output external delay                   -0.50     1.50
  data required time                                1.50
  ---------------------------------------------------------------
  data required time                                1.50
  data arrival time                                -6.35
  ---------------------------------------------------------------
  slack (VIOLATED)                                 -4.85
```

Design Compiler Report Timing Options

The timing report is generated by the DC `report_timing` command. The timing report header lists the command options used to generate the report as shown in "Clocks" on page 159.

- `path full:` Report the full path between the start point and end point (as opposed to only start and end point).
- `delay max:` Report paths that did not meet the setup timing requirement.
- `nets:` Display the net names associated with the timing points.
- `max_paths 1:` Report only the worst timing path (in this path group).

Virtual Clocks

Virtual clocks are used as a reference to constrain the interface pins, thereby relating the data arrival time at the input and output ports. Each physical clock is assigned a corresponding equivalent virtual clock, named `<clock>_virt`. This allows a more accurate measurement of the timing paths without the need to factor in additional clock tree timing logic.

Wire Delays

When a design is synthesized, the synthesis tool needs to estimate the wire delays. Historically, wireload models were used to estimate path delays in synthesis. Since no placement information is taken into account, a sufficient timing margin needs to be used to ensure that the Place and Route (P&R) tool is able to meet timing.

DC Topographical leverages the Synopsys physical implementation solution to derive a virtual layout of the design so that the tool can accurately predict and use real net capacitances instead of statistical net approximations based on wire load models. This is done in an iterative fashion, with capacitances updated as synthesis progresses.

If you are using the DC Topographical flow, you should be able to ease the timing constraints compared to when using wireload models and still be able to meet timing in P&R.

The example timing report in Figure D-1 is generated using DC Topographical, therefore Wire Load Model Mode is reported as inactive.

Path Groups

Design Compiler optimizes timing based on (timing) path groups. By default, there is one path group for each clock in the design. Also, there are separate path groups for input and output ports vs. register-to-register groups. Example: The timing report could contain the following path groups:

- FEEDTHROUGH: Path that begins at an input and ends at an output
- REGIN: Path that begins at an input and ends at a register
- REGOUT: Path that beings at a register and ends at an output
- core_clk (REG-to-REG): Path that begins at a register and ends at a register clocked by `core_clk`[1]
- pipe_clk (REG-to-REG): Path that begins at a register and ends at a register clocked by `pipe_clk`[2]
- aux_clk (REG-to-REG): Path that begins at a register and ends at a register clocked by `aux_clk`[3]

Design compiler optimizes timing independently for each path group. If it cannot meet timing in one path group, it does not try to optimize timing on other paths in that group. Therefore, it is not worth looking into other timing violations until the critical path in that group has been fixed.

1. If a path begins in one clock domain and ends in another clock domain it belongs to the path group of the terminating clock.
2. If a path begins in one clock domain and ends in another clock domain it belongs to the path group of the terminating clock.
3. If a path begins in one clock domain and ends in another clock domain it belongs to the path group of the terminating clock.

Figure D-1 gives an example of where the path group is reported in the timing report.

Note In the PCIe controller, core_clk, pipe_clk, and aux_clk are considered synchronous, because during normal operation they function as the same clock.

D.1.2 Input and Output Delay

Input and output delay are set on the input and output ports in the synthesis constraints and defines how much of the clock cycle time is reserved for external logic. The remainder of the clock cycle time is what the synthesis tool has to work with to meet timing on a path that begins on an input or that ends on an output.

By default, Synopsys sets input and output delays to 30% and 60%, respectively. This is just an estimate, because the actual time required by the external logic not known.

In the example timing report in Figure D-1, both the input and output delays come into effect, as this is a feedthrough path with only combinatorial logic between the input and output. In this case the solution is to reduce the output delay, which do not need to be 60%. There is no logic in this path, so no optimization can be done. It is already as fast as it can possibly be.

D.1.3 Clocks

To model timing, the following values are defined for the clocks in the design (pre-layout)

- Clock network delay: Delay from the clock definition point to the clock input of the register
- Clock uncertainty: Clock source jitter + clock skew (difference in delay between source and destination register clock branches)

In a post-layout clock report, the clock network delay is calculated (propagated) and is the actual time from clock definition point to register clock input. Clock uncertainty includes only the clock source jitter. When using virtual clocks, the clock uncertainty on the output path is set to zero since the clock tree delays can be excluded. Only the input and output delays need to be considered. The clock network delay in this case is considered ideal.

D.1.4 Anatomy of a Timing Path

The timing path slack is calculated by taking the difference between the delay through the source register and the delay to the destination register, represented in Timing Path by a red and a green path:

Figure D-2 Timing Path

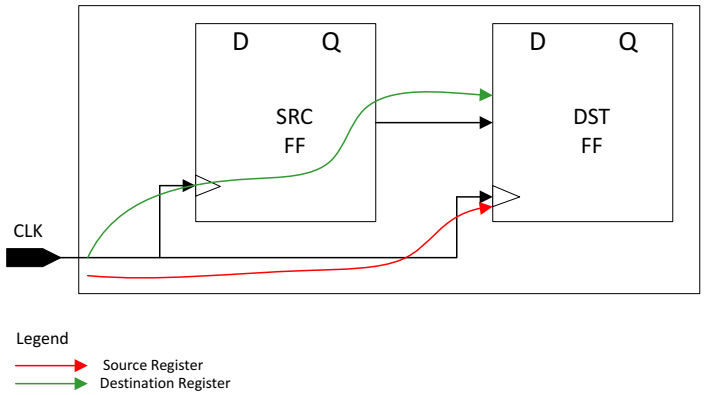


Figure D-2 shows an example of a register-to-register timing path. The numbers mentioned in this section correspond to the numbers in the Figure D-3.

Source Path:

Delay from clock definition point to data input of destination register. This includes the following components

1. clock network delay from clock definition point to clock input of source register.
2. clock-to-output delay in source register.
3. Delay from source register output to destination register input (wires and combinatorial logic).

Destination Path:

Delay from clock definition point to clock input of destination register. This includes

1. Clock period. One clock cycle delay to represent the difference between the launch and capture cycles.
2. Clock network delay from clock definition point to clock input of destination register.
3. Clock uncertainty (jitter + skew).

Figure D-3 Register-to-Register Timing Path

```
Startpoint: u_DWC_pcie_core_u_radm_sw/u_radm_q_seg_buf_u_radm_order_mgr_u_delay_mem_reg_1__7_
        (rising edge-triggered flip-flop clocked by radm_clk_g)
Endpoint: u_DWC_pcie_core_u_radm_sw/u_radm_q_seg_buf_u_data_DW_sbc_u_DW_multi_sbc_info_sb_full_m_2_o_reg_8_
        (rising edge-triggered flip-flop clocked by radm_clk_g)
Path Group: radm_clk_g
Path Type: max

Point                                                                    Fanout    Incr     Path
-----------------------------------------------------------------------------------------------------
clock radm_clk_g (rise edge)                                                        0.00     0.00 ─┐ 1
clock network delay (ideal)                                                         0.00     0.00
u_DWC_pcie_core_u_radm_sw/u_radm_q_seg_buf_u_radm_order_mgr_u_delay_mem_reg_1__7_/CK (SEP_FSDPRBQO_8)    0.00     0.00 r ├ 2
u_DWC_pcie_core_u_radm_sw/u_radm_q_seg_buf_u_radm_order_mgr_u_delay_mem_reg_1__7_/Q (SEP_FSDPRBQO_8)    0.09     0.09 f ┘
u_DWC_pcie_core_u_radm_sw/u_radm_q_seg_buf_req_rd (net)    8    0.00    0.09 f
u_DWC_pcie_core_u_radm_sw/U36769/X (SEP_INV_10)        0.03 *    0.12 r
u_DWC_pcie_core_u_radm_sw/n72111 (net)      4      0.00      0.12 r
u_DWC_pcie_core_u_radm_sw/U31293/X (SEP_NR2_T_6)       0.01 *    0.13 f
u_DWC_pcie_core_u_radm_sw/n12150 (net)      1      0.00      0.13 f
u_DWC_pcie_core_u_radm_sw/U13287/X (SEP_ND2_T_6)       0.02 *    0.15 r
u_DWC_pcie_core_u_radm_sw/n12152 (net)      2      0.00      0.15 r
u_DWC_pcie_core_u_radm_sw/U38116/X (SEP_ND2_24)        0.04 *    0.19 f
u_DWC_pcie_core_u_radm_sw/u_radm_q_seg_buf_hdrq_rd_seg_num[1] (net)    6    0.00    0.19 f
u_DWC_pcie_core_u_radm_sw/u_radm_q_seg_buf_u_data_DW_sbc_u_DW_multi_sbc_pop_logic/IN3 (DW_multi_sbc_pop_logic_18_5_130_13_2)    0.00    0.19 f
.....
u_DWC_pcie_core_u_radm_sw/n23699 (net)      1      0.00      1.68 f
u_DWC_pcie_core_u_radm_sw/U7476/X (SEP_OAI21_2)        0.02 *    1.70 r
u_DWC_pcie_core_u_radm_sw/u_radm_q_seg_buf_u_data_DW_sbc_u_DW_multi_sbc_info_next_sb_full_m_2_o[8] (net)    1    0.00    1.70 r
u_DWC_pcie_core_u_radm_sw/u_radm_q_seg_buf_u_data_DW_sbc_u_DW_multi_sbc_info_sb_full_m_2_o_reg_8_/D (SEP_FSDPRBQO_8)    0.00 *    1.70 r ┤ 3
data arrival time                                                                   1.70

clock radm_clk_g (rise edge)                                                        2.00     2.00 ─┐ 1
clock network delay (ideal)                                                         0.00     2.00 ─┤ 2
clock uncertainty                                                                  -0.20     1.80 ─┤ 3
u_DWC_pcie_core_u_radm_sw/u_radm_q_seg_buf_u_data_DW_sbc_u_DW_multi_sbc_info_sb_full_m_2_o_reg_8_/CK (SEP_FSDPRBQO_8)    0.00    1.80 r
library setup time                                                                 -0.13     1.67
data required time                                                                           1.67

-----------------------------------------------------------------------------------------------------
data required time                                                                           1.67
data arrival time                                                                           -1.70
-----------------------------------------------------------------------------------------------------
slack (VIOLATED)                                                                            -0.03
```

In this example there is a negative slack that needs to be addressed. For suggestions on resolving timing issues, see "Resolving Timing Violations in the Synopsys Controller" on page 163.

Hierarchy Optimizations

The synthesis tool may have optimized the design hierarchy, the path would indicate this by concatenating the names of the merged modules. For example,

```
u_DWC_pcie_core_u_radm_sw/u_radm_q_seg_buf_u_radm_order_mgr_u_delay_mem_reg_1__7/C
K
```

Here the merged modules are marked in bold. For more information on possible hierarchical design optimizations, see "Optimization Across Hierarchy Boundaries" on page 164.

D.1.5 Opening a coreConsultant Generated Netlist in DC Shell

After you have finished synthesis in coreConsultant, you can easily read in the synthesis data base in a DC shell by using this command:

```
restore_checkpoint.scr <synthesis stage>
```

If you are using the default synthesis strategy that is all that is required. If you are instead using DCTCL_opto_strategy you can type a different stage name other than initial (Example: incr1 or incr2) depending on which stage output, you want to open. In all cases this invokes DC, sets up the libraries and then loads the final DDC file.

Example:

```
% cd <work space>/syn
% ./restore_checkpoint.scr initial
```

D.1.6 Reporting Timing

When you have opened the net list in DC shell (as described in section "Opening a coreConsultant Generated Netlist in DC Shell" on page 161), you can report timing on the violating paths.

Example: consider the following worst violating path:

```
Start-point:
u_DWC_pcie_edma_axi_bridge/u_axi_bridge/u_sif/u_srsp/u_wrordr/r_lut_pop_cnt_reg_1_
/CK End-point:
u_DWC_pcie_edma_axi_bridge/u_axi_bridge/u_sif/u_srsp/u_wrordr/r_lut_id_reg_31__8_/
D
```

To report timing on this path, use the following command:

```
% report_timing -from
u_DWC_pcie_edma_axi_bridge/u_axi_bridge/u_sif/u_srsp/u_wrordr/r_lut_pop_cnt_reg_1_
/CK -to
u_DWC_pcie_edma_axi_bridge/u_axi_bridge/u_sif/u_srsp/u_wrordr/r_lut_id_reg_31__8_/
D -path_type full
```

To report timing on the 10 worst paths starting at any of the register called `r_lut_pop_cnt_reg`, execute the following command

```
% report_timing -from
u_DWC_pcie_edma_axi_bridge/u_axi_bridge/u_sif/u_srsp/u_wrordr/r_lut_pop_cnt_reg_*_
/CK -path_type full -max_paths 10
```

If you only know the register name where the path starts, you can list the full path to all the cells with a name containing `r_lut_pop_cnt_reg` with the following command:

```
% get_cells -hier *r_lut_pop_cnt_reg*
```

Note Depending on the technology library used, the clock port name might vary. For example, the clock port could be either CK or CP. It is important to use the correct name when specifying the start point. This command would list the clock port name and could be useful if you want to report timing on the same path in net lists synthesized with different libraries.

D.2 Resolving Timing Violations in the Synopsys Controller

This section covers the timing report checks and possible ways to resolve the timing issues.

D.2.1 Clocks

- **Clock Period**: Is the clock period correct/reasonable?
  - Clocks can be over-constrained to give margin to the back-end tools. However, if physical synthesis flow is used, you can expect more accurate results in synthesis, thus reducing the need to over-constrain the clocks.
  - AXI clocks are not required to have a fixed frequency. If there is a timing violation it is possible that the AXI clock frequency can be reduced while still maintaining functionality and performance. System level requirements determine the frequency.
  - Is the AXI clock frequency required by other components in the system?
  - Is the bandwidth on the AXI side larger than the bandwidth on the PIPE side? If so, it is possible that the AXI frequency can be reduced.
  - `core_clk`, `aux_clk`, and `pipe_clk` is determined by the controller configuration parameters (62.5/125/250 or 500MHz).
    - If you have trouble meeting timing in one clock domain, you can reconfigure the controller to run at a lower speed, keeping the PHY running at a higher speed. When the controller and the PHY are run at different frequencies `(CX_MAC_SMODE_GEN1` and `CX_PHY_SMODE_GEN1)`the frequency step module is automatically instantiated. The frequency step module converts between lower frequency/wider data path width on the controller side and higher frequency/narrower data path on the PHY side.
- **Clock Uncertainty:** What is the ratio of clock uncertainty to clock period? Is it reasonable?
  - Similar to the clock period, some methodologies set a large clock uncertainty to give margin to back-end flows. This may be unnecessary if physical synthesis is used, as the estimated wire delays are more accurate to begin with.
- **Clock Network Delay:** For post-layout net list, the network delay is calculated (propagated). If there is a large difference between the clock network delay for the source and destination clock branches, chances are that there is a problem in the clock tree synthesis (CTS) and that the clock tree is not balanced. Note: This delay is expected to be ideal when using virtual clocks.

D.2.2 False Paths

- Check if the violating paths are between asynchronous clock domains.
- When the controller configuration parameters `SLV_CLK_DIFF_ENABLE` and/or `MSTR_CLK_DIFF_ENABLE` are set to 0, `core_clk`, `aux_clk`, and `pipe_clk` are asynchronous to AXI clocks. You need to declare these clocks as asynchronous or set max delay between them.
- Some signals originating in the configuration register module (CDM), can be considered quasi-static. These paths can be declared as false paths.
- The controller synthesis constraints can be written out from coreConsultant with the `write_sdc` command. For more information, see"Synthesizing to a Device Outside of coreConsultant" on page 64. These constraints set false paths on applicable paths.
  - Ensure that you have included these constraints in your own top-level constraint file.

- Check the Spyglass CDC waivers to see if the violating path is a false path. For more information and a link to the full CDC waiver report, see "Advanced Information: Clock and Data Crossing (CDC)" appendix in the *Databook*.
  - If it is a false path, declare it in the synthesis constraint file.

D.2.3 Input and Output Delay

- Is the path a register-to-output or input-to-register path?
  - If yes, are the input and output delays set to realistic values? For example, the input and output delays set in the coreConsultant generate synthesis constraints are just an estimate (30% and 60%, respectively). Consequently, timing violations on input-to-register paths and register-to-output paths may not be critical depending on the actual timing budget in your external logic.

D.2.4 Levels of Logic

Normally, you do not see excessive levels of logic in the controller path, but a path could be long if the tool inserts many inverters (buffers).

- This may be an indication of a routing issue. To maintain signal strength on long wires, buffers need to be inserted.
- This is not an indication that the RTL itself generates many levels of logic.
- Some configuration options generate a lot of register used by other logic. This may force some destination flops to be placed far away from the registers, causing long wires. For more information, see "Configuration Parameter Settings to Improve Timing" on page 165 for tips on configuration changes that can reduce the number of registers.

D.2.5 Optimization Across Hierarchy Boundaries

There are two types of optimizations that can be done across hierarchical boundaries: boundary optimization and automatic ungrouping. Both are turned on by default in Design Compiler Ultra. If you have not turned them on in your synthesis tool, doing so helps in timing.

D.2.5.1 Boundary Optimization

Boundary optimization propagates information across hierarchical boundaries to optimize the logic. A few examples are as follows.

Note: Boundary optimization does not change or remove the hierarchical boundaries.

- Propagation of constants across boundaries.

  Example: An input port tied to 0 connected to an OR-gate results in the OR-gate being removed.
- Pushing of inverters across the hierarchy (phase inversion).

  Example: Inverters on both sides of the hierarchy are collapsed and removed.

D.2.5.2 Automatic Ungrouping

Automatic ungrouping merges logic from a sub-block into the parent-block and is enabled by default in Design Compiler. It removes hierarchical boundaries and allows Design Compiler to improve timing by

reducing the levels of logic and to improve area by sharing logic. Some flows may not allow this optimization to simplify formal verification with 3rd party tools and to facilitate debug. Another option is to allow automatic ungrouping but define exceptions where automatic ungrouping is not to be performed, such as; setting the `ungroup` attribute to false on certain hierarchies or cells.

D.2.6 Multiple Voltage Threshold Libraries

Some libraries contain functionally equivalent cells with different voltage thresholds. Which ones to use is a trade-off. High voltage threshold means lower leakage power but slower switching.

- **HVT:** High Voltage Threshold (slow switching, low leakage)
- **SVT:** Standard Voltage Threshold (medium switching, medium leakage)
- **LVT:** Low Voltage Threshold (fast switching, high leakage)

Some synthesis flows call for not using LVT cells at all in synthesis, to leave margin for the back-end flow where LVT cells can be used to fix timing during place and route.

When timing becomes difficult to be met, it would be necessary to allow LVT cells to be used during synthesis. In this case it is not recommended to have any restriction of faster cell usage.

Note: LVT is recommended for 1GHz applications.

D.2.7 Configuration Parameter Settings to Improve Timing

The configuration parameter choices you make when you generate the controller have a big impact on timing and area.Table D-2 lists the parameters that have the biggest impact. Depending on which area your critical path is located in, you can re-configure the controller to improve timing.

Table D-1 Parameters that make Timing Worse

Parameter Name	Description	RTL block	Suggestions
CC_DMA_ENABLE	Embedded DMA controller.	The DMA controller increases the area making it harder to meet timing, particularly in the core_clk domain in the AXI bridge.	Evaluate if your usage model would allow you to reduce the number of DMA channels. For example, your application may do mostly outbound traffic, in which case you can reduce the number of DMA read channels.
CC_NUM_DMA_WR_CHAN			
CC_NUM_DMA_RD_CHAN			
CX_RASDP	Data path protection with parity or ECC.	Native Controller	Evaluate if feature is required. Also, parity has smaller timing impact than ECC.
CX_INTERNAL_ATU_ENABLE	Internal address translation unit.	Native Controller	Reduce number of regions.
CX_ATU_NUM_OUTBOUND_REGIONS			
CX_ATU_NUM_INBOUND_REGIONS			

Parameter Name	Description	RTL block	Suggestions
CX_MAX_MTU	Max supported payload size.	AXI Bridge and Native Controller	Reduce CX_MAX_MTU.
CX_VF_STRIDE_ALWAYS_ONE	Enable logic optimization when you never use a VF stride greater than one.	Native Controller	When VF stride value is never used greater than 1, enable CX_VF_STRIDE_ALWAYS_ONE
DYNAMIC_VF_ENABLE	Enable programmable allocation of VFs to PFs.	Native Controller	Disable DYNAMIC_VF_ENABLE when you are using static VF allocation. This saves gates and eases timing closure.
CX_MAX_TAG	Maximum number of simultaneous outbound PCIe non-posted requests.	Native Controller	Reduce the number of requests.
CX_NVC	The number of Virtual Channels (VCs) to be supported.	Native Controller	Reduce the number of virtual channels. This parameter increases the area of the CPl queue (unless it is in bypass mode) which makes accessing it in one cycle difficult for wide data paths.
CX_APP_RD_REQ_SIZE	Max Outbound Read Request Size for Buffered CPLs.	Native Controller	Reduce the max request size. This parameter increases the area of the CPl queue (unless it is in bypass mode) which makes accessing it in one cycle difficult for wide data paths.
CX_SRIOV_ENABLE	Single-Root IO Virtualization.	Native Controller	Evaluate if feature is required (reducing the number of VFs does not have a big impact).
CX_NL	Number of lanes.	Native Controller	Reduce number of lanes so that data path width < 256 bits.
CC_MAX_MSTR_TAGS_AXI	Number of AXI master tags.	AXI Bridge	Reduce number of tags.

Parameter Name	Description	RTL block	Suggestions
CC_SLV_NUM_OUTSTND_CPU_WR_REQ	Max Number of Outstanding AXI Slave NP Write Requests. This parameter is meant to match the maximum number of NP WR requests the application can send to the slave and is used to size an internal set-aside buffer used to offload NP Wr requests and keep P Wr flowing.	AXI Bridge (critical path in u_wrordr)	Reduce number of requests (set both parameters to the same value). The effect of reducing this parameter is different for EP and RP. EP: Potential reduction in slave write request performance as less of them is accepted at a time. RP: Increased risk of deadlocks if value is less than max number of NP Wr issued by application.
CC_SLV_NUM_OUTSTND_WR_REQ (Hidden)			

Table D-2 Parameters that Improve Timing

Parameter Name	Description	RTL block	Suggested Change
CX_CUSTOM_PIPELINING	Enable custom pipeline stages.	All	Enable pipeline stages in the module where the timing issue is to improve timing. For more information about available pipelining parameters, see "Advanced Pipeline Config Parameters" section in the *Reference Manual*. This parameter does not add any pipelines, it only gives you the option to enable pipelines in modules that need it. Note: Keep pipelines to the minimum required to meet timing, because they add latency. When the controller is configured for a 1 GHz clock, additional pipelines are enabled by default.
CX_RADM_RAM_WR_REGOUT	Pipeline is inserted at the Receive Header and Data Queue RAM Write interfaces.	Native Controller	Enable this parameter if there is a timing issue on the RAM write data path.

Parameter Name	Description	RTL block	Suggested Change
CX_RETRYSOTRAM_REGOUT	Pipeline is inserted for the SOT RAM output signals.	Native Controller	Enable this parameter if there is a timing issue on the RAM write data path.
CX_RETRYRAM_REGOUT	Pipeline is inserted for the Retry RAM output signals.	Native Controller	Enable this parameter if there is a timing issue on the RAM write data path.
CX_RETRY_RAM_RD_LATENCY	Retry RAM read cycle latency (controller samples the data after this many cycles).	User logic	If the RAM read access time > 1 clock cycle, this parameter can be increased to allow for pipeline stages to be inserted.
CX_RETRY_SOT_RAM_RD_LATENCY	Retry SOT RAM read cycle latency (controller samples the data after this many cycles).	User logic	If the RAM read access time > 1 clock cycle, this parameter can be increased to allow for pipeline stages to be inserted.
CX_RADM_RAM_RD_LATENCY	Receive Queue Header and Data RAM read cycle latency (controller samples the data after this many cycles).	User logic	If the RAM read access time > 1 clock cycle, this parameter can be increased to allow for pipeline stages to be inserted.
CX_RADM_FORMQ_RAM_RD_LATENCY	Receive Serialization Queue RAM read cycle latency (controller samples the data after this many cycles). Applicable if CX_NW==16.	User Logic	If the RAM read access time > 1 clock cycle, this parameter can be increased to allow for pipeline stages to be inserted.

Parameter Name	Description	RTL block	Suggested Change
CC_MSTR_CPL_SEG_BUF_RAM_RD_LATENCY	MCB RAM read cycle latency (controller samples the data after this many cycles). Applicable to AXI configurations.	User Logic	If the RAM read access time > 1 clock cycle, this parameter can be increased to allow for pipeline stages to be inserted.
CC_OB_CCMP_DATA_RAM_RD_LATENCY	Slave Completion Composer RAM read cycle latency (controller samples the data after this many cycles). Applicable to AXI configurations.	User Logic	If the RAM read access time > 1 clock cycle, this parameter can be increased to allow for pipeline stages to be inserted.

E

Clock Domain Crossing (CDC) Methodology

This appendix provides detailed information about the Clock Domain Crossing (CDC) strategy used in PCIe. The following topics are discussed:

- "Synthesis Constraints" on page 172
- "Synchronizers Used in PCIe" on page 173

E.1 Synthesis Constraints

The synthesis constraints applied to the design follows a specific CDC methodology by adding timing exceptions. It overrides the default flop-to-flop timing paths that are defined for each individual clock domain.

- The `set_max_delay` constraint overrides the setup timing checks reaching the first synchronization flip-flops in the destination clock domain.
- The `set_min_delay` constraint overrides the hold timing checks into the destination flip-flops
- The `-ignore_clock_latency` switch in the `set_max_delay/set_min_delay` constraints is required to remove the clock tree latency from the timing calculations.

Note: The `set_false_path` constraint has higher priority than `set_max_delay/set_min_delay` constraints. `set_max_delay/set_min_delay` constraints are ignored when `set_false_path` uses the same path.

The CDC Methodology obeys the rules described in Table E-1.

Table E-1 CDC Methodology Rules

Path on Which CDC is Applied	Constraint Setting or Constraint Value		Notes
	set_max_delay -ignore_clock_latency	**set_min_delay -ignore_clock_latency**	
Gray-coded signals	‘1’ source clock period (or less)	0	Ensures only valid Gray codes are sampled in the destination clock domain.
Qualifier-based Data Bus signals	(Number of Sync stages-0.5) x destination clock period (or less)	0	Ensures that the Data Bus signals are stable when sampled in the destination clock domain.
CDC crossing[a]	‘1’ destination clock period (or less)	0	Limits the separation between the flip-flops connecting the source and destination clock domains. Maintains the assumptions for a safe CDC implementation.
Clock group information	`set_clock_groups-allow_paths`		The `set_clock_groups` constraint is used to provide clock group information required in Signal Integrity analysis during place and route only if the switch `-allow_paths` is included, otherwise `set_max_delay/set_min_delay` constraints are ignored.
Quasi-static signals at the output of the bus delay components	`set_false_path-through` (DWC_pcie_bcm36_nhs)		Quasi-static signals are not required to meet timing. The `set_false_path-through` constraint removes the timing checks on the paths.

a. Gray-coded and Qualifier-based Data bus signals must be excluded

E.2 Synchronizers Used in PCIe

A synchronizer is instantiated for every signal, register, or data-bus crossing between different clock domains, except for quasi-dynamic static registers defined in "Synthesis Constraints" on page 172. PCIe uses proven blocks, called Basic Core Modules (BCMs), to implement CDC. It uses a library of design-specific CDC parts that are mapped to the standard DesignWare library parts in http://www.synopsys.com/dw/buildingblock.php as follows:

Table E-2 PCIe to DesignWare Library Part Mapping

PCIe Part Name	DesignWare Library Part Name	Type of Synchronizer
DWC_pcie_bcm01.sv	DW_minmax	Minimum/Maximum Value
DWC_pcie_bcm21.sv	DW_sync	Single Clock Data Bus Synchronizer
DWC_pcie_bcm22.sv	DW_pulse_sync	Dual Clock Pulse Synchronizer
DWC_pcie_bcm23.sv	DW_pulseack_sync	Pulse Synchronizer with Acknowledge
DWC_pcie_bcm24.sv	DW_gray_sync	Gray Coded Synchronizer
DWC_pcie_bcm25.sv	DW_data_sync	Data Bus Synchronizer with Acknowledge
DWC_pcie_bcm36_nhs.sv		Bus Delay Component
DWC_pcie_bcm37.sv	DW_reset_sync	Reset Sequence Synchronizer *Note:* without parameter CLK_D_FASTER
DWC_pcie_bcm41.sv	DW_sync	Single Clock Data Bus Synchronizer with Configurable Polarity Reset *Note:* Same as DWbb_bcm21 but with additional RST_VAL parameter
DWC_pcie_bcm46_a.sv	DW_ecc	Error Checking and Correction--6 check bits (up to 26 data bits)
DWC_pcie_bcm46_b.sv	DW_ecc	Error Checking and Correction--7 check bits (up to 57 data bits)
DWC_pcie_bcm46_c.sv	DW_ecc	Error Checking and Correction--8 check bits (up to 120 data bits)
DWC_pcie_bcm46_d.sv	DW_ecc	Error Checking and Correction--9 check bits (up to 247 data bits)
DWC_pcie_bcm46_e.sv	DW_ecc	Error Checking and Correction--10 check bits (up to 502 data bits)
DWC_pcie_bcm48.sv		Parallel CRC Generator
DWC_pcie_bcm53.sv	DW_arb_2t	Two-Tier Arbiter with Dynamic/Fair-Among-Equal Scheme
DWC_pcie_bcm55.sv	DW_arb_rr	Arbiter with Round Robin Priority Scheme

PCIe Part Name	DesignWare Library Part Name	Type of Synchronizer
DWC_pcie_bcm57.sv	DW_ram_r_w_s_dff	Sync. Write-Port, Async. Read-Port RAM (Flip-Flop-Based)
DWC_pcie_bcm60.sv		Single port RAM with retiming registers (Flip-Flop Based)
DWC_pcie_bcm62.sv		Three port RAM with retiming registers (Flip-Flop Based)
DWC_pcie_bcm64_td.sv	DW_fifo_s1_df	Synchronous (Single Clock) FIFO with Dynamic Flags with transport delay on the RAM interface (using DWbb_bcm86 FIFO controller)
DWC_pcie_bcm65_td.sv	DW_fifo_s1_sf	Synchronous (Single Clock) FIFO with Static Flags
DWC_pcie_bcm86.sv		Single Clock Dynamic Flag FIFO Controller with Transport Delay
DWC_pcie_bcm87.sv	DW_fifoctl_s2_sf	Dual-Clock Static Flag FIFO Controller with Transport Delays

F

Simulation Troubleshooting Guide

This guide assists you in performing first level debugging of failing PCIe simulations in your SoC test bench. It highlights some commonly occurring issues and lists key controller signals (visible in the waveform viewer) that can be used to debug issues related to link training, enumeration, and memory transactions. For issues that are not covered in this guide, or not addressed by the guidelines provided here, contact Synopsys support. The following topics are discussed:

- "Gen1/Gen2 Link Training"
- "Gen3 and Above Equalization" on page 187
- "Gen5 and Above Features" on page 196
- "Upstream and Downstream Port Enumeration" on page 198
- "Memory Transactions" on page 201
- "Top Level Reference File Generation" on page 206

F.1 Gen1/Gen2 Link Training

F.1.1 Basic Clock, Reset, and Simulation Setup Checks

To ensure correct operation of the controller, check the following signals from hierarchy ***.u_pcie_core** in the simulation waveform viewer:

- `core_clk` – The `core_clk` should be up and running at Gen1 base frequency.
- `perst_n, pwr_rst_n, core_rst_n, sticky_rst_n,` and `non_sticky_rst_n` – All the resets should be de-asserted (low to high).
- `app_ltssm_enable` – Should be asserted high, to allow LTSSM to proceed with link training after your application has completed programming of controller registers through DBI (if any).

View the following signals from hierarchy: ***.u_pcie_core. u_DWC_pcie_core**

- `cfg_fast_link_mode`– Fast Link Mode should be enabled to speed up timers and reduce simulation time. For more information, see "Troubleshooting the Integration of Third-party PHYs" on page 83. This scales down all LTSSM related timeouts in the controller by a scaling factor determined by the `FAST_LINK_SCALING_FACTOR` field in `TIMER_CTRL_MAX_FUNC_NUM_OFF` register. For example, if the scaling factor is 0, a 2ms timeout occurs in 2us.

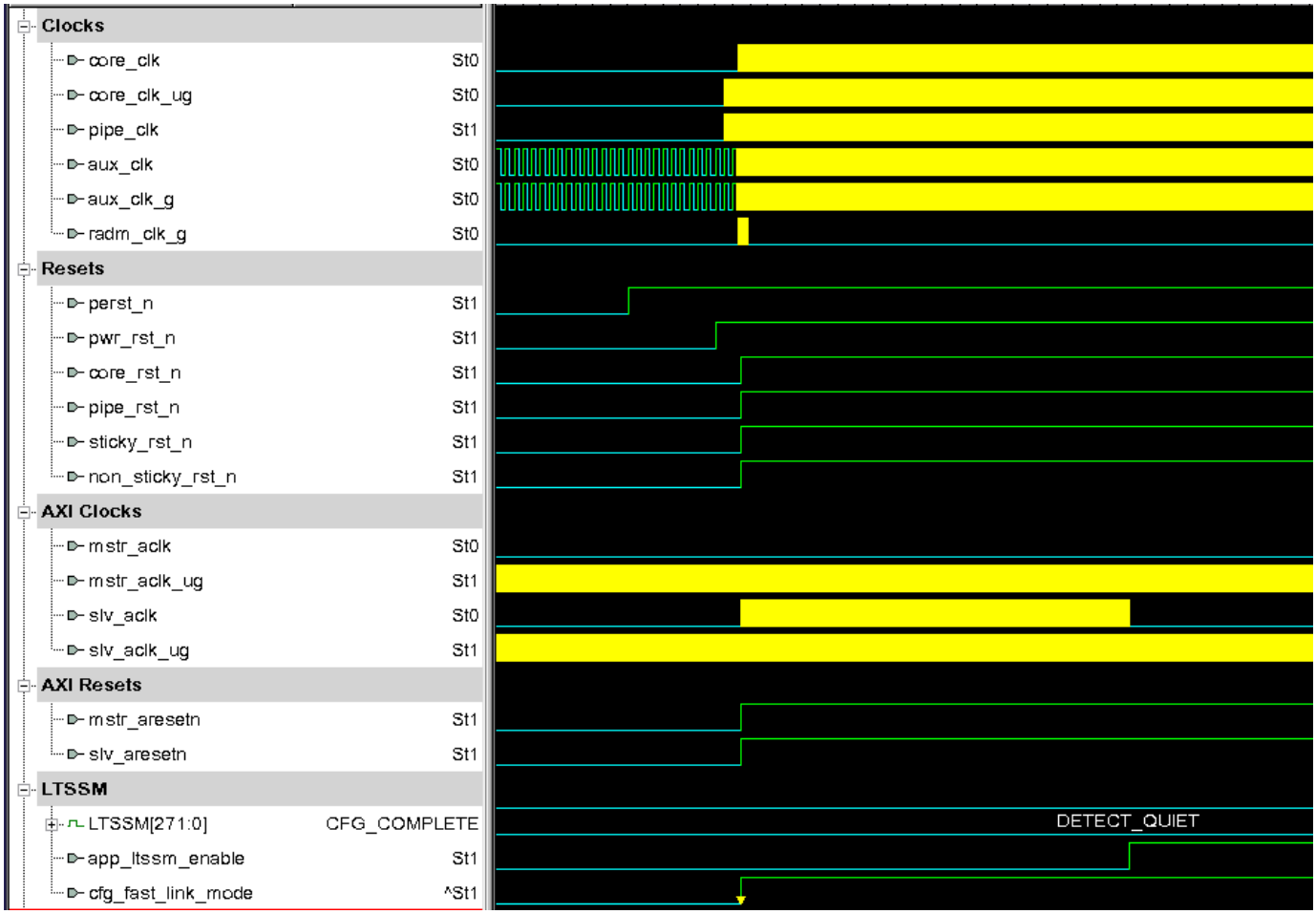
Note The cfg_fast_link_mode signal must be asserted in simulation only, it must be set to '0' in real applications.

View the following signals from hierarchy
***.u_pcie_core.u_DWC_pcie_core.u_cx_pl.u_smlh.u_smlh_ltssm**

- `LTSSM` – This signal shows the current LTSSM state.

Note For easy readability set the radix of this bus to ASCII.

Figure F-1 Clocks and Resets After Power-up



Note For more information on clocks and resets of the controller, see “Clock Requirements” and “Reset Requirements” sections of the *Databook*.

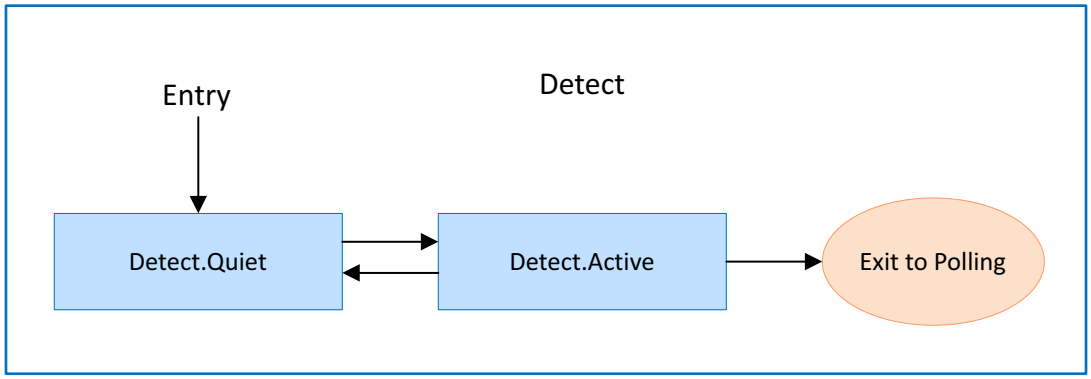
After initial power up, the LTSSM goes through the following states during link training; before reaching the normal operating state L0.

- *Detect*
- *Polling*
- *Configuration*

F.1.2 LTSSM in Detect Substates

In *Detect* state the controller performs receiver detection on all lanes to detect when a far end termination is present. In this state the controller operates at 2.5GT/s data rate.

Figure F-2 Detect Sub-state State Machine



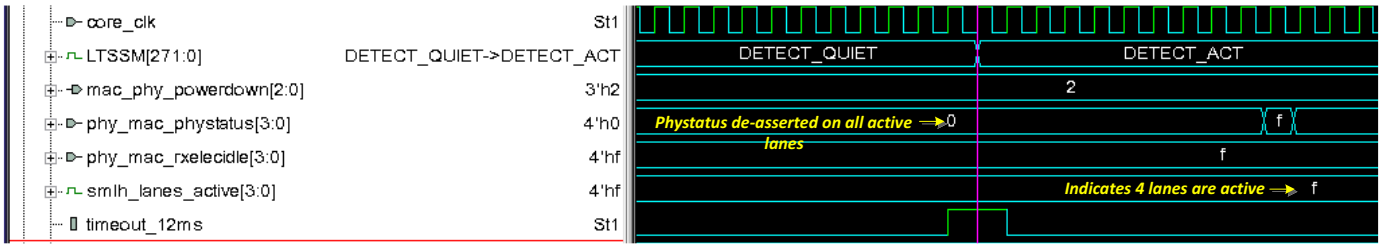
Detect.Quiet

In this state the controller LTSSM moves from *Detect.Quiet* to *Detect.Active* when the PHY power state is P1, and `phy_mac_phystatus` is de-asserted on all active lanes. The controller remains in *Detect.Quiet* until any lane exits electrical idle or 12ms timeout occurs.

Check the following signals from hierarchy ***.u_pcie_core**

- `mac_phy_powerdown:` This is the PIPE output from the controller, it should be driven to P1(2'b10).
- `phy_mac_phystatus:` This is the PIPE input to the controller, it should be de-asserted on all active lanes (`smlh_lanes_active`).
- `smlh_lanes_active:` Shows the active lanes (1 bit per lane). By default, `smlh_lanes_active` is set to the max link width of the controller.
- `phy_mac_rxelecidle:` This is the PIPE input to the controller. If `phy_mac_rxelecidle` is not de-asserted on even one of the active lanes, after the 12ms timeout, LTSSM transitions to *Detect.Active*.

Figure F-3 Successful LTSSM transition from Detect.Quiet to Detect.Active



Detect.Active

In this state, the controller performs receiver detection on all active lanes and checks the number of lanes that have detected a receiver. LTSSM is stuck in *Detect.Active* if PHY does not complete receiver detection sequence on all active lanes, as indicated by the assertion of `phy_mac_phystatus`. If all the lanes do not detect a receiver, the sequence is repeated one more time to check if the same number of lanes detect receiver again.

View the following signals from hierarchy ***.u_pcie_core**

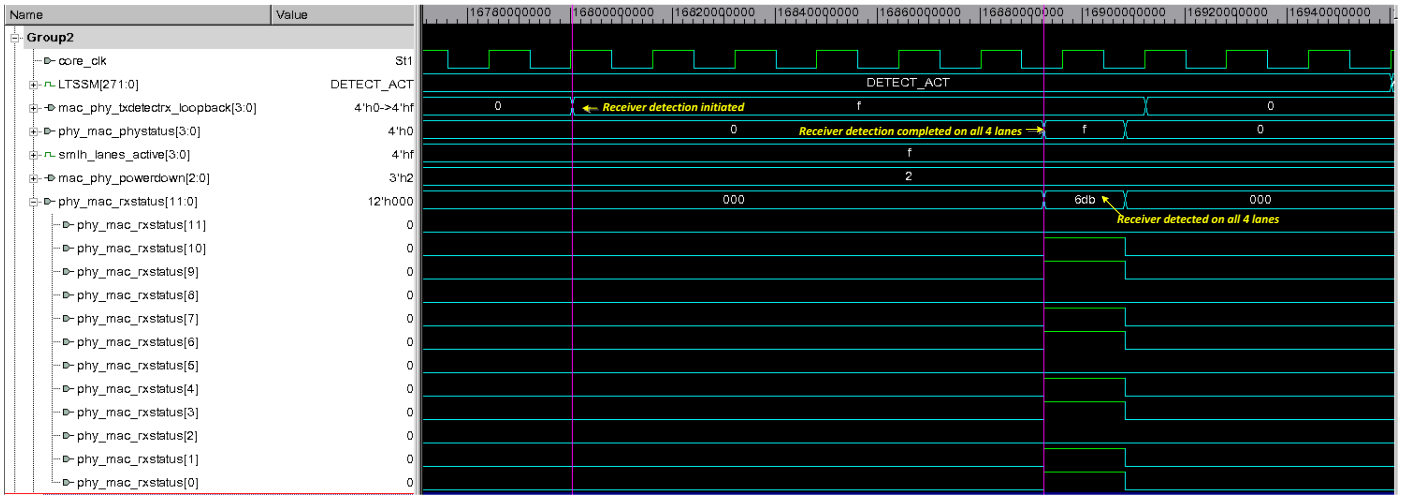
- `mac_phy_txdetectrx_loopback:` Controller asserts this signal to begin receiver detection.
- `phy_mac_phystatus:` PHY asserts this signal to indicate completion of receiver detection sequence.

Note PHY must assert this signal on all active lanes regardless of whether the lane's receiver is detected (phy_mac_rxstatus[2:0] = 3'b011) or not (phy_mac_rxstatus[2:0] = 3'b000).

- `phy_mac_rxstatus:` PHY asserts this signal to indicate receiver detected (3'b011) or receiver not preset (3'b000).

Note Ensure that the PHY model and link partner (VIP) are also operating in a fast simulation mode with the similar scaling factor as the controller. This avoids timeout mismatches which can result in unexpected behavior.

Figure F-4 Receiver Detection Completed and Receiver Detected on all Active Lanes



Frequently Asked Questions

Q: *Can the controller link up if receiver is not detected on lane0?*

A. Yes, if you have enabled the Auto Lane Flip and Reversal feature during controller configuration, (`CX_AUTO_LANE_FLIP_CTRL_EN` = 1) and receiver is not detected on lane0, the controller can automatically flip the lanes to form a narrow link.

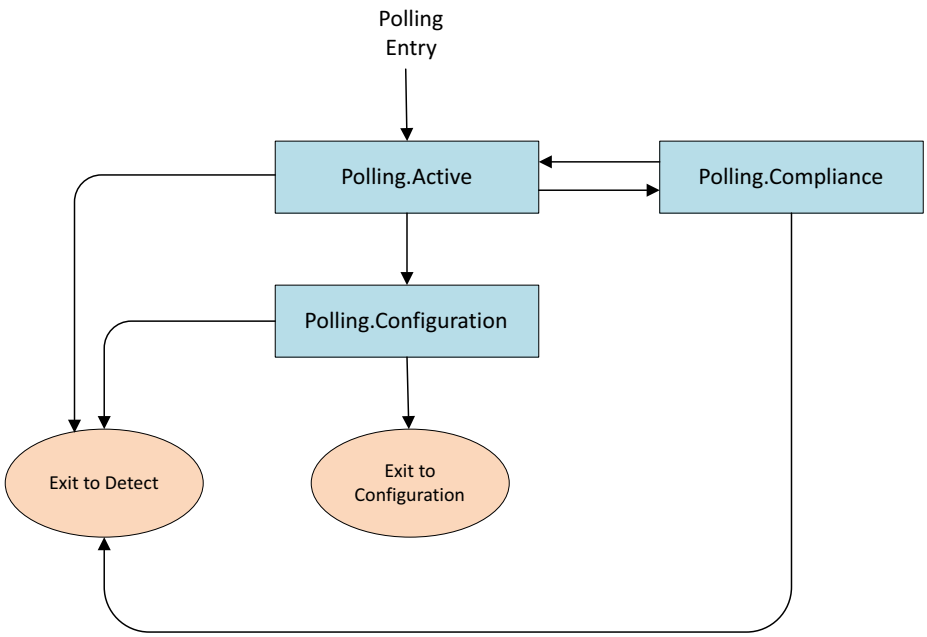
Q: *Can I restrict the maximum link width before link up? For example, can I force a x4 controller to link up as x2?*

A. Yes, you can do this by programming the `LINK_CAPABLE` field in `PORT_LINK_CTRL_OFF` to 3'b010 before link up (before `app_ltssm_enable` is asserted by application logic). This limits the number of active lanes and is reflected on the signal `smlh_lanes_active`.

F.1.3 LTSSM in Polling Substates

The *Polling* sub-state is used for establishing bit lock, symbol lock, and configuring lane polarity by the transmission and reception of Training Ordered Sets.

Figure F-5 Polling Sub-state Machine



LTSSM loops between *Polling.Active* <->*Polling.Compliance* or *Polling.Active* <-> *Detect*.

- In *Polling.Active* state the controller sends TS1 on all lanes that had detected a receiver during *Detect*-state. LTSSM moves to *Polling.Configuration* in one of the following two scenarios:
  - When all these lanes receive eight consecutive TS1/TS2 Ordered Sets, or
  - After 24 ms timeout a predetermined subset of the lanes exits electrical idle and any lane receives eight consecutive TS1/TS2 Ordered Sets.
- The LTSSM moves to *Polling.Compliance*, if all the lanes from the predetermined set of lanes (set by the `NUM_OF_LANES` field in `GEN2_CTRL_OFF` register) do not detect an exit from electrical idle. This indicates there is a problem with some lanes of the link.
- The LTSSM moves to *Detect*, if all the predetermined set of lanes exited electrical idle but did not receive the required TS Ordered Sets. This requires the link to be reset and re-trained.
- LTSSM loops between *Polling.Active* <-> *Polling.Compliance* or *Polling.Active* <-> *Detect* until the conditions to move to *Polling.Configuration* are met.

Note To prevent the LTSSM from going to *Polling.Compliance*, and allowing the link to come up at a narrow width when a subset of the lanes is broken, program the `NUM_OF_LANES` field in `GEN2_CTRL_OFF` to a smaller link width. The controller then moves to *Polling.Configuration*, if this subset of lanes exit electrical idle in *Polling.Active*.

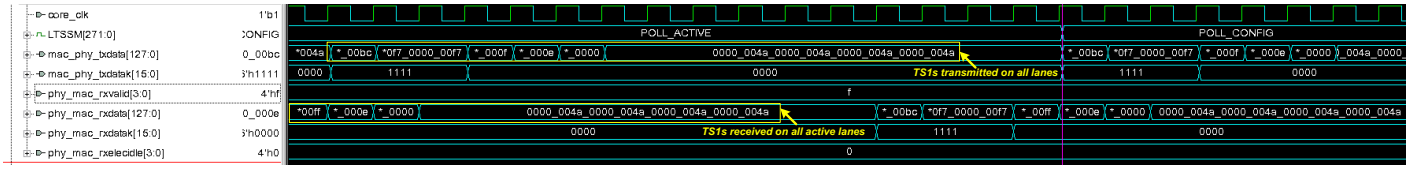
View the following signals from hierarchy ***.u_pcie_core**

- `mac_phy_txdata:` Data transmitted by the controller on the PIPE.
- `mac_phy_txdatak:` Data/Control for the symbols of transmitted data. 1-bit per byte of txdata.

- `phy_mac_rxdata:` Data received by the controller on the PIPE.
- `phy_mac_rxdatak:` Data/Control for the symbols of received data. 1-bit per byte of rxdata.
- `phy_mac_rxvalid:` Indicates symbol lock and valid data on `phy_mac_rxdata` and `phy_mac_rx-datak`. Controller samples the `phy_mac_rxdata` only when `phy_mac_rxvalid` is high.
- `phy_mac_rxelecidle:` De-assertion (1->0) of this signal indicates the receiver detected an exit from electrical Idle.

When Fast Link Mode is enabled (`cfg_fast_link_mode=1'b1`), in order to speed up simulation, in *Polling.Active* the controller sends only 16 TS1s as opposed to 1024 TS1s required by the PCIe Base Specification.

Figure F-6 Polling.Active to Polling.Configuration Normal Transition



Frequently Asked Questions

Q: *Can the controller link up if receiver is detected on Lane0 in Detect state but Lane0 is broken in Polling.Active?*

A. Yes, in this scenario the controller can form a x1 link if `GEN2_CTRL_OFF.AUTO_LANE_FLIP_CTRL_EN = 1`. Depending on the capabilities of the link partner, it may also be possible to form a multi-lane link in this scenario. To form a multi-lane link in this scenario you need to do the following:

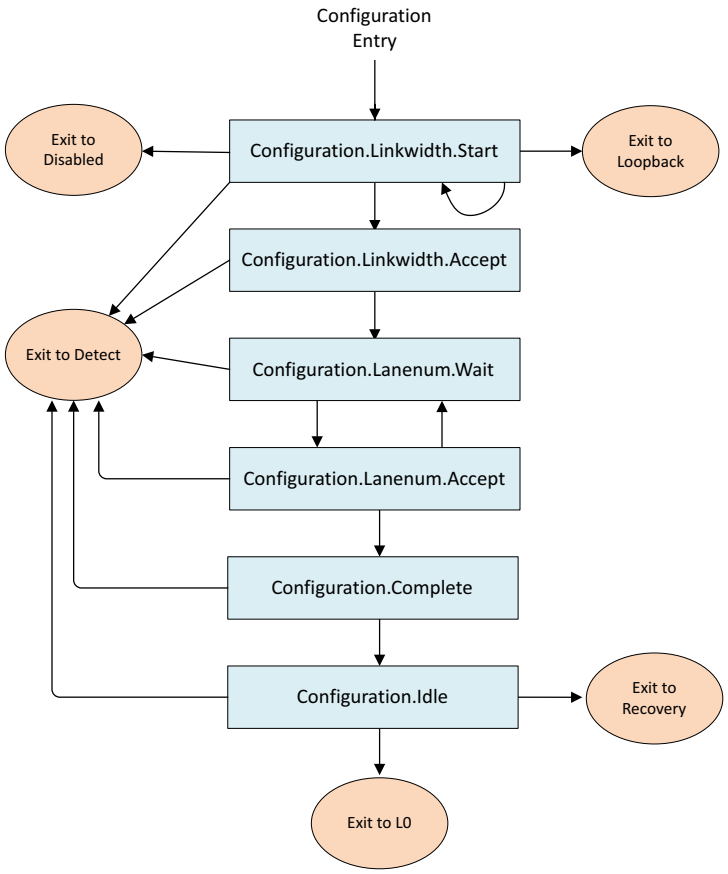
- set `GEN2_CTRL_OFF.AUTO_LANE_FLIP_CTRL_EN = 0`, and
- set input pins `tx_lane_flip_en = 1`, and
- set `rx_lane_flip_en = 1` for a manual lane flip.

Note For more information, see "Lane Reversal and Broken Lanes" appendix in the *Databook*.

F.1.4 LTSSM in Configuration Substates

In configuration substates both link partners exchange TSs to determine the link number and lane numbers. At the end of *Configuration.Idle* both sides have link number and lane numbers assigned.

Figure F-7 Configuration Sub-state Machine



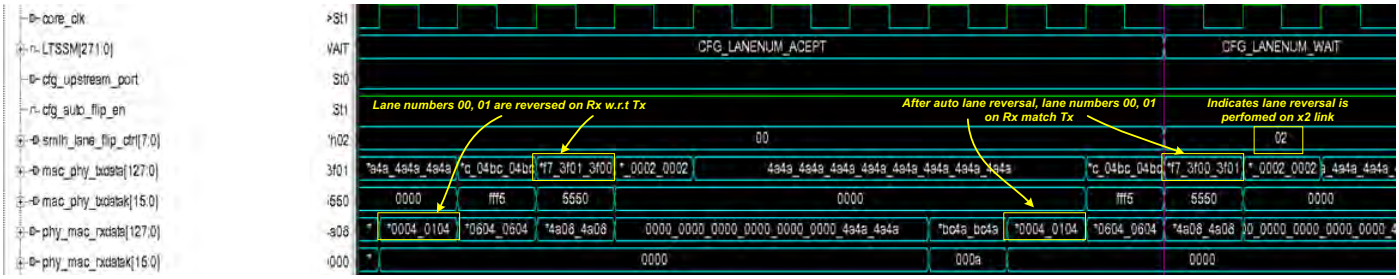
On transition to the *Configuration* state from *Polling*, any lane that detected a receiver during *Detect* is considered as an active lane. If lanes are reversed, the controller can perform lane reversal in this state. If the LTSSM is stuck in *Configuration.Lanenum.Accept* state, it is likely that it did not receive the required TS1s on the active lanes or the lane numbers do not match as expected. In this case you should check the following:

- Check that the controller is configured to support both full and partial lane reversal (`CX_AUTO_LANE_FLIP_MUX_ARCH =2`).
- Check that the `NUM_OF_LANES`, `PRE_DET_LANES`, and `AUTO_LANE_FLIP_CTRL_EN` fields in `GEN2_CTRL_OFF` are programmed to proper values as discussed in "Lane Reversal and Broken Lanes" appendix in the *Databook*.

View the following signals from hierarchy
***.u_pcie_core.u_DWC_pcie_core.u_cx_pl.u_smlh.u_smlh_ltssm**

- `cfg_upstream_port`: Shows if the controller is operating as an upstream port (EP mode).
- `cfg_auto_flip_en`: Indicates auto lane flip and reversal is enabled.
- `smlh_lane_flip_ctrl`: Indicates the number of lanes being reversed.

Figure F-8 Auto Lane Reversal During Configuration.Lanenum.Wait



In *Configuration.Complete*, both link partners transmit TS2s with LinkNum!=PAD and LaneNum!=PAD and move to *Configuration.Idle* after 8 such consecutive TS2s are received with identical data rate, link number, and lane number. In *Configuration.Idle* both link partners transmit idle data symbols on all active lanes and move to L0 if 8 idle data symbols are received.

View the following signal from hierarchy ***.u_pcie_core. u_DWC_pcie_core.u_cx_pl.u_xmlh**

- `xmtbyte_ts2_sent:` Each pulse indicates one TS2 training sequence that has been transmitted.

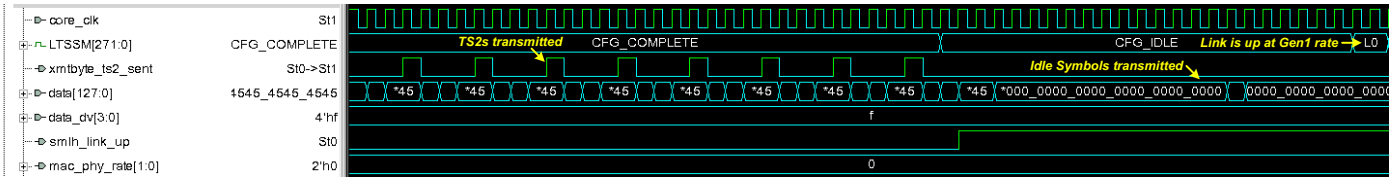
View the following signals from hierarchy ***.u_pcie_core. u_DWC_pcie_core.u_cx_pl.u_xmlh.u_scramble**

- `data[*]:` Shows the PIPE Tx data before it is scrambled by the scrambler. The transmitted data pattern can be easily identified using this signal because the data is not scrambled at this point.
- `data_dv[*]:` Data valid, one bit per DWORD of data[*].

View the following signals from hierarchy ***.u_pcie_core**

- `smlh_link_up:` Assertion of this signal indicates the link is up.
- `mac_phy_rate:` Shows the PIPE data rate.

Figure F-9 Configuration to L0 Transition



F.1.5 Smaller Than Expected Link Width

If width of the link after link up is smaller than expected, you should check the following:

- Did all expected lanes detect a receiver during *Detect* as shown in Figure F-4?

  If not, check the PHY's receiver detection logic for correct operation.
- Did all lanes exit electrical idle and receive TS1s during *Polling.Active* as shown in Figure F-6?

  If not, check connections to the PHY and max supported link width of the link partner.
- If you have reversed or broken lanes, check that your controller is configured to support these as described in section "LTSSM in Configuration Substates" on page 181.

Note For more information, see “Lane Reversal and Broken Lanes” appendix in the *Databook*.

Frequently Asked Questions

Q: *Can the link width be changed dynamically after the link is up?*

A. Yes, the controller supports dynamic link width change. For more information, see “Link Establishment” section in “Controller Operations” chapter of the *Databook*.

Note To complete dynamic link width change successfully, the `FAST_LINK_SCALING_FACTOR` field in `TIMER_CTRL_MAX_FUNC_NUM_OFF` should be set to 2’b01 or higher value when Fast Link Mode is enabled (`cfg_fast_link_mode` = 1’b1).

F.1.6 Speed Change to Gen2 Data Rate

If both the sides of the link support Gen2 data rate, after the link comes up at Gen1 data rate, either side can initiate speed change to Gen2. If speed change from Gen1 to Gen2 does not happen, you should check the following:

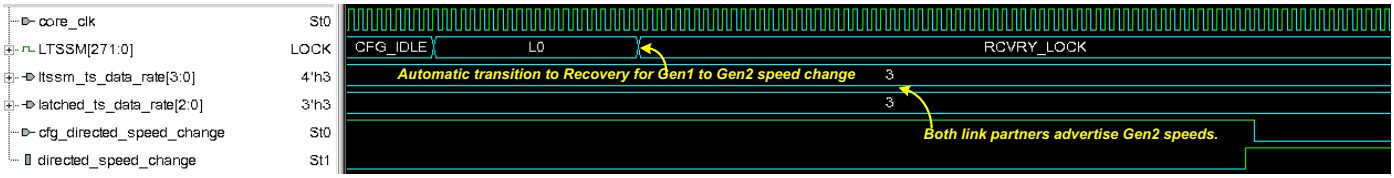
- Is controller `GEN2_CTRL_OFF[17]` (Directed Speed Change) set to 1’b1?

  This allows the controller to automatically initiate speed change from Gen1 to Gen2.
- Do both link partners advertise Gen2 support?

View the following signals from hierarchy
***.u_pcie_core.u_DWC_pcie_core.u_cx_pl.u_smlh.u_smlh_ltssm**

- `cfg_directed_speed_change`: Shows the programmed value of `GEN2_CTRL_OFF[17]` (Directed Speed Change).
- `directed_speed_change`: directed_speed_change variable as defined by the PCIe Base Specification.
- `latched_ts_data_rate`: Shows data rate advertised by the remote link partner.
- `ltssm_ts_data_rate`: Shows the data rate advertised by the controller.

Figure F-10 Gen1 to Gen2 Speed Change Initiation

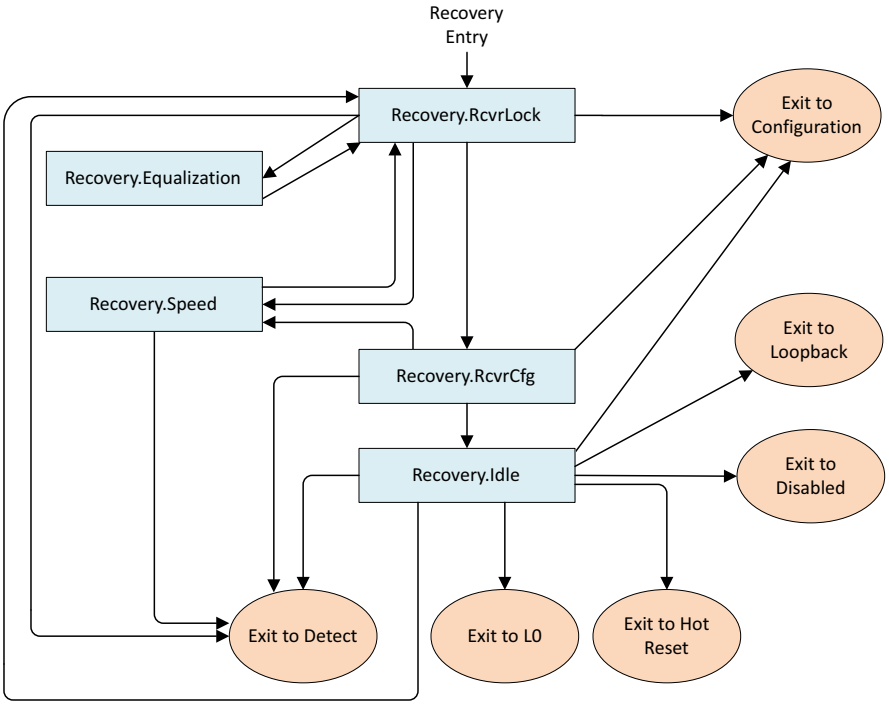


F.1.7 LTSSM in Recovery Substates

In *Recovery.RcvrLock* both link partners must receive eight consecutive TS1s to move to *Recovery.RcvrCfg*. In *Recovery.RcvrCfg* both link partners must receive eight consecutive TS2s; if directed_speed_change variable is set, LTSSM transitions to *Recovery.Speed*. In *Recovery.Speed*, controller changes the rate signal on PIPE interface and waits for PHY to acknowledge the rate change by asserting `phy_mac_phystatus` on all active

lanes. If Dynamic Frequency speed mode is used, the `core_clk` frequency switches to Gen2 frequency at this time.

Figure F-11 Recovery Sub-state Machine



View the following signals from hierarchy ***.u_pcie_core**

- `cxpl_debug_info[34]:` Each pulse indicates one TS1 training sequence received on Lane0.
- `cxpl_debug_info[33]:` Each pulse indicates one TS2 training sequence received on Lane0.

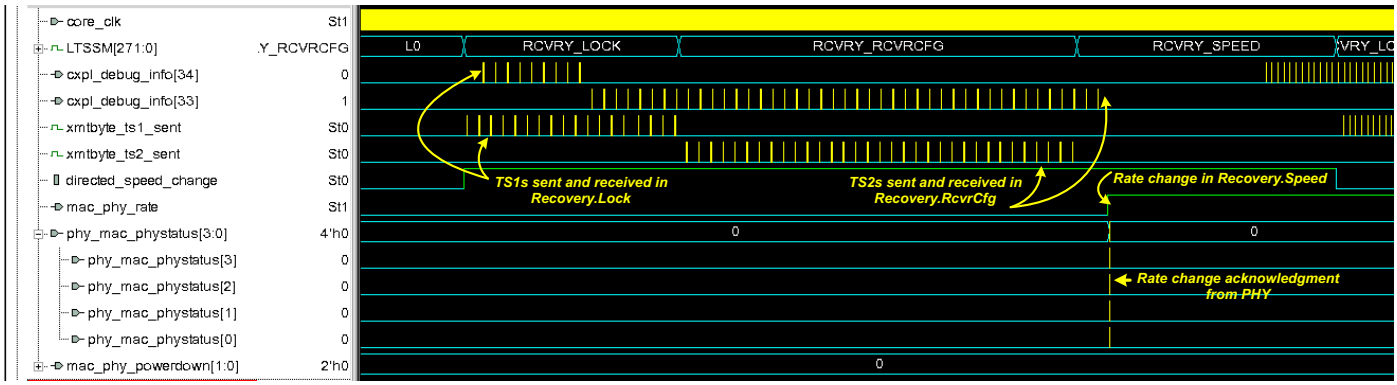
View the following signals from hierarchy ***.u_pcie_core.u_DWC_pcie_core.u_cx_pl.u_xmlh**

- `xmtbyte_ts1_sent:` Each pulse indicates one TS1 training sequence transmitted.
- `xmtbyte_ts2_sent:` Each pulse indicates one TS2 training sequence transmitted.

View the following signals from hierarchy ***.u_pcie_core**

- `mac_phy_rate:` PIPE output from controller. Changes from 0-> 1 for speed change from Gen1 to Gen2.
- `phy_mac_phystatus:` PIPE input driven by PHY. Asserted to acknowledge rate change.

Figure F-12 Speed Change from Gen1 to Gen2



After rate is changed in *Recovery.Speed*, the PHY is required to assert `phystatus` on all lanes before the 48ms timeout. When controller is in fast link mode, the 48ms timeout occurs in 48us by default. Check that the PHY is also running in fast link mode (fast simulation mode) and is able to complete rate change and assert `phystatus` within 48us. If not, reprogram the `FAST_LINK_SCALING_FACTOR` field in `TIMER_CTRL_MAX_FUNC_NUM_OFF` to reduce the scaling factor.

When fast link mode is enabled (`cfg_fast_link_mode` = 1'b1), in order to speed up simulation in *Recovery.RcvrLock*, the controller sends only 16 TS1s as opposed to 1024 TS1s required by the PCIe Base spec when extended sync bit is 1.

Figure F-13 core_clk Frequency Switching from Gen1 to Gen2

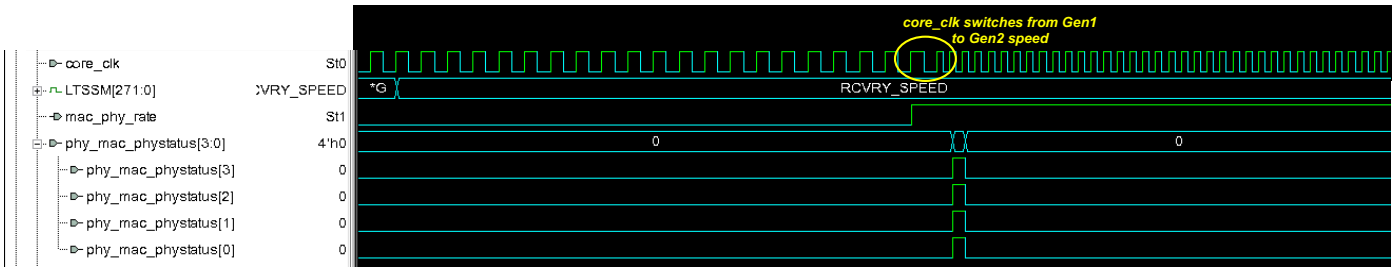
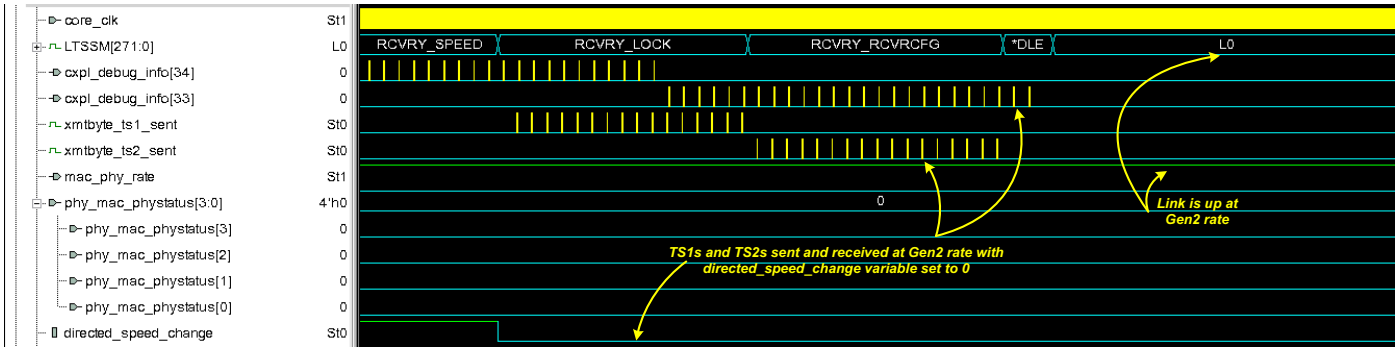


Figure F-14 Recovery to L0 Transition at Gen2 Rate



F.2 Gen3 and Above Equalization

F.2.1 Gen3 Equalization Setup Checks

- During configuration, the preset-to-coefficient mapping mode of the controller is selected using parameter `CX_GEN3_EQ_PSET_COEF_MAP_MODE`. If this is set to Dynamic PHY(0) mode, check that the following inputs and outputs of the controller are connected to the PHY.
  - `phy_mac_local_tx_pset_coef[*]`
  - `phy_mac_local_tx_coef_valid[*]`
  - `mac_phy_local_pset_index[*]`
  - `mac_phy_getlocal_pset_coef[*]`

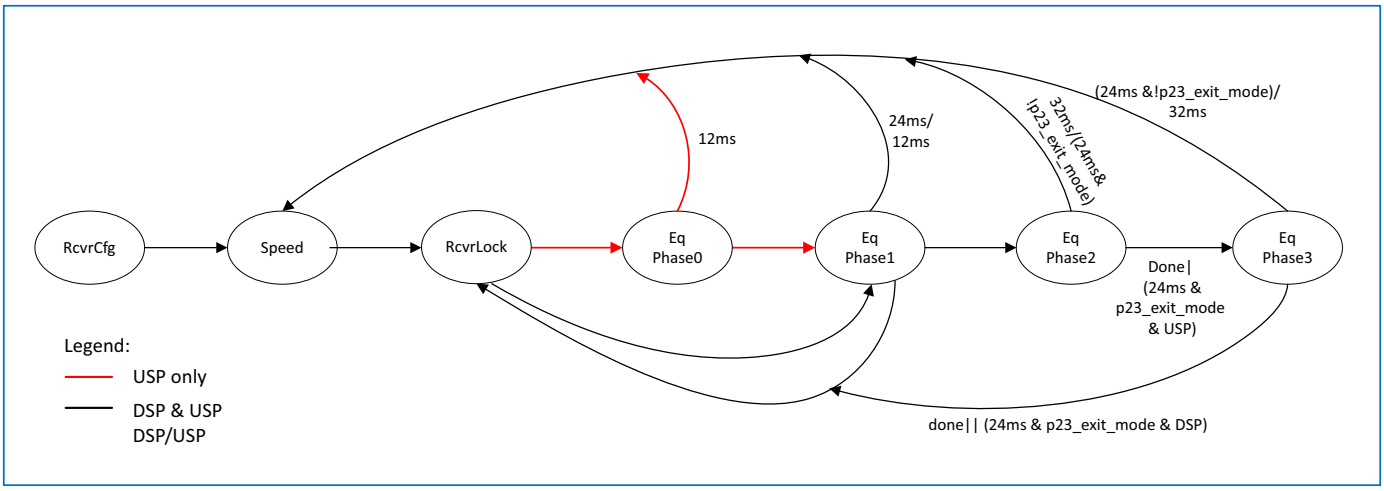
  If `CX_GEN3_EQ_PSET_COEF_MAP_MODE` is Dynamic MAC(1) or Programmable Table (2), the inputs mentioned in this section should be tied off to '0' externally and outputs should be left unconnected.
- The following inputs should be connected and driven by the PHY when Dynamic PHY or Dynamic MAC mode is used.
  - `phy_mac_localfs[*]`
  - `phy_mac_locallf[*]`

Note If the controller is configured with PIPE5 version (CX_PIPE_VER>=3), these signals does not exist. Controller implements Low Pin Count Interface and exchange equalization settings with PHY using PHY/MAC Message Bus registers using phy_mac_messagebus or mac_phy_messagebus signals.

- If `CX_GEN3_EQ_PSET_COEF_MAP_MODE` is Programmable Table(2) then your application software must program the preset-to-coefficient mapping lookup table in the controller through the port logic register `GEN3_EQ_PSET_COEF_MAP__0`.
- If your PHY uses the RxStandby PIPE signal, check that the default values of the following configuration parameters have been set to the values required by your PHY.
  - `CX_RXSTANDBY_DEFAULT`
  - `CX_RXSTANDBY_CONTROL`
- Check that the following controller PIPE signals are connected to the PHY, if these are used by your PHY.
  - `mac_phy_rxstandby`
  - `phy_mac_rxstandbystatus`
- If your controller is a downstream port, check that the following parameters have been configured to legal values; so that the upstream and downstream receiver hints and transmitter presets in the Lane Equalization Control register (`SPCIE_CAP_OFF_0CH_REG`) for each lane are set, as required by your PHY.
  - `DEFAULT_USP_RX_PRESET_HINT0`
  - `DEFAULT_USP_TX_PRESET0`
  - `DEFAULT_DSP_RX_PRESET_HINT0`
  - `DEFAULT_DSP_TX_PRESET0`

F.2.2 Speed Change to Gen3 and Above

Figure F-15 Equalization States in LTSSM



Both link partners enter *Recovery* from L0. In *Recovery.RcvrCfg,* downstream port sends eqTS2s to the upstream port with preset fields reflecting the values programmed in its secondary PCIe registers. The upstream port latches these preset values in the RO fields of its Lane Equalization Control registers.

View the following signals from hierarchy ***.u_pcie_core.u_DWC_pcie_core**

Upstream port

The signals that shows the transmitter preset and receiver preset hint captured by the upstream port from received EqTS2s are shown in Table F-1.

Table F-1 Signals that Shows the Transmitter Preset

Signal	Bits per lane	Preset	Speed Rate Change
`smlh_lec_pset_ltx[4*CX_NL-1:0]`	4	Transmitter Preset	Gen3
`smlh_lec_g4_pset_ltx[4*CX_NL-1:0]`			Gen4
`smlh_lec_g5_pset_ltx[4*CX_NL-1:0]`			Gen5
`smlh_lec_pset_lrx[3*CX_NL-1:0]`	3	Receiver Preset Hint	Gen3
`smlh_lec_g4_pset_lrx[3*CX_NL-1:0]`			Gen4
`smlh_lec_g5_pset_lrx[3*CX_NL-1:0]`			Gen5

Downstream port:

The signals that shows the upstream port and downstream port programmed in register `SPCIE_CAP_OFF_0CH_REG` are listed in Table F-2.

Table F-2 Signals that Shows the Upstream Port and Downstream Port Preset

Signal	Port	Bits per lane	Preset	Speed
`cfg_lec_pset_ltx[4*CX_NL-1:0]`	DSP	4	Transmitter Preset	8 GT/s
`cfg_lec_pset_rtx[4*CX_NL-1:0]`	USP			
`cfg_lec_pset_lrx[3*CX_NL-1:0]`	DSP	3	Receiver Preset Hint	
`cfg_lec_pset_rrx[3*CX_NL-1:0]`	USP			
`cfg_lec_g4_pset_ltx[4*CX_NL-1:0]`	DSP	4	Transmitter Preset	16 GT/s
`cfg_lec_g4_pset_rtx[4*CX_NL-1:0]`	USP			
`cfg_lec_g4_pset_lrx[3*CX_NL-1:0]`	DSP	3	Receiver Preset Hint	
`cfg_lec_g4_pset_rrx[3*CX_NL-1:0]`	USP			
`cfg_lec_g5_pset_ltx[4*CX_NL-1:0]`	DSP	4	Transmitter Preset	32 GT/s
`cfg_lec_g5_pset_rtx[4*CX_NL-1:0]`	USP			

Figure F-16 Upstream port receives Initial Tx Preset and Rx Preset Hints from Downstream Port (Gen3)

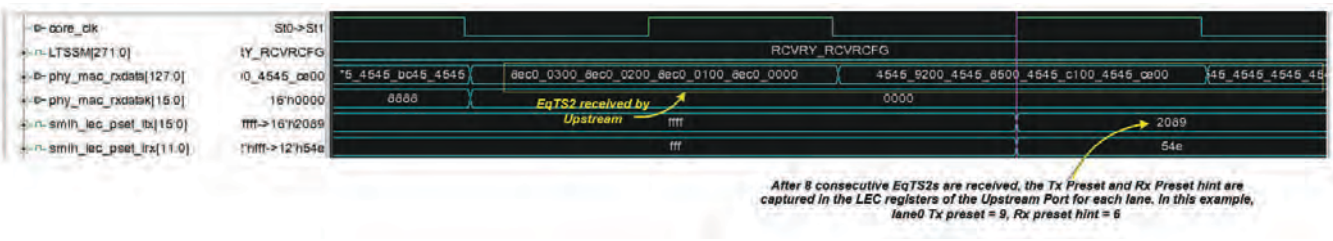
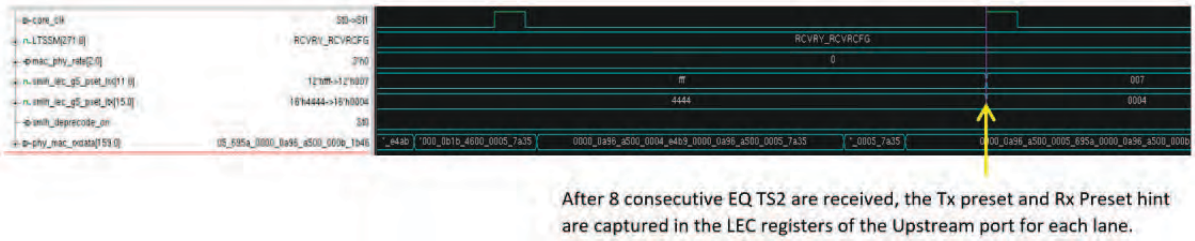


Figure F-17 Upstream port receives Initial Tx Preset and Rx Preset Hints from Downstream Port (Gen5)



In Recovery.Speed Controller initiates speed change to Gen3 or above. PHY provides the FS and LF values of its transmitter to the controller. If Dynamic PHY mapping mode is used, controller requests the PHY to map the Tx preset to coefficients. Using the FS and LF values, controller checks if the coefficients returned by the PHY are legal.

View the following signals from hierarchy ***.u_pcie_core.u_DWC_pcie_core**

- `phy_mac_locallf:` LF value of the PHY's transmitter (6-bits per lane).
- `phy_mac_localfs:` FS value of the PHY's transmitter (6-bits per lane).

- `mac_phy_getlocal_pset_coef`: Asserted by controller to request PHY to perform preset-to-coefficient mapping (1-bit per lane).
- `mac_phy_local_pset_index`: Preset to be mapped (4-bits per lane).
- `phy_mac_local_tx_coef_valid`: Asserted by PHY to return mapped coefficient (1-bit per lane).
- `phy_mac_local_tx_pset_coef`: Coefficients mapped to the preset (18-bits per lane in the format phy_mac_local_tx_pset_coef[17:0] = {post-cursor, cursor, pre-cursor}).
- `mac_phy_txdeemph`: If the mapped co-efficients are legal, controller reflects the same value on this output to the PHY.

In Recovery.Lock,

- Both link partners transmit ec0TS1s (TS1 with Symbol6[1:0] = Equalization Control = 2'b00).
- LTSSM determines if equalization procedure must be performed.

Figure F-18 First Speed Change to Gen3 Data Rate

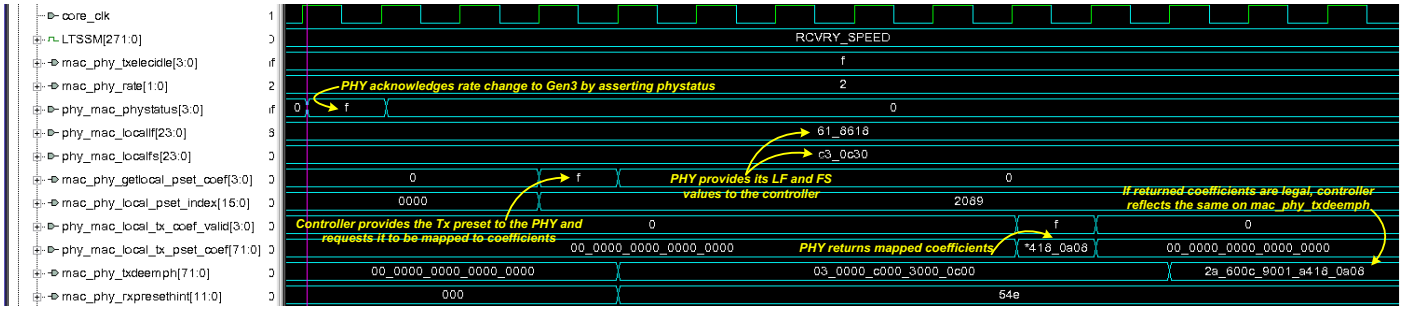
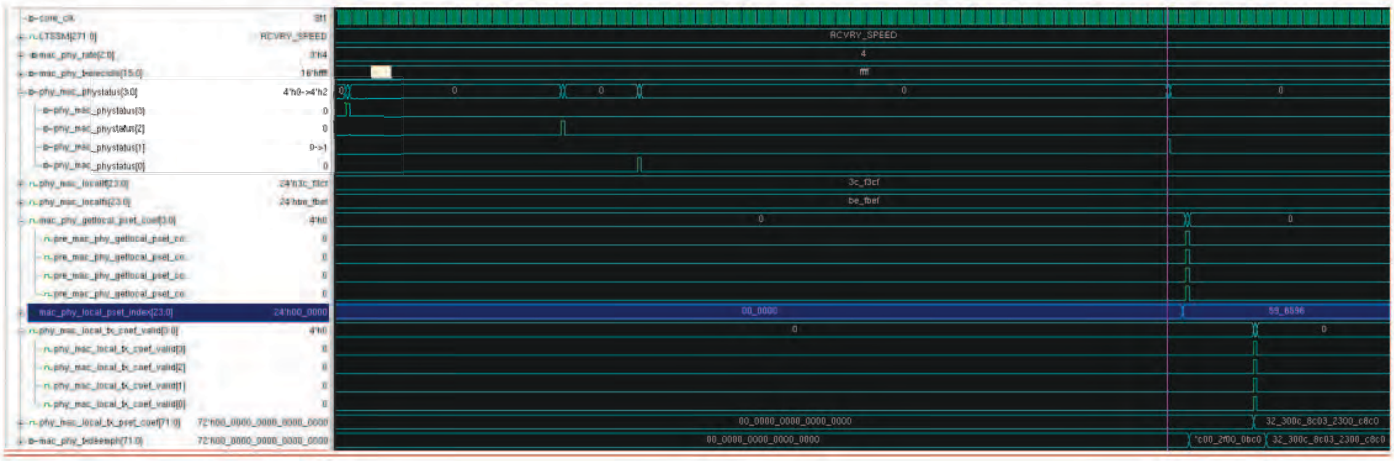


Figure F-19 Speed Change to Gen5 Data Rate



Frequently Asked Questions

Q: *What happens when illegal coefficients are returned by the PHY during Rcvry.Speed?*

A. If the returned coefficients are not legal, the `mac_phy_txdeemph` output is driven with the co-efficients corresponding to Preset 4 (6'b0, `phy_mac_localfs`[5:0], 6'b0) and equalization continues.

Q: *Can equalization be disabled in the controller for debug purposes?*

A. Yes, local software can set bit 16 of `GEN3_RELATED_OFF` to disable Gen3 equalization. The controller bypasses the equalization procedure and linkup at Gen3.

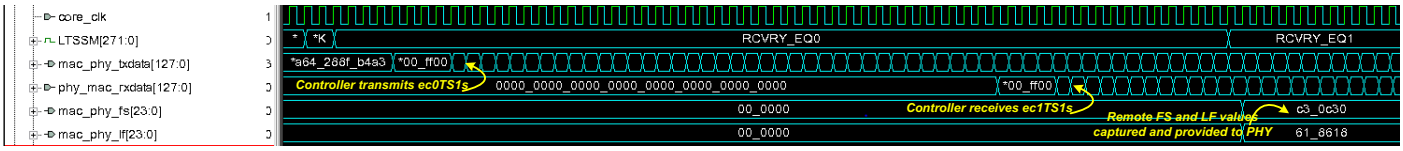
F.2.3 Equalization Phase0

Upstream port transmits ec0TS1s, with the coefficients in use by the transmitter and the initial Tx presets. Downstream port is in Phase1 and transmits ec1TS1s (TS1 Symbol6[1:0] = Equalization Control = 2'b01) with its own LF and FS values. Upstream port captures these LF and FS values and reflects them on `mac_phy_fs` and `mac_phy_lf` outputs to the PHY. PHY can use these to adjust the remote transmitter settings in Phase2. Upstream Port proceeds to Phase1 after receiving two ec1TS1s.

View the following signals from hierarchy: ***.u_pcie_core**

- `mac_phy_lf:` LF value of remote link partner (6-bits per lane).
- `mac_phy_fs:` FS value of remote link partner (6-bits per lane).

Figure F-20 Upstream Port Equalization.Phase 0



F.2.4 Equalization Phase1

Upstream port transmits ec1TS1s with the FS and LF values corresponding to the lane's current transmitter settings. Both link partners move to Equalization.Phase2 after two consecutive ec1TS1s are received.

F.2.5 Equalization Phase2/Phase3

Check the following characteristics of the PHY you are using:

- Does your PHY support Figure of Merit feedback mode?

  If it does, check with your PHY vendor which presets should be selected for evaluation in Phase2. Check if the Preset Request Vector field of `GEN3_EQ_CONTROL_OFF`[23:8] is programmed to select the required presets.

- Does your PHY require more than 2ms (2us in simulation or more depending on `FAST_LINK_SCALING_FACTOR`) to complete Preset/Co-efficient evaluation?

  If it does, you can program `GEN3_EQ_CONTROL_OFF`[5] to disable the 2ms timeout and allow more time for the PHY to complete evaluation.

- Does your PHY support Direction Change Fine Tuning?

  If it does not, program the `GEN3_EQ_CONTROL_OFF`[3:0] (Feedback Mode) to 0001b: Figure Of Merit.

  During Phase2, upstream port is the EQ Master and it fine tunes the downstream port's remote transmitter. During Phase 3, downstream port is the EQ Master and it fine tunes the upstream port's remote transmitter. Figure of Merit evaluation is performed first followed by Direction Change fine tuning, if enabled. Equalization is performed on each lane independently; each lane has a separate EQ master state machine.

  Figure F-20 show an Upstream port, lane 0 example. View the following signals from hierarchy *.u_pcie_core.u_DWC_pcie_core.u_cx_pl.u_smlh.u_smlh_eqpa.u_smlh_eqpa_slv[0]

- `EQ_MASTER_STATE:` Equalization Master State Machine state for lane0. Set the radix of this bus to ASCII for easy readability.
- `cfg_eq_pset_req_vector:` Shows the value programmed in Preset Request Vector field in `GEN3_EQ_CONTROL_OFF[23:8]`.

View the following signals from hierarchy ***u_pcie_core**

- `mac_phy_rxeqeval:` Controller asserts this signal to request PHY to perform a Preset/Co-efficient evaluation
- `mac_phy_rxeqinprogress:` Controller keeps this signal asserted for entire duration of Eq.Phase2/Eq.Phase3
- `mac_phy_invalid_req:` Controller asserts this signal to indicate to the PHY, that the feedback on latest evaluation generated an invalid preset or set of co-efficients.
- `phy_mac_fomfeedback:` 8-bit FOM feedback from local PHY.
- `phy_mac_dirfeedback:` Feedback from local PHY to fine tune the coefficients. 6-bits per lane.

Figure F-21 Upstream Port Equalization.Phase 1: Selected Preset Request

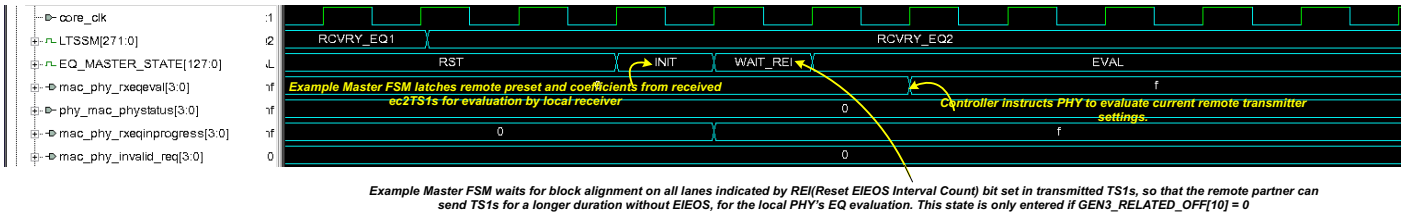
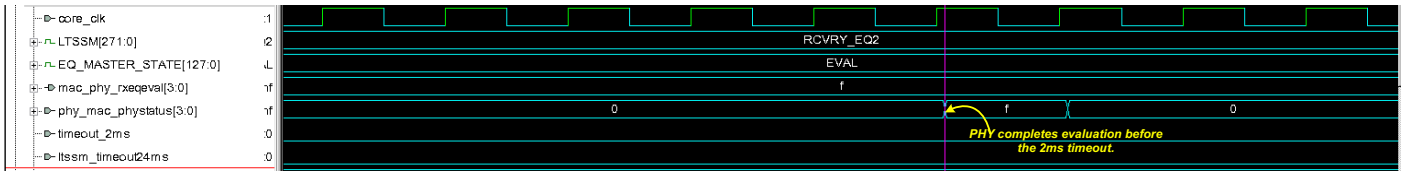


Figure F-22 Upstream Port Equalization.Phase 1: Preset Request Accepted



- Does your PHY require more than 2ms (2us in simulation) to complete Preset/Co-efficient evaluation?

  If it does, you can program GEN3_EQ_CONTROL_OFF[5] to disable the 2ms timeout and give more time for the PHY to complete evaluation. By default, GEN3_EQ_CONTROL_OFF[5] is 0, if phystatus is not asserted within 2ms, evaluation aborts and Phase2 exits unsuccessfully.

Figure F-23 Upstream Port Equalization.Phase 2: Selected Preset Request

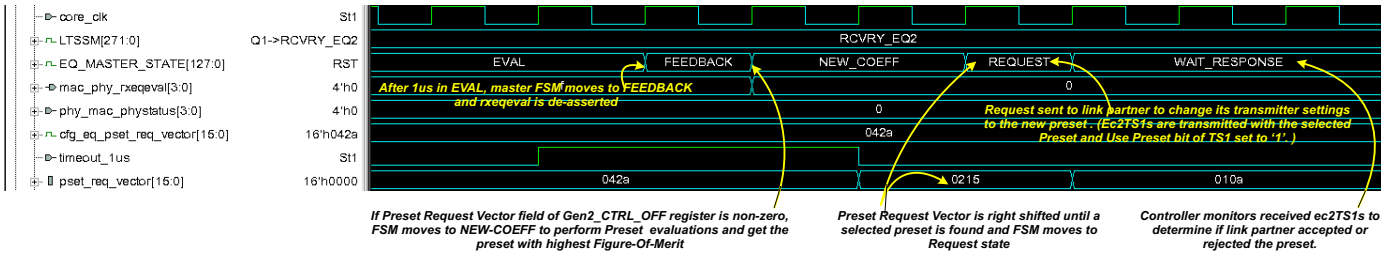
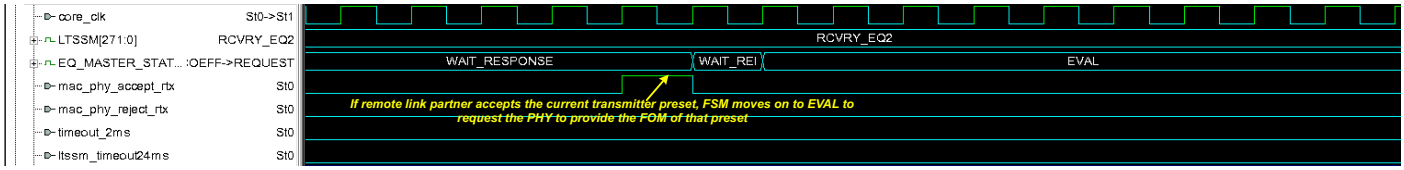


Figure F-24 Upstream Port Equalization.Phase 2: Preset Request Accepted



Frequently Asked Questions

Q: *What happens if the remote link partner rejects the current transmitter preset?*

A. The current evaluation is skipped and controller moves on to evaluate the next selected preset from the Preset Request Vector.

Q: *What happens if controller does not receive the ec2TS1s in WAIT_RESPONSE state until 2ms timeout occurs?*

A. Controller skips the current EQ evaluation and send the next selected preset. This process continues until all the selected presets are sent. Else, after a 24ms timeout, Phase2 exits unsuccessfully.

If the 24ms timeout occurs, for debug purposes you can program the GEN3_EQ_CONTROL_OFF[4] (Behavior After 24 ms Timeout) to '1'. This forces the controller to continue link training as normal (LTSSM moves to Eq.Phase3 if USP, Recovery.RcvrLock if DSP).

Figure F-25 Upstream Port Equalization.Phase 2: Figure-Of-Merit Feedback

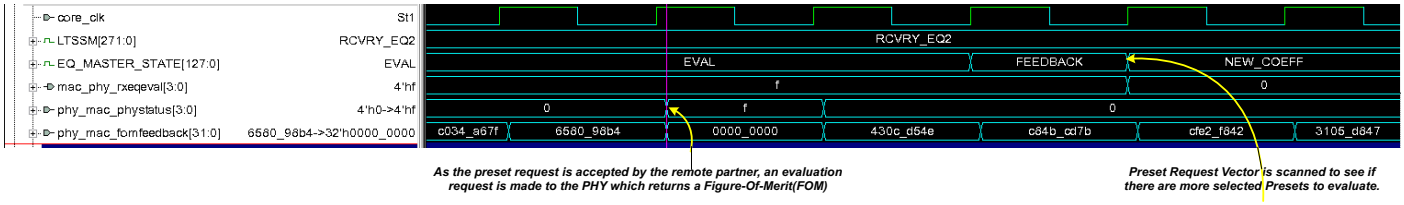
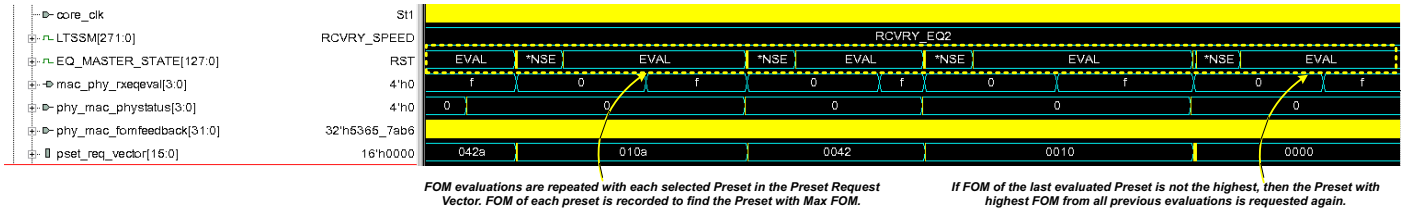


Figure F-26 Upstream Port Equalization.Phase 2: Selected Presets Evaluated



Phase2/3 FOM evaluation always begins with the unmodified transmitter settings (Initial Tx Preset received from DSP) and then moves on to evaluation of Presets selected in the Vector. You can choose to include the FOM of the Initial Preset in the Max FOM calculation by programming `GEN3_EQ_CONTROL_OFF[24]` to '1'.

Figure F-27 Upstream Port Equalization.Phase 2: Co-efficient Request

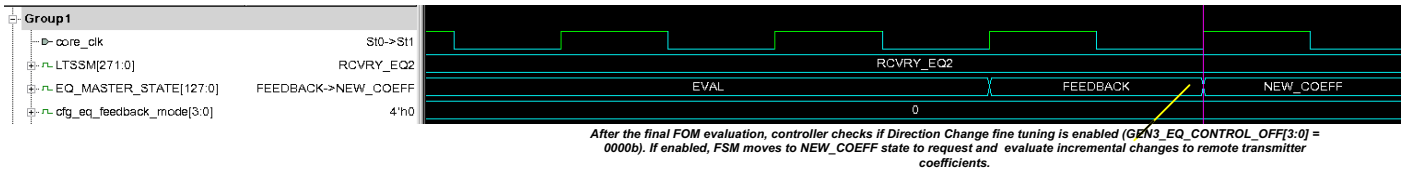


Figure F-28 Upstream Port Equalization.Phase 2: Direction Change Fine Tuning

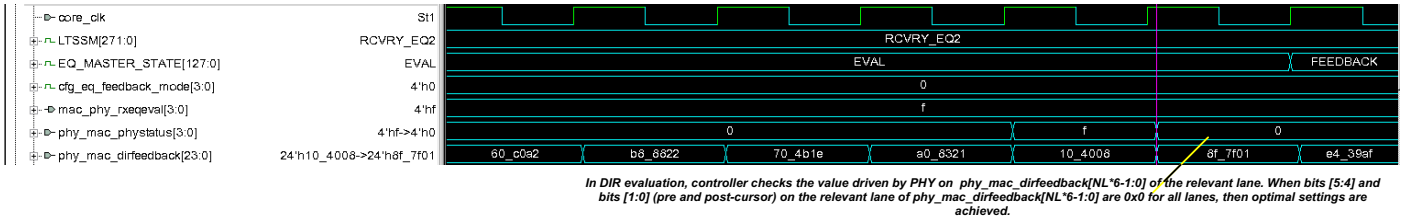
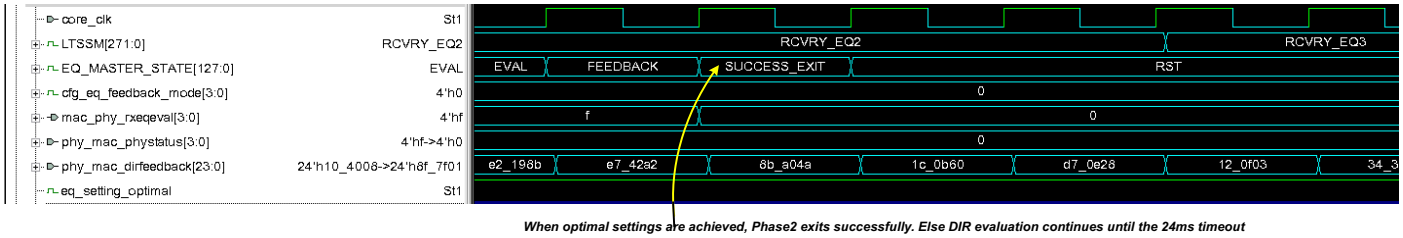


Figure F-29 Upstream Port Equalization.Phase 2: Successful Completion



Q: *What is the impact of setting CX_GEN3_EQ_COEF_CONV_SUPPORTED parameter to '1'?*

A. When this parameter is set to '0', the criteria for optimal settings is that the pre and post cursor on the relevant lane of `phy_mac_dirfeedback` is 0x0.

Whereas when this parameter is set to '1', the criteria for optimal settings is more relaxed. It requires that the coefficients of the remote transmitter, requested and evaluated in the last "N" attempts have a maximum delta that does not exceed a value "D". For more information, see "DIR Convergence Criteria" section in "Gen3/4/5 Equalization Details and Example" appendix of the *Databook*.

Q: *What happens if the co-efficients do not converge (optimal settings not achieved) after 24ms timeout?*

A. By default this is considered as Equalization failure, the LTSSM moves back to Recovery.Speed and link speed falls back to Gen1. But you can change this behavior by programming the `GEN3_EQ_CONTROL_OFF[4]`(Behavior After 24 ms Timeout) to '1'. This forces the controller to continue link training as normal (LTSSM moves to Eq.Phase3 if USP, Recovery.RcvrLock if DSP).

Q: *If equalization fails and link falls back to Gen1 speed, will it automatically re-initiate speed change to Gen3?*

A. No, in order to re-initiate speed change to Gen3, software should reprogram the `GEN2_CTRL_OFF[17]` (Directed Speed Change) to '1', because this field gets automatically cleared after every speed change.

Q: *Is Gen3 Equalization performed every time data rate changes to 8GT/s?*

A. The equalization procedure is executed only during the first data rate change to 8GT/s.

Note The equalization transition can be repeated in the autonomous equalization sequence by using Request Equalization bits in the TS2s.

After the first data rate change to 8GT/s, software can execute equalization procedure by setting the following in downstream port.

- `LINK_CONTROL3_REG. PERFORM_EQ` = 1
- `LINK_CONTROL2_LINK_STATUS2_REG. PCIE_CAP_TARGET_LINK_SPEED` = 4'h3
- `LINK_CONTROL_LINK_STATUS_REG. PCIE_CAP_RETRAIN_LINK`= 1

Note For more information, see “Gen3/4/5 Equalization Details and Example” appendix in the *Databook*.

F.3 Gen5 and Above Features

For Gen5 and above data rate controller supports the following features which impacts the equalization process.

F.3.1 Equalization Bypass to Highest Data Rate

You can enable the Equalization Bypass to Highest Data Rate feature by setting the `CX_SKIP_LOWER_RATE_EQ` parameter to '1'. If this feature is enabled and controller receives TS1 from remote partner in Configuration state with Symbol 5 (bit[7:6]) set as 01b, link bypass Gen3/Gen4 equalization and directly performs Gen5 equalization. Otherwise the link goes through speed change and equalization steps as described in the above section for Gen3->Gen4->Gen5 data rates after initial Gen1 link up. The default value of the `CX_SKIP_LOWER_RATE_EQ` parameter is '1'.

Figure F-30 Link Up Sequence When Equalization Bypass to Highest Data Rate is Disabled

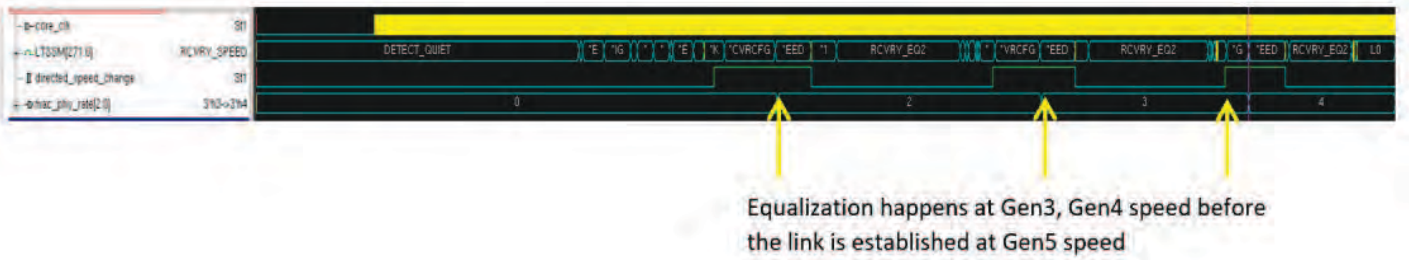
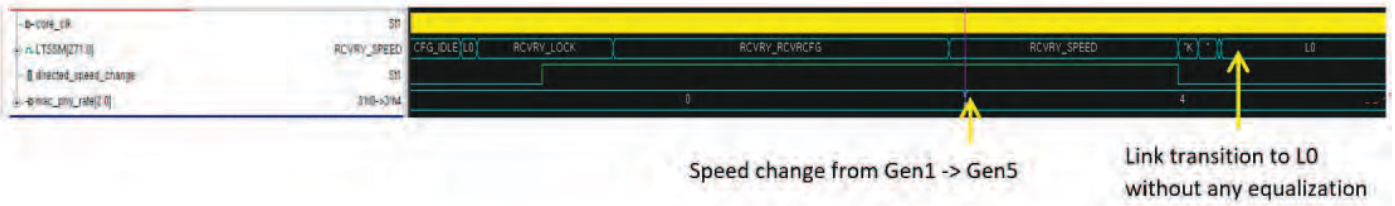


Figure F-31 Link Up Sequence When Equalization Bypass to Highest Data Rate is Enabled

F.3.2 No Equalization Needed

You can enable the No Equalization Needed feature by setting the `CX_NO_EQ_NEEDED` parameter to '1'. If this feature is enabled and controller receives 2 consecutive TS1 ordered set from remote partner in configuration state with Symbol 5 (bit[7:6]) set as 10b, link need not perform equalization at any rate. The link can transition directly to L0 after the *Recovery.Speed* state.

Figure F-32 Link Up Sequence When No Equalization Needed is Enabled



F.3.3 Precoding

The precoding feature is enabled using `CX_DEFAULT_TX_PRECODE_REQ` parameter. When it is enabled, controller requests the link partner to use precoding by setting bit0 of Symbol 6 in the `EQ TS` transmitted prior to entry to *Recovery.Speed* state. The RxData received by controller is deprecoded before descrambling. The same applies to transmit data, if the link partner sends `EQ TS` with `CX_DEFAULT_TX_PRECODE_REQ` register bit set.

When `CX_NO_EQ_NEEDED` feature is enabled the controller uses sticky precoding value based on the last equalization or on the value set during `core_rst_n`.

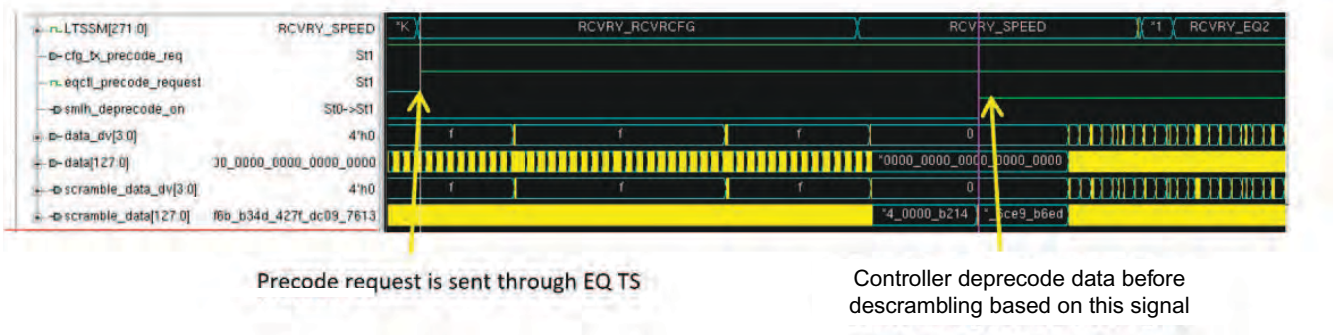
View the following signals from hierarchy ***.u_pcie_core.u_DWC_pcie_core.u_cx_pl.u_smlh**

- `cfg_tx_precode_req`: This signal is based on `TX_PRECODE_REQ` register setting.
- `eqctl_precode_request`: This signal is set when the controller sends `EQ TS` with Precoding Request bit as 1.
- `smlh_deprecode_on`: This signal controls deprecode at Rx side before descrambling the data.
- `smlh_precoding_on`: This signal controls precoding at Tx side after scrambling the data.

View the following signals from hierarchy ***.u_DWC_pcie_core.u_cx_pl.u_rmlh.u_scramble**

- `data[*]`: Scrambler Input data received at PIPE Rx side.
- `data_dv`[*]: Data valid, one bit per DWORD of `data[*]`.
- `scramble_data[*]`: Scrambler output data after descrambling and deprecoding PIPE Rx Data.
- `scramble_data_dv[*]`: Output data valid, one bit per DWORD of `scramble_data[*]`.

Figure F-33 RxData Decoding When Precoding Request is Enabled



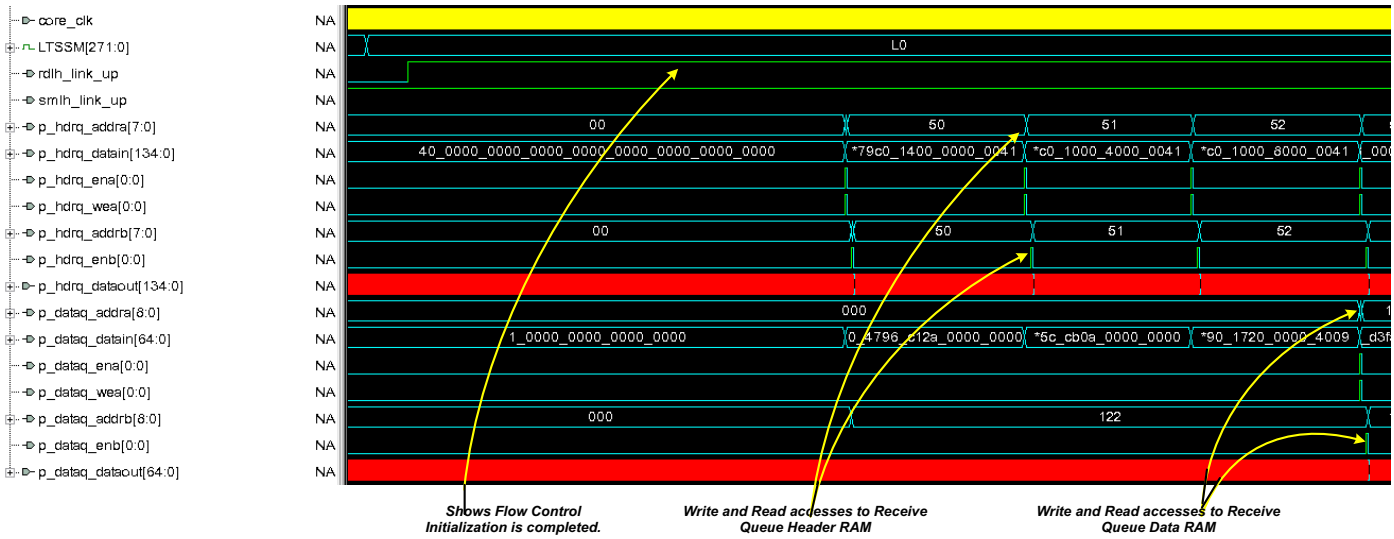
F.4 Upstream and Downstream Port Enumeration

F.4.1 Upstream Port: No Completion Transmitted in Response to Inbound Cfg TLP

The controller generates Completions (CPLs) in response to received Cfg TLPs. If this does not happen in your simulation, check the following:

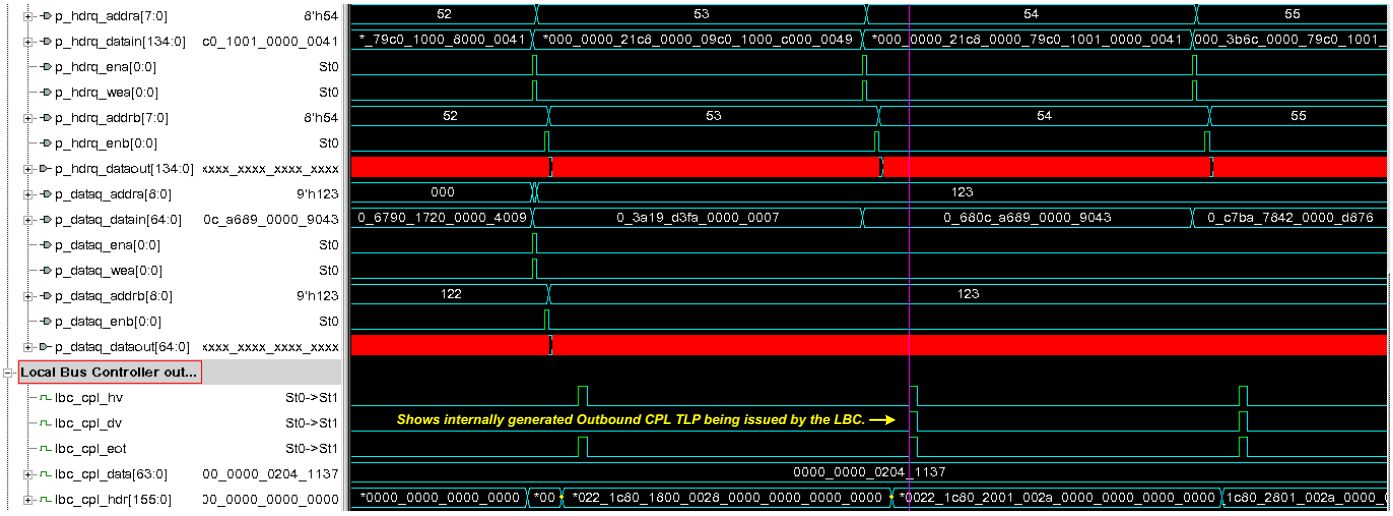
- Check that the link is up, and flow control initialization has completed.
- View the following signals from hierarchy ***.u_pcie_core**
  - `smlh_link_up`: Indicates link is up.
  - `rdlh_link_up`: Indicates that data link layer has completed Flow Control Initialization.
- Check that the receive queue RAM interfaces of the controller (`p_hdqr_*`, `p_dataq_*`) are connected correctly to your RAM model as shown in the reference top level file `pcie_iip_device.sv` (See "Top Level Reference File Generation" on page 206)
- Check that the RAM model is receiving `core_clk` and the clock is running at expected frequency.
- Check receive queue RAM interface to see if write and read accesses are happening.
- View the following signals from hierarchy ***.u_pcie_core**
  - `p_hdrq_*`: Receive Header Queue RAM interface.
  - `p_dataq_*`: Receive Data Queue RAM interface.

Figure F-34 Write and Read Accesses to Receive Queue RAMs



- View the following signals from hierarchy ***.u_pcie_core**
  - `lbc_cpl_*`: Internal interface of Local Bus Controller (LBC) module that transmits internally generated Completions.

Figure F-35 Outbound Completion Generated and Issued by Local Bus Controller



F.4.2 Downstream Port: Outbound CFG TLP not Transmitted

The controller transmits outbound Cfg TLPs when your application issues them on the client* interface or AXI Slave interface. If the Cfg request issued by your application is not transmitted by the controller on the PCIe wire, check the following.

- Check that the link is up, and flow control initialization has completed.
- If using AXI interface, check that all the AXI bridge RAM interfaces of the controller are connected to the RAM models as shown in the reference top level file `pcie_iip_device.sv` (See "Top Level Reference File Generation" on page 206).
- Check that all RAMs are receiving the correct clocks.
- If using iATU to translate an AXI request to a Cfg TLP, check that the iATU registers are programmed correctly as described in "Internal Address Translation Unit (iATU)" section in "Controller Operations" chapter of the *Databook*.
- If you are using the sideband bus `slv_a*misc_info` instead of iATU to issue the outbound Cfg request, check that the `slv_a*misc_info`[4:0] signals are driven correctly to indicate Type "Cfg".

Figure F-36 Outbound Cfg TLP Issued on AXI Slave Interface

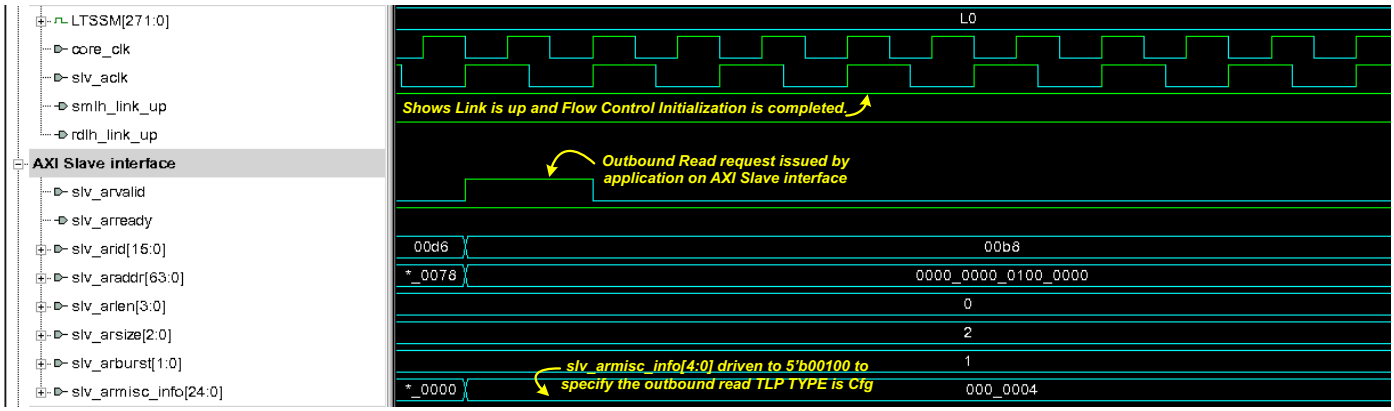
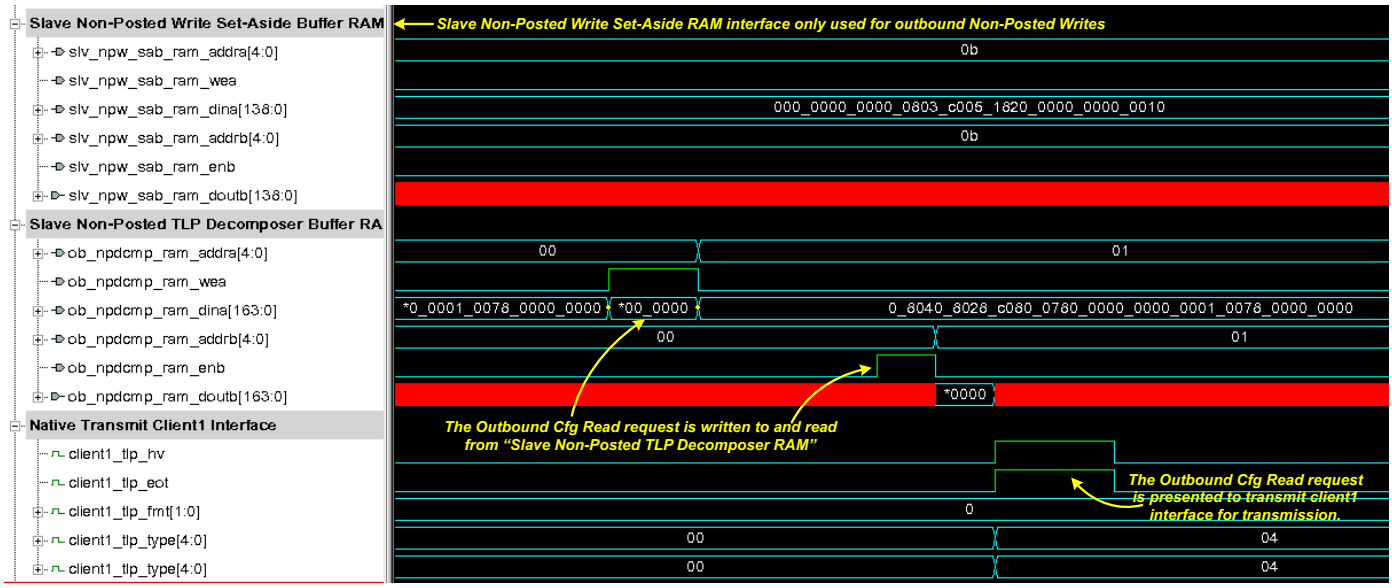


Figure F-37 Accesses to AXI Bridge Slave RAMs on the Outbound Tx Path



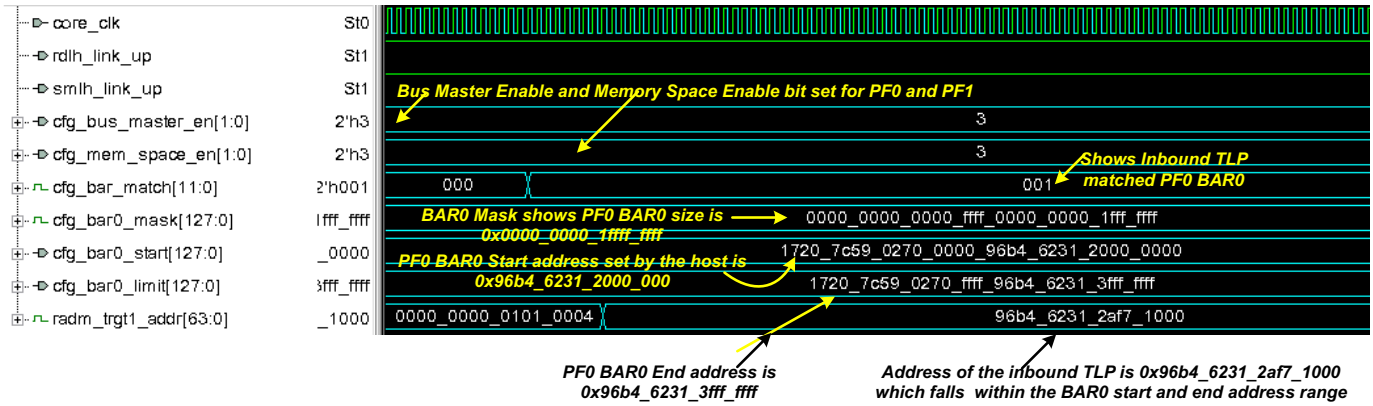
F.5 Memory Transactions

F.5.1 Inbound Transaction: MWr/MRd TLP not Seen on Application Interface

The controller delivers all error-free MWr/MRd TLPs to your application through the `radm_trgt1*` interface (native controller) or AXI Master interface `mstr_a*`. If an inbound TLP is not delivered on the application interface, check the following.

- Check that the link is up, and flow control initialization has completed.
- View the following signals from hierarchy ***.u_pcie_core**
  - `smlh_link_up:` Indicates link is up.
  - `rdlh_link_up:` Indicates that data link layer has completed Flow Control Initialization.
- Check that the Memory Space Enable (MSE) and Bus Master Enable (BME) bits of the Command register in the controller are set before the MWr/MRd TLPs are received.
- View the following signals from hierarchy ***.u_pcie_core**
  - `cfg_bus_master_en:` Shows value of the BME bit per physical function.
  - `cfg_mem_space_en:` Shows value of the MSE bit per physical function.
- If your controller is an Upstream Port, check that the Inbound TLP is addressed to a location within the enabled BARs.
- View the following signals from hierarchy ***.u_pcie_core**
  - `cfg_bar_match:` Shows the BAR hit by the inbound TLP. This bus has 6-bits (bar5-bar0) per physical function.
  - `cfg_bar0_mask:` Shows the value of the BAR0 Mask register which determines the size of the BAR.
  - `cfg_bar0_start:` Shows the start address of BAR0 set by the host during enumeration.
  - `cfg_bar0_limit:` Shows the end address of BAR0 calculated from the start address and BAR size.
  - `radm_trgt1_addr:`Shows the address of the inbound TLP.

Figure F-38 Successful BAR Match for Inbound Memory TLP



- If your controller is a Downstream Port, check that the Inbound TLP is addressed to a location outside the range specified by the Memory Base and Limit register and outside the range specified by the Prefetchable Memory Base and Limit registers of the Downstream port's configuration space.
- If you are using iATU to translate an inbound PCIe TLP to an AXI request, check that the iATU registers are programmed correctly as described in "Internal Address Translation Unit (iATU)" section in "Controller Operations" chapter of the *Databook*.
- Check that the receive queue RAMs and AXI bridge RAMs (if using AXI interface) are connected correctly to the controller's RAM interfaces and the RAMs are receiving the correct clocks are shown in reference top level file `pcie_iip_device.sv` (See "Top Level Reference File Generation" on page 206).

Figure F-39 RAM Accesses Along the Rx Path of MWr TLP

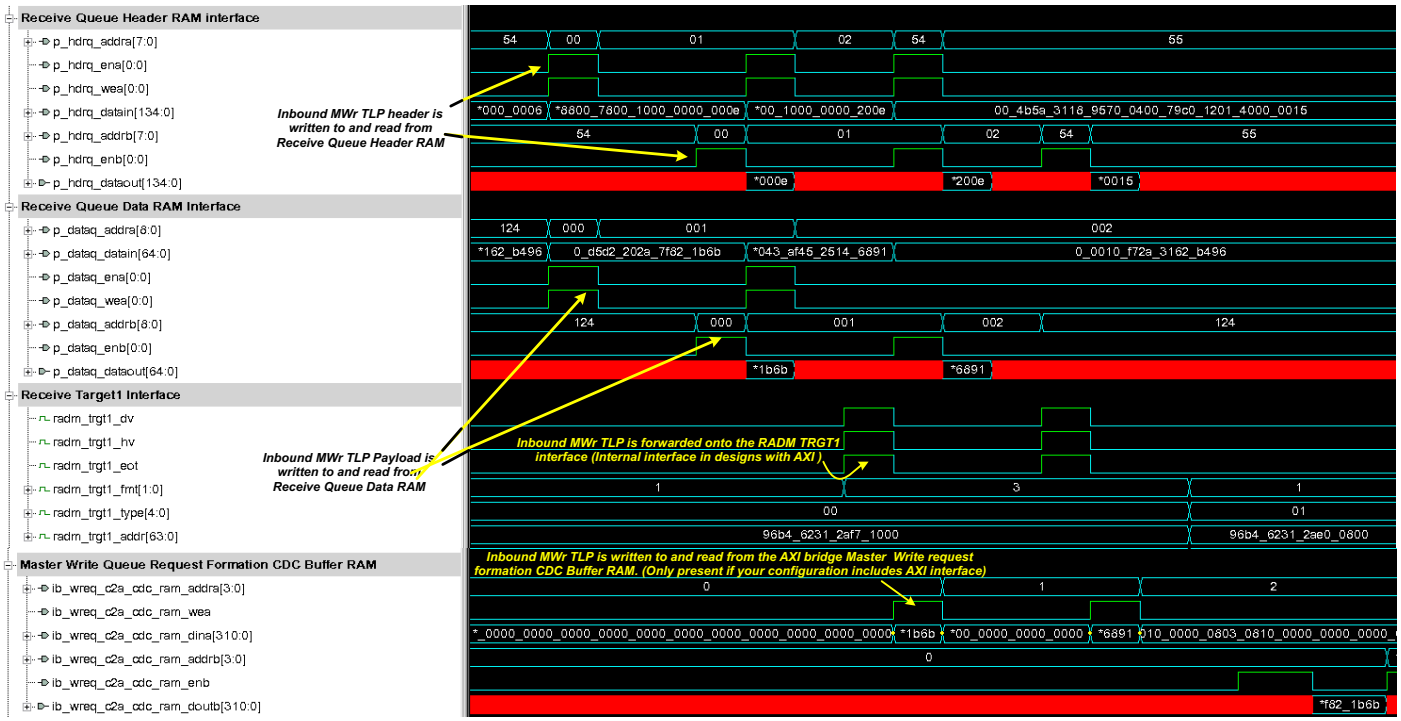


Figure F-40 RAM Accesses Along the Rx Path of MRd TLP

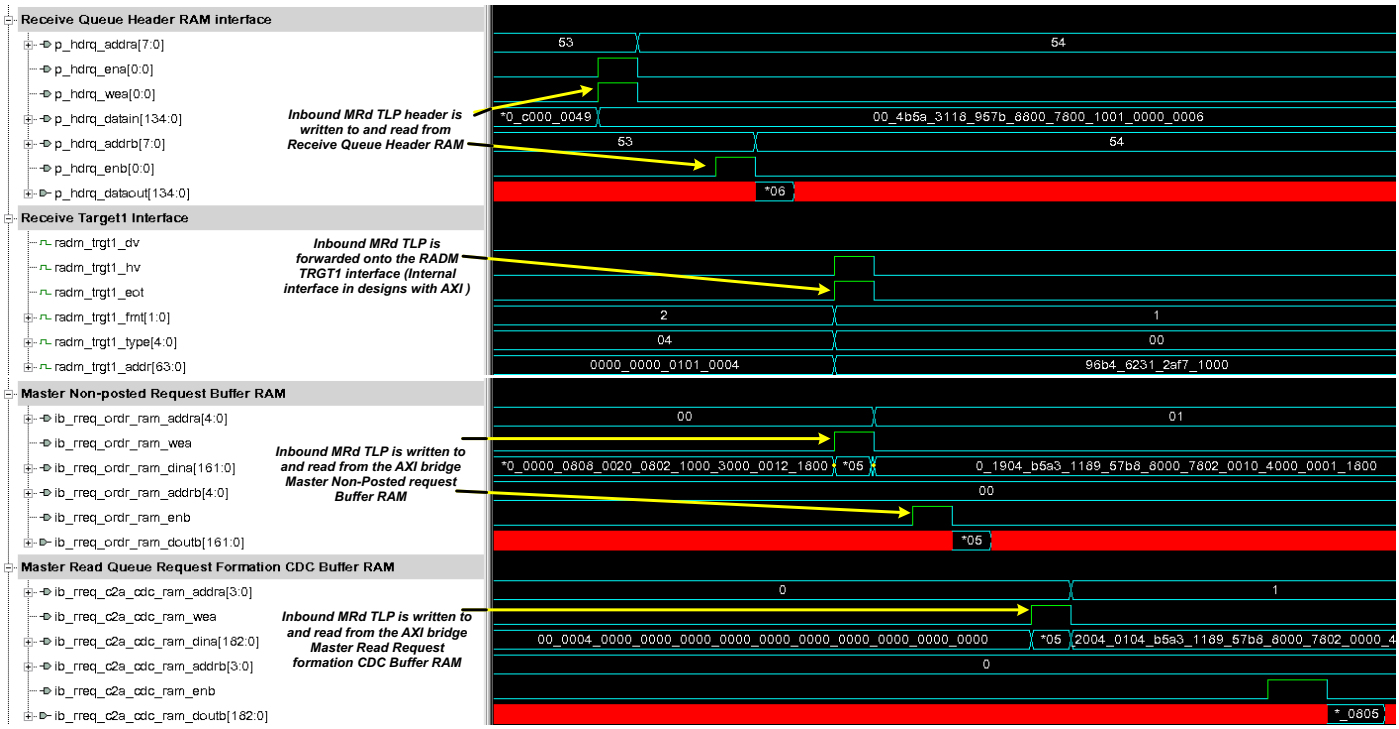
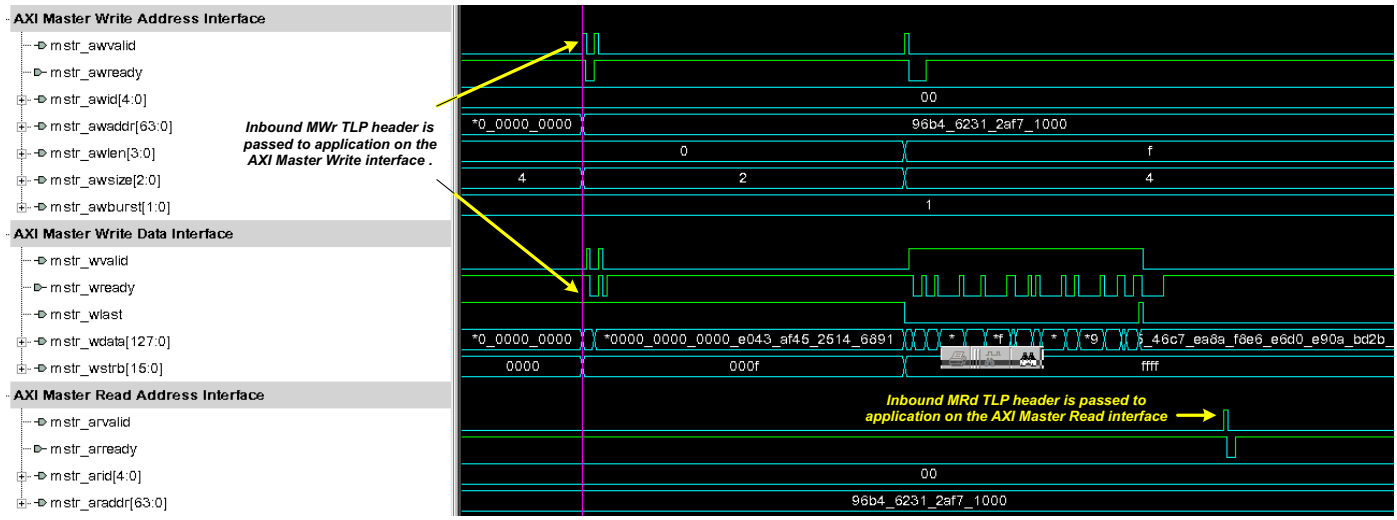


Figure F-41 Inbound Memory TLPs Delivered to Application on AXI Master Interface



F.5.2 Outbound Transaction: MWr/MRd TLP not Received by Link Partner

The controller transmits all TLPs issued by the application on the client* or AXI Slave interface on the PCIe wire. It does not perform any error checking or buffering on the transmit path. But, the controller performs credit checking on all TLP types. If an outbound MWr/MWr TLP issued by your application is not seen by the remote link partner, check the following:

- Check that the link is up, and flow control initialization has completed, as shown in Figure F-38.

- Check that the Memory Space Enable (MSE) and Bus Master Enable (BME) bits of the Command register in the controller are set before the MWr/MRd TLPs are issued by your application. See Figure F-38.
- Check if lack of header/data credits of the particular TLP type is preventing the outbound TLP from being transmitted.
- View the following signals from hierarchy ***.u_pcie_core**
  - `xadm_ph_cdts:` Shows the amount of posted header buffer space currently available at the remote end of the link (in units of posted header credits).
  - `xadm_pd_cdts:` Shows the amount of posted data buffer space currently available at the remote end of the link (in units of posted data credits).
  - `xadm_nph_cdts:` Corresponds to non-posted header credit.
  - `xadm_npd_cdts:` Corresponds to non-posted data credit.
  - `xadm_cplh_cdts:` Corresponds to completion header credit.
  - `xadm_cpld_cdts:` Corresponds to completion data credit.
- Check that the Replay buffer and SOT buffer are sized correctly as per the RamSizes.html report generated by coreConsultant. Check that these RAM interfaces are connected correctly as shown in reference top level file `pcie_iip_device.sv` (See, "Top Level Reference File Generation" on page 206). This is necessary for the controller to replay a TLP when a TLP is NAK'd by the link partner or there is a Replay timeout.
- If using AXI interface, check that the Slave RAMs are sized correctly and getting the correct clocks are shown in reference top level file `pcie_iip_device.sv` (See, "Top Level Reference File Generation" on page 206).
- If you are using iATU to translate an Outbound AXI request to a PCIe TLP, check that the iATU registers are programmed correctly as described in "Internal Address Translation Unit (iATU)" section in "Controller Operations" chapter of the *Databook*.
- If not using iATU and you are using the sideband bus `slv_a*misc_info` to issue the outbound Mem request, check that the `slv_a*misc_info`[4:0] signals are driven correctly to indicate Type "Mem".

Figure F-42 Credit Checking for Outbound TLPs

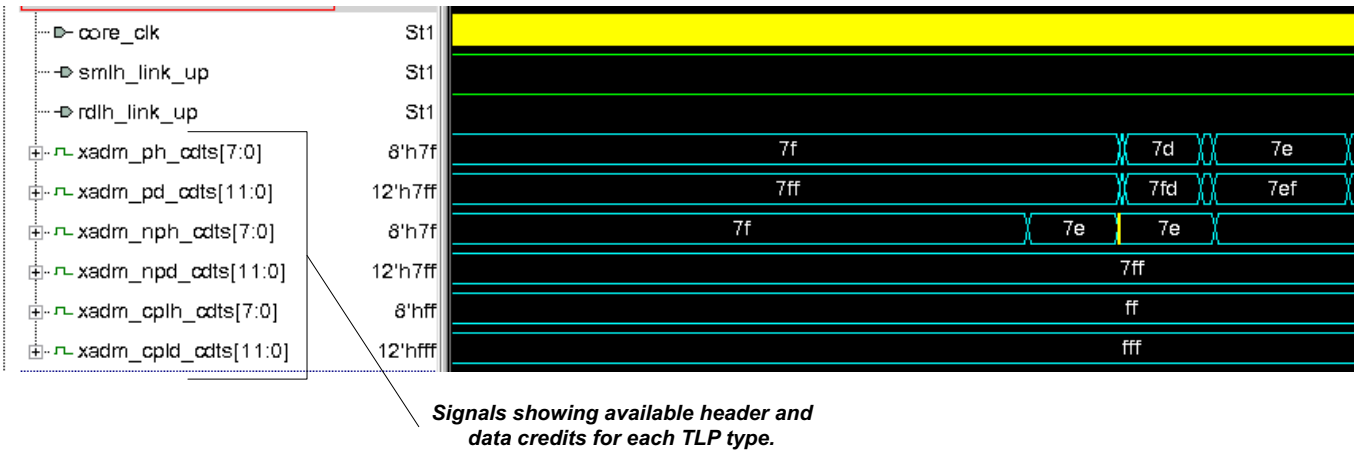
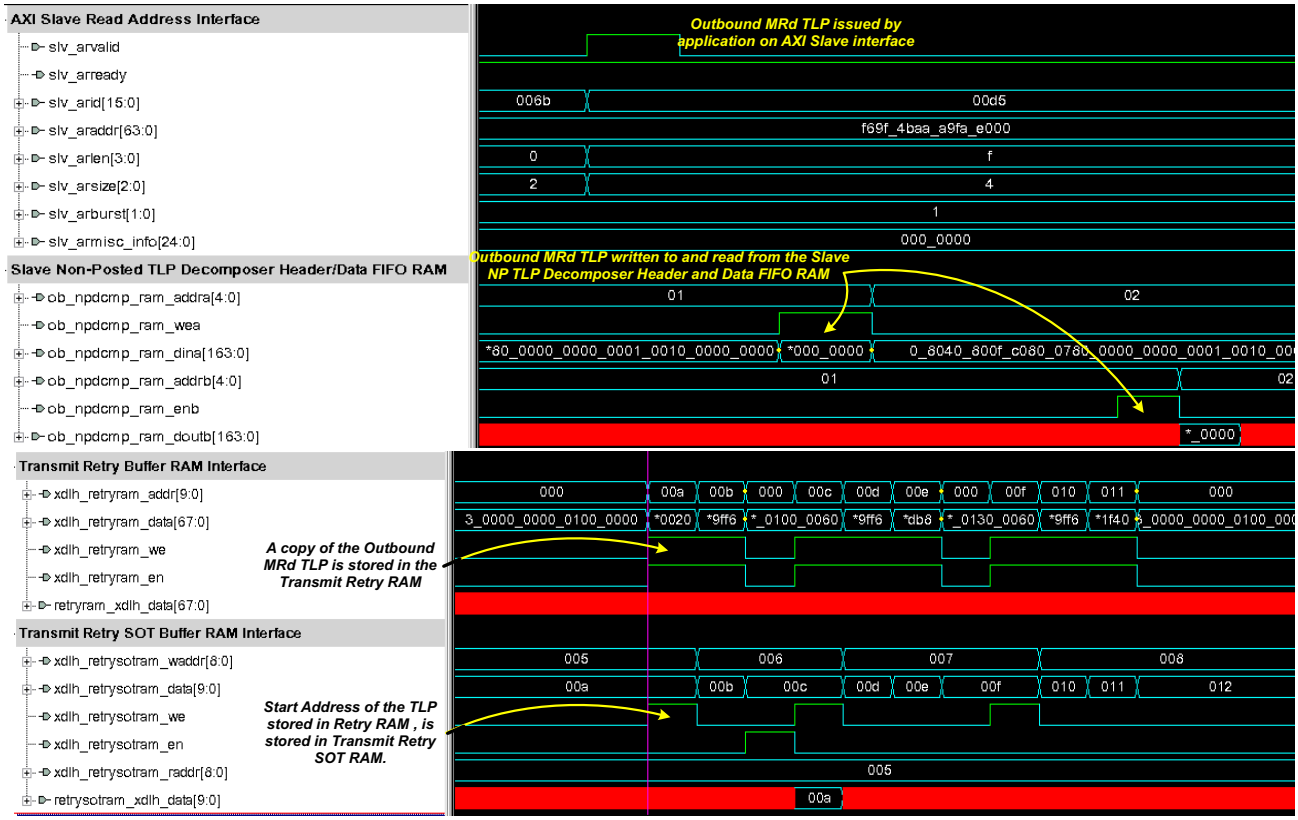


Figure F-43 AXI RAM Accesses Along the Tx Path of MRd TLPs



F.6 Top Level Reference File Generation

Do the following to generate the reference top level instantiation file `pcie_iip_device.sv`:

- Download and install the specified versions of VIPs as described in "Download and Install Verification IP (VIP)" section in the *Install Guide*.
- Open your configured workspace or create a new workspace in coreConsultant and configure the PCIe controller. Make sure you are using the version of coreConsultant specified in "Supported Tool Versions" section of the *Install Guide*.
- In your configuration, change the parameter PHY_TYPE to "Example PIPE PHY" (if not already done) by entering the following command on the prompt:
- set_configuration_parameter PHY_TYPE 8
- Click **Apply** to generate RTL
- Go to **Edit -> Tool Installation Roots** to make sure you are using the version of VCS specified in "Supported Tool Versions" section of the *Install Guide*.
- Go to **Verify Component -> Setup and Run Simulations**.
- Check the **Execution options**, change them if required.
- Click **Apply** to run the Simulate activity.
- The top-level instantiation file is generated under

  `<workspace>/examples/pcie_iip_device.sv`

Did This Help?

If these troubleshooting instructions did not help resolve your problem, open a SolvNetPlus case mentioning the name of your project and provide the simulation waveform dump file.

G

Silicon Debug Guide

The PCIe controller supports the following Silicon Debug options:

- RAS DES (`CX_RAS_DES_ENABLE =1`)
  - Software accessible registers and top-level ports which provide internal data and control information.
  - Software accessible event counters which provide a count of error and non-error events.
  - Error injection capabilities for system validation.
- SII: Debug Registers and Signals
  - Software accessible debug registers (`PL_DEBUG0_OFF` and `PL_DEBUG1_OFF`).
  - Top level debug ports (`cxpl_debug_info` and `cxpl_debug_info_e`).
- SII: Diagnostic Control and Status Signals (`DIAGNOSTIC_ENABLE =1` or `CX_RAS_DES_ENABLE =1`)
  - Top level diagnostic status and control bus (`diag_status_bus`and `diag_ctrl_bus`).

Note: RAS DES is the recommended option because it provides more extensive debug information.

This guide can be used as reference to systematically debug PCIe controller silicon issues related to the following areas:

- "System Integration" on page 208
- "Link Training" on page 216
- "Link Stability" on page 226
- "Power Management" on page 235

G.1 System Integration

The following topics are discussed:

- "System Checks"
- "Clocking" on page 210
- "Reset" on page 212
- "PHY" on page 213

G.1.1 System Checks

This section lists system level checks that you can perform to isolate the problem area and determine the next steps for debugging the problem.

Failure Rate

- How many total devices were tested and how many of them failed?
- What are the observed differences between a passing and failing test?
- Can the issue be reproduced on more than one devices?
- What is the failure rate of the failing devices?
- What is the average time before failure?
  - Did the failure occur immediately after speed change?
  - Did the failure occur after few seconds of data traffic, post the speed change?
- Is the failure intermittent or continuous?
  - Are RX errors detected continuously?
  - Are RX errors detected intermittently, that is, once in every few seconds?

Process Corner, Speed Grade, and Silicon Revision

- What is the process corner of the failing and passing devices?
- Is the failure rate dependent on process variations?
  - Does the failure rate increase when SS process corner devices are used?
  - Does the failure rate decrease when FF process corner devices are used?
- Is the failure rate dependent on speed grade variations?
  - Does the failure rate increase when a slower speed grade FPGA is used?
  - Does the failure rate decrease when a faster speed grade FPGA is used?
- Is the failure observed on previous silicon revisions, under the same test conditions?

Voltage and Power Supply

- What are the PHY and PCIe Controller voltage measurements when the failure is detected?

  Note: Some applications may have in-system monitors for voltage measurements.
- Are the on-board power status LEDs reporting any errors?

- Are the voltages stable without any glitches, and within data sheet limits after power-on?

  Note: An earlier issue such as, PHY operating out of voltage range, can lead to issues later in time.

  - Is Voltage drop observed during power-on-reset?
  - Is Voltage spike observed during FPGA programming?
- Is the failure rate dependent on voltage variations?
  - Does the failure rate increase when Vcc is 5% under nominal range?
  - Does the failure rate decrease when Vcc is 5% over nominal range?
- Does the failure rate decrease if powering-on sequence is staggered across multiple lanes?

  Note: Some power supplies may not be able to support powering-on all lanes simultaneously. In that case:

  - Try powering-on one lane at a time.
  - For applications where 4 lanes share the same PLL, try powering-on one quad at a time.
- Does the failure rate decrease after adjusting the voltage level?

  Note: Some applications may have on-board voltage regulators for voltage adjustment.
- Does the failure rate decrease after switching to an external cleaner power supply?

Temperature

- What is the device temperature measurement when the failure is detected?

  Note: Some applications may have in-system monitors for temperature measurements.
- Is the device temperature within the data sheet limits after power-on?
- Is the failure rate dependent on temperature variations?
  - Does the failure rate increase when operating at -50C?
  - Does the failure rate decrease when operating at room temperature?
- Is the failure rate dependent on cold or warm starts?
  - Does the failure rate increase if test starts after device is already warm?
  - Does the failure rate decrease if test starts after device is cooled down?

Board and Connectors

- Is the channel insertion loss of the link within PCIe specification range? For more information, see *PCI Express Base Specification, Revision 6.0, Version 1.0*.
- Does the failure rate decrease on shorter channels or channels with less insertion loss?
- Is the failure rate dependent on the board?

  Note: If the device is on a socket, try swapping the devices on different boards.
- Is there an active interposer connected on the link?

  Note: Active interposer can sometimes condition or improve the signal quality of the link.
- Are the cables and connectors rated for the highest operating speed for a high speed link such as 8 GT/s in Gen3 or 16 GT/s in Gen4?

Link Partner

- Is the link partner reporting any errors?
- Is the link partner PCI-SIG compliant?
- Is the link partner's PCIe electrical compliance report available for review?
- Has the link partner worked with other devices?
- What is the link width of the application?
- Is the failure rate the same on all lanes across all boards? Try lane reversal and lane flip to resolve issue.
- Is the failure rate dependent on link width?
  - Did Lane0 pass in x4, but failed in x8?
  - Did the test pass in x4, but failed in x8?

  Note: Some PHY architectures share one common PLL for every 4 lanes (quad). It is possible that the test fails when the second quad is powered-on for x8 applications.
- Does the failure rate decrease on Lane0 if all other lanes are powered-down or disabled?

  Note: Variables such as crosstalk can cause issues. You can stagger the power-on sequence across multiple lanes to reduce load.
- Reproduce issue in x1 application or Lane0 to simplify debug. Select the device and board combination with most consistent failing results.

G.1.2 Clocking

Clocking issues are common and should be one of the first things to check during silicon debug. An unstable clock with high jitter may leave little or no margin for the PHY and PCIe Controller to work. The primary clocks discussed in this section are the reference clock, pipe clock, core clock, aux clock, and AXI clocks.

Reference Clock (REFCLK)

- **Clocking Architecture:** The REFCLK can have various architectures, for example Common Clock (CC), Separate REFCLK with No SSC (SRNS), or Separate REFCLK with Independent SSC (SRIS). Identify the clocking architecture of your system. Switch the clocking architecture from SRNS or SRIS to CC to see if the relaxed clocking resolves the clocking issue.

  Note: This change is only for debug purpose. It may not be supported by all systems.
- **SSC:** When SSC is enabled, the low frequency REFCLK jitter limits must be met. Some systems or motherboards allow SSC of the REFCLK to be disabled. For such systems, you can disable SSC of the REFCLK and to resolve issues with the REFCLK. For more information, see PCI Express Base Specification, Revision 6.0, Version 1.0.
- **Frequency:** Check if REFCLK is operating at the expected frequency, and the parameters are set within the specification limits, when the failure is detected. It is recommended to use oscilloscope to get the most accurate measurement. For more information, see PCI Express Base Specification, Revision 6.0, Version 1.0.
- **Stability:** Check if REFCLK is stable and free-running after power-on. It is recommended to use the oscilloscope's infinite persistence mode for monitoring stability.
- **REFCLK Jitter:** Check if the REFCLK jitter limits are compliant with PCI Express Base Specification, Revision 6.0, Version 1.0. REFCLK jitter limits are defined only for CC architecture, and not for SRNS

or SRIS architecture. The REFLCK jitter limits is expected to be lower in SRNS and SRIS architecture. For more information, see PCI Express Base Specification, Revision 6.0, Version 1.0.

- **Compliance Test:** Use SIGtest to test REFCLK for compliance. Some scope vendors have software applications based on the SIGtest engine. SIGtest and the methodology for testing REFCLK can also be downloaded from PCI-SIG.
- **External REFCLK:** In some scenarios switching to an external REFCLK source may resolve the issue.

  Note: This change is only for debug purpose. It may not be supported by all systems.
- **Replace on-board REFCLK:** In some scenarios replacing the on-board crystal generating the REFCLK to one having less jitters may resolve the issue.

  Note: This change is only for debug purpose. It may not be supported by all systems. Some low-quality crystals generating REFCLK that supports CC architecture may not be able to support SRNS or SRIS architectures.

PIPE Clock (PCLK)

- **Frequency:** Check if PCLK is operating at the expected frequency when the failure is detected.It is recommended to use oscilloscope to get most accurate measurement.
- **Stability:** Check if PCLK is stable after reset and during normal operation. It is recommended to use the oscilloscope's infinite persistence mode for monitoring stability. Ensure PCLK is not slowly drifting away from reference. Use a stable REFCLK as reference for monitoring drift.
- **PLL:**Check if the PLL circuit responsible for generating PCLK is locked. Most PLLs have a lock status indicator. Use oscilloscope or FPGA debug trigger to monitor the PLL lock status. Contact your PHY vendor to see if there are signals or registers to improve and speedup the PLL lock. For more information, see "PHY" on page 213. If the PLL does not lock in time, it can cause PCIe Controller issues like:
  - PLL intermittently losing lock during normal operation in L0, leading to link down issues.
  - PLL taking too long to lock after exiting L1SS or speed change, leading to timeout issues.
- **Speed Change:** PCLK can temporary flat line during speed change. This is OK if PCLK recovers and is stable before the PHY asserts PHYSTATUS in response to speed change. Ensure PCLK is operating at expected frequency after speed change. For more information, see "Speed Change" on page 219.
- **Reducing PCLK Frequency:** For FPGA applications, try reconfiguring the PCIe Controller to operate at a lower PCLK frequency. Changing PCLK in Gen3 from 500 MHz to 250 MHz may help resolve issues caused by timing violations.

  Note: This reconfiguration is only for debug purpose. It may not be supported by all systems.

Core Clock (core_clk)

The `core_clk` used in the controller is derived from PCLK. Debugging guidelines for PCLK applies to `core_clk` also.

Aux Clock (aux_clk)

During normal operation `aux_clk`is driven by `core_clk` and must be completely synchronous to `core_clk`. `aux_clk` and`core_clk`clock trees must be balanced during layout. During low power mode `aux_clk` may switch to a low frequency system auxiliary clock.

G.1.3 Reset

Resets are a critical part of link bring up and ensure the correct operation of the controller. An unsuccessful reset may cause the PCIe link to be stuck in reset. In some cases, the reset may complete unsuccessfully, and cause issues later. The common issues with the PHY and controller's main reset and their workarounds are discussed in this section.

PIPE and Controller Reset

- **PIPE Protocol:** Ensure that PHY reset is executed at Gen1 in P1 power state, as defined in the PIPE specification. Resetting the PHY and controller at any speed brings the PCIe link down to Gen1. Ensure PHY reset completes successfully before initiating any other PIPE related operation, to avoid the PHY and controller from becoming out-of-sync.

  RESET# is PHY reset and PHYSTATUS is PHY reset status indicator,

  - PHYSTATUS should assert immediately when RESET# is asserted.
  - PHYSTATUS should de-assert when PHY is done with reset.
- **Timeout:** Check for any PHY and controller reset timeout.
- **Multi-lane Application:** Check that all active PHY lanes de-assert PHYSTATUS to indicate reset complete before the controller begins link training.
- **Applying Reset:** Try applying resets immediately upon power-on. Some PHY circuits need to be protected using resets during power-on until REFCLK and PHY are stable.
- Releasing Reset: Ensure PHY reset is released after power and REFCLK is stable. Some applications may have a power good signal to indicate when the voltage is within nominal operating range.
- **Extended Reset:** Some PHY circuits may have a minimum reset assertion requirement. Try extending the reset assertion time to see if it makes any difference.
- **Back-to-Back Resets:** Try applying two or more consecutive or back-to-back resets to see if it makes any difference.
- **Internal PHY Resets:** Most PHYs typically have internal resets for each sub-block, such as PMA reset, PCS reset, PLL reset, CDR reset, and RX elastic buffer reset. You can apply PHY resets at sub-block level to isolate the issue with reset. For FPGA applications, it is possible to use virtual I/O or debug logic to apply the resets.
- **Timing Diagram:** See the PHY reset timing diagram provided by your PHY vendor and to identify the differences, if any.
- **LTSSM:** During reset, check if the LTSSM is waiting in Detect.Quiet until PHY reset completes for all active lanes.
- **FPGA Programming:** For FPGA applications, try to keep the design in reset while the FPGA is being programmed to provide extra protection.
- **FPGA Debug:** Monitor the following PIPE signals for reset status.
  - RESET# (PHY reset)
  - PHYSTATUS
  - TXDETECTRX_LOOPBACK =0b
  - TXELECIDLE =1b (TX electrical idle)
  - TXCOMPLIANCE =0b
  - RXPOLARITY =0b

- POWERDOWN =0010b (P1 power state)
- TXMARGIN =000b
- TXDEEMPH =1b (3.5 dB de-emphasis)
- RATE =00b (reset always restores the PCIe link back to Gen1)

Set analyzer to trigger on the first PHYSTATUS falling edge in response to RESET# released. Review the PIPE interface signal conditions during reset.

- FPGA configuration should be completed before system power-on. FPGA configuration generally takes a much longer time than system power-on and a link partner could give up link establishment during this time, resulting in link up failure.

G.1.4 PHY

Silicon issues should be isolated as PHY issues or controller issues, for easy debugging. This section provides general guidance and quick tests to determine if the PHY is operational. Consult your PHY vendor for additional guidance, if required.

- **Loopback Test:**Set the PHY to transmit a PRBS or PCIe compliance pattern in test mode. Locally loop-back the TX to RX. Use a PRBS or PCIe compliance pattern checker to ensure the PHY can operate error free in loopback mode. Most PHYs have certain loopback modes for debug, consult your PHY vendor for information.
  - **Step 1 - PCS loopback:** Check if the local TX to RX loopback data path through PCS is OK. This loopback mode bypasses the PMA, so should be the easiest to pass.
  - **Step 2 - PMA loopback:** If PCS loopback can pass, check if the local TX to RX loopback data path through PCS and PMA is OK.
  - **Step 3 - Cable loopback:**If PMA lookback passes, check if cable loopback is OK. Add a compliance channel between the cables loopback between TX+/TX- and RX+/RX- for most accurate result.
- **TX Test:** Check if the PHY's TX data path is operational and runs error free. Set the PHY to transmit a PRBS or PCIe compliance pattern. Connect the PHY's TX+/TX- to the analyzer's input. Ensure that the analyzer locks to the PRBS or PCIe compliance pattern and operates error free. Review the TX eye diagram on the analyzer.
- **RX Test:**Check if the PHY's RX data path is operational and runs error free. Set the analyzer to generate and transmit PRBS or PCIe compliance pattern to the PHY's RX. Try stressing the transmitted pattern by injecting different components of jitter.
  - **Step 1:** Ensure that the PHY recovers the RX data, locks to the pattern, and runs error free with BER = 10-12 or better
  - **Step 2:** Set the PHY to loopback RX to TX. Connect the PHY's TX to the pattern detector and checker. Ensure the analyzer can lock to the pattern and run error free with BER = 10-12 or better.
- **RX Eye Scan:** Perform and review RX eye scan to ensure there is a good eye height and width opening with enough margin. Most PHYs have an in-system RX eye scan feature to evaluate the RX signal quality before and after the equalizer (CTLE/DFE) circuit.
- **PRBS and Compliance Pattern:**Most PHYs have built in PRBS or PCIe compliance pattern generators and checkers for debug. For example, use PRBS15 for Gen1/Gen2 and PRBS23 for Gen3/Gen4.
- **PLL:** Ensure that the PLL locks to new frequency after speed change. Most PHYs have a PLL lock indicator which can be used to check the status of PLL lock.

- **Speed Change**: Some PHYs may use different PLLs between low line rates (Gen1/Gen2) and high line rates (Gen3/Gen4). The PLL may not be able to lock to a new frequency if the PLL settings do not switch correctly.
- **Feedback Dividers:** Some PHYs use different PLL feedback dividers. For example, a PLL operating at 10G and may use a feedback divider of 4 to achieve Gen1(2.5G), and a feedback divider of 2 to achieve Gen2(5G). The PLL may not be able to lock to a new frequency if the PLL feedback divider does not switch correctly.

- **CDR:** Ensure that the CDR locks and recovers RX data after speed change or after exiting RX electrical idle. Some PHYs provide a CDR lock indicator, but this indicator might be a statistical lock and may not represent a true lock. It is recommended to use oscilloscope or FPGA debug trigger to monitor the CDR lock status. Contact your PHY vendor to determine CDR lock if a lock indicator is not available.
  - **Speed Change:** The PHY may use different CDR configurations for different speeds, if the PHY does not auto adapt after speed change. PHY may not be able to re-lock or recover the RX data if there are error in speed dependent settings.
  - **RX Electrical Idle**: Ensure that the CDR re-locks and recovers the RX data after exiting RX electrical idle. The CDR may lose lock and wander off if it is in RX electrical idle for a long period of time. Ensure that the re-lock time meets the timeout requirements for exiting L0s.
- **Recovered Clock (RECCLK):**
  - **Frequency:** Ensure that the RECCLK is operating at the expected frequency when the failure is detected. It is recommended to use oscilloscope to get most accurate measurements.
  - **Stability:** Ensure that the RECCLK is stable after reset and during normal operation. it is recommended to use the oscilloscope's infinite persistence mode for monitoring stability. Ensure RECCLK is not slowly drifting away from reference. Use a stable REFCLK as reference for monitoring drift.

    Note:

    - RECCLK operating at correct frequency, but slowly drifting from reference point can cause intermittent RX bit errors.
    - The controller may report these RX bit errors as correctable errors.
  - **CDR Lock:** Ensure that the CDR circuit responsible for generating RECCLK is locked. Check if there are signals or registers to improve and speedup the CDR lock. If the CDR does not lock in time or the lock does not remain stable, it can cause controller issues, for example:
    - CDR intermittently losing lock during normal operation in L0 can cause RX errors.
    - CDR not locking quickly after exiting L0s can cause timeout issues.
- **Electrical Compliance:**Ensure that the PHY passes electrical compliance, check the PCI-SIG compliance report if available.
  - **PLL Bandwidth:**Ensure PLL bandwidth and peaking are within the PCI Express Base Specification, Revision 6.0, Version 1.0 limits. Consult your PHY vendor for debug features to allow you to adjust the PLL bandwidth and peaking performance.
  - **TX Output Jitter and Differential Amplitude:**Ensure that TX output jitter and differential amplitude are within the PCI Express Base Specification, Revision 6.0, Version 1.0. Consult your PHY vendor for debug features to allow you to adjust the PHY to reduce TX output jitter and to optimize TX differential amplitude.
  - **RX Jitter Tolerance:**Ensure that RX jitter tolerance is within the PCI Express Base Specification, Revision 6.0, Version 1.0 limits. The RX jitter tolerance limits depend on the clocking mode, such

as CC (Common Clock) or IR (Independent REFCLK). Consult your PHY vendor for PHY adjustments (such as CTLE, DFE, and CDR) to improve RX jitter tolerance margin.

- **TX and RX Return Loss:**Ensure TX and RX common mode and differential return loss are within the PCI Express Base Specification, Revision 6.0, Version 1.0 limits. Consult your PHY vendor for debug features that allow you to adjust the common mode and differential return loss performance.

- **FPGA Debug:** The following PIPE interface signals helps for silicon debug:
  - POWERDOWN
  - TXELECIDLE
  - RATE
  - PHYSTATUS
  - RXELECIDLE
  - RXVALID
  - RXSTATUS

  Note: These signals should be sampled using PCLK. If PCLK is turned-off, use REFCLK.

G.2 Link Training

Link up failure on silicon commonly occurs due to issues in the following areas:

- "Receiver Detection"
- "Broken Lanes" on page 218
- "Speed Change" on page 219
- "Link Equalization" on page 220

This section provides guidelines to identify and debug these issues.

G.2.1 Receiver Detection

The link performs receiver detection after completing PHY reset successfully to determine if a remote link partner is available for establishing a linkup.

Common Symptoms

- Link does not come up. LTSSM state loops between Detect.Quiet and Detect.Active.
- Link comes up but with link width narrower than expected.

Debug Guidelines

- **Debug using RAS DES**

  If your controller has RAS DES enabled (`CX_RAS_DES_ENABLE =1`) you can monitor the following registers for receiver detection issues.;

Table G-1 Debug Using RAS DES

Issue	RAS DES Register	Usage
Receiver Detection Status	`SD_STATUS_L1LANE_REG`	Application software can read the status of receiver detection by selecting the lane number using `SD_STATUS_L1LANE_REG[3:0]` and read the value of `SD_STATUS_L1LANE_REG[17]` to check if receiver is detected on the selected lane number.
Bypass Receiver Detection	`SD_CONTROL1_REG`	If the receiver detection feature is not working properly, bypass receiver detection to see if link training progresses for debug. Application software can set `FORCE_DETECT_LANE_EN` field of the `SD_CONTROL1_REG[16]`register to 1b to instruct the controller to ignore receiver detection from PHY during LTSSM Detect state and use receiver detection status from `FORCE_DETECT_LANE` field of `SD_CONTROL1_REG[15:0]` register instead. **Note:**Each bit in `SD_CONTROL1_REG[15:0]` register corresponds to one lane.
Receiver Detection Timeout	`SD_CONTROL2_REG`	Check for any receiver detection related timeout. If the PHY requires more time for receiver detection, the application software can hold LTSSM in Detect.Active by setting the `HOLD_LTSSM` field of `SD_CONTROL2_REG[0]`register.

- **Debug Using Common Controller Signals and Registers**

  Check the current LTSSM state by reading `PL_DEBUG0_OFF[5:0]` or by monitoring top level port `cxpl_debug_info[5:0]` .

- **General Debug Guidelines**
  - **PIPE:** Check if receiver detection is executed in Gen1 and P1 power state as defined in the PHY Interface for the PCI Express, SATA, USB 3.2, DisplayPort, and USB4 Architectures (PIPE) Version 4.4.1, Version 5.1.1, Version 6.1.1. Ensure receiver detection completes successfully before initiating another PIPE related operation to avoid the PHY and controller from becoming out-of-sync.
  - **Multi-lane applications:** Ensure receiver is detected on at least one lane.
  - **DC single ended impedance:** Ensure remote PCIe link partner's receiver DC single ended impedance (**ZRX-DC**) is between 40 and 60 Ω. For more information, see PCI Express Base Specification, Revision 6.0, Version 1.0.
  - **Receiver detection or receiver issue:** You can determine if the issue is with receiver detection or the receiver itself, as follows:
    - Perform receiver detection on a known good receiver that can always be detected by other PCIe devices.
    - Perform receiver detection on a load impedance between 40 and 60 Ω.

    Try swapping passing and failing lanes to see if the passing lane still passes.

- FPGA Debug Guidelines
  - Determine the status of receiver detection on the PIPE interface as follows:
    - RXSTATUS =000b with PHYSTATUS =1b and TXDETECTRX =1b; indicates receiver is not present.
    - RXSTATUS =011b with PHYSTATUS =1b and TXDETECTRX =1b; indicates receiver present.
  - Monitor the following PIPE signals for receiver detection status.
    - POWERDOWN
    - TXDETECTRX
    - PHYSTATUS
    - RXSTATUS

    Set analyzer to trigger on the first PHYSTATUS rising edge after TXDETECTRX is asserted. Review RXSTATUS along with the other PIPE signals. For more information, see PHY Interface for the PCI Express, SATA, USB 3.2, DisplayPort, and USB4 Architectures (PIPE) Version 4.4.1, Version 5.1.1, Version 6.1.1.
  - Unused card edge connectors implemented on a FPGA board can be incorrectly detected by a link partner resulting in link up failure.For example, if your FPGA board implements x8 card edge connectors, and they are connected to high speed IO inside FPGA which has a x1 MAC core, the upper lanes could be incorrectly detected by a link partner (that is x2 or more). As a result, link training does not complete successfully.

    To avoid this issue ensure:
    - PRSNT# is correctly connected, and
    - Unused lanes of high speed IO are in electrical idle state. (For example, Lane0 is connected with MAC, and txelecidle =1 for Lane1-7.), or

- Unused card edge connectors are masked by insulator.
- Ensure that FPGA configuration is completed before system power-on, because FPGA configuration takes longer time than system power on.

G.2.2 Broken Lanes

After receiver detection is completed, the LTSSM goes through Polling -> Configuration -> Recovery states before reaching L0 state at Gen1 data rate. If some lanes are broken after receiver detection, the link may not reach L0 at the desired link width.

The LTSSM can go into Polling.Compliance state if all lanes that detected receiver during Detect, do not receive expected TS Ordered Sets in Polling. This can happen if a subset of lanes is broken after detecting.

Common Symptoms

- Link comes up (LTSSM is in L0) with narrower link width than expected.
- LTSSM goes to Polling.Compliance state unintentionally.

Debug Guidelines

- Debug using RAS DES

  If your controller has RAS DES enabled (`CX_RAS_DES_ENABLE =1`) you can monitor the following registers for broken lane issues.

Table G-2 Debug Using RAS DES

Issue	RAS DES Register	Usage
PIPE signals RXELECIDLE and RXVALID of each lane	`SD_STATUS_L1LANE_REG`	Program the lane number in `SD_STATUS_L1LANE_REG[3:0]`and read the value of `SD_STATUS_L1LANE_REG[19:18]` to check the RXELECIDLE and RXVALID values of the selected lane number .
Receiver Errors	`EVENT_COUNTER_CONTROL_REG` `EVENT_COUNTER_DATA_REG`	Check if any lane had a receiver error like elastic buffer overflow/underflow, by programming `EVENT_COUNTER_CONTROL_REG` to select the Group number and Event number, lane number and enabled the counter. For more information, see “Reliability, Availability, and Serviceability (RAS)” section in the “Controller Operations” chapter of the *Databook*. **Note**: You can also use the top level signals `cdm_ras_des_ec_info_common[10](rmlh_deskew_alignment_err)` and `cdm_ras_des_ec_info_common[6](rmlh_rcvd_err)` as interrupts for error indication.

- Debug Using Common Controller Signals and Registers

  Check the following top-level ports if present in your design:

  - `rmlh_deskew_alignment_err:`Indicates deskew logic overflow
  - `rmlh_ts_rcv_err:` Indicates a problem in received TS on lane0

  - `rmlh_lanes_rcving[NL-1:0]`: Indicates the lanes active in link training
- General Debug Guidelines
  - In a multi-lane setup, to isolate the broken lane, link up at a smaller link width. For example, if your x4 link goes to Polling.Compliance, try to link up at x2 or x1.
  - If the LTSSM is going to Polling.Compliance in a multi-lane setup, reduce the link width to x1 and see if link up is successful .
  - Try a lane reversal setup (Connect Lane0 to Lane n-1 of the link partner).
- FPGA Debug Guidelines

  Trigger on Polling.Active or Polling.Compliance LTSSM state and monitor the following PIPE signals
  - RXELECIDLE
  - RXVALID
  - RXSTATUS

G.2.3 Speed Change

For link speed change to complete successfully, it is required that both link partners support the new speed rate, and the PHY changes rate successfully. This section provides silicon debug guidelines for speed change issues. Gen3 and Gen4 speed change also involves link equalization, for more information, see "Link Equalization" on page 220.

Common Symptoms

- Link does not come up at the new speed.
- Link falls back to a lower speed after speed change.
- Link is not stable after speed change, LTSSM keeps going to Recovery.
- Link goes down after speed change .
- Link quality degrades after speed change, correctable errors are detected.
- In a multi-lane application, link comes up at a lower link width after speed change.

Debug Guidelines

- Debug using RAS DES

  If your controller has RAS DES enabled (`CX_RAS_DES_ENABLE =1`) you can monitor the following registers for speed change issues.

Table G-3 Debug Using RAS DES

Issue	RAS DES Register	Usage
Speed change related LTSSM variables	`SD_STATUS_L1LTSSM_REG`	Read the value of bits[20:16] to check value of variables: directed_speed_change, changed_speed_recovery, successful_speed_negotiation, upconfigure_capable, select_deemphasis.

Issue	RAS DES Register	Usage
Receiver Errors	`EVENT_COUNTER_CONTROL_REG` `EVENT_COUNTER_DATA_REG`	Check if any lane had a receiver error like elastic buffer overflow/underflow and so on, by programming `EVENT_COUNTER_CONTROL_REG` to select the Group number and Event number, lane number and enabled the counter. For more information, see "Reliability, Availability, and Serviceability (RAS)" section in the "Controller Operations" chapter of the *Databook*. **Note:**You can also use the top level signals `cdm_ras_des_ec_info_common[10]` (`rmlh_deskew_alignment_err`) and `cdm_ras_des_ec_info_common[6]`(`rmlh_rcvd_err`) as interrupts for error indication.

- Debug Using Common Controller Signals and Registers

  Check the following top-level ports if present in your design:

  - `rmlh_deskew_alignment_err`:Indicates deskew logic overflow.
  - `rmlh_ts_rcv_err`: Indicates a problem in received TS on lane0
  - `rmlh_lanes_rcving[NL-1:0]`: Indicates lanes active in link training
- General Debug Guidelines
  - Check if there are errors in the link after speed change by reading the Advanced Error Reporting Status Registers. For more information, see "Link Stability" on page 226.
  - If using Dynamic Frequency speed change mode, check that PCLK and controller clock frequencies change to the new frequency after speed change. It is recommended to use analyzer to measure actual clock frequency and clock stability over a period.
  - Use a protocol analyzer trace to check that both link partners advertise support for the new data rate.
  - If failure occurs on change to Gen3/Gen4 speed, see "Link Equalization" on page 220 for more information.
- FPGA Debug Guidelines

  Monitor the following PIPE signals for speed change status.

  - TXELECIDLE
  - POWERDOWN
  - RATE
  - PHYSTATUS

  Set analyzer to trigger on first PHYSTATUS rising edge after RATE change. Review PCLK frequency along with the other PIPE signals.

G.2.4 Link Equalization

The link may be required to perform link equalization after entering Gen3 or Gen4. Link equalization involves both the PHY and the controller. The PHY evaluates the RX signal, provides feedback results, and adapts to the RX signal. The controller adjusts the TX coefficient of the local and remote link partner's PHY based on the feedback, to optimize the RX signal quality. This section provides silicon debug guidance for

link equalization issues. For silicon debug guidance on general speed change issues , see "Speed Change" on page 219.

Common Symptoms

- Link fails to come up at Gen3/Gen4 data rate.
- Link is unstable after speed change to Gen3/Gen4 data rate.
- Link does not come up at Gen3/Gen4 on all lanes of a multi-lane link.
- Link equalization phase 1 exits unsuccessfully or times out.
- Link equalization phase 2 exits unsuccessfully or times out.
- Link equalization phase 3 exits unsuccessfully or times out.
- Link repeats equalization indefinitely.

Debug Guidelines

- Debug using RAS DES

  If your controller has RAS DES enabled (`CX_RAS_DES_ENABLE =1`) you can monitor the following registers for link equalization issues.

Table G-4 Debug Using RAS DES

Issue	RAS DES Register	Usage
Completion of link equalization on each lane	`SD_EQ_STATUS1_REG` `SD_EQ_CONTROL1_REG`	Program the `SD_EQ_CONTROL1_REG[3:0]` with the selected lane number and `SD_EQ_CONTROL1_REG[4]`with the selected data rate. Read the `SD_EQ_STATUS1_REG[2:1]`to check the equalization status.
Local receiver preset hint	`SD_EQ_STATUS2_REG` `SD_EQ_CONTROL1_REG`	The local receiver preset hint in both EP and RP is set by the DSP during Recovery.RcvrCfg. Program the selected lane number and data rate in `SD_EQ_CONTROL1_REG` and read `SD_EQ_STATUS2_REG[20:18]` to check that this value is as expected.
Current local figure of merit	`SD_EQ_STATUS2_REG` `SD_EQ_CONTROL1_REG`	For debug purpose, you can check the current local figure of merit. Program the selected lane number and data rate in `SD_EQ_CONTROL1_REG` and read `SD_EQ_STATUS2_REG[31:24]`.

Issue	RAS DES Register	Usage
Force local transmitter preset and receiver preset hint	SD_EQ_CONTROL1_REG SD_EQ_CONTROL2_REG	If the local PHY is known to work well with a particular preset, for debug purpose, you can force that preset value to be used as the local transmitter preset and receiver preset hint. Program the selected lane number and data rate in SD_EQ_CONTROL1_REG and program the selected preset values in SD_EQ_CONTROL2_REG[27:24] (local transmitter preset) and SD_EQ_CONTROL2_REG[20:18] (local receiver preset).
Force pre-cursor, cursor, and post-cursor values on local and remote transmitters	SD_EQ_CONTROL1_REG SD_EQ_CONTROL2_REG	For debug purpose, you can force the local and remote transmitter to use a predefined set of co-efficients. Program the selected lane number and data rate in SD_EQ_CONTROL1_REG and program the selected co-efficient values in SD_EQ_CONTROL2_REG[17:0].
Rejection of preset or co-efficient request during EQ.Phase2(USP) and EQ.Phase3(DSP) by the link partner	SD_EQ_STATUS1_REG SD_EQ_CONTROL1_REG	If equalization has not completed successfully on any of the lanes, you can check if the link partner rejected the preset/co-efficient requests by reading SD_EQ_STATUS1_REG[7]. Program SD_EQ_CONTROL1_REG to select the lane number and data rate.
Check if the direction change feedback returned by the PHY resulted in invalid co-efficient values	SD_EQ_STATUS1_REG SD_EQ_CONTROL1_REG	If Equalization has not completed successfully on any of the lanes, check if the direction change feedback from the PHY caused RuleA/RuleB/RuleC co-efficient rule violation by reading SD_EQ_STATUS1_REG[6:4]. Program SD_EQ_CONTROL1_REG to select the lane number and data rate.

- **Debug Using Common Controller Signals and Registers**
  - Check if all phases of link equalization has completed successfully by reading the LINK_CONTROL2_LINK_STATUS2_REG[20:18].
  - Read the contents of the Lane Equalization Control Register of the Secondary PCIe Capability.
    - For a DSP port this register reflects the transmitter preset and receiver preset hints that the DSP uses and sends to the USP during Recovery.RcvrCfg.
    - For a USP port this register reflects the values captured by the USP from the received TS2 Ordered Sets during Recovery.RcvrCfg.
  - Phases 2 and 3 can be bypassed to minimize link EQ time in PCIe applications, such as chip-to-chip, where the PHYs for both USP and DSP are already optimized using TX EQ presets.
  - DSP can be programmed to bypass Phases 2 and 3 by setting GEN3_RELATED_OFF[9] =1b. All four phases of link EQ can be bypassed by setting GEN3_RELATED_OFF[16] =1b. You must use this for debug purpose only.
  - Link EQ can quickly exit Phases 0 or 1 when TS1s are successfully received. Some PHY implementations may need more time for the receiver to adapt and recover the data, especially if Phases 2 and 3 are bypassed. The controller can be programmed to hold in Phase 0 (USP) or Phase 1 (DSP) for 10 ms to give the receiver more time by setting GEN3_RELATED_OFF[12] =1b.

- Link evaluation latency is PHY implementation specific. Latency may be different for each lane. Some PHY implementations may have longer link evaluation latencies and may need more link evaluation iterations to optimize the RX signal quality. Link EQ for each phase must complete within the timeout limits defined in PCI Express Base Specification, Revision 6.0, Version 1.0.
  - Phase 0 Timeout: DSP =N/A, USP 12 ms
  - Phase 1 Timeout: DSP =24 ms, USP =12 ms
  - Phase 2 Timeout: DSP =32 ms, USP =2 ms or 24 ms (exception)
  - Phase 3 Timeout: DSP =2 ms or 24 ms (exception), USP =32 ms

  The controller can be programmed to increase the link evaluation timeout limit from 2 ms to 24 ms to support PHYs with longer latencies by setting `GEN3_EQ_CONTROL_OFF[5] =1b`.
- If Link EQ exits unsuccessfully, the controller can be programmed to move on to the next LTSSM state and continue to link up at Gen3/Gen4 rate, instead of falling back to Recovery.Speed by setting `GEN3_EQ_CONTROL_OFF[4] =1b`.

- **General Debug Guidelines**
  - Use a protocol analyzer trace to check the TS Ordered Sets being received by both link partners during EQ Phase1, Phase2 and Phase3.
  - During EQ.Phase2(USP)/Phase3(DSP) figure of merit evaluation, carefully select the TX EQ presets to be evaluated based on the knowledge of the PHY and system. Some PHYs may not be able to adapt and recover RX data using certain TX EQ presets and channel loss combinations. Application software can read the `GEN3_EQ_CONTROL_OFF[23:8]`to see which presets have been selected for evaluation. Contact your PHY vendor for recommended TX EQ Preset settings.
  - During EQ.Phase2(USP)/Phase3(DSP) Direction Change fine tuning, link equalization can timeout if the PHY cannot converge the TX coefficients to optimize signal quality. Consult your PHY vendor if the advanced co-efficient convergence support in the controller should be used to relax the convergence criteria. For more information, see “Gen3/4 Equalization Details and Example” appendix of the *Databook*.
  - Use a protocol analyzer trace to check the contents in TS1 and TS2 formatted for link EQ.
    - The EC field in Symbol 6 is used to communicate the current link EQ Phase .
    - TX link EQ uses presets when the Use Preset field in Symbol 6 is set to 1b, otherwise TX link EQ uses coefficients .
    - Bits [5:0] of Symbols 7, 8, and 9 stores the TX EQ pre-cursor, main-cursor, and post-cursor values respectively (in Phase 1 or EC = 01b, FS and LF values are stored instead) .
    - The RCV field in Symbol 9 is used to reject the new coefficient value requested by the remote LP if it is does not satisfy the coefficient rules.
  - Link EQ is initiated autonomously by hardware or can be initiated by software after speed change into Gen3 or Gen4. Controller autonomously performs link EQ when entering Gen3/Gen4 the first time. It is not required to perform link EQ when re-entering Gen3 or Gen4 if there are no significant changes in the electrical environment.
    - Check if the PHY can store the current “good” TX EQ settings for reuse when re-entering Gen3 or Gen4.
  - In Recovery.RcvrCfg, EP is required to evaluate and report coefficient/preset mismatches that is agreed in Phase 2 equalization.
  - If coefficient/preset mismatch is reported, the link can re-enter recovery and redo equalization

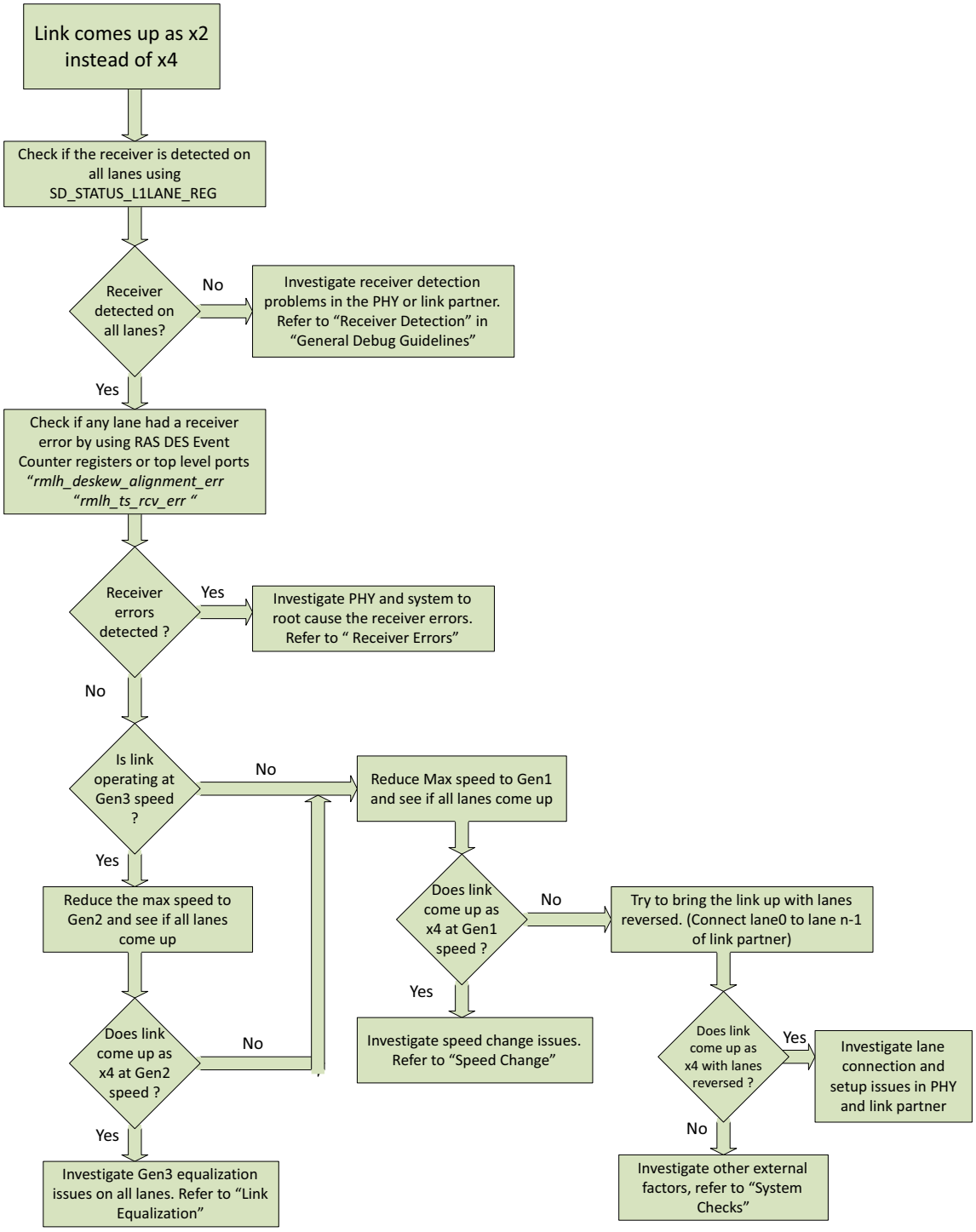
- In Recovery.RcvrLock, check if RP is setting TS1 OS fields with correct coefficients/presets agreed in Phase 2 equalization. For more information, see the *PCI Express Base Specification, Revision 6.0, Version 1.0*.
- If equalization redo does not resolve the issue, equalization redo can repeat indefinitely.
- Setting `EQ_REDO` disable bit in the `GEN3_RELATED_OFF` register to 1b disables the controller from requesting equalization redo.

- **FPGA Debug Guidelines**
  - **TX Link EQ:** Monitor the following PIPE signals for TX link equalization status (used in all 4 phases of link equalization). To check TX EQ preset to coefficient mapping, set analyzer to trigger on LOCAL_TX_COEFFICIENT_VALID. Review LOCAL_TX_PRESET_COEFFICIENT and the following PIPE signals:
    - LOCAL_LF
    - LOCAL_FS
    - GET_LOCAL_PRESET_COEFFICIENTS
    - LOCAL_PRESET_INDEX
    - LOCAL_TX_COEFFICIENT_VALID
    - LOCAL_TX_PRESET_COEFFICIENT
    - TXDEEMPH
  - **RX Link EQ:** Monitor the following PIPE signals for RX link equalization status (used in Phase 2 and 3 of link equalization):
    - LF
    - FS
    - RXEQEVAL
    - RXEQINPROGRESS
    - FOMFEEDBACK
    - DIRFEEDBACK
    - PHYSTATUS
    - INVALID_REQ

    To check PHY evaluation and feedback status, set analyzer to trigger on first PHYSTATUS rising edge after RXEQEVAL assertion. Review FOMFEEDBACK and DIRFEEDBACK along with the other PIPE signals. To check for invalid feedback request, set analyzer to trigger on INVALID_REQ rising edge after RXEQEVAL assertion.

G.2.5 Example Debug Flow for Link Training Issues

Figure G-1 Example Debug Flow for Link Training Issues



G.3 Link Stability

The PCIe link enters L0 and begins to transmit and receive data in normal operation after reset or rate change. The PCIe link needs to maintain stability so the controller can exchange data with the link partner successfully. The PHY and controller have mechanisms to detect and report errors that can cause the link to be unstable. This section provides general guidelines for debugging the following link stability issues in L0:

- "Receiver(Rx) Errors"
- "Correctable Errors" on page 229
- "Uncorrectable Errors" on page 231
- "Example Debug Flow" on page 234

G.3.1 Receiver(Rx) Errors

The PHY can detect RX elastic buffer errors due to link stability issues and report them on the PIPE interface. The controller reports the following receiver errors as correctable errors:

- Decode Errors
- Disparity Errors
- RX Elastic Buffer Overflow Errors
- RX Elastic Buffer Underflow Errors

Common Symptoms

- PCIe link does not remain stable in L0 and goes down to Recovery.
- PCIe Controller reports correctable errors.

Debug Guidelines

- Debug using RAS DES

  If your controller has RAS DES enabled (`CX_RAS_DES_ENABLE =1`) you can monitor the following registers for receiver(rx) errors.

Table G-5 Debug Using RAS DES

Issue	RAS DES Register	Usage
Detect lane with receiver errors	`EVENT_COUNTER_CONTROL_REG` `EVENT_COUNTER_DATA_REG`	Program`EVENT_COUNTER_CONTROL_REG[27:24]` to select Group0 Program `EVENT_COUNTER_CONTROL_REG[11:8]` to select the Lane number Program `EVENT_COUNTER_CONTROL_REG[23:16]`to select one of the following events: 0x00 = RX Elastic Buffer Overflow Error 0x01 = RX Elastic Buffer Underflow Error 0x02 = Decode Error 0x03 = Disparity Error 0x04 = SKP OS Parity Error (Gen3/Gen4) 0x05 = SYNC Header Error (Gen3/Gen4) 0x06 = RX valid de-assertion without EIOS Enable the Event counter and read the `EVENT_COUNTER_DATA_REG` to see if the selected receiver error has occurred on the selected lane. **Note:** You can also use the top level signals `cdm_ras_des_ec_info_common[10]`(`rmlh_deskew_alignment_err`) and `cdm_ras_des_ec_info_common[6]`(`rmlh_rcvd_err`) as interrupts for error indication.

- **Debug Using Common Controller Signals and Registers**

  The controller reports RX errors as correctable errors. The application software can read the Correctable Error Status Register of the AER Capability or `CORR_ERR_STATUS_OFF[0]`to see if RX error has been reported. For more information, see "Correctable Errors" on page 229.

- **General Debug Guidelines**

  - Decode and Disparity Errors: Check if the PHY is reporting any decode and/or disparity errors. PHY reports decode or disparity errors by updating RXSTATUS. Some PHYs may replace bad RXDATA with EDB. Decode error has higher priority over disparity error. The controller treats decode or disparity errors the same and report them as correctable errors. RXSTATUS reporting is valid only when RXVALID is asserted. Disparity errors are valid for Gen1/Gen2 only.
    - RXSTATUS =100b represents Decode Error
    - RXSTATUS =111b represents Disparity Error
  - Overflow and Underflow Errors: Check if the PHY is reporting any overflow and/or underflow errors. PHY reports overflow and underflow errors by updating RXSTATUS. There could be missing RXDATA in overflow conditions. The PHY may insert EDB in RXDATA in underflow conditions. Overflow errors has higher priority over underflow errors. Check if the PHY is properly adding and/or removing SKP if overflow or underflow errors are reported. The PCIe Controller does not do anything with overflow or underflow errors reported.
    - RXSTATUS =101b represents Overflow Error
    - RXSTATUS =110b represents Underflow Error
  - SKP Add and Remove Status: Check if the PHY is properly adding and removing SKP to handle clock correction. SKP are used in clock correction to avoid RX elastic buffer overflow or underflow conditions. PHY reports SKP added or removed by updating RXSTATUS. PHY adds or removes SKP on RXDATA. The controller does not do anything with SKP add or remove reported. Check

if the PHY is not removing all SKP in the ordered set. The controller requires at least one SKP immediately following the COM when removing SKP.

- RXSTATUS =001b reports SKP added
- RXSTATUS =010b reports SKP removed

- Multi-lane Application: Check which PHY lane is reporting errors. Each active PHY lane reports RXSTATUS independently.
- TX De-emphasis (Gen1 and Gen2): Try adjusting the TX de-emphasis and margin of the link partner if there is a RX link stability issue in Gen1 or Gen2. Tuning may be necessary since TX settings optimized for Gen1 may not always be ideal for Gen2. Adjust TXDEEMPH to optimize the TX signal quality by changing the TX de-emphasis of the signal. A larger TX de-emphasis may improve the TX signal quality over larger channel insertion loss. TXDEEMPH adjustment is for Gen2 only, but can be used for debugging Gen1 also, if the PHY allows it. Adjust the TXDEEMPH of EP and/or RP depending on the issue. For example, if the EP does not have enough RX margin to recover the data and is having link stability issues, then adjust the TXDEEMPH of the RP.
  - TXDEEMPH =00b selects -6.0 dB
  - TXDEEMPH =01b selects -3.5 dB
  - EP: EP's TX de-emphasis is selected by the select_deemphasis variable as described in PCI Express Base Specification, Revision 6.0, Version 1.0. This variable is set to the SDE bit of the TS1 OS received from RP during LTSSM Recovery. This variable can also be set to the `GEN2_CTRL_OFF[20]` bit in LTSSM Detect and Loopback states. The value read from `LINK_CONTROL2_LINK_STATUS2[16]`register depends on the LTSSM state in which this register is read.
  - RP: To adjust the RP's TX de-emphasis in Gen2, the application software can update the `PCIE_CAP_SEL_DEEMPHASIS` field of the `LINK_CONTROL2_LINK_STATUS2` register. This is a standard PCIe register. The default value of this bit is configurable through coreConsultant using the `SEL_DE_EMPHASIS` parameter. After the `SEL_DEEMPHASIS`field is updated, the MAC requests the PHY to adjust its TX de-emphasis using the TXDEEMPH control of the PIPE interface. The current value of TXDEEMPH in Gen2 updates in the `PCIE_CAP_CURR_DEEMPHASIS` field (*bit 16*) of the `LINK_CONTROL2_LINK_STATUS2` register.This is a standard PCIe register. The default value of this bit is 1b. Application logic can read this register while in Gen2 to get the current TX de-emphasis of the local PHY.
- TX De-emphasis (Gen3 and Gen4): The TX de-emphasis setting for Gen3 and Gen4 is controlled during link equalization. Review the link equalization settings to determine if there are more optimal TX Presets to use. For more information, see "Link Equalization" on page 220.
- TX Margining: Adjust TXMARGIN to optimize the TX signal quality by changing the TX differential amplitude. A higher TX differential amplitude may typically improve the TX signal quality over larger channel insertion loss. Adjust the TXMARGIN of EP and/or RP depending on the issue. For example, if the EP does not have enough RX margin to recover the data and is having RX link stability issues, then adjust the TXMARGIN of the RP.
  - TXMARGIN =000b selects normal operating range.
  - Normal operating range is typically programmable in the PHY.
- RX Electrical Idle: Check if the PHY is receiving valid RX data and not inadvertently entering and exiting RX electrical idle during normal L0 operation. Consult your PHY vendor for debug

features that allow you to adjust the PHY to increase or decrease the RX electrical idle detection threshold.

Gen1 and Gen2: RXELECIDLE can be used for indicating RX electrical idle entry or exit. During normal L0 operation, RXELECIDLE toggling can mean the link is unstable.

- Entry: Check if the voltage amplitude of the RX data is below the threshold for RX electrical idle entry detection. PHY reports RX electrical idle detected by asserting RXELECIDLE when the RX+/RX- voltage amplitude is below the threshold. PHY loses bit and symbol lock and de-asserts RXVALID. The controller must stop using the invalid RXDATA.
- Exit: Check if the voltage amplitude of the RX data is above the threshold for RX electrical idle exit detection. PHY reports RX electrical idle exit by de-asserting RXELECIDLE when the RX+/RX- voltage amplitude is above the threshold. PHY obtains bit and symbol lock and asserts RXVALID after detecting COM. The controller can start using the valid RXDATA.

  Gen3 and Gen4: RXELECIDLE signaling is not reliable at high line rates.
- Entry: Check if the controller can detect EIOS and enter RX electrical idle successfully. RXELECIDLE cannot be used for indicating RX electrical idle entry. The controller is expected to infer RX electrical idle entry by using EIOS.
- Exit: Check if the PCIe Controller can detect EIEOS and exit RX electrical idle successfully. While in RX electrical idle, the first RXELECIDLE falling edge can be used for indicating RX electrical idle exit. EIEOS is a low frequency reported one-zero pattern to assist PHY in RX electrical idle exit detection.

- **FPGA Debug**

  Monitor the following PIPE signals for RX error status:

  - RXVALID
  - RXSTATUS
  - RXELECIDLE

  Set analyzer to trigger on RXSTATUS not equal to 000b when RXVALID is asserted. Also set analyzer to trigger on RXELECIDLE assertion. For more information, see PHY Interface for the PCI Express, SATA, USB 3.2, DisplayPort, and USB4 Architectures (PIPE) Version 4.4.1, Version 5.1.1, Version 6.1.1.

G.3.2 Correctable Errors

Correctable errors are the most common consequence of poor link quality and can lead to link instability. The following are commonly seen correctable errors:

- RX Errors
- Bad TLP
- Bad DLLP
- Replay Number Rollover
- Replay Timer Timeout
- Advisory Non-Fatal Error
- Corrected Internal Error
- Header Log Overflow

Common Symptoms

- Link does not remain stable in L0 and goes down to Recovery

Debug Guidelines

- Debug using RAS DES

  If your controller has RAS DES enabled (`CX_RAS_DES_ENABLE =1`) you can monitor the following registers for correctable errors.

Table G-6 RAS DES Debug

Issue	RAS DES Register	Usage
Framing Errors	`SD_STATUS_L1LTSSM_REG`	Read bit[7] to check the detection of framing error. Read bits[6:0] to know the type of detected Framing error. **Note:** You can also use top level signal `cdm_ras_des_ec_info_common[9](rmlh_framing_err)` as interrupt for error indication.
Malformed TLPs	`SD_STATUS_L3_REG`	Read bit[7] to check the detection of a malformed TLP. Read bits[6:0] to know what condition caused the malformed TLP error. For example, TLP length mismatch, or invalid TC.
Disable transition to Recovery due to Framing Error	`SD_CONTROL2_REG`	For debug purpose, you can set bit[16] of `SD_CONTROL2_REG` register to disable transition to Recovery due to Framing error.
Force transition to Recovery	`SD_CONTROL2_REG`	For debug purpose, you can set bit[1] of `SD_CONTROL2_REG` register to force a transition to Recovery from L0 or L0s.

- **Debug Using Common Controller Signals and Registers**

  Read the Correctable Error Status Register (`CORR_ERR_STATUS_OFF`) to know which of the mentioned errors in Tables G-6 have been detected by the controller. This register can be read at both sides of the link, to know if errors are in TX or RX direction.

- **General Debug Guidelines**
  - RX Errors: For more information, see "Receiver(Rx) Errors" on page 226.
  - Bad TLP: This error is reported if the received TLP fails LCRC check, or has incorrect sequence number. This can occur as a result of bit errors in the link or due to receiver errors. Check if any Rx error is also reported along with Bad TLP.
  - If an analyzer trace is available, check if any sequence number errors are reported. This can help identify the Bad TLP.
  - Bad DLLP: This error is reported if the received DLLP fails CRC check. This can also occur as a result of bit errors or RX errors.
  - Replay Number Rollover: This error is reported if no ACK or NACK is received from the link partner for a particular TLP before the replay timer expires for three consecutive times. This can occur if the ACK/NACK DLLP is corrupted due to bit errors on the link and is not detected by the

controller. If an analyzer trace is available, check if all TLPs are receiving ACK/NACK from the link partner.

- Replay Timer Timeout: This happens if no ACK or NACK is seen by the PCIe Controller for a transmitted TLP before the Replay timer expires. For debug purpose, you can try increasing the Replay Timer timeout limit using the `TIMER_MOD_REPLAY_TIMER` field of the `TIMER_CTRL_MAX-_FUNC_NUM_OFF`register.
- Advisory Non-Fatal: This bit indicates that any of the following errors have occurred and the severity of these has been set to Non-Fatal in the Uncorrectable Error Severity Register (`UNCOR-R_ERR_SEV_OFF`).
  - Poisoned TLP received
  - TLP with ECRC error received
  - Requester with Completion Timeout
  - Received Completion with UR status
  - Received Completion with CA status
  - Received Unexpected CPL

  To further debug an Advisory Non-Fatal error, increase the severity of all errors in Uncorrectable Error Severity Register to Fatal, this helps to identify the errors which caused the Advisory Non-Fatal bit to be set.
- Corrected Internal Error: The PCIe Controller does not use this bit, therefore it is always 0.
- Header Log Overflow: A Header Log Overflow occurs when an error that requires header logging is detected and either the number of recorded headers supported by the PCIe Controller has been reached, or the Multiple Header Recording Enable bit is not Set and the First Error Pointer register is valid.

- **FPGA Debug Guidelines**

  Trigger on LTSSM transition from L0 to RCVRY_LOCK to capture an analyzer trace for debug of correctable errors.

G.3.3 Uncorrectable Errors

Uncorrectable errors are further classified as Fatal Errors and Non-Fatal Errors based on the Severity programmed in the Uncorrectable Error Severity Register.

Fatal Errors: These errors render the link and related hardware unreliable. These are only seen when any of the link components is severely broken and a reset of the link components is needed to recover from these. The following errors are programmed to be Fatal errors by default:

- Malformed TLP Error
- Data Link Protocol Error
- Uncorrectable Internal Error
- Surprise Down
- Receiver Overflow
- Flow Control Protocol Error

Non-Fatal Errors: These errors cause a particular transaction to be unreliable, but the Link is otherwise fully functional. These can be recovered from without resetting the components on the link. The following errors are programmed as Non-Fatal errors by default:

- Poisoned TLP Received
- ECRC Check Failed
- Unsupported Request Received
- Completion Timeout
- Completer Abort
- Unexpected Completion

Common Symptoms

- Link goes down to Detect
- Application logic sees Completion timeouts

Debug Guidelines

- Debug using RAS DES

Table G-7 RAS DES Debug

Issue	RAS DES Register	Usage
Check Malformed TLP Status	`SD_STATUS_L3_REG`If your controller has RAS DES enabled (`CX_RAS_DES_ENABLE =1`) you can monitor the following registers for uncorrectable errors.	Read bit[7] to check the detection of Malformed TLP. Read bits[6:0] to know what condition caused the Malformed TLP error. For example, TLP length mismatch or invalid TC.

Note: You can also use signals from the top-level bus `cdm_ras_des_ec_info_common` and `diag_status_bus` as interrupts for indication of these errors. For more information, see "Signal Descriptions" chapter of the *Reference Manual*.

- **Debug Using Common Controller Signals and Registers**
  - Application software can read the Uncorrectable Error Status Register of the AER Capability or `UNCORR_ERR_STATUS_OFF`register to see the reported uncorrectable or fatal errors.
  - If more than one bits are set in the Uncorrectable Error Status Register, read the `ADV_ERR_CAP_C-TRL_OFF[4:0]`(First Error Pointer) to know which of these errors occurred first. Read the `HDR_LOG_0_OFF`register to get the header of the errored TLP
- **General Debug Guidelines**

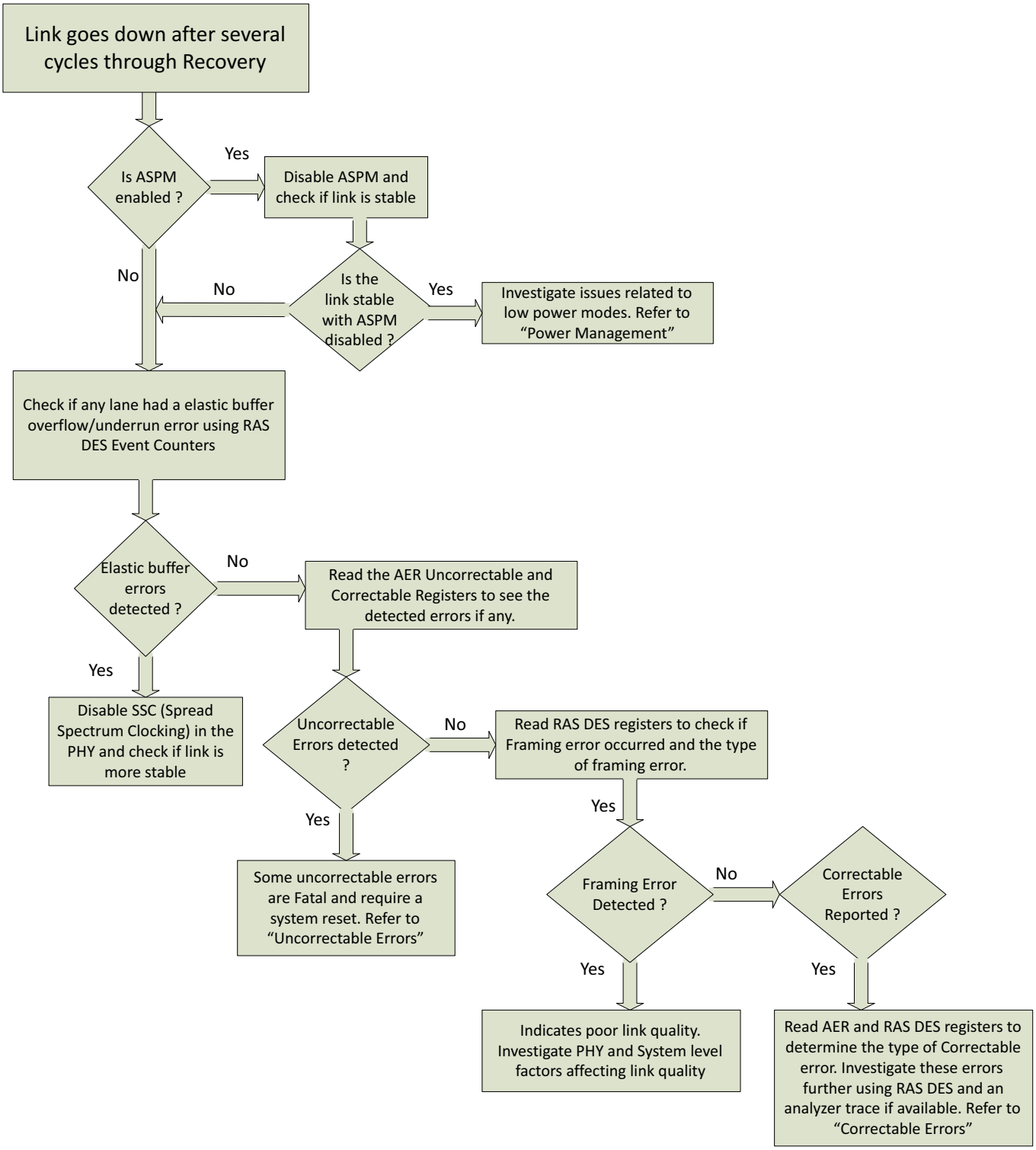
  Fatal Errors
  - **Malformed TLP Error:**Reported if any header field of the TLP sent by the link partner is incorrect. Read `SD_STATUS_L3_REG[6:0]`to know what condition caused the malformed TLP error.
  - **Data Link Protocol Error:**Reported if the sequence number of received TLP is invalid. If this error is reported, check the analyzer trace to see received TLPs and find out which TLP has invalid sequence number.
  - **Uncorrectable Internal Error:** Reported if your application logic drives the `app_err_bus[9]`PCIe Controller input to '1' or if the PCIe Controller detects an uncorrectable datapath or RAM parity/ecc error.

- **Receiver Overflow:**Reported when the credit check on a received TLP fails, which means the receive queue buffer does not have enough space to accept the received TLP. This can happen if the link partner ignores flow control updates or is not receiving correct flow control updates. Use an analyzer trace to check correct exchange of FC Update DLLPs.
- **Flow Contr**ol Protocol Error: Occurs if no DLLP is received within a 200us window (Watch Dog Timer expiration limit). This indicates that the link quality is severely deteriorated.

Non-Fatal Errors

- **Poisoned TLP Received:** Reported if the link partner sends a TLP with the EP bit set.
- **ECRC Check Failed:** Reported when the ECRC of received TLP does not match the calculate ECRC. This indicates corruption of the TLP header or payload.
- **Unsupported Request Received:**Reported, for example, when a received MEM TLP does not hit any of the enabled BARs of a device. In that case, check the address of the received TLP is valid.
- **Completion Timeout:**Reported when an outbound non-posted request does not receive a CPL before the CPL timer expires. This can happen, for example, if the received CPL is malformed, or had an ECRC error and is dropped by the PCIe Controller. It could also indicate that the CPL is not sent by the link partner. Check the analyzer trace to find the expected CPL.
- Completion Abort: Reported when a CPL TLP is received with status “Completer Abort”.
- Unexpected Completion: Indicates that a CPL TLP is received for which the corresponding non-posted request is not transmitted or is no longer outstanding. This can happen if the TAG field or other header fields of the received CPL do not match the corresponding request header fields.

Figure G-2 Example Debug Flow



G.4 Power Management

After successful link training and linkup, the PCIe link can enter low power states to save power. Most of the power savings is achieved in the PHY and through clock gating. The PCIe Controller uses the power management capabilities to request the PHY to enter low power states and to negotiate clock removal/restoration. The following are the primary PCIe low power link states.

- "L0s"
- "L1 and L1SS" on page 237
- "L2" on page 238

For more information, see PCI Express Base Specification, Revision 6.0, Version 1.0.

G.4.1 L0s

The PCIe link can enter L0s to save power. The PHY normally operates in P0s power state when the PCIe Controller is in L0s link state. P0s is a low power state with low exit latency compared to other low power states. The PHY is expected to recover RX data and lock quickly when exiting P0s. PCLK is operational in L0s. The PHY operates on Vmain power and only the TX is in electrical idle. This section provides silicon debug guidelines for common L0s entry and exit issues.

Common Symptoms

- PCIe link not able to enter L0s successfully
- PHY does not respond to P0s entry request on all active lanes
- PCIe Controller is unable to enter L0s due to pending requests
- PCIe link not able to exit L0s successfully
- PHY does not respond to P0s exit request on all active lanes
- PHY is unable to recover RX data and lock quickly after exiting L0s
- PCIe Controller exits L0s to Recovery after N_FTS timeout
- PCIe Controller is unable to linkup after exiting L0s

Debug Guidelines

- Debug using RAS DES

  If your controller has RAS DES enabled (`CX_RAS_DES_ENABLE =1`) you can monitor the following registers for L0s errors.

Table G-8 RAS DES Debug

Issue	RAS DES Register	Usage
Check PIPE powerdown signal	`SD_STATUS_L1LTSSM_REG`	Read bits[10:8] to check the value of PIPE Powerdown signal. Controller drives the powerdown signal to P0s to request L0s entry to PHY.
Check PIPE Txelecidle	`SD_STATUS_L1LANE_REG`	Read bit[20] to check the current value of Txelecidle PIPE signal of the selected lane.

Issue	RAS DES Register	Usage
Check PIPE Rxelecidle	`SD_STATUS_L1LANE_REG`	Read bit[19] to check the current value of Rxelecidle PIPE signal of the selected lane. PHY asserts Rxelecidle to indicate RX electrical idle entry and de-asserts Rxelecidle to indicate electrical idle exit.
Check if N_FTS timeout occurred	`EVENT_COUNTER_CONTROL_REG` `EVENT_COUNTER_DATA_REG`	Enable event counter group #1,event #0x08 to check if any N_FTS timeout occurs.

- **Debug Using Common Controller Signals and Registers**

  Read the `PL_DEBUG0_OFF[5:0]` to know the current LTSSM state. L0s involves the following LTSSM substates:

  - Tx_L0s.Entry
  - Tx_L0s.Idle
  - Tx_L0s.FTS
  - Rx_L0s.Entry
  - Rx_L0s.Idle
  - Rx_L0s.FTS

  Read the `Link Control[1:0]`to check if L0s is enabled through the ASPM Control field.

  - **00b:** Disable
  - **01b:** L0s Entry Enabled
  - **10b:** L1 Entry Enabled
  - **11b:** L0s and L1 Entry Enabled

  Read `ACK_F_ASPM_CTRL_OFF[23:16]`(if PHY uses Common REFCLK architecture) or `ACK_F_ASPM_CTRL_OFF[15:8]`(if PHY uses independent REFCLK) to check the number of Fast training sequences (N_FTS) the controller advertises. Try increasing N_FTS if the PHY is not able to lock within the N_FTS timeout.

- **General Debug Guidelines**

  - **L0s Entry:** Check if no higher stage of power down is requested if the USP does not enter L0s.
  - **L0s Exit:** Check if any of the following conditions are met if USP does not exit L0s:
    - DLLP or TLP is pending to be send
    - Link partner requested PCIe link to enter Recovery.
  - **TX L0s Entry:**
    - Check if the Controller transmits EIOS, asserts TXELECIDLE, and sets POWERDOWN =01b to request PHY to enter P0s.
    - After transmitting the last UI of the last EIOS, check if the PCIe link enters TX electrical idle within the maximum latency defined in the PCIe Base specification. The maximum latency is TTX-IDLE-SET-TO-IDLE (8ns).
    - Check if the PCIe link remains in TX electrical idle for at least the minimum duration defined in the PCIe Base specification. The minimum duration is TTX-IDLE-MIN (20 ns).

- Check if the PHY asserts PHYSTATUS for 1 PCLK to indicate successful P0s entry.
- Check if the PCIe link is entering and exiting TX electrical idle repeatedly. This behavior can cause an unstable PCIe link. For more information, see "Link Stability" on page 226.

❑ **TX L0s Exit:**

- Check if the PCIe Controller sets POWERDOWN =00b to request PHY to exit P0s.
- Check if the PHY exits P0s by asserting PHYSTATUS for 1 PCLK to indicate done.
- Check if the PCIe Controller de-asserts TXELECIDLE and transmits enough FTS to assist link partner to quickly achieve bit and symbol lock. Try increasing N_FTS if the PHY is not able to lock within the N_FTS timeout. Up to 255 N_FTS can be send or 4096 if Extended Synch bit is set.
- Check if the Controller transmits EIEOS follow by SDS. The Controller my exit TX L0s and enter Recovery if the link partner enters Recovery.

❑ **RX L0s Entry:**

- Check if the PHY asserted RXELECIDLE to indicate RX electrical idle entry.
- Check if the Controller detects RX electrical idle for at least the minimum duration defined in the PCIe Base specification. The minimum duration is TTX-IDLE-MIN (20 ns).
- Check if the PCIe link is entering and exiting RX electrical idle repeatedly. This behavior can cause an unstable PCIe link. For more information, see "Link Stability" on page 226.

❑ **RX L0s Exit:**

- Check if the PHY de-asserted RXELECIDLE to indicate RX electrical idle exit.
- Check if the Controller can detect FTS and is it able to lock within the N_FTS timeout. Check if N_FTS timeout occurred using RAS DES event counter group #1 (event #0x08) or detect a RX L0s exit to Recovery transition. Request the link partner to increase N_FTS, if timeout occurs.
- Check if the Controller detected EIEOS follow by SDS.
- Check if the Controller exits RX L0s into L0.

- **FPGA Debug Guidelines**

  Monitor the following PIPE signals for L0s entry and exit status:

  ❑ POWERDOWN

  ❑ PHYSTATUS

  Set analyzer to trigger on first PHYSTATUS rising edge after POWERDOWN change to 01b for entering L0s. Set analyzer to trigger on first PHYSTATUS rising edge after POWERDOWN change out of 01b for exiting L0s.

G.4.2 L1 and L1SS

The PCIe link can enter L1 to save power or enter L1 sub-state (L1SS) to save even more power. L1.1 and L1.2 are the two sub-states making up L1SS. L1.2 has more power savings as compared to L1.1. The PHY normally operates in P1 power state when the PCIe Controller is in L1 or L1SS link state. P1 is a lower power state with longer exit latency compared to P0s power state. PCLK is operational in L1 but can be off in L1SS. PIPE signaling is asynchronous when PCLK is off. The PHY operates in Vmain power, and both TX and RX should be in electrical idle. The PHY is expected to reduce the necessary power after the PCIe Controller puts the PHY in P1. This section provides silicon debug guidelines for common L1 and L1SS entry and exit issues.

G.4.3 L2

The PCIe link can enter L2 to save power. The PHY normally operates in P2 power state when the Controller is in L2 link state. P2 is the lowest power state with longest exit latency compared to P0s and P1 power states. PCLK is off and PIPE signaling is asynchronous in L2. The PHY typically operates in Vaux power, and both TX and RX should be in electrical idle. The PHY is expected to reduce the necessary power after the PCIe Controller puts the PHY in P2. This section provides silicon debug guidance for common L2 entry and exit issues.

Common Symptoms

- PCIe link not able to enter L2 successfully
- PHY does not respond to P2 entry request on all active lanes
- Application logic is unable to remove PCLK when entering L2
- Controller is unable to enter L2
- PCIe link not able to exit L2 successfully
- PHY does not respond to P2 exit request on all active lanes
- Application logic is unable to restore PCLK when exiting L2
- Controller is unable to wake up the link partner from L2
- Controller is unable to linkup after exiting L2

Debug Guidelines

- Debug using RAS DES

  If your controller has RAS DES enabled (`CX_RAS_DES_ENABLE =1`) you can monitor the following registers for L2 errors.

Table G-9 RAS DES Debug

Issue	RAS DES Register	Usage
Check the state of Power Management FSM	`SD_STATUS_PM_REG`	Read bits [11:8] to know the state of the power management master FSM and bits [4:0] for slave FSM.

- Debug Using Common Controller Signals and Registers
  - If your device is an Upstream port, check that the application has asserted `app_ready_entr_l23` to allow L2 entry.
  - Monitor top level signal `cxpl_debug_info_ei[0]`(EIOS detected) to check if EIOS is received after L2 entry initiation.
  - Check if `apps_pm_xmt_pme` is asserted by upstream port application to initiate L2 exit.
- **General Debug Guidelines**

  L2 Entry

  - L2 link state is entered when PM software transitions a device into D3 power state or when the Upstream Port receives a PME_Turn_Off message from the Downstream Port. Check the analyzer trace to see PME_Turn_Off/PME_TO_Ack handshake has been completed.

- Check the analyzer trace to see that Upstream Port sends PM_Enter_L23 DLLP and RP responds with PM_Request_Ack.
- If link fails to enter L2, check if EIOS is received after L2 entry initiation.

L2 Exit

- L2 exit can be initiated by the Upstream Port by assertion of `apps_pm_xmt_pme` input to the Controller, or by the assertion of board level PESRT#.
- If your controller does not have Vaux and does not support the slow auxiliary clock (`aux_clk`), a cold reset (power on reset) should be asserted to exit L2.

- **FPGA Debug Guidelines**

  Monitor the following PIPE signals for L2 entry and exit status between the PCIe Controller and PHY:

  - POWERDOWN
  - TXELECIDLE
  - PHYSTATUS

  Set analyzer to trigger on first PHYSTATUS rising edge after POWERDOWN change to 11b for entering L2

H

Application Interface Integration Guide

The following groups of Input/Output (IO) interfaces are driven to/from your application logic. These interfaces are configuration dependent.

- "AXI Interfaces" on page 242
- "Native Transmit and Receive Interfaces (XALI0, XALI1, XALI2, Bypass, RTRGT1)" on page 249
- "Data Bus Interface (DBI)" on page 250
- "External Local Bus Interface (ELBI)" on page 251
- "Message Signaled Interface (MSI)" on page 252
- "MSI-X Interface" on page 254
- "Vendor Message Interface (VMI)" on page 256
- "Power Budgeting" on page 257
- "System Information Interface (SII)" on page 258
- "FLR" on page 272
- "Target Completion LUT" on page 273
- "RAS D.E.S Signals" on page 274
- "Configuration Intercept Interface (CII) Signals" on page 275
- "PHY Register Bus Interface" on page 276

The following sections describe the suggested connection and usage of these interfaces in your SoC.

H.1 AXI Interfaces

The controller can have the following AXI interfaces depending on your configuration:

- AXI Master Interface
- AXI High Priority Master Interface (2nd Master)
- AXI Slave Interface
- AXI DBI Slave Interface

Table H-1 describes the connection of these interfaces.

Table H-1 AXI Master Interface Connections

Signal	Direction	Description	Connection/Notes
mstr_*id mstr_*valid mstr_a*addr mstr_a*len mstr_a*size mstr_a*burst mstr_a*lock mstr_a*cache mstr_a*prot mstr_*ready mstr_a*qos mstr_a*bar mstr_a*domain mstr_a*snoop mstr_wstrb mstr_*data mstr_*last mstr_*resp	As per AXI Specification	Standard AXI Master Interface signals as per AXI specification	■ Connect to the Standard AXI Master interface of your AXI Fabric
mstr_a*addrp	Output	Write/Read address parity bits	■ Can be used by your application logic to check AXI Master address bus parity
mstr_a*misc_info	Output	Sideband bus carrying received PCIe TLP header information	■ Your application can capture this information in registers also and use it to obtain the BDF of received CFG TLPs, message code of Msg TLPs, and other header fields of MRd/MWr TLPs ■ If the iATU is enabled, this information can be obtained through the iATU, so this bus need not be used ■ Leave open, if not used
mstr_a*misc_info_last_dcmp_tlp	Output	Sideband signal that is asserted to indicate the last beat of an inbound decomposed request	■ Your application can use this signal to determine last beat of an inbound decomposed request ■ Leave open, if not used

Signal	Direction	Description	Connection/Notes
mstr_a*misc_info_hdr_34dw	Output	Sideband bus carrying the 3rd and 4th header DWs of received PCIe Msg/MsgD	■ Your application can use this for reception of Vendor Defined Messages and ATS Messages ■ Leave open, if not used
mstr_a*misc_info_tlpprfx	Output	Sideband bus carrying TLP Prefixes of received TLPs	■ Can be used by applications/configurations that support TLP Prefixing
mstr_a*misc_info_tlpln	Output	Sideband signal carrying the Lightweight Notification (LN) bit of received TLP header	■ Can be used by applications/configurations that support Lightweight Notification
mstr_a*misc_info_tph	Output	Sideband bus carrying TLP processing hint and steering TAG information of the received TLP.	■ Can be used by applications/configurations that support TLP processing hints.
mstr_a*misc_info_dma	Output	Sideband bus carrying information related to DMA transfers.	■ Can be used by configurations using PCIe controller DMA for debugging purposes ■ Leave open, if not used
mstr_a*misc_info_nw mstr_a*misc_info_ats	Output	Sideband signals carrying the received TLP's NW and AT header fields.	■ Can be used by applications that support PCI-SIG ATS
mstr_a*misc_info_func_num mstr_a*misc_info_vfunc_num	Output	Sideband signal carrying function number/virtual function number of received TLP when SRIOV or ARI is enabled	■ Can be used by applications to obtain the function number/virtual function number of received TLP. ■ Leave open, if not used
mstr_a*misc_info_vfunc_active	Output	Sideband signal, indicating if the received TLP matched a VF BAR	■ Can be used by your application to determine if the received TLP matched a VF BAR ■ Leave open. if not used
mstr_a*misc_info_ido	Output	Sideband signal carrying IDO field of received TLP header	■ Can be used by your application to determine IDO field of the received TLP header ■ Leave open, if not used
mstr_*misc_info_cpl_stat	Input	Sideband signal to control the PCIe Non-Posted Write/Read completion status to be sent on the wire	■ Your application can use this to override the completion status sent out for a successful completion ■ Tie to 0, if not used
mstr_armisc_info_zeroread	Output	Sideband signal indicating when a zero-length read is received from the PCIe wire	■ Leave open, if not used

Signal	Direction	Description	Connection/Notes
mstr_csysack mstr_cactive mstr_csysreq	Standard AXI low power interface signals	The controller does not support AXI low power modes	■ Tie inputs to inactive (1'b1) ■ Leave outputs open

Table H-2 AXI High Priority Master Interface Connections

Signal	Direction	Description	Connection/Notes
mstrh_*	This interface is present only if you enable the "High Priority Master" feature in your configuration. The description and usage of signals in this interface is identical to the "AXI Master Interface".		

Table H-3 AXI DBI Slave Interface Connections

Signal	Direction	Description	Connection/Notes
dbi_a* dbi_r* dbi_b* dbi_w*	This interface is present only if you enable the "Slave DBI" feature in your configuration. The description and usage of signals in this interface is identical to the "AXI Slave Interface".		

Table H-4 AXI Slave Interface Connections

Signal	Direction	Description	Connection/Notes
slv_*id slv_*valid slv_a*addr slv_a*len slv_a*size slv_a*burst slv_a*lock slv_a*cache slv_a*prot slv_*ready slv_a*qos slv_a*bar slv_a*domain slv_a*snoop slv_wstrb slv_*data slv_*last slv_*resp	As per AXI Specification	Standard AXI Slave interface signals as per AXI specification	■ Connect to the standard AXI slave interface of your AXI Fabric
slv_a*addrp	Input	Write/Read address parity bits.	■ Can be used by your application to drive parity bits for the AXI Slave write/read address bus

Signal	Direction	Description	Connection/Notes
slv_a*misc_info	Input	Sideband bus carrying outbound PCIe TLP header information	■ Your application can drive this bus to provide TLP header information ■ You can also tie this input to 0 and use the Internal Address Translation Unit (iATU) to provide the TLP header information
slv_a*misc_info[21]	Input	Used to indicate a shared DBI access	■ If your configuration uses shared DBI option, application should drive this bit to indicate that the access is a DBI access
slv_a*misc_info_hdr_34dw	Input	Sideband bus carrying the 3rd and 4th header DWs of outbound PCIe Msg/MsgD	■ Your application can use this for generation of Vendor Defined Messages and ATS Messages ■ Tie to 0, if not used
slv_a*misc_info_tlpprfx	Input	Sideband bus carrying TLP Prefixes of outbound TLPs	■ Can be used by applications/configurations that support TLP Prefixing
slv_a*misc_info_tlpln	Input	Sideband signal carrying the Lightweight Notification (LN) bit of outbound TLP header	■ Can be used by applications/configurations that support Lightweight Notification
slv_awmisc_info_p_tag	Input	Sideband bus carrying TAG number for outbound posted requests	■ Your application must drive this to 0, except when generating ATS invalidate requests
slv_a*misc_info_tph	Input	Sideband bus carrying TLP Processing Hint and Steering TAG information of outbound TLP	■ Can be used by applications/configurations that support TLP Processing hints
slv_awmisc_info_atu_bypass	Input	Allows bypass of Internal Address Translation on a per transaction basis	■ Can be asserted by your application when a request needs to bypass address translation ■ Tie to 0, if not used
slv_a*misc_info_nw slv_a*misc_info_ats	Input	Sideband signals carrying the outbound TLP's NW and AT header fields.	■ Can be used by applications that support PCI-SIG ATS
slv_a*misc_info_func_num slv_a*misc_info_vfunc_num	Input	Sideband signal carrying function number/VF number of outbound TLP when SRIOV or ARI is enabled	■ Can be used by your application to indicate the function/virtual function initiating the request. ■ Tie to 0, if iATU is used
slv_a*misc_info_vfunc_active	Input	Sideband signal, indicating if the outbound TLP is initiated by a VF	■ Can be used by your application to indicate when a virtual function initiates the request ■ Tie to 0, if iATU is used

Signal	Direction	Description	Connection/Notes
slv_a*misc_info_ido	Input	Sideband signal carrying IDO field of outbound TLP header	■ Can be used by your application to determine the IDO field of outbound TLP header ■ Leave open, if not used
slv_wmisc_info	Input	Sideband signal used to poison outbound write requests	■ Your application can drive this to set the EP bit of outbound TLP ■ Tie to 0, if not used
slv_csysack slv_cactive slv_csysreq	Standard AXI low power interface signals.	The controller does not support AXI low power modes	■ Tie input to inactive (1'b1) ■ Leave outputs open

H.1.1 Other Integration Considerations

The following should also be considered while integrating your application with the PCIe AXI interface:

- "Mechanisms for Transaction Conversion Between AXI and PCIe"
- "Register Access Through Application Interface" on page 247
- "AXI Memory Mapping and Address Translation" on page 247

H.1.1.1 Mechanisms for Transaction Conversion Between AXI and PCIe

The standard AXI bus does not carry all the information contained in a PCIe TLP header, there are no corresponding signals for fields such as Type, TC, Attr and function number. The following mechanisms are available for communicating these header fields in inbound and outbound directions:

- **Option 1:** Hardware driven sideband signals on AXI Interfaces (*misc_info_*)
- **Option 2:**Software Programmable internal Address Translation Unit (iATU) – **RECOMMENDED**

Table H-5 discusses the Pros and Cons of these two options.

Table H-5 Pros and Cons of Options for Communicating the Header Files

Option	Pros	Cons
Option 1	■ No iATU needed, reduces the logic area of PCIe controller	■ Standard AXI fabric may not support sideband signals. ■ Additional application logic is needed to drive the sideband buses
Option 2	■ No AXI sideband signals needed ■ Easy to use through software programmable registers ■ iATU can also be used for address translation, type translation (for example Mem to Cfg translation), or TLP header field substitution	■ Some increase in logic area

H.1.1.2 Register Access Through Application Interface

During controller configuration you can select the interface to be used by your application to access PCIe controller registers. Table H-6 describes the options available and their pros and cons.

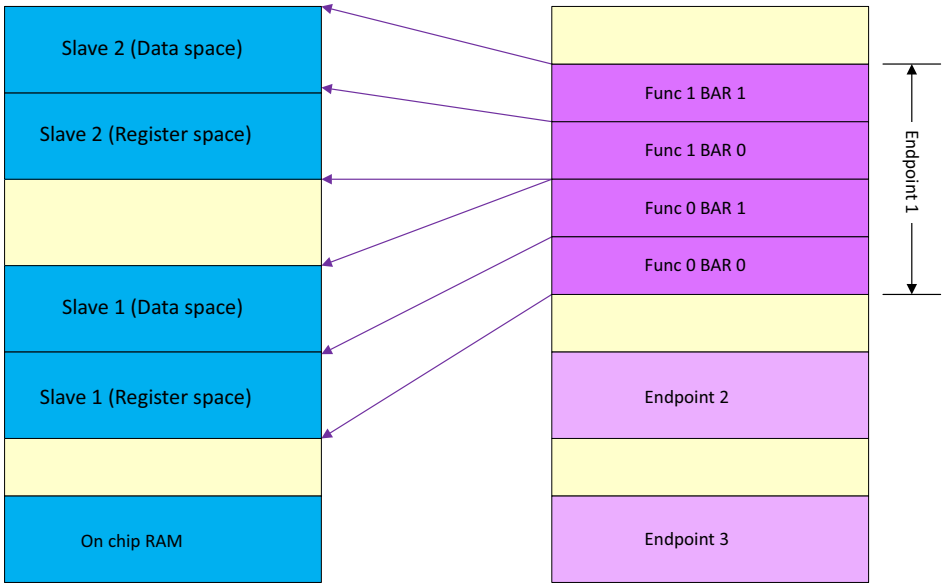
Table H-6 Register Access Options: Pros and Cons

Register Access Option	Interface Standard	Pros	Cons
Dedicated AXI Slave AXI Slave port used for register access only	AXI	■ No extra application logic required	■ One additional AXI Slave port
Shared AXI Slave AXI Slave port, shared for sending outbound packets as well as for register access	AXI	■ Reduces one AXI Slave port	■ Additional application logic required to drive signal slv_a*misc_info[21] which is used to differentiate between outbound and register access (1 for register, 0 for outbound request) ■ Outbound requests and DBI requests cannot be issued simultaneously
Native DBI Synopsys proprietary interface for register access	Native	■ Simple, low gate count, register read/write interface ■ Reduces one AXI Slave port	■ If your system uses a standard interface like APB for register access, you must design a bridge (such as apb2dbi) to drive this interface

H.1.1.3 AXI Memory Mapping and Address Translation

- The AXI Memory ranges allocated to the AXI master and slave ports of the controller in your AXI memory map are application dependent. These AXI address regions may be different from the PCIe memory space of the controller. The internal Address Translation Unit (iATU) provides a convenient mechanism for mapping your application's AXI address regions to address regions in the PCIe memory space.
- The iATU can be setup through software programmable registers to translate the address of inbound PCIe TLPs from the original PCIe address to a target address in the AXI memory map.
- Similarly, the outbound iATU can be programmed to translate the original AXI address of an outbound packet to a target PCIe address.

Figure H-1 Example Mapping of AXI Memory Regions to PCIe BARs of an Endpoint



H.2 Native Transmit and Receive Interfaces (XALI0, XALI1, XALI2, Bypass, RTRGT1)

- When the AXI bridge is not enabled, there are two transmit client interfaces (XAL10, XALI1) and an optional transmit client interface (XALI2) available for your application to send outbound PCIe requests or completions. For more information, see "Transmit Interfaces (XALI0/1/2)" section in the "Signal Interfaces" chapter of the *Databook*.
- Normally, applications use one client interface for sending one type of TLP (P on XALI0, NP on XALI1, and CPL on XALI2).
- Data should be provided continuously (without any gaps) on these transmit interfaces, once the transaction begins. It is recommended to use a FIFO based mechanism to drive these interfaces.
- By default, the RTRGT1 Interface is used to pass inbound error free TLPs to your application. If any receive queue is configured to be in Bypass mode, RBYP interface is used to pass those TLPs to your application.
- Your application can back pressure or halt the RTRGT1 interface if it is temporarily unable to accept more packets. To avoid any potential deadlock, it is recommended to design a TLP tracker in your application logic to assert the halt signal `trgt1_radm_pkt_halt` when your application buffer for a particular TLP type reaches a certain level. For more information, see "TRGT1 Packet Grant and Halt" section in the "Signal Interfaces" chapter of the *Databook*.

H.3 Data Bus Interface (DBI)

- Your application can use this interface to locally access the controller's internal registers in the CDM, or your external application registers on the ELBI.
- If present in your configuration, this interface should be driven from your application logic that converts your local CPU accesses to DBI protocol. For instance, an apb2dbi bridge.
- The DBI protocol is a simple register read/write protocol described in "DBI Protocol Transactions" section in the "Signals Interfaces" chapter of the *Databook*.
- The `dbi_addr[0]`along with `dbi_cs2` determine the target of a DBI transaction. Your application should implement logic to drive these appropriately for accessing different regions of the PCIe register space. For more information, see "Data Bus Interface (DBI) Access" in "Register Module, LBC, and DBI" section in the "Controller Operations" chapter of the *Databook*.
- If your configuration includes an AXI DBI Interface, the address bit that is used to determine the target of DBI access, depends on other parameters in your configuration. For more information, see "Data Bus Interface (DBI) Access" in "Register Module, LBC, and DBI" section in the "Controller Operations" chapter of the *Databook*.

H.4 External Local Bus Interface (ELBI)

- This interface is provided for your application to implement any registers in the PCIe configuration space above the location specified by `CONFIG_LIMIT` parameter.
- The ELBI enables access to these registers over the wire or by local CPU through the DBI. For more information, see "Register Configuration Space Overview" in "Register Module, LBC, and DBI" section in the "Controller Operations" chapter of the *Databook* and "External Local Bus Interface (ELBI)" section in the "Signals Interfaces" chapter of the *Databook*.
- ELBI can be used, for example, to implement application programmable registers to control the AXI sideband signals.
- If ELBI is not used, corresponding outputs should be left open and inputs should be tied to 0, except for the`ext_lbc_ack`. To avoid any potential deadlock due to unintended ELBI access, drive `ext_lbc_ack` with null data when`lbc_ext_cs` is 1.

H.5 Message Signaled Interface (MSI)

Your upstream port application logic can issue MSI requests using,

- MSI interface (Method A), or
- Normal Traffic Interface (XALI0/XALI1/XALI2/or AXI slave interface) (Method B)

Figure H-2 and Figure H-3 show how to connect the MSI interface I/O to your application logic using Method A and Method B, respectively.

Figure H-2 Guidelines for Connection of MSI and Normal traffic Interface Signals (Method A)

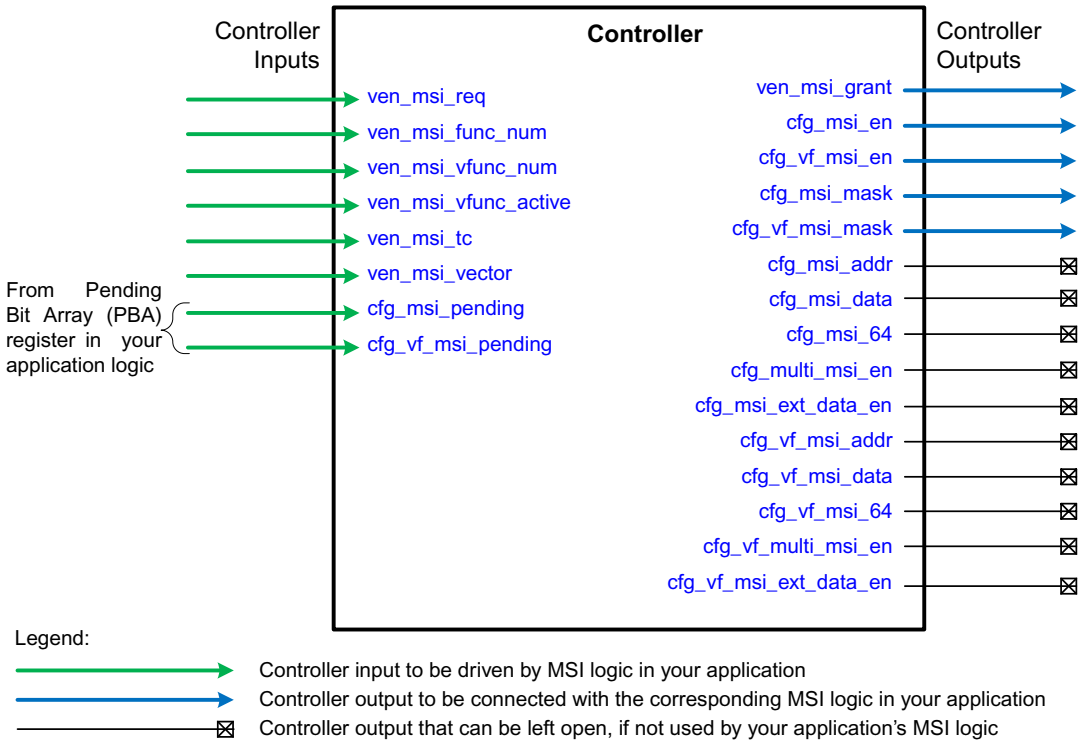
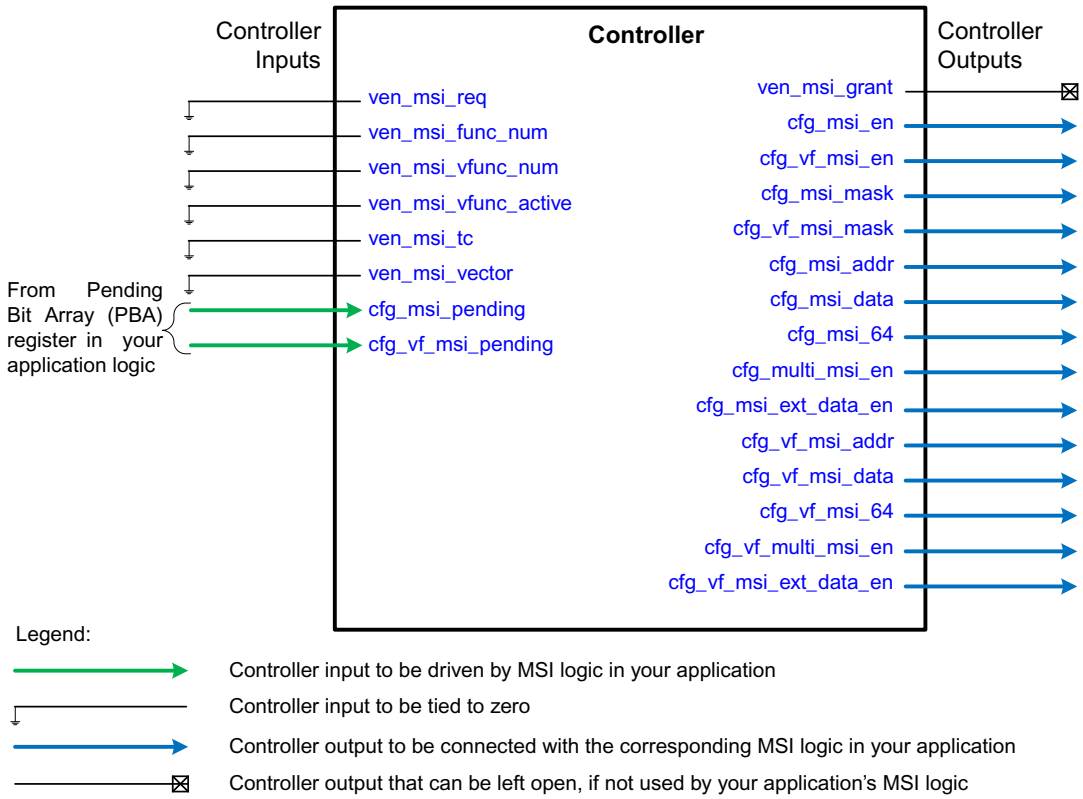


Figure H-3 Guidelines for Connection of MSI and Normal Traffic Interface Signals (Method B)



H.6 MSI-X Interface

You can use the MSI-X (MSI-X Table, PBA RAMs, and the associated logic) feature of the controller by setting the parameter `MSIX_TABLE_EN` to 1. If you are not using the controller provided MSI-X implementation, your application logic must implement the MSI-X Table, PBA RAMs, and the associated logic. For more information, see "Interrupt Generation (USP)" in "Interrupts" section of the "Controller Operations" chapter of the *Databook*.

When Integrated MSIX Table is not enabled, then similar to MSI, to generate MSI-X your application logic can use either the MSI Interface (Method A) or the Normal Traffic Interface (Method B) as shown in Figure H-4 and Figure H-5.

Figure H-4 Guidelines for Connecting MSI-X and Normal Traffic Interface Signals (Method A)

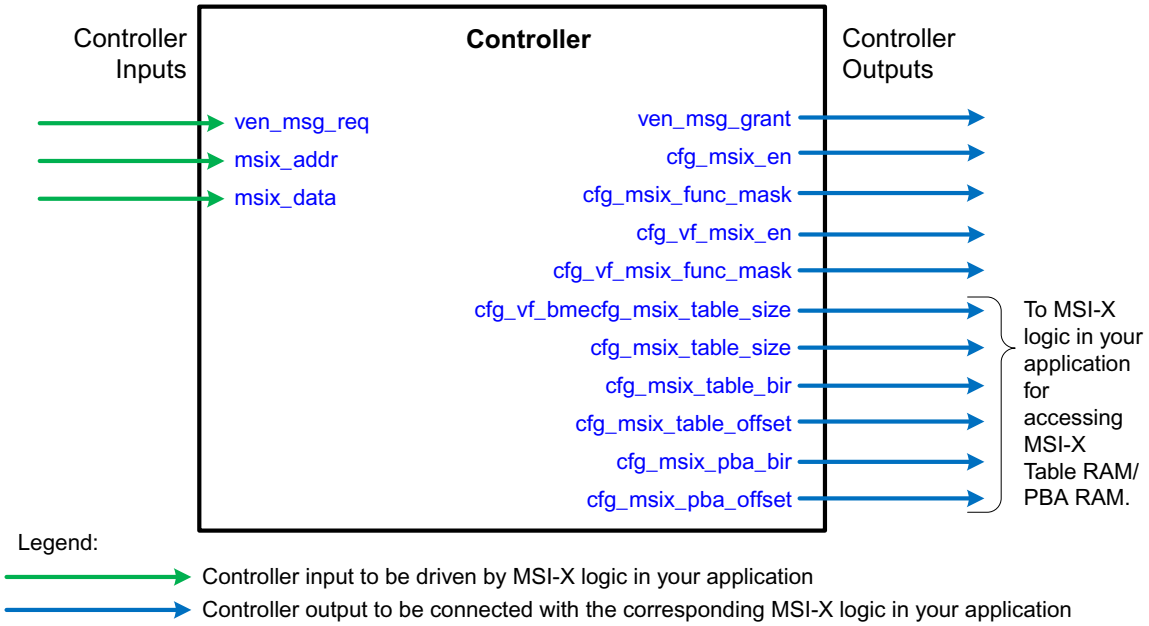
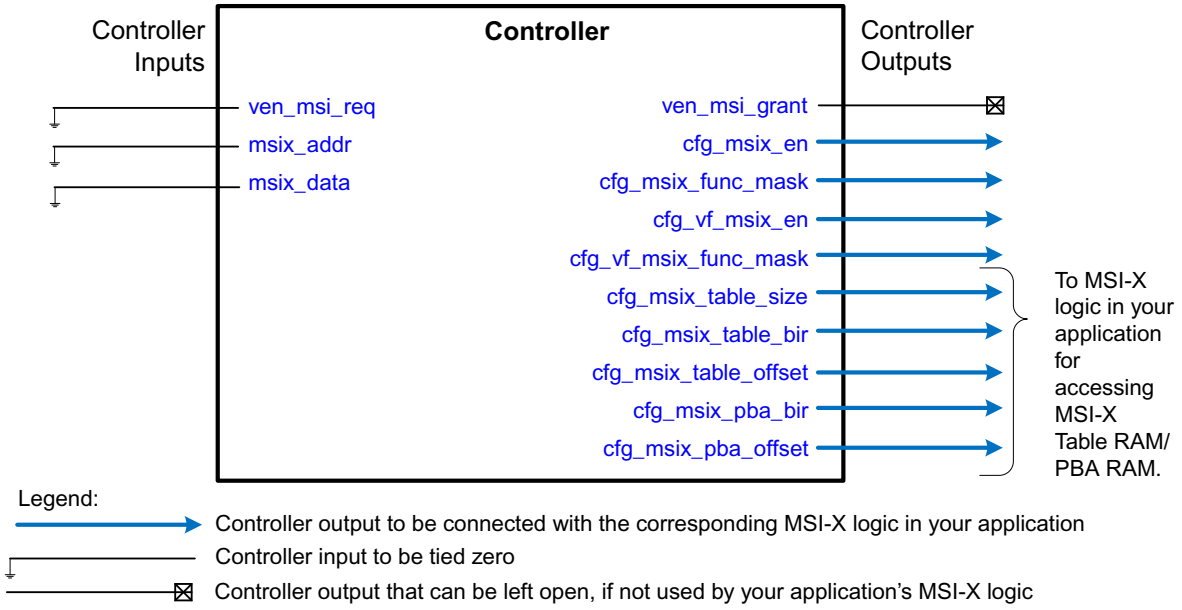


Figure H-5 Guidelines for Connecting MSI-X and Normal Traffic Interface Signals (Method B)

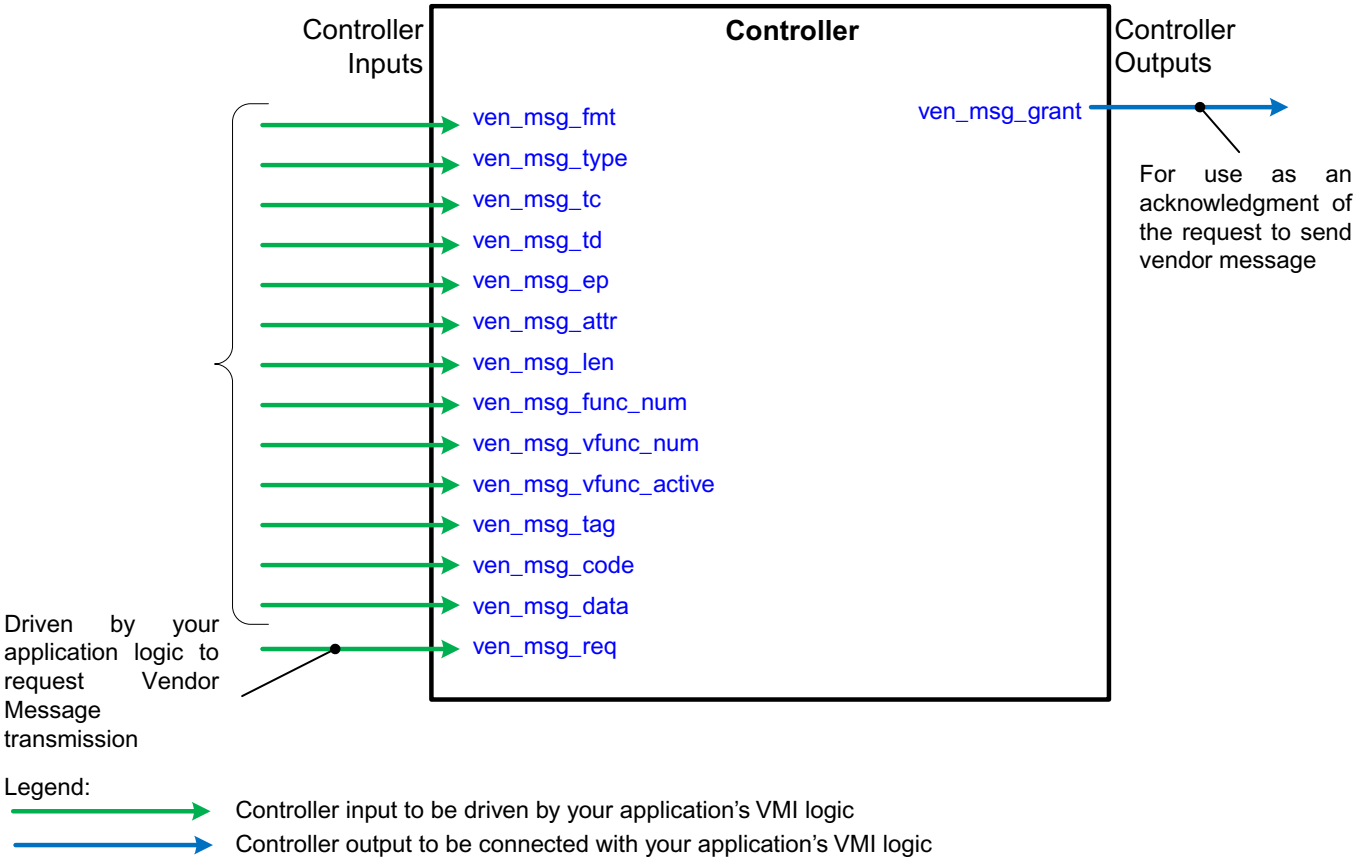


H.7 Vendor Message Interface (VMI)

You can send Vendor Defined Messages using VMI. Alternatively, the iATU can be used to send Vendor Defined Messages by setting up an Outbound iATU region for Mem to Msg type translation. For more information, see "iATU Outbound MSG Handling" in "Internal Address Translation Unit (iATU)" section of the "Controller Operations" chapter of the *Databook*.

If not used, the inputs of this interface can be tied to 0, and outputs can be left open.

Figure H-6 Guidelines for Connecting VMI Signals



H.8 Power Budgeting

If your configuration supports Power Budgeting capability `PWR_BUDGET_CAP_ENABLE =1`, then your application logic must connect to this interface as described in "Power Budgeting Signals" section in the "Signal Descriptions" chapter of the *Reference Manual*.

H.9 System Information Interface (SII)

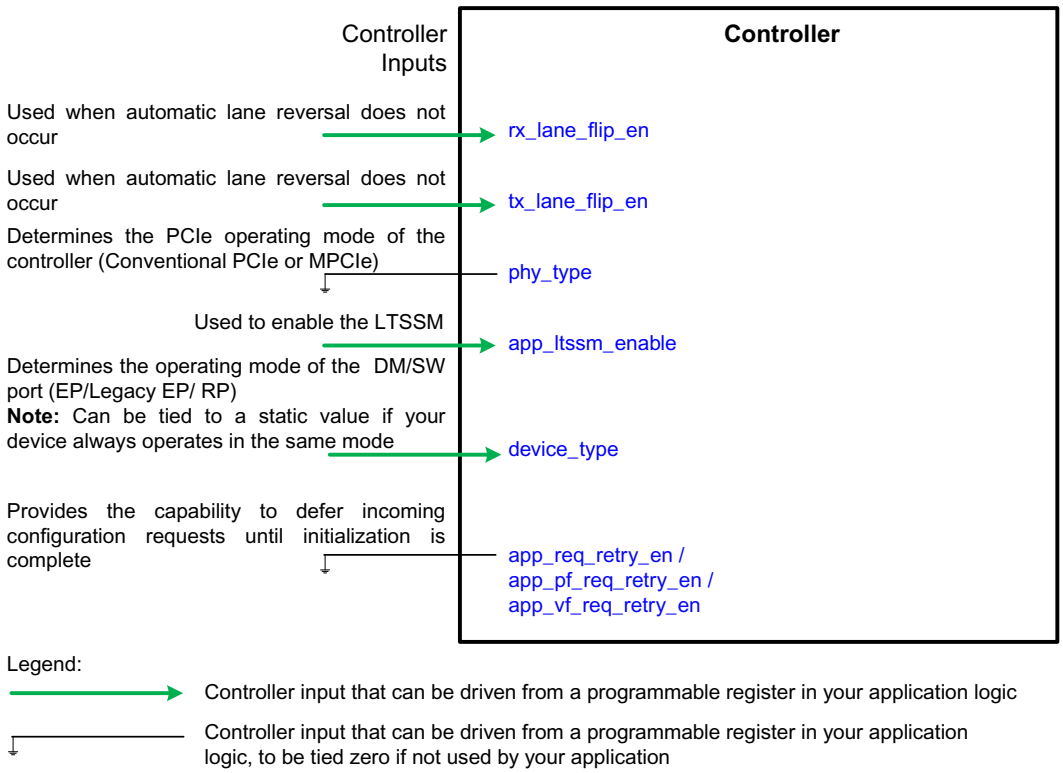
- The SII signal interface is a collection of various signals that carry system control, interrupt, and debug information. Many of these signals are provided for flexibility and can be used by your application. This section identifies the SII signals that must be connected to your application and provides usage examples for the ones that are optional.
- One of the most common implementations is to selectively mux out a set of SII signals into a register space which can be accessed by your application through a standard interface like APB.

H.9.1 SII: General Core Control Signals

The SII signals described in Figure H-7 are used for specific controller operations. Some of these are required in all configurations while others are optional and can be left unused

For detailed description of the signals mentioned in Figure H-7, see "SII: General Core Control Signals" section in the *Reference Manual*.

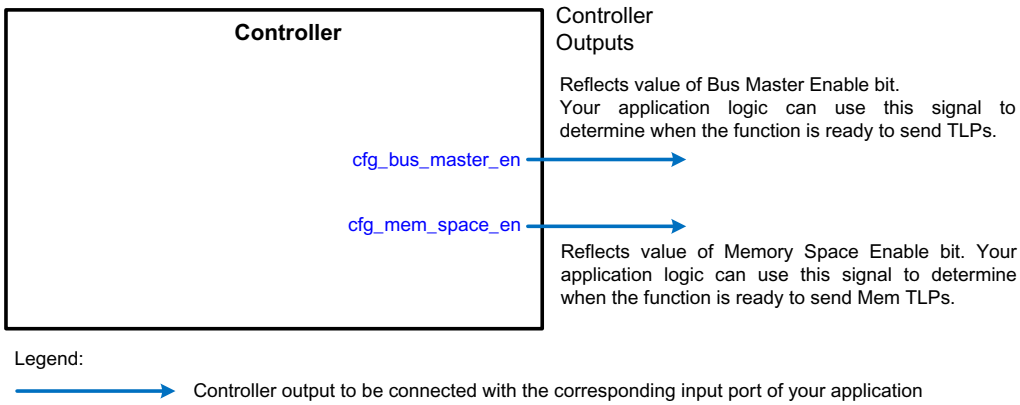
Figure H-7 Guidelines for Connecting General Core Control Signals



H.9.2 SII: Configuration Information Signals

The Configuration Information Signals mentioned in Figure H-8 reflect important controller functions and must be connected to your application logic and used as described. For more information, see "SII: Configuration Information Signals" in the "Signal Descriptions" chapter of the *Reference Manual*.

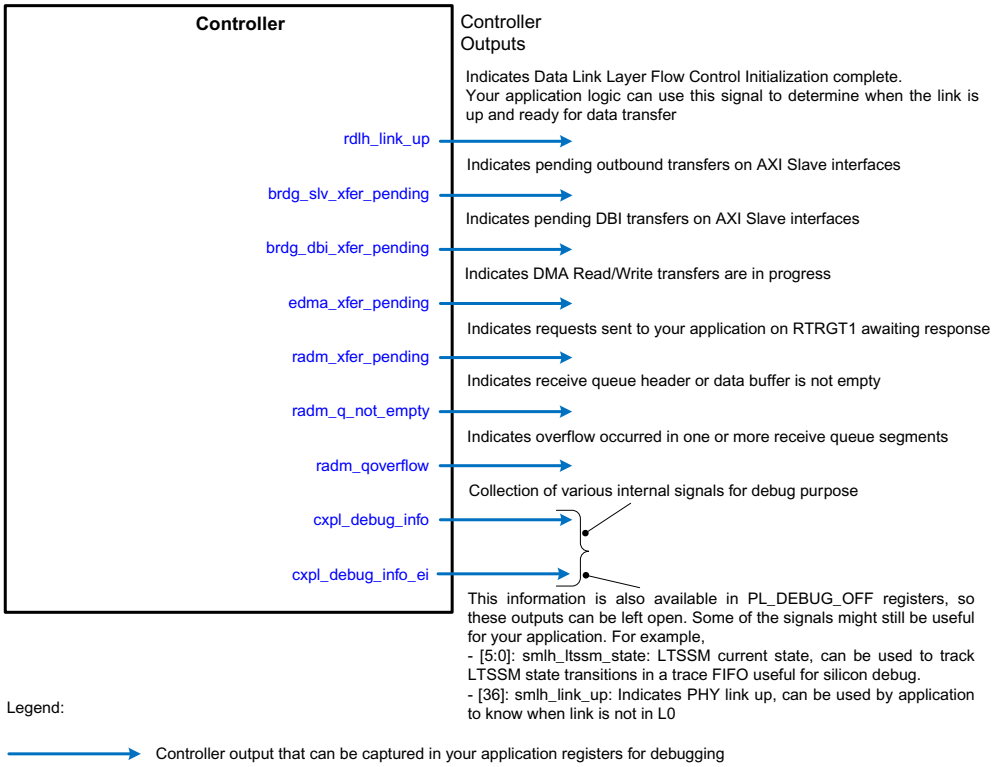
Figure H-8 Guidelines for Connecting Configuration Information Signals



H.9.3 SII: Debug Signals

The value of these debug signals can be captured in your application registers and used for debug purposes. If not used, they can be left open. Figure H-9 describes signals useful for debugging common issues.

Figure H-9 Guidelines for Connecting SII Debug Signals



H.9.4 SII: Diagnostic Control Signals [Optional]

`diag_ctrl_bus[2:0]` can be used to insert LCRC/ECRC error and to enable Fast Link Mode for simulation. If unused, tie off this input bus to 0. LCRC/ECRC error injection can also be performed using

the RAS DES feature of the controller. Fast Link Mode can also be enabled using `PORT_LINK_CTRL_OFF[7]`register.

H.9.5 SII: Diagnostic Status Signals [Optional]

- In addition to SII Debug Signals, there is an optional diagnostic status bus (`diag_status_bus`) that can be used for diagnostic purposes. Some of these signals can be treated as interrupts in your SoC while others can be captured in read-only status registers. A few important signals can be muxed out to the top level of your SoC for silicon debug.
- The width of this bus is configuration dependent.
- If your controller configuration has RAS DES enabled, most of the information provided on the `diag_status_bus` signals can also be obtained through the RAS DES registers and ports.

Table H-7 describes the suggested usage of the diag_status_bus signals.

Table H-7 Guidelines for Connecting Diagnostic Status Signals

Signal	Description	Suggested Usage with (CX_RAS_DES_ENABLE = 0)	Suggested Usage with (CX_RAS_DES_ENABLE = 1)
Data Link Layer Diagnostic Signals			
rtfcgen_incr_amt	Payload flow control credits consumed.	Capture in status register	Leave open
rtfcgen_incr_enable	Header flow control credit consumed.	Capture in status register	Leave open
rtfcgen_fctype	Flow control type consumed (P=0, NP=1, CPL=2).	Capture in status register	Leave open
rtcheck_rtfcgen_vc	Virtual channel of received TLP.	Capture in status register	Leave open
xdlh_xtlh_halt	Layer2 is not accepting data to transmit this cycle.	Multiplex to debug port at top level of your SoC.	Multiplex to debug port at top level of your SoC.
xtlh_xdlh_data	Transmit data at the interface between Layer3 and Layer2. If RASDP is enabled, includes the ECC or parity protection code bits.	Capture in status register	Capture in status register
xtlh_xdlh_badeot	Nullify this transmit TLP (invert CRC, append EDB).	Capture in status register	Capture in status register
xtlh_xdlh_eot	Transmit End of TLP this cycle.	Capture in status register	Capture in status register
xtlh_xdlh_sot	Transmit Start of TLP this cycle.	Capture in status register	Capture in status register
ecrc_err_asserted	End-to-end CRC corrupted for this packet.	Level interrupt	Level interrupt

Signal	Description	Suggested Usage with (CX_RAS_DES_ENABLE = 0)	Suggested Usage with (CX_RAS_DES_ENABLE = 1)
lcrc_err_asserted	Link CRC corrupted for this packet.	Level interrupt	Level interrupt
xmlh_xdlh_halt	PHY Layer not accepting data this cycle.	Multiplex to debug port at top level of your SoC. Useful for debugging performance issues.	Multiplex to debug port at top level of your SoC.Useful for debugging performance issues.
xdlh_xmlh_data	Transmit packet payload (completely framed).	Capture in status register	Capture in status register
xdlh_xmlh_sdp	Transmit Start of DLLP (per dword).	Capture in status register	Capture in status register
xdlh_xmlh_stp	Transmit Start of TLP (per dword).	Capture in status register	Capture in status register
xdlh_xmlh_eot	Transmit end of TLP/DLLP (per dword).	Capture in status register	Capture in status register
rdlh_xdlh_req_acknack_seqnum	Sequence Number for ACK/NAK DLLP.	Capture in status register	Leave open
rdlh_xdlh_req2send_nack	DataLink Layer request to send NAK.	Capture in status register	Leave open
rdlh_xdlh_req2send_ack_due2dup	Request to send ACK due to duplicate TLP.	Capture in status register	Capture in status register
rdlh_xdlh_req2send_ack	DataLink Layer request to send ACK.	Capture in status register	Leave open
rdlh_xdlh_rcvd_acknack_seqnum	Sequence number corresponding to NAK/ACK.	Capture in status register	Leave open
rdlh_xdlh_rcvd_ack	DataLink Layer received ACK DLLP.	Capture in status register	Leave open
rdlh_xdlh_rcvd_nack	DataLink Layer received NAK DLLP.	Capture in status register	Leave open
cfg_link_retrain	Software programmed link retrain request.	Capture in status register	Capture in status register
rtlh_req_link_retrain	Receive watchdog timer expired, retrain link.	Single cycle pulse Interrupt	Leave open
xdlh_smlh_start_link_retrain	Max retries attempted, request to retrain link.	Single cycle pulse Interrupt	Leave open
rdlh_rtlh_tlp_dv	Receive data interface from DLL to Transaction Layer valid this cycle (rdlh_rtlh_*).	Capture in status register, use as trigger for next two signals.	Capture in status register, use as trigger for next two signals.

Signal	Description	Suggested Usage with (CX_RAS_DES_ENABLE = 0)	Suggested Usage with (CX_RAS_DES_ENABLE = 1)
rdlh_rtlh_tlp_eot	End of TLP (per dword).	Capture in status register	Capture in status register
rdlh_rtlh_tlp_sot	Start of TLP (per dword).	Capture in status register	Capture in status register
MAC Layer Diagnostic Signals			
rmlh_rdlh_pkt_data	Received packet payload	Capture in status register	Capture in status register
rmlh_rdlh_pkt_err	Physical Error detected (per dword).	Capture in status register	Capture in status register
rmlh_rdlh_pkt_dv	Receive data interface from PHY to DLL valid this cycle (rmlh_rdlh_*).	Capture in status register, use as trigger for capturing other rmlh_rdlh_* signals.	Capture in status register, use as trigger for capturing other rmlh_rdlh_* signals.
rmlh_rdlh_pkt_edb[NW-1:0]	Packet terminated with EDB (per dword).	Capture in status register	Capture in status register
rmlh_rdlh_pkt_end[NW-1:0]	DLLP/TLP ending (per dword)	Capture in status register	Capture in status register
rmlh_rdlh_tlp_start[NW-1:0]	TLP start (per dword)	Capture in status register	Capture in status register
rmlh_rdlh_dllp_start[NW-1:0]	MAC layer detected DLLP start (per dword)	Capture in status register	Capture in status register
rmlh_rdlh_nak[NW-1:0]	MAC layer detected STP with receiver error (per dword).	Capture in status register	Capture in status register
smlh_lanes_rcving[NL-1:0]	lanes active in link training	Multiplex to debug port at top level of your SoC.	Leave open.
rmlh_rcvd_eidle_set	Received EIDLE ordered set, any active lane.	Capture in status register	Capture in status register
rmlh_rcvd_idle1	Logical Idle seen for 1+ symbols on all active Lanes.	Capture in status register	Capture in status register
rmlh_rcvd_idle0	Logical Idle seen for 8+ symbols on all active Lanes.	Capture in status register	Capture in status register
rmlh_rcvd_lane_rev	Receive logic detected logical lane reversal.	Capture in status register	Leave open
rmlh_ts_link_num_is_k237	Received Link number (lane 0) is PAD (k237).	Capture in status register	Capture in status register

Signal	Description	Suggested Usage with (CX_RAS_DES_ENABLE = 0)	Suggested Usage with (CX_RAS_DES_ENABLE = 1)
rmlh_deskew_alignment_err	Deskew logic overflow. Unable to align lanes [pulse].	Capture in status register	Leave open
rmlh_ts_lane_num_is_k237	Received lane number (lane 0) is PAD (k237).	Capture in status register	Capture in status register
rmlh_ts2_rcvd	At least one active lane received TS2 this cycle.	Capture in status register	Capture in status register
rmlh_ts1_rcvd	At least one active lane received TS1 this cycle.	Capture in status register	Capture in status register
rmlh_ts_rcv_err	During training, lane 0 TS sequence problem.	Capture in status register	Leave open
rmlh_inskip_rcv	Skip character is received this cycle	Capture in status register	Capture in status register
PM Diagnostic Bus			
xadm_no_fc_credit	No credits of any type are available (per VC).	Capture in status register	Leave open
xadm_notlp_pending	No TLP transmit requests currently pending (from internal or external clients).	Capture in status register	Capture in status register
xadm_had_enough_credit	The controller has enough transmit credits of each type (per VC) to meet the PM entry criteria.	Capture in status register	Leave open
xdlh_not_expecting_ack	All transmitted TLPs have been acknowledged by the link partner.	Capture in status register	Capture in status register
xdlh_xmt_pme_ack	DataLink layer just transmitted PME_TO_ACK msg.	Capture in status register	Capture in status register
xdlh_nodllp_pending	There are no pending or in-progress packets going to the PHY.	Capture in status register	Capture in status register
l1sub_state	L1 sub state (Tied 0 if L1 sub state is not enabled).	Multiplex to debug port at top level of your SoC	Leave open
RADM Diagnostic Bus			
unexpected_cpl_err	Received an unexpected completion.	Local Interrupt (One cycle pulse)	Interrupt

Signal	Description	Suggested Usage with (CX_RAS_DES_ENABLE = 0)	Suggested Usage with (CX_RAS_DES_ENABLE = 1)
cpl_ca_err	Received completion with CA status.	Local Interrupt (One cycle pulse)	Interrupt
cpl_ur_err	Received completion with UR status.	Local Interrupt (One cycle pulse)	Interrupt
flt_q_cpl_last	Final completion for the transaction.	Capture in status register	Capture in status register
flt_q_cpl_abort	Current completion is being aborted (trashed).	Local Interrupt (One cycle pulse)	Interrupt
cpl_mlf_err	Malformed completion	Local Interrupt (One cycle pulse)	Interrupt
flt_q_header_cpl_status	Indicates the filtering result for the current RX request or the completion status for the current RX received completion.	Capture in status register	Capture in status register
flt_q_header_destination	Destination interface.	Capture in status register	Capture in status register
form_filt_ecrc_err	This TLP had an ECRC error	Local Interrupt (One cycle pulse)	Interrupt
form_filt_malform_tlp_err	This TLP is malformed	Local Interrupt (One cycle pulse)	Interrupt
form_filt_dllp_err	This TLP has a DataLink Layer Error (for example, LCRC)	Local Interrupt (One cycle pulse)	Interrupt
form_filt_eot	End of TLP received this cycle	Capture in status register	Capture in status register
form_filt_dwen	Dword enables	Capture in status register	Capture in status register
form_filt_data	Packet data from receive Transaction Layer.	Capture in status register	Capture in status register
form_filt_dv	Packet is in payload stage	Capture in status register	Capture in status register
form_filt_hdr	Header data (size of stored header is configurable).	Capture in status register	Capture in status register
form_filt_hv	Information to the receive packet filter block is valid	Capture in status register, and use as trigger for capturing other form_filt_* signals	Capture in status register
XADM Diagnostic Bus			

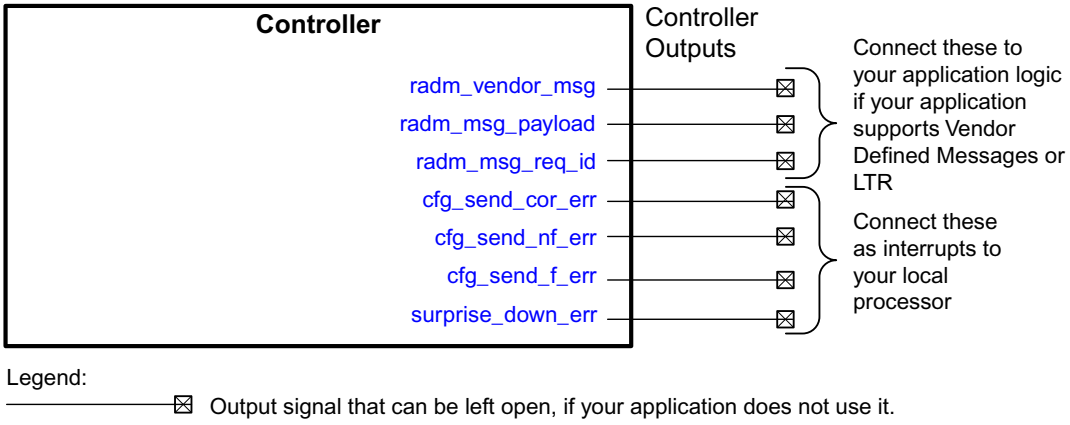
Signal	Description	Suggested Usage with (CX_RAS_DES_ENABLE = 0)	Suggested Usage with (CX_RAS_DES_ENABLE = 1)
active_grant	Currently granted client (1-hot)	Capture in status register	Capture in status register
grant_ack	This client has completed his request (1-hot)	Capture in status register	Capture in status register
fc_cds_pass	Credit check passed for each Request	Capture in status register	Capture in status register
arb_reqs	Transmit requests from each client	Capture in status register	Capture in status register
CDM Diagnostic Bus			
cfg_pbus_num	Bus number; 8 bits per function when MULTI_DEVICE_AND_BUS_PER_FUNC_EN =1, else 8 bits in total.	Capture in status register	Capture in status register
cfg_pbus_dev_num	Device number; 5 bits per function when MULTI_DEVICE_AND_BUS_PER_FUNC_EN=1, else 5 bits in total.	Capture in status register	Capture in status register
xdlh_replay_timeout_err	Retry timer expired [pulse].	Local Interrupt (One cycle pulse)	Leave open
xdlh_replay_num_rlover_err	Max retries exceeded [pulse].	Local Interrupt (One cycle pulse)	Leave open
rdlh_bad_dllp_err	Received DLLP with DataLink Layer error [pulse].	Local Interrupt (One cycle pulse)	Leave open
rdlh_bad_tlp_err	Received TLP with DataLink Layer error [pulse].	Local Interrupt (One cycle pulse)	Leave open
rdlh_prot_err	DLLP protocol error (out of sequence DLLP) [pulse].	Local Interrupt (One cycle pulse)	Leave open
rtlh_fc_prot_err	Flow control protocol violation (watchdog timer) [pulse].	Multiplex to debug port at top level of your SoC	Leave open
rmlh_rcvd_err	Received PHY error this cycle [pulse].	Local Interrupt (One cycle pulse)	Leave open
int_xadm_fc_prot_err	Flow control update protocol violation (opt. checks).	Multiplex to debug port at top level of your SoC	Leave open
radm_cpl_timeout	A request failed to complete in the allotted time.	Local Interrupt (One cycle pulse)	Leave open

Signal	Description	Suggested Usage with (CX_RAS_DES_ENABLE = 0)	Suggested Usage with (CX_RAS_DES_ENABLE = 1)
radm_qoverflow	Receive queue overflow. Normally happens only when flow control advertisements are ignored.	Local Interrupt (One cycle pulse)	Interrupt
radm_unexp_cpl_err	Received an unexpected completion [pulse].	Local Interrupt (One cycle pulse)	Leave open
radm_rcvd_cpl_ur	Received a completion with UR status.	Local Interrupt (One cycle pulse)	Interrupt
radm_rcvd_cpl_ca	Received a completion with CA status.	Local Interrupt (One cycle pulse)	Interrupt
radm_rcvd_req_ca	Completer aborted request.	Local Interrupt (One cycle pulse)	Interrupt
radm_rcvd_req_ur	Received a request which device does not support.	Local Interrupt (One cycle pulse)	Interrupt
radm_ecrc_err	Received a TLP with ECRC error.	Local Interrupt (One cycle pulse)	Interrupt
radm_mlf_tlp_err	Received a malformed TLP [pulse].	Local Interrupt (One cycle pulse)	Interrupt
radm_rcvd_cpl_poisoned	Received a completion with poisoned payload.	Local Interrupt (One cycle pulse)	Leave open
radm_rcvd_wreq_poisoned	Received a write with poisoned payload.	Local Interrupt (One cycle pulse)	Leave open
cdm_lbc_ack	Local bus Acknowledge.	Leave open	Leave open
lbc_cdm_wr	Local bus Write enable (per byte).	Leave open	Leave open
lbc_cdm_cs	Local bus chip select.	Leave open	Leave open
lbc_cdm_data	Local bus Data.	Leave open	Leave open
lbc_cdm_addr	Local bus Address.	Leave open	Leave open
cfg_sys_err_rc_cor	System Error caused by Correctable Error. (One-clock-cycle pulse) (Tied 0 if not Root Complex).	Local Interrupt (One cycle pulse)	Interrupt
cfg_sys_err_rc_nf	System Error caused by Non-Fatal Error (One-clock-cycle pulse) (Tied 0 if not Root Complex).	Local Interrupt (One cycle pulse)	Interrupt

Signal	Description	Suggested Usage with (CX_RAS_DES_ENABLE = 0)	Suggested Usage with (CX_RAS_DES_ENABLE = 1)
cfg_sys_err_rc_f	System Error caused by Fatal Error (One-clock-cycle pulse) (Tied 0 if not Root Complex).	Local Interrupt (One cycle pulse)	Interrupt

H.9.6 SII: General Message Reception Signals

Figure H-10 Guidelines for Connecting SII: General Message Reception Signals



H.9.7 SII: Interrupt Signals

The outputs of the controller described in Table H-8, if present in your configuration, can be treated as interrupts to your local processor.

Table H-8 Local Interrupts

Signal	Interrupt Type	Upstream Port (USP)	Downstream Port (DSP)
cfg_vpd_int	One cycle pulse	Yes	No
cfg_aer_rc_err_int	One cycle pulse	Yes	No
cfg_aer_rc_err_msi	Level	Yes	No
cfg_pme_int	Level	Yes	No
cfg_pme_msi	One cycle pulse	Yes	No
radm_inta_asserted	One cycle pulse	Yes	No
radm_intb_asserted	One cycle pulse	Yes	No
radm_intc_asserted	One cycle pulse	Yes	No
radm_intd_asserted	One cycle pulse	Yes	No
radm_inta_deasserted	One cycle pulse	Yes	No

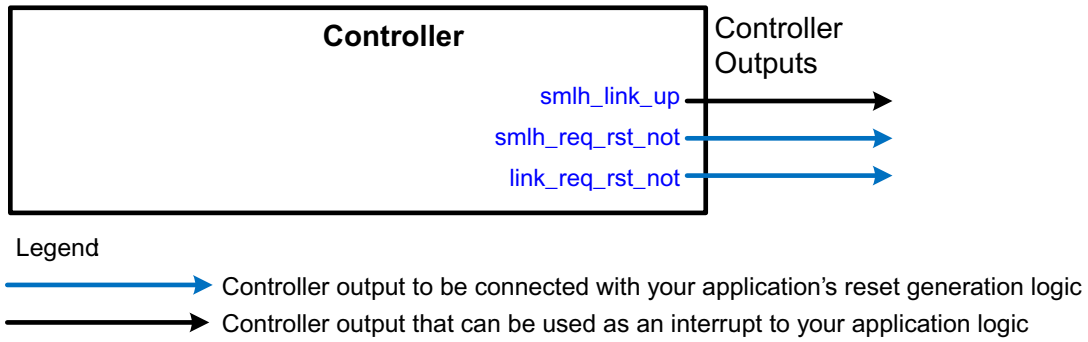
Signal	Interrupt Type	Upstream Port (USP)	Downstream Port (DSP)
radm_intb_deasserted	One cycle pulse	Yes	No
radm_intc_deasserted	One cycle pulse	Yes	No
radm_intd_deasserted	One cycle pulse	Yes	No
hp_pme	One cycle pulse	Yes	No
hp_int	Level	Yes	No
hp_msi	One cycle pulse	Yes	No
cfg_link_auto_bw_int	Level	Yes	No
cfg_bw_mgt_int	Level	Yes	No
cfg_link_eq_req_int	Level	No	No
edma_int	Level	No	No
smlh_link_up	Level	No	No
radm_correctable_err	One cycle pulse	Yes	No
radm_nonfatal_err	One cycle pulse	Yes	No
radm_fatal_err	One cycle pulse	Yes	No
mstr_rasdp_err_mode	Level	No	No
slv_rasdp_err_mode	Level	No	No
cfg_rasdp_error_mode	Level	No	No

In addition to these, some of the diagnostic bus signals can be treated as interrupts. For more information, see Table H-7.

H.9.8 SII: Link Reset/Status Signals

Figure H-11 lists the output signals used in reset generation of the controller. For more information, see "Clocks and Resets" section in the "Architecture" chapter of the Databook.

Figure H-11 Guidelines for Connecting SII: Link Reset/Status Signals



H.9.9 SII: LTR Message Generation/Reception Signals

If your configuration supports LTR, use this interface as described in "SII: LTR Message Generation Signals"/ "SII: LTR Message Reception Signals" in "Signals Descriptions" chapter of the *Reference Manual*.

H.9.10 SII: OBFF Message Generation/Reception Signals

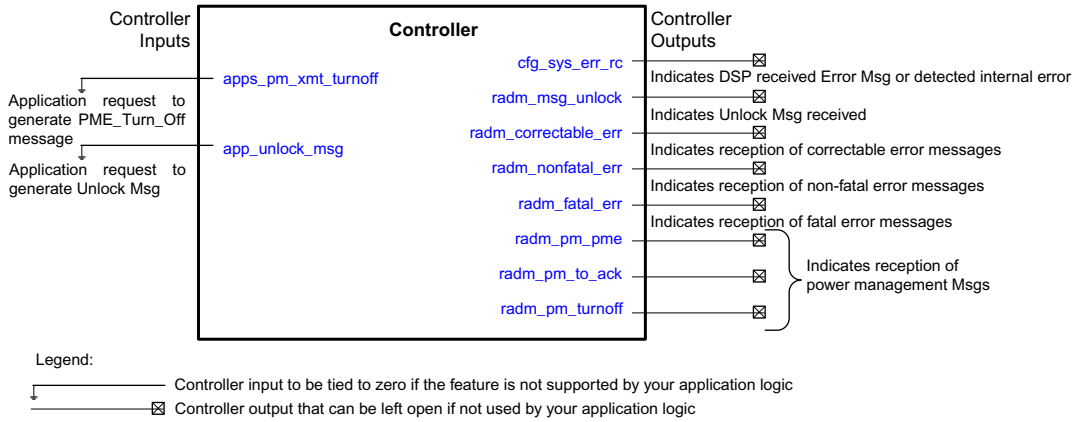
If your configuration supports OBFF, use this interface as described in "SII: OBFF Message Generation Signals" / "SII: OBFF Message Reception Signals" in "Signals Descriptions" chapter of the *Reference Manual*.

H.9.11 SII: PM Unlock and Error Message Signals

The signals listed in Figure H-12 can be used by your application to generate outbound PME_Turn_Off Message, Unlock Message. These signals can also be used to know when the controller receives PM Messages (PM_PME, PME_TO_Ack, PME_Turn_off), Unlock, and Error Messages.

Connect the inputs to your application logic or tie to 0 if not used. Leave outputs open if unused.

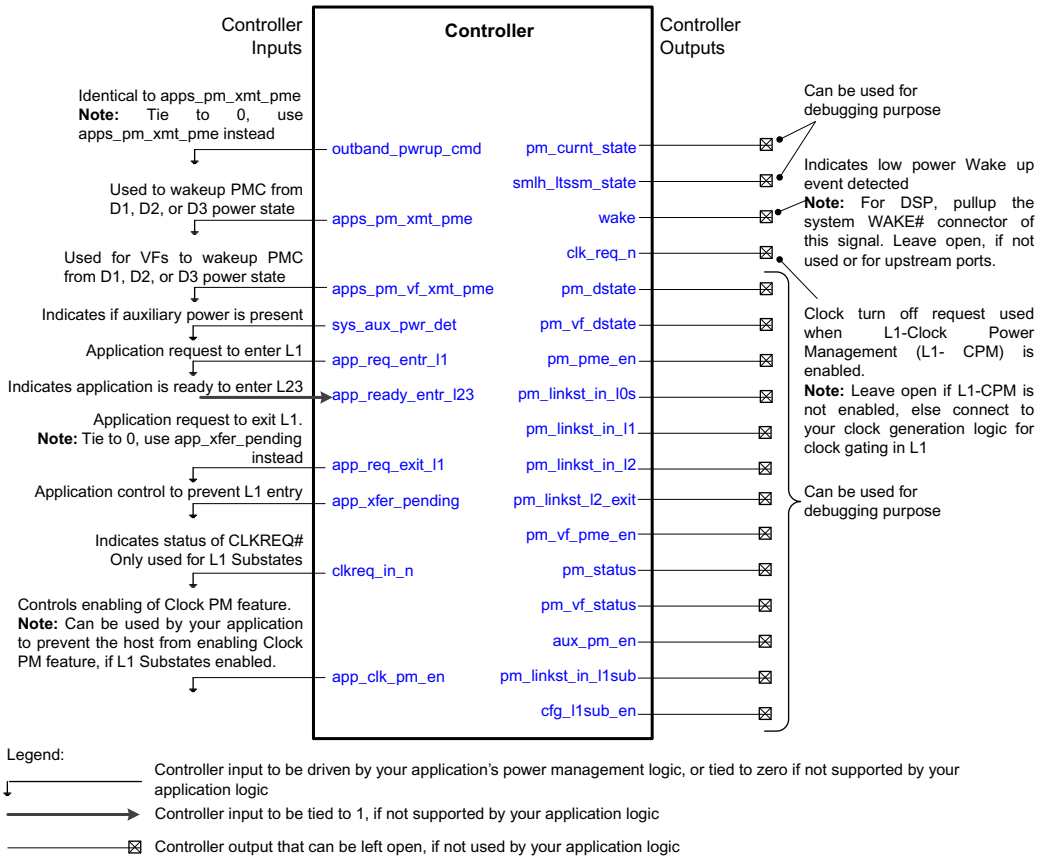
Figure H-12 Guidelines for Connecting SII: PM Unlock and Error Message Signals



H.9.12 SII: Power Management Signals

Figure H-13 provides integration guidance for power management related IOs present in all configurations.

Figure H-13 Guidelines for Connecting SII: Power Management Signals

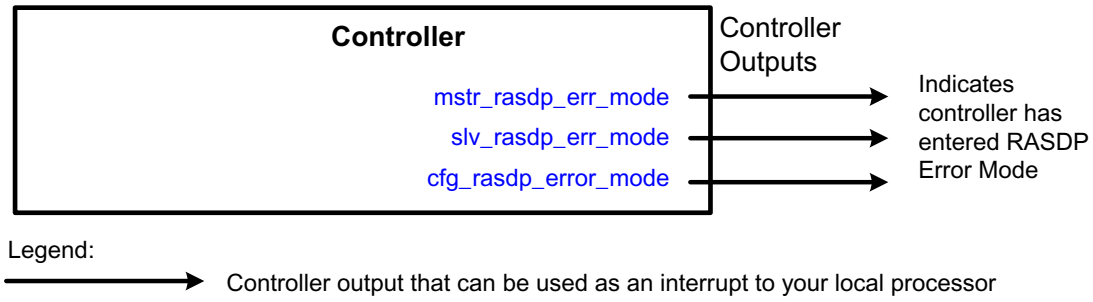


H.9.13 SII: Precision Time Management Signals

If your configuration supports PTM, connect this interface as described in "SII: Precision Time Management Signals" in "Signal Descriptions" chapter of the *Reference Manual*.

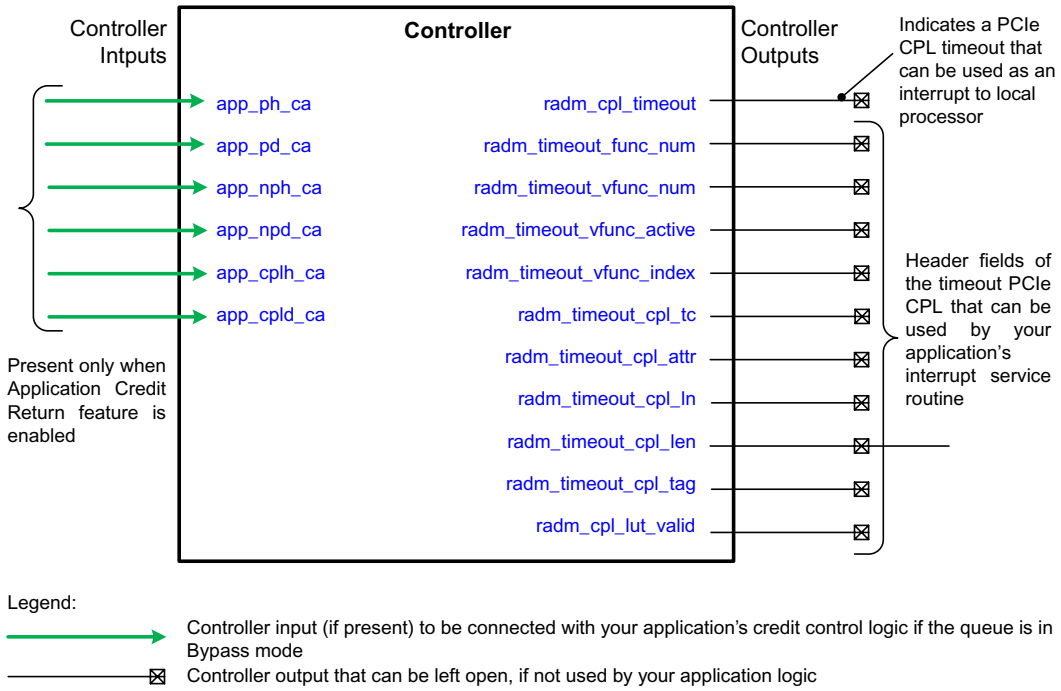
H.9.14 SII: RAS Data Protection Signals

Figure H-14 Guidelines for Connecting SII: RAS Data Protection Signals



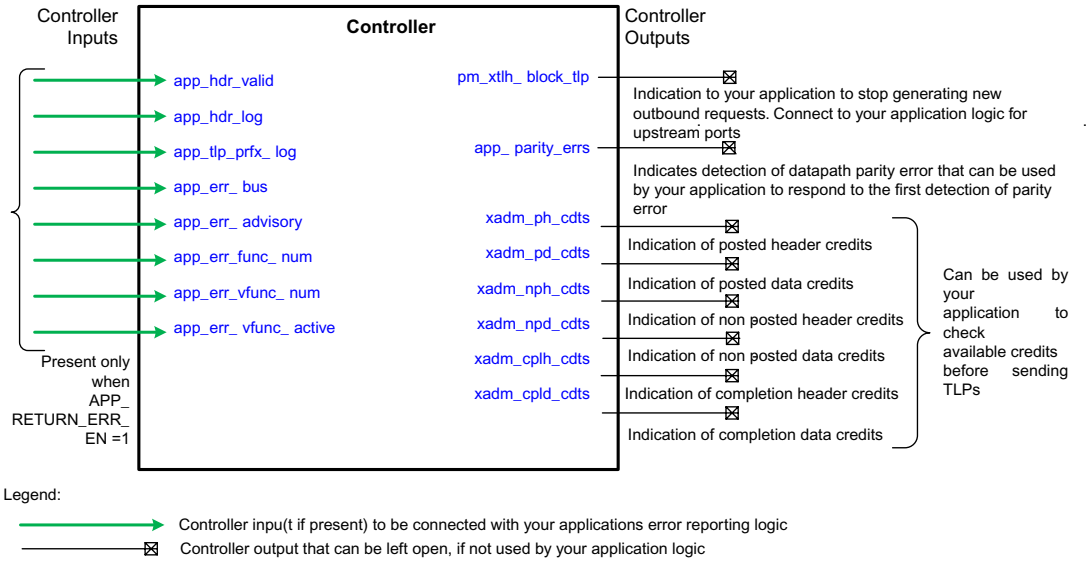
H.9.15 SII: Receive Control/CPL Timeout Signals

Figure H-15 Guidelines for Connecting SII: Receive Control/CPL Timeout Signals



H.9.16 SII: Transmit Control Signals

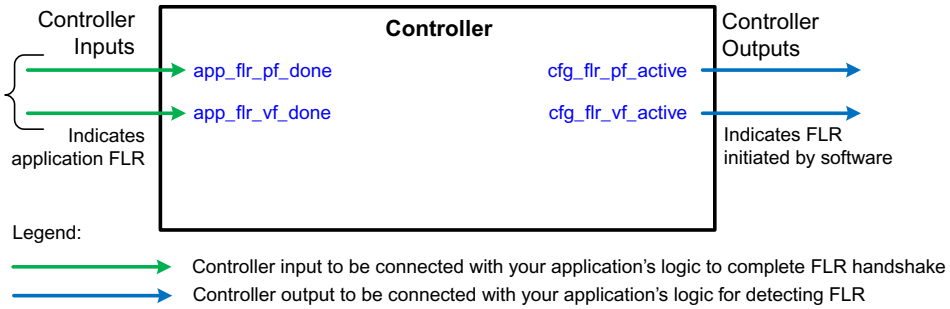
Figure H-16 Guidelines for Connecting SII: Transmit Control Signals



H.10 FLR

If your configuration supports FLR, connect the signals mentioned in Figure H-17 to your application logic and use them as described in the "Function Level Reset (FLR)" section in the "Controller Operations" chapter of the *Databook*.

Figure H-17 Guidelines for Connecting FLR Signals

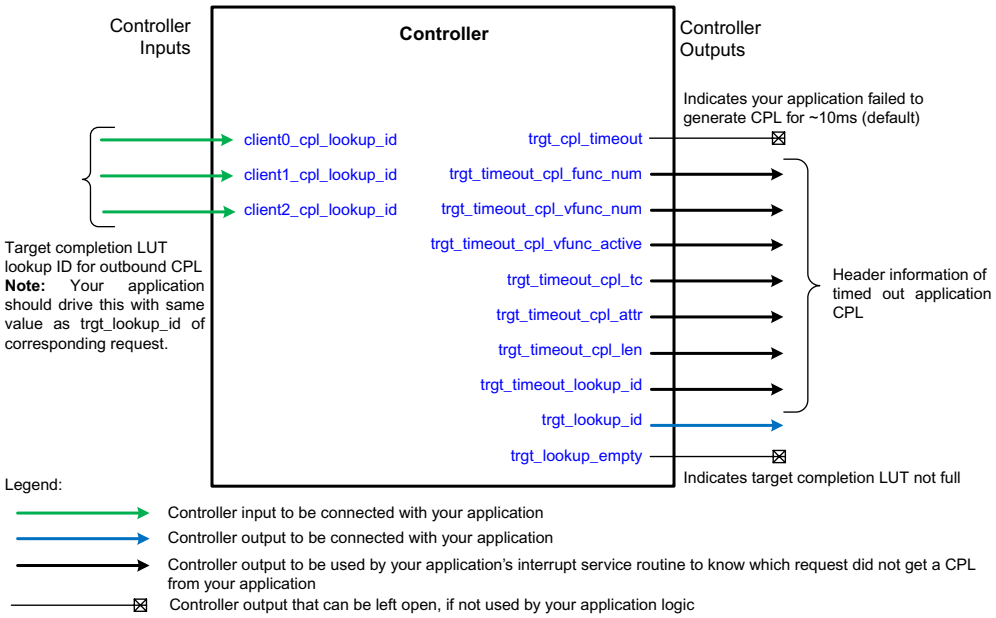


H.11 Target Completion LUT

Configurations that do not include the AXI bridge can enable the Target Completion LUT interface by selecting parameter `TRGT_CPL_LUT_EN`. This feature saves some application logic as application does not have to provide the Byte Count, Tag, Attributes, requester ID, and Address values for outbound CPLs, these are instead taken from the Target Completion LUT.

If this feature is enabled, use application logic to connect to the signals mentioned in Figure H-18.

Figure H-18 Guidelines for connecting Target Completion LUT Signals



H.12 RAS D.E.S Signals

If your configuration has RAS D.E.S Event Counters, Silicon Debug, or Time Based Analysis features enabled (`CX_RAS_DES_EC_ENABLE = 1`, `CX_RAS_DES_SD_ENABLE = 1`, or `CX_RAS_DES_TBA_ENABLE =1`), then you can use the top-level signals mentioned in Figure H-19 to obtain RAS DES related information.

Figure H-19 Guidelines for Connecting RAS D.E.S. Event Counters Debug Signals

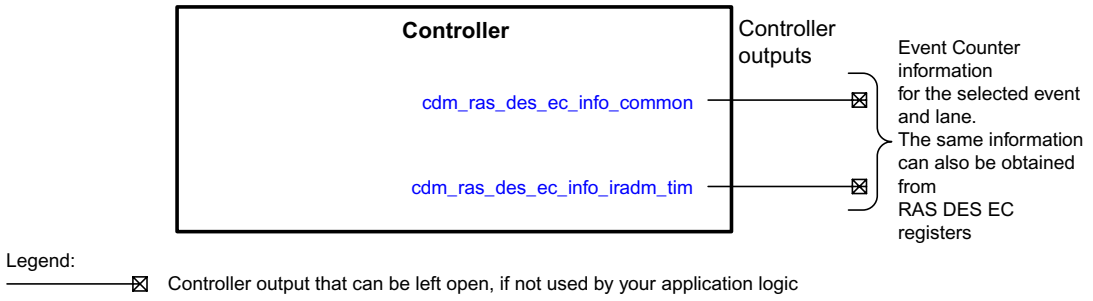


Figure H-20 Guidelines for Connecting RAS D.E.S. Silicon Debug Internal Signals

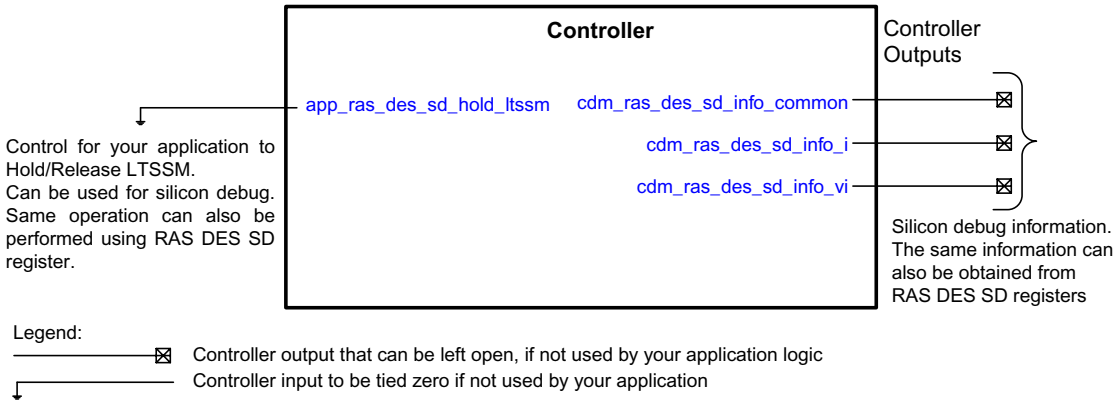
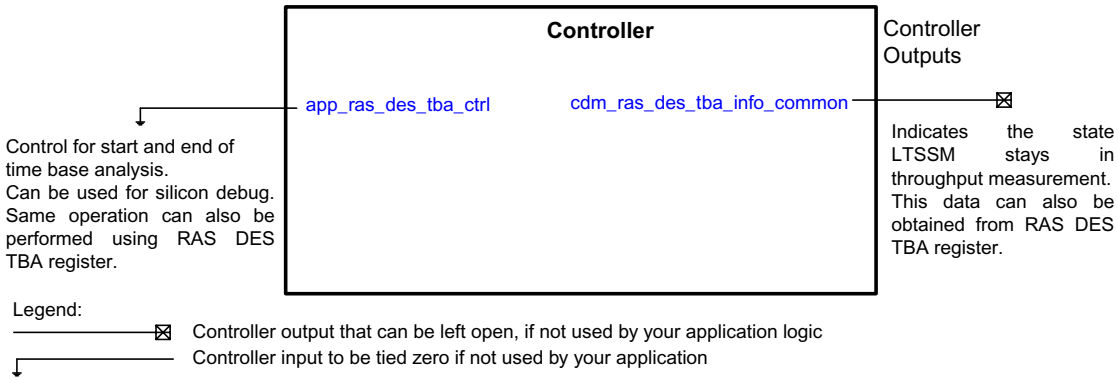


Figure H-21 Guidelines for Connecting RAS D.E.S. Time-Based Analysis Debug Signals



H.13 Configuration Intercept Interface (CII) Signals

If your upstream port configuration has CII enabled (`CX_CONFIG_INTERCEPT_ENABLE =1`), then you must implement logic that takes the CFG TLP information provided on this interface, perform the required override and return the data as described in "Configuration Intercept Controller (CIC) for USP" appendix of the *Databook*. "Configuration Intercept Interface (CII) Signals" section in the "Signal Descriptions" chapter of the *Reference Manual* describes all the signals of this interface.

H.14 PHY Register Bus Interface

If you are using a Synopsys PHY that has the Control Register Parallel Interface (CRPI) and you have enabled the PHY register viewport in your controller configuration (`CX_PHY_VIEWPORT_ENABLE =1`), then this interface should be connected between the controller and the PHY to allow the host/local application to access control registers in the PHY as described in "PHY Port Logic Registers" in "Register Module, LBC, and DBI" section of the "Controller Operations" chapter of the *Databook*.

For guidance on connecting the signals of this interface, see "PHY Register Bus Interface Signals" section in the "Signal Descriptions" chapter of the *Reference Manual*.