DesignWare Cores SuperSpeed USB 3.0 Controller

User Guide

DWC_usb3 – Product Codes

Version 3.30b
May 2018

Synopsys, Inc.
690 E. Middlefield Road
Mountain View, CA 94043

www.synopsys.com

Contents

Revision History

Date	Version	Description
May 2018	3.30b	No updates to this document
February 2018	3.30a	Moved sections from Chapter 2, “Configuring the Controller” into respective chapters relating to coreConsultant activities - Chapter 3, “Simulating the Controller” - Chapter 4, “Performing Synthesis and Post-Synthesis Activities” - Chapter 5, “Integrating the Controller” Replaced: ■ “Product Overview” chapter with Chapter 1, “Design Flow” ■ Term “core” with “controller” ■ Section heading “GUSB3PIPECTL*n*” with “GUSB3PIPECTL*n* Register” Removed support for the following features: ■ OTG ■ ADP ■ SSIC Chapter 3, “Simulating the Controller”: Replaced VTB with PVE Chapter 4, “Performing Synthesis and Post-Synthesis Activities”: Added “Performing Power Analysis” on page 129

Date	Version	Description
February 2017	3.20a	Chapter 2, “Configuring the Controller” ■ Added “Running VCS Xprop Analyzer” on page 74 ■ Updated “Design Flow” on page 24 Chapter 5, “Integrating the Controller” ■ Removed reference to rm_non_DFF_timing.pl file in “Running the Simulation” on page 70 ■ Updated: - Information on Synopsys PHY in “GUSB3PIPECTL[18]” on page 150 - Notes on debug_u2pmu and debug_u3pmu in Table 5-5 on page 181 Chapter 3, “Simulating the Controller” ■ Fixed typos in Table 4-2 on page 150 Chapter 4, “Performing Synthesis and Post-Synthesis Activities” ■ Updated: - Direction of timing paths in “Timing Checks for Reset Signals” on page 106 - “Synthesizing Using UPF Power Intent” on page 118
March 2016	3.10a	Chapter 2, “Configuring the Controller” ■ Added 3 new analysis modes in Table 2-3 on page 37 ■ Removed section “Running VC CDC using cC” because VC CDC is not supported Chapter 5, “Integrating the Controller” ■ Renamed the sleep signal as power_switch_control signal ■ Replaced debug_u2pmu[0] with debug_u2pmu[DWC_USB3_NUM_U2_ROOT_PORTS*DWC_USB3_DEBUG_U2WAKEUP_W] in Table 5-5 on page 181 and “Code Example” on page 183 Chapter 4, “Performing Synthesis and Post-Synthesis Activities” ■ Added new section “Timing Checks for Reset Signals” on page 106 Appendix B, “FPGA Implementation of DWC_usb3 Controller” ■ Updated to include information about the USB 3.0 IP Prototyping kits

Date	Version	Description
May 2015	3.00a	Chapter 2, "Configuring the Controller" ■ Added a note on adding prefix for the VTB testbench in "Running the Simulation" on page 70 ■ Added new analysis modes in Table 2-3 on page 37 ■ Added a new section "Running VC CDC using cC" on page 65 Chapter 5, "Integrating the Controller" ■ Updated "Clocks" on page 140 ■ Updated sleep signal in Table 5-5 on page 181 Chapter 3, "Simulating the Controller" ■ Added the following figures: - Figure 4-4 on page 147 - Figure 4-17 on page 162 - Figure 4-18 on page 162 - Figure 4-19 on page 163 - Figure 4-20 on page 163 ■ Updated Table 4-2 on page 150 ■ Updated the following sections: - "Device Testbench and Test Files" on page 164 - "Host Testbench and Test Files" on page 178
May 2015 *(Contd)*	3.00a *(Contd)*	*(Continued)* ■ Added the following sections: - "Toggle Coverage Report for RMMI Interface" on page 226 - "SVA Checks for Specific SSIC Controller Interface Signals" on page 227 - "Toggle Coverage" on page 259 Chapter 4, "Performing Synthesis and Post-Synthesis Activities" ■ Updated "Clock Gating Cell" on page 105 ■ Updated "Synthesizing Using UPF Power Intent" on page 118

Date	Version	Description
September 2014	2.90a	Chapter 2, “Configuring the Controller” ■ Added new section, “Running Spyglass Lint and CDC” on page 37 Chapter 5, “Integrating the Controller” ■ Updated: - “Pipelining on the PIPE Interface” on page 141 - “Integrating the Dual Power Rail (Hibernation) Feature” on page 176 - Table 5-5 on page 181 Chapter 3, “Simulating the Controller” ■ Updated: - Table 4-1 on page 148 - “Device Testbench and Test Files” on page 164 - Table 4-3 on page 166 - Table 4-4 on page 181 - “Debugging VTB Tests” on page 234 ■ Added “Integrating with M-PHY in VTB” on page 209 Chapter 4, “Performing Synthesis and Post-Synthesis Activities” ■ Added - Figure 5-2 on page 181 - Example 5-2 on page 185 ■ Updated - Figure 4-14 on page 113 - Example 4-2 on page 115 - “Synthesizing Using UPF Power Intent” on page 118
February 2014	2.80a	Chapter 5, “Integrating the Controller” ■ Updated section “Little-Endian and Big-Endian” on page 188 to reflect the AXI specification that states that most little-endian components can connect directly to a byte-invariant interface. Components that support only big-endian transfers require a conversion function for byte-invariant operation. Chapter 3, “Simulating the Controller” ■ Added new VTB source file to Table 4-1 on page 148. ■ Added more SSIC VTB tests to “Device Testbench and Test Files” on page 164 ■ Added more SSIC VTB tests to “Host Testbench and Test Files” on page 178.

Date	Version	Description
December 2013	2.70a	■ Added details on SuperSpeed InterChip (SSIC) feature ■ Direct connect (Connecting DWC_usb3 controller to USB 3.0 VIP directly through PIPE3 interface without USB 3.0 PHY) option is no longer supported. Removed the documentation on the same. Chapter 2, "Configuring the Controller" ■ Added information on the USB 3.0 VIP (usb_svt) and AMBA VIP – These VIPs are no longer included in the DWC_usb3 image. To run simulations, these VIPs must be downloaded and installed separately (see "Setting Up Your Environment" on page 28 and "Running the Simulation" on page 70) ■ Updated "Simulating the Controller" on page 49 ■ Added "Running Leda RTL Checker" Chapter 3, "Simulating the Controller" ■ Changes and additions made to following tables: "List of Device Tests Provided in the SoC Integration Example", "List of Host Tests Provided in the SoC Integration Example", and "Hub VTB Test Details". ■ New figures added—Figure 4-15 on page 160 and Figure 4-16 on page 161. ■ Added DWC_USB3_HSIC_H_UTMI_DEVICE, DWC_USB3_HSIC_H_ULPI_DEVICE, DWC_USB3_HSIC_H_UTMI_HOST, and DWC_USB3_HSIC_H_ULPI_HOST parameters to Table 4-2 on page 150. ■ Added "DWC_usb3_fifogasket Test Module" on page 214.
July 2013	2.60a	Preface ■ Added a new section on list of product codes for DWC_usb3 controller ("Product Codes" on page 16) Chapter 2, "Configuring the Controller" ■ Added a note on rm_non_DFF_timing.pl example script in section "Running the Simulation" on page 70 ■ Added information on posedge_ffs.rpt, negedge_ffs.rpt and latches.rpt files in section "Checking ASIC Synthesis Results" on page 98 Chapter 5, "Integrating the Controller" ■ Updated "GUSB3PIPECTL[21:19]" on page 149 Chapter 3, "Simulating the Controller" ■ Added two device tests (Test507_LPM_Errata.v and Test508_LPM_Errata.v) in Table 4-3 on page 166 Chapter 4, "Performing Synthesis and Post-Synthesis Activities" ■ Updated "Synthesis Constraints for DWC_usb3 Controller" on page 102 (added Figure 4-9 and updated Example 4-1) ■ Updated "Clock Frequency Requirements for DFT" on page 110 (added Figure 4-11 and Figure 4-12, and updated wpc_clk_in generation (Figure 4-13 on page 112)

Date	Version	Description
November 2012	2.50a	Chapter 5, “Integrating the Controller” ■ Fixed a databook error: Updated utmiotg_bvalid as utmisrp_bvalid ■ Updated “Integrating with USB 2.0 PHY” on page 154 ■ Added the following new sections: - “Global User Control Register (GUCTL)” on page 153 - “TX/RX Data FIFO Sizes and TX/RX Threshold Control Register Settings” on page 194 Chapter 3, “Simulating the Controller” ■ Updated the following device tests in Table 4-3 on page 166 - Test90, Test91, Test158, and Test911 ■ Updated description of Test17 in Table 4-4 on page 134 ■ Updated the following host tests in Table 4-4 on page 181 - Test490, Test495, Test496, Test520, Test590, Test595, and Test596 ■ Added the following in “Timescale Related Precautions During SoC Integration” on page 204 - A bullet about the new *-diag timescale* switch in VCS - Information on the timescale precision for the VIP Chapter 4, “Performing Synthesis and Post-Synthesis Activities” ■ Updated synthesis reference script (changed pipe3_rx_pclk[0] to suspend_clk in two lines) in “Reference Script – DRD Configuration With ULPI and UTMI” on page 103
August 2012	2.40a	Chapter 5, “Integrating the Controller” Section “Integrating the Dual Power Rail (Hibernation) Feature”: Added a note on multi-port OTG configuration Chapter 3, “Simulating the Controller” Added Debug Capability tests to Host tests Appendix B, “FPGA Implementation of DWC_usb3 Controller” Updated the list of Reference Documents

Date	Version	Description
July 2012	2.30a	Removed the vaux_reset_n input to the DWC_usb3 core, and moved it's behavior to the vcc_reset_n signal. The vcc_reset_n signal now has two behaviors depending on whether the hibernation feature is enabled or not Preface: Updated xHCI errata from January 17 2011 to June 13 2011 Chapter 5, "Integrating the Controller" Section "PHY and Link-Related Controls and Synthesis Options": Added information on the new bit U2EXIT_LFPS Added a new section "Transfer TRB Error" Added a new section "Save and Restore" Chapter 3, "Simulating the Controller" Added 3 host tests (Test100_cache_exceptions.v, Test101_cache_exceptions.v, and Test220_dbc_set_config.v) Chapter 4, "Performing Synthesis and Post-Synthesis Activities" ■ Updated - Figure: "DFT Bypass/Control Details when "Enable Additional DFT Control Ports" Set to No - Figure: "DFT Bypass/Control Details when "Enable Additional DFT Control Ports" Set to Yes" ■ Added Figure "Hibernation Mode MUX when DWC_USB3_EN_PWROPT Set to 2" ■ Updated "Reference Script – DRD Configuration with ULPI and UTMI"
April 2012	2.20a	Chapter 5, "Integrating the Controller" ■ Added: - Information on Multiport connection (see Section 'Host/DRD Mode – Multiport Connection') - Updated note after Figure 5-20 on page 157 - New diagram Figure 5-17 on page 155 ■ Updated section "PHY Loopback" ■ Moved section "Host Mode – Miscellaneous Topics" one level up Chapter 3, "Simulating the Controller" ■ Added "Integrating with Synopsys PHY" Added Chapter 4, "Performing Synthesis and Post-Synthesis Activities"
January 2012	2.10a	■ Moved Hardware Integration and Verification Chapter from the *DWC SuperSpeed USB 3.0 Controller Databook* to this User Guide. ■ Updated Table 4-3 on page 166 and Table 4-4 on page 181

Preface

This document describes the Synopsys DesignWare® Cores SuperSpeed USB 3.0 Controller. The controller implements the USB (Universal Serial Bus) 3.0 functionality, and can be configured as a Device, Host, Dual-Role Device (DRD), or Hub based on licenses purchased. The DWC USB 3.0 Controller corresponds to DWC_usb3 in the SolvNet database. The terms "DWC_usb3 controller", "DWC_usb3", and "hardware" all refer to the USB 3.0 Controller product. The terms "system bus" and "SoC bus" are used interchangeably.

This section includes the following topics:

- "Product Codes" on page 16
- "User Guide Organization" on page 17
- "Related Product Documentation" on page 18
- "Reference Documentation" on page 19
- "Web Resources" on page 20
- "Customer Support" on page 21

Product Codes

Table 4-1 lists the products and product codes for the DWC_usb3 controller.

Table 4-1 Product Codes for DWC SuperSpeed USB 3.0 Controller

Component Name or Add-On	Product Code
Base Products	
DWC USB 3.0 Device-AHB[1]	6793-0
DWC USB 3.0 Host-AHB[2]	6897-0
DWC USB 3.0 Dual Role Device-AHB[3]	7567-0
DWC USB 3.0 Hub	6921-0
DWC AP USB 3.0 Device	C234-0
DWC AP USB 3.0 Host	C233-0
DWC AP USB 3.0 Dual Role Device	C232-0
Add-Ons	
DWC USB 3.0 Device-Isoch Add-On	7571-0
DWC USB 3.0 Host-Isoch Add-On	7593-0
DWC USB 3.0 DRD-Isoch Add-On	7568-0
DWC USB 3.0 Hibernation (LCA) Add-On	9228-0
DWC USB 3.0 HSIC Add-On	7387-0

1. DWC USB 3.0 Device-AHB includes DWC USB 3.0 Device-AXI Add-On and DWC USB 3.0 Device-Isoch Add-On.
2. DWC USB 3.0 Host-AHB includes DWC USB 3.0 Host-AXI Add-On and DWC USB 3.0 Host-Isoch Add-On.
3. DWC USB 3.0 Dual Role Device-AHB includes DWC USB 3.0 DRD-AXI Add-On and DWC USB 3.0 DRD-Isoch Add-On.

User Guide Organization

The purpose of this manual is to show you how to configure the controller, verify, and integrate it into your chip design. The chapters of this user guide are organized as follows:

- Chapter 1, "Design Flow,"introduces you to the coreConsultant tool, and describes the design flow of the DWC_usb3 controller.
- Chapter 2, "Configuring the Controller," describes how to configure the DWC_usb31 controller using coreConsultant tool, generate reports and views. It also describes how to run Spyglass Link and CDC.
- Chapter 3, "Simulating the Controller," describes the provided SoC Integration Example Testbench and Tests, and how to use them to verify the DWC_usb3 controller after integrating it into your SoC.
- Chapter 4, "Performing Synthesis and Post-Synthesis Activities," describes the synthesis flow for the DWC_usb3 controller. It also shows you how to run formal verification, use the coreConsultant tool to insert DFT, run automatic test pattern generation (ATPG), and perform static timing analysis (STA) and power analysis.
- Chapter 5, "Integrating the Controller," provides example integrations, and details on integrating the DWC_usb3 controller with USB PHYs.
- Appendix A, "Additional coreConsultant Information", provides miscellaneous information related to coreConsultant such as writing a batch script, accessing help information, and workspace directory contents.
- Appendix B, "FPGA Implementation of DWC_usb3 Controller", provides the guidelines for FPGA implementation of the DWC_usb3 controller.

Related Product Documentation

USB 3.0 Controller

Before installing the USB 3.0 Controller, you can download the latest document set, including the Datasheet, Installation Guide, QuickStart, Databook, Programming Guide, and Release Notes, at:

Device: https://www.synopsys.com/dw/ipdir.php?c=dwc_usb_3_0_device

Host: https://www.synopsys.com/dw/ipdir.php?c=dwc_usb_3_0_host

Dual-Role Device (DRD): https://www.synopsys.com/dw/ipdir.php?c=dwc_usb_3_0_drd

Hub: https://www.synopsys.com/dw/ipdir.php?c=dwc_usb_3_0_hub

(SolvNet ID required)

USB 3.0 Controller Linux Driver Software

Before installing the USB 3.0 Controller Linux Driver Software, you can download the latest *User Guide* and *Release Note* at:

http://www.synopsys.com/dw/ipdir.php?c=dwc_usb_3_0_device_software

(SolvNet ID required)

The driver software includes Doxygen-generated API documentation.

Verification IP (VIP)

To run simulations in coreConsultant, you must download and install the Synopsys Verification IP for USB 3.0 and AMBA from the SolvNet Download Center, at:

https://solvnet.synopsys.com/DownloadCenter/dc/product.jsp

For more information, refer to the *DesignWare Cores SuperSpeed USB 3.0 Installation Guide*.

Synopsys Tools

- *coreConsultant User Guide*, Synopsys, Inc. (included with the coreConsultant tool)
  http://www.synopsys.com/dw/doc.php/doc/coretools/latest/coreconsultant_user.pdf

Reference Documentation

The following standards are applicable to the USB 3.0 Controller product:

USB Protocol (from USB Implementers Forum)

- *Universal Serial Bus 3.0 Specification*, Revision 1.0, November 12, 2008
- *On-The-Go Supplement to the USB 2.0 Specification*, Revision 1.3, USB-IF, December 5, 2006
- *Universal Serial Bus Mass Storage Class UAS Protocol Specification,* Revision 0.6, June 1, 2008
- *Universal Serial Bus Specification*, Revision 2.0, USB-IF, April 27, 2000
- Errata for "USB Revision 2.0 April 27 2000" as of May 28, 2002, USB-IF
- *High-Speed Inter-Chip USB Electrical Specification, Version 1.0,* USB-IF, September 23, 2007
- *Inter-Chip Supplement to the USB Specification, Version 1.0,* USB-IF, March 13, 2006
  http://www.usb.org/developers/docs
- *On-The-Go Supplement to the USB 2.0 Specification*, Revision 2.0, May 8, 2009
- *On-The-Go and Embedded Host Supplement to the USB Revision 3.0 Specification* (Revision 1.0, May 9, 2011)
- *Battery Charging Specification*, Revision 1.2, Nov 2, 2010

USB Protocol (other)

- *Universal Serial Bus PC Legacy Compatibility Specification*, 0.9 Draft Revision, May 30, 1996

  http://wwwindev.synopsys.com/~dr/products/usb30/uploads/Main.ExternalSpecs/usb_le9.pdf
- *Inter-Chip Supplement to the USB Revision 3.0 Specification*, Revision 1.01, February 11, 2013

  http://www.usb.org

PHYs and Transceivers

- *PHY Interface for the PCI Express™, SATA, and USB 3.0 Architectures*, Version 4.00, Intel Corp.

  http://www.intel.com/content/dam/www/public/us/en/documents/white-papers/phy-interface-pci-express-sata-usb30-architectures.pdf
- *USB 2.0 Transceiver Macrocell Interface (UTMI) Specification*, Revision 1.05, Intel Corp., March 29, 2001

  http://www.intel.com/technology/usb/download/2_0_Xcvr_macrocell_1_05.pdf
- *UTMI+ Specification*, Revision 1.0, ULPI Working Group, February 25, 2004

  http://www.ulpi.org/ (May require registration)
- *UTMI+ Low Pin Interface (ULPI) Specification*, Revision 1.1, ULPI Working Group, October 20, 2004

  http://www.ulpi.org/ (May require registration)

- *PDIUSBP11A Universal Serial Bus Transceiver Product Specification*, Philips Semiconductor, June 12, 2001

  http://www.semiconductors.philips.com/acrobat_download/datasheets/PDIUSBP11A_3.pdf
- *Mini USB Analog Carkit Interface Specification*, CEA-936, Revision 2, December 2002

  http://global.ihs.com/ (Available for purchase)

SoC Buses

- *AMBA™ Specification*, Revision 2.0, ARM Ltd, ARM IHI 0001, 1999 (May require registration)

  http://www.arm.com/products/solutions/AMBA_Spec.html
- *AMBA® AXI(3) Protocol Specification'* v2.0, ARM Ltd, ARM IHI 0022C, 2003-2010 (May require registration)

  http://www.arm.com/products/solutions/axi_spec.html

Host Controllers

- *eXtensible Host Controller Interface for Universal Serial Bus (xHCI)*, Revision 0.96, Intel Corp., May 8, 2009 (Request from USB-IF)
- *eXtensible Host Controller Interface for Universal Serial Bus (xHCI)*, Revision 1.1, Intel Corp., December 20, 2013 (Request from USB-IF)

  http://www.intel.com/technology/usb/xhcispec.htm

Design Methodology

- *Reuse Methodology Manual: For System-On-A-Chip Designs*, Third Edition, Kluwer Academic Publishers, 2002

  http://www.springer.com/engineering/circuits+%26+systems/book/978-0-387-74098-0

Web Resources

- DesignWare IP product information: http://www.designware.com
- Your custom DesignWare IP page: http://www.mydesignware.com
- Documentation through SolvNet: http://solvnet.com (Synopsys password required)
- Synopsys Common Licensing (SCL): http://www.synopsys.com/keys

Customer Support

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

- First, prepare the following debug information, if applicable:
  - For environment setup problems or failures with configuration, simulation, or synthesis that occur within coreConsultant or coreAssembler, use the following menu entry:

    File > Build Debug Tar-file

    Check all the boxes in the dialog box that apply to your issue. This menu entry gathers all the Synopsys product data needed to begin debugging an issue and writes it to the file <*core tool startup directory*>/debug.tar.gz.
  - 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
- Then, contact the Support Center, with a description of your question and supply the above information, using one of the following methods:
  - *For fastest response*, use the SolvNet website. If you fill in your information as explained below, your issue is automatically routed to a support engineer who is experienced with your product. The **Sub Product** entry is critical for correct routing.

    Go to http://solvnet.synopsys.com/support/open_case.action.
    Provide the requested information, including:
    - Product: DesignWare Cores
    - Sub Product: USB 3.0 Device, USB 3.0 Host, USB 3.0 Hub, USB 3.0 DRD
    - Version: 3.30b
    - Problem Type
    - Priority
    - Title: Provide a short summary of the issue or list the error message you have encountered
    - Description: For simulation issues, include the timestamp of any signals or locations in waveforms that are not understood

    After creating the case, attach any debug files you created in the previous step.
  - 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 name, Sub Product name, Product version, and Tool Version number in your e-mail (as identified above) 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 in the previous step.
  - 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 Design Flow

This chapter provides an overview of the Synopsys coreConsultant tool. It also discusses the design flow of the DWC_usb3 controller. The topics in this chapter are as follows:

1.1 Overview of coreConsultant

DWC_usb3 controller to your design flow. The final output from the coreConsultant design flow is a set of RTL files (the configured controller) and scripts to allow you to run various tools standalone within your own design flow. The coreConsultant features and tasks include:

- A graphical user interface (GUI) to guide you through design flow activities

  You can also perform all activities through a command line interface. For details, 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

This user guide gives more detailed information on how to configure your controller and generate RTL files, as well as running synthesis and simulation.

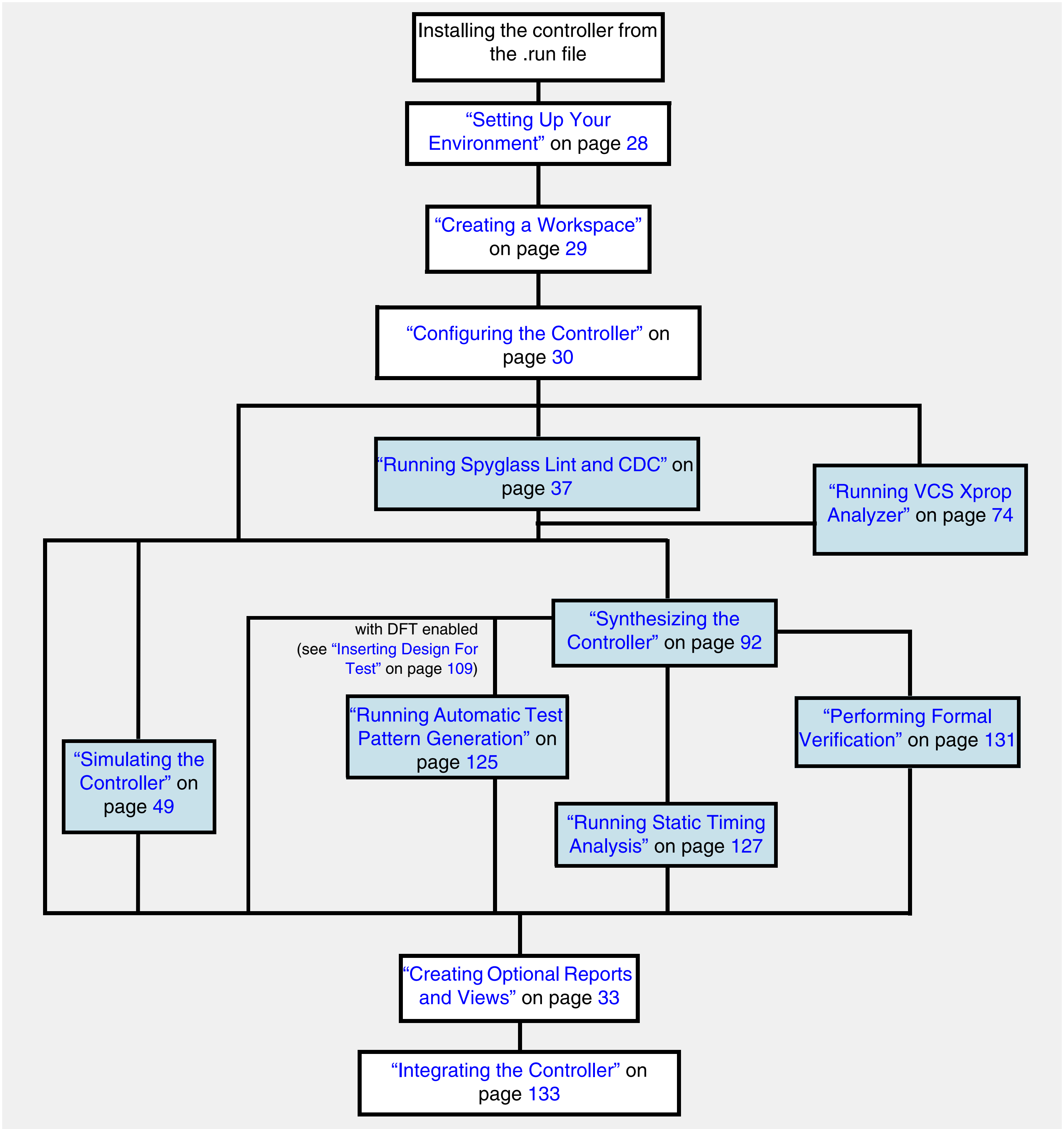
Hint When you are using more than one DesignWare Core, coreAssembler is recommended. The coreAssembler tool includes all of the features of coreConsultant but allows you to configure, automatically inter-connect, synthesize, and export a subsystem as opposed to a single component. For more information, see the coreAssembler documents mentioned in "Help Information" on page 200.

1.2 Design Flow

Figure 1-1 on page 25 gives the design flow for the DWC_usb31 controller. The coreConsultant GUI guides you through the controller design flow in your workspace. A workspace is a local Unix directory structure containing your configured copy of the USB 3.0 Controller (see "Creating a Workspace" on page 29).

Most coreConsultant activities are optional as indicated by the various paths shown in Figure 1-1 on page 25 and do not need to be completed before exporting the configured RTL to your chip design flow.

Figure 1-1 Design Flow


Installing the controller from the .run file
"Setting Up Your Environment" on page 28
"Creating a Workspace" on page 29
"Configuring the Controller" on page 30
"Running Spyglass Lint and CDC" on page 37
"Running VCS Xprop Analyzer" on page 74
with DFT enabled (see "Inserting Design For Test" on page 109)
"Synthesizing the Controller" on page 92
"Running Automatic Test Pattern Generation" on page 125
"Simulating the Controller" on page 49
"Performing Formal Verification" on page 131
"Running Static Timing Analysis" on page 127
"Creating Optional Reports and Views" on page 33
"Integrating the Controller" on page 133

🟦 To perform these activities, you need the supported versions of the tools. For more information, refer to *DWC SuperSpeed USB 3.0 Controller Installation Guide*.

2 Configuring the Controller

This chapter shows you how to use the coreConsultant tool to configure the DWC_usb3 controller, generate optional reports and views, and run Spyglass Lint and CDC.

The topics in this chapter are as follows:

Hint When you are using more than one DesignWare controller, coreAssembler is recommended. The coreAssembler tool includes all of the features of coreConsultant but allows you to configure, automatically inter-connect, synthesize, and export a subsystem as opposed to a single component. For more information, see the coreAssembler documents mentioned in "Help Information" on page 200.

2.1 Prerequisites for Configuring the Controller

This section describes how to set up your environment and create a workspace so that you can start configuring the controller.

2.1.1 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, perform these steps:

- Confirm that you have installed the controller design files and fully set up your environment as described in the *DWC SuperSpeed USB 3.0 Controller Installation Guide*(provide hyperlink to pdf if applicable).
- Confirm that you have installed the required version of Discovery Verification IP for USB 3.0 and AMBA Verification IP as specified 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 Directory Structure" appendix of the *Installation Guide*.

  ```
  % ls $DESIGNWARE_HOME
  ```
- If you are using Synopsys Design Compiler, check that $SYNOPSYS points to the Synopsys tools tree.
- Check that the USB 3.0 Controller licenses are installed correctly on your license server by using the `lmstat` command. For example, enter:

  ```
  % lmstat -a -c $LM_LICENSE_FILE | grep <LICENSE_NAME>
  ```

  For more information, see the "Setting License File Environment Variable" and "Checking License Requirements" sections of the *Installation Guide.*

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

2.1.2 Creating a Workspace

A workspace is a local UNIX directory structure containing your configured copy of the DWC_usb3 controller. For more details about this directory structure, refer to "Workspace Directory Structure Overview" on page 201. You can create several workspaces to experiment with different design alternatives.

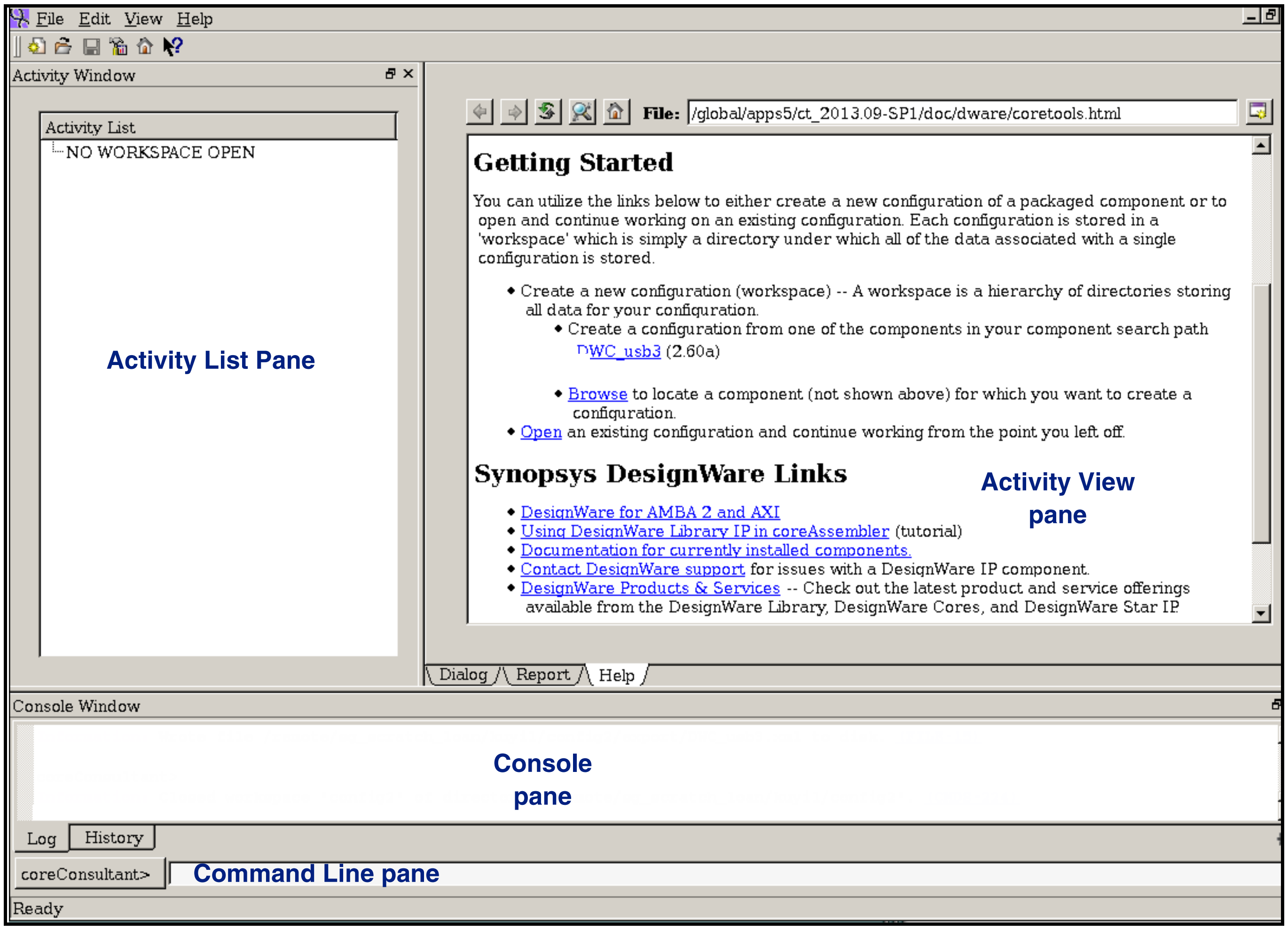
To create a workspace, follow these steps:

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

   ```
   % coreConsultant &
   ```

   Figure 2-1 shows the initial coreConsultant screen.

Figure 2-1 Initial coreConsultant Screen



Getting Started

You can utilize the links below to either create a new configuration of a packaged component or to open and continue working on an existing configuration. Each configuration is stored in a 'workspace' which is simply a directory under which all of the data associated with a single configuration is stored.

- Create a new configuration (workspace) -- A workspace is a hierarchy of directories storing all data for your configuration.
  - Create a configuration from one of the components in your component search path
    DWC_usb3 (2.60a)
  - Browse to locate a component (not shown above) for which you want to create a configuration.
- Open an existing configuration and continue working from the point you left off.

Synopsys DesignWare Links

- DesignWare for AMBA 2 and AXI
- Using DesignWare Library IP in coreAssembler (tutorial)
- Documentation for currently installed components.
- Contact DesignWare support for issues with a DesignWare IP component.
- DesignWare Products & Services -- Check out the latest product and service offerings available from the DesignWare Library, DesignWare Cores, and DesignWare Star IP

3. Create a workspace.

   From this screen, click on the name of the IP controller that you wish to configure.

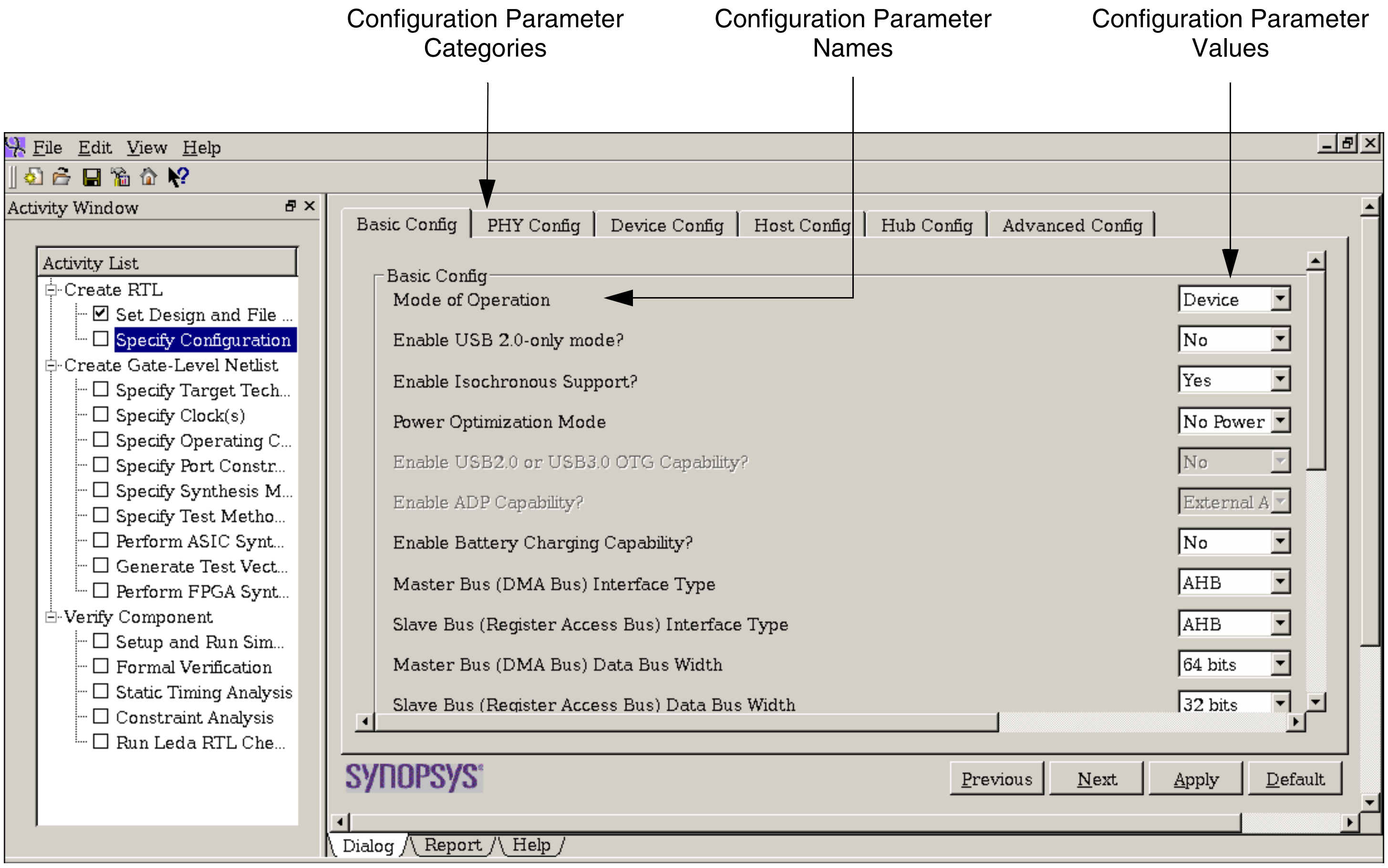
Hint Here is a common problem that can occur:

The DWC_usb3 controller is not visible in the initial coreConsultant page. The $DESIGNWARE_HOME environmental variable is not set (or is not pointing to your IP installation directory) in the shell from which you started coreConsultant. For more details, refer to "Setting Up Your Environment" on page 28

2.2 Configuring the Controller

After you have created a workspace, configure the controller and create the RTL using the Create RTL activity dialog boxes as illustrated in Figure 2-2.

Figure 2-2 Specify Configuration Activity



Basic Config

Mode of Operation	Device
Enable USB 2.0-only mode?	No
Enable Isochronous Support?	Yes
Power Optimization Mode	No Power
Enable USB2.0 or USB3.0 OTG Capability?	No
Enable ADP Capability?	External A
Enable Battery Charging Capability?	No
Master Bus (DMA Bus) Interface Type	AHB
Slave Bus (Register Access Bus) Interface Type	AHB
Master Bus (DMA Bus) Data Bus Width	64 bits
Slave Bus (Register Access Bus) Data Bus Width	32 bits

1. **Specify your configuration.**

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

   - You can use default values 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 value 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 device mode of operation, coreConsultant disables all configuration parameters that are related only to the host mode of operation. For example, when you select a single-function configuration, 0, coreConsultant disables the parameter selection for functions 1–7.
   - For more information about the configuration parameters, refer to the "Parameters" chapter in the *DWC USB 3.0 Controller Databook*.

Note Use the Set Design 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.

2. **Generate RTL.**

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

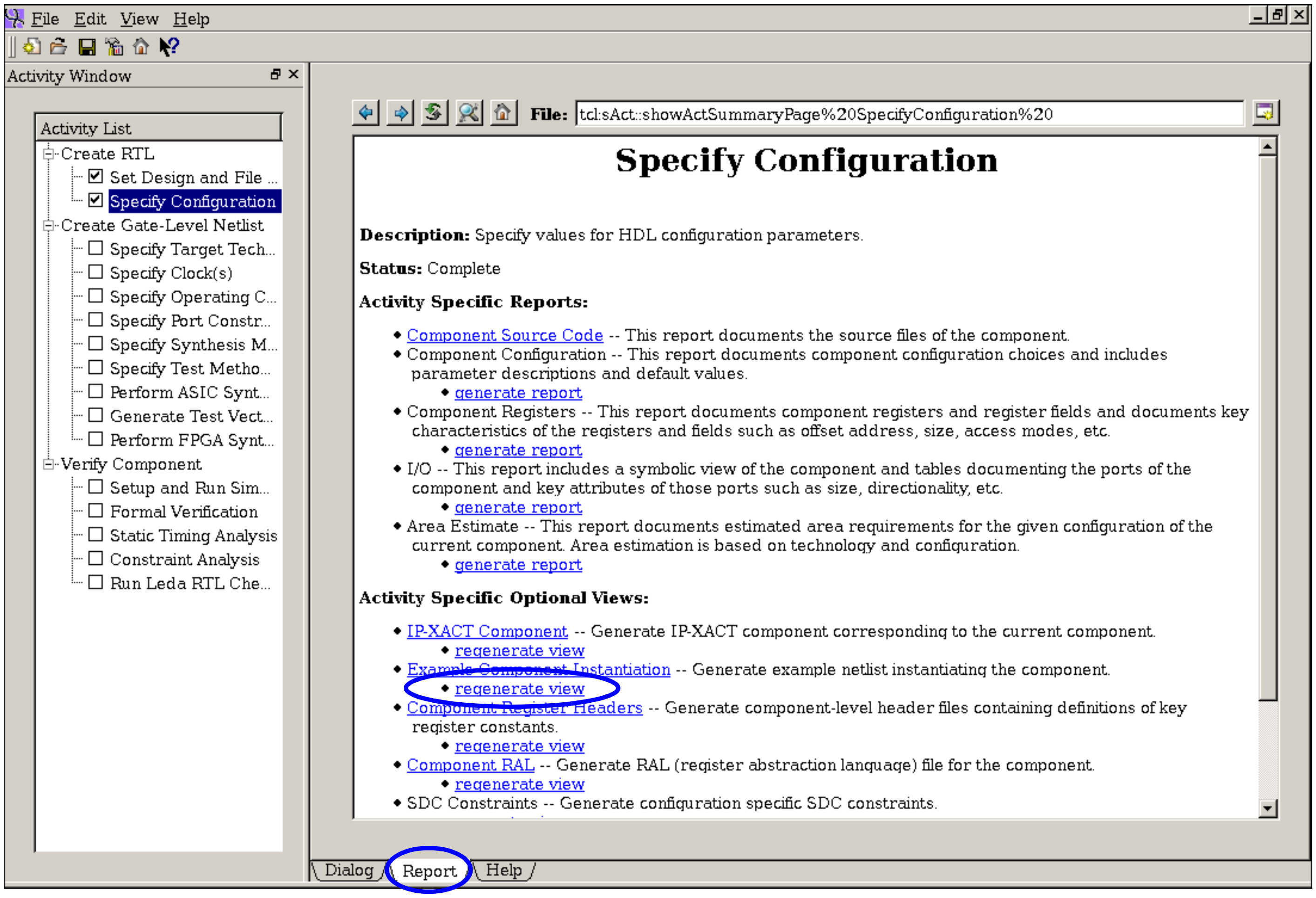
   You can return to the Configuration activity to configure the controller again (create new RTL) at any time.

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 details, see "Creating a Batch Script" on page 199.

3. **View reports and generate optional views/reports.**

   After you have configured the controller, coreConsultant generates several configuration reports. You can access these from the Report tab (shown in Figure 2-3 on page 32).

Figure 2-3 Configuration Reports



tcl:sAct::showActSummaryPage%20SpecifyConfiguration%20

Specify Configuration

Description: Specify values for HDL configuration parameters.

Status: Complete

Activity Specific Reports:

- Component Source Code -- This report documents the source files of the component.
- Component Configuration -- This report documents component configuration choices and includes parameter descriptions and default values.
  - generate report
- Component Registers -- This report documents component registers and register fields and documents key characteristics of the registers and fields such as offset address, size, access modes, etc.
  - generate report
- I/O -- This report includes a symbolic view of the component and tables documenting the ports of the component and key attributes of those ports such as size, directionality, etc.
  - generate report
- Area Estimate -- This report documents estimated area requirements for the given configuration of the current component. Area estimation is based on technology and configuration.
  - generate report

Activity Specific Optional Views:

- IP-XACT Component -- Generate IP-XACT component corresponding to the current component.
  - regenerate view
- Example Component Instantiation -- Generate example netlist instantiating the component.
  - regenerate view
- Component Register Headers -- Generate component-level header files containing definitions of key register constants.
  - regenerate view
- Component RAL -- Generate RAL (register abstraction language) file for the component.
  - regenerate view
- SDC Constraints -- Generate configuration specific SDC constraints.

Also, you can generate Activity Specific Reports and Activity Specific Optional Views. For more information, refer to "Creating Optional Reports and Views" on page 33. For example, to generate example component instantiation, click the **generate view** link as indicated in Figure 2-3.

Note If any problems occur during or after the Specify Configuration activity, refer to "Troubleshooting" on page 198.

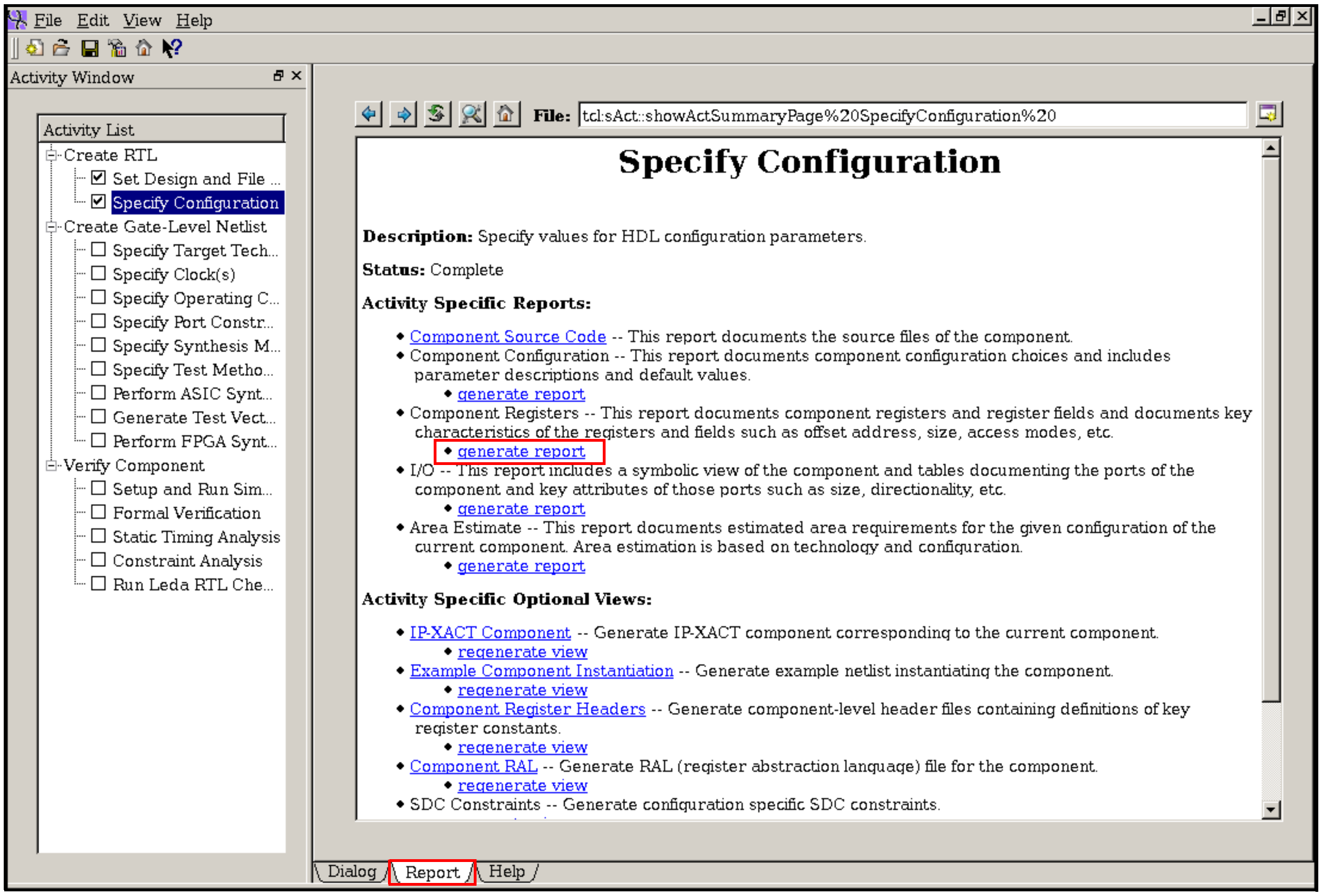
At this point, you can verify the controller ("Simulating the Controller" on page 49) or generate a gate-level netlist ("Synthesizing the Controller" on page 92).

2.3 Creating Optional Reports and Views

After you have configured the controller, coreConsultant generates several configuration reports and displays the a list of activity-specific views you can optionally generate. You can access these by clicking the **Report** tab under the **Specify Configuration** activity shown in Figure 2-4.

Attention Some of the activity-specific report creation may not be fully functional with your controller. For assistance, contact Synopsys Customer Support.

Figure 2-4 Generating Activity-Specific Reports and Views



Specify Configuration

Description: Specify values for HDL configuration parameters.

Status: Complete

Activity Specific Reports:

- Component Source Code -- This report documents the source files of the component.
- Component Configuration -- This report documents component configuration choices and includes parameter descriptions and default values.
  - generate report
- Component Registers -- This report documents component registers and register fields and documents key characteristics of the registers and fields such as offset address, size, access modes, etc.
  - generate report
- I/O -- This report includes a symbolic view of the component and tables documenting the ports of the component and key attributes of those ports such as size, directionality, etc.
  - generate report
- Area Estimate -- This report documents estimated area requirements for the given configuration of the current component. Area estimation is based on technology and configuration.
  - generate report

Activity Specific Optional Views:

- IP-XACT Component -- Generate IP-XACT component corresponding to the current component.
  - regenerate view
- Example Component Instantiation -- Generate example netlist instantiating the component.
  - regenerate view
- Component Register Headers -- Generate component-level header files containing definitions of key register constants.
  - regenerate view
- Component RAL -- Generate RAL (register abstraction language) file for the component.
  - regenerate view
- SDC Constraints -- Generate configuration specific SDC constraints.

2.3.1 Activity-Specific Reports and Views

You can generate reports that document I/O signals, parameters, and register descriptions. component view that you can use.

Table 2-1 describes the current optional reports.

Table 2-1 Activity-Specific Reports

Report	Filenames (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, access mode and so on
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, directionality, and so on
Area Estimate	AreaEstimate.html AreaEstimate.xml	Documents estimated area requirements for the given configuration of the current component. Area estimation is based on the technology and configuration.

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 2-2 describes the activity-specific optional views.

Table 2-2 describes the current optional reports.

Table 2-2 Activity-Specific Optional Views

Report	Filenames (in *workspace*/export)	Description
IP-XACT Component	DWC_usb3.xml	Generates IP-XACT component corresponding to the current component
Example Component Instantiation	DWC_usb3_inst.v	Generates an example netlist instantiating the component
Component Register Headers	./headers/*	Generates component-level header files containing definitions of key register constants
Component RAL	DWC_usb3.ralf	Generates RAL (register abstraction language) file for the component
SDC Constraints	./compress_sdc/DWC_usb3.sdc	Generates configuration-specific SDC constraints file

Note If you want reports to be automatically generated, select the appropriate check boxes in the **File > Generate Reports** or **File > Generate Optional Views** dialog box.

2.3.2 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 applying an XSLT style sheet.

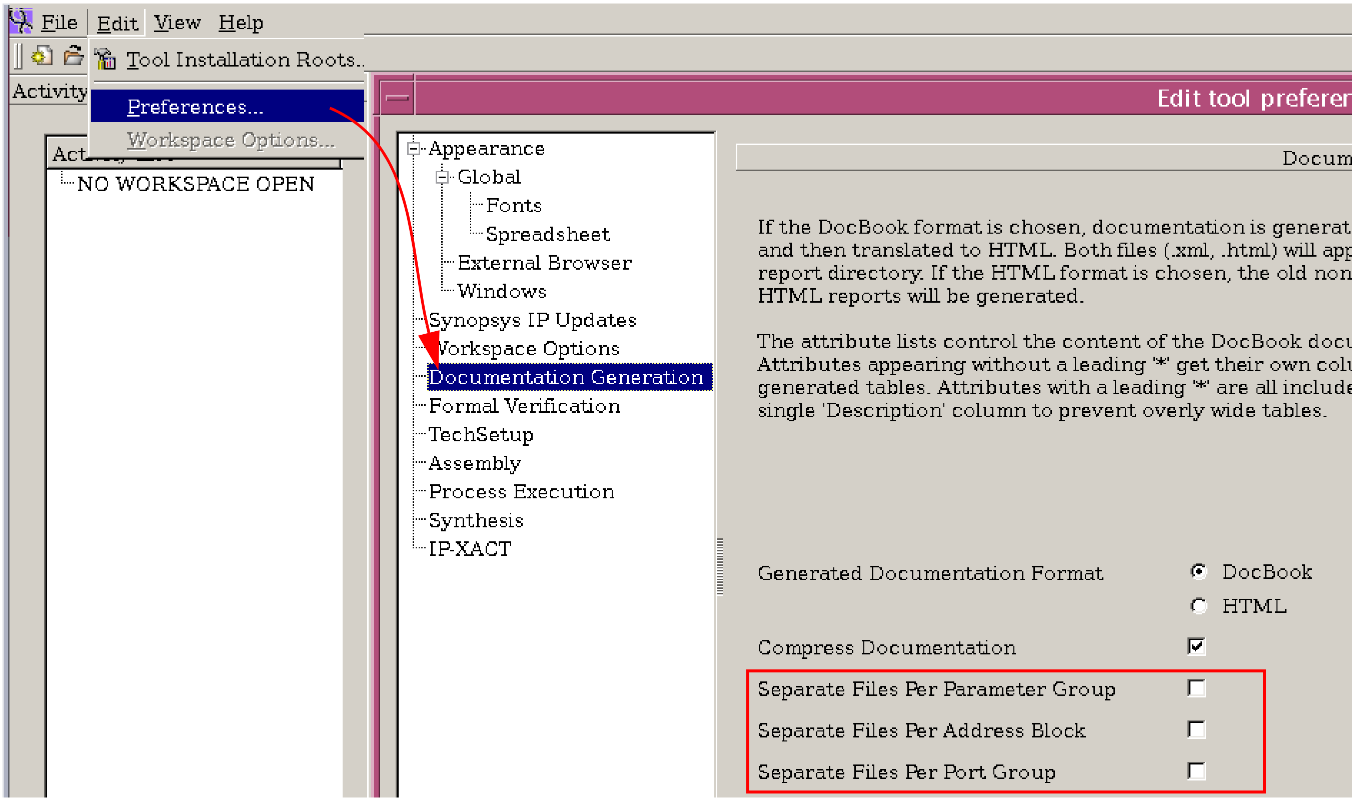
2.3.3 Setting Preferences for Report Generation

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 2-5.

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 2-5 Separate Files Options



If the DocBook format is chosen, documentation is generat
and then translated to HTML. Both files (.xml, .html) will app
report directory. If the HTML format is chosen, the old non
HTML reports will be generated.

The attribute lists control the content of the DocBook docu
Attributes appearing without a leading '*' get their own colu
generated tables. Attributes with a leading '*' are all include
single 'Description' column to prevent overly wide tables.

Generated Documentation Format	[✓ DocBook] HTML
Compress Documentation	✓
Separate Files Per Parameter Group	☐
Separate Files Per Address Block	☐
Separate Files Per Port Group	☐

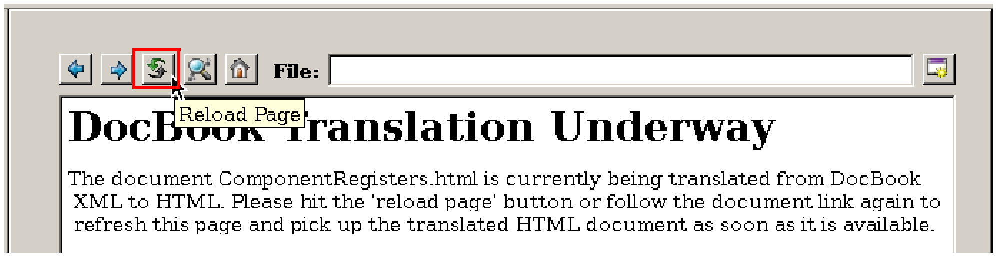
2.3.4 Generating Activity-Specific Reports

To generate the required report, click the respective **generate report**. For example, to generate component registers-related reports, click the one highlighted in Figure 2-4 on page 33.

2.3.5 Accessing Reports

To view the report, click **Reload Page** after clicking **generate report** as shown in Figure 2-6.

Figure 2-6 Viewing Reports

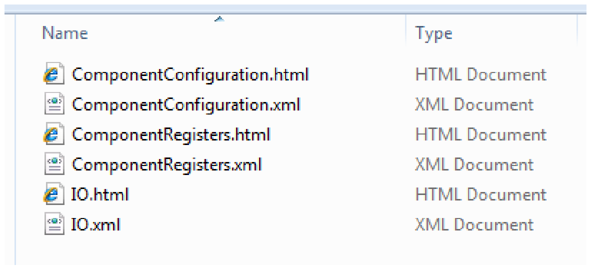


DocBook Translation Underway

The document ComponentRegisters.html is currently being translated from DocBook XML to HTML. Please hit the 'reload page' button or follow the document link again to refresh this page and pick up the translated HTML document as soon as it is available.

You can also access these report files from the **<your workspace>/report** directory. The content of this directory is similar to that shown in Figure 2-7:

Figure 2-7 Contents of the report Directory



Name	Type
ComponentConfiguration.html	HTML Document
ComponentConfiguration.xml	XML Document
ComponentRegisters.html	HTML Document
ComponentRegisters.xml	XML Document
IO.html	HTML Document
IO.xml	XML Document

2.3.6 Generating and Accessing Activity-Specific Views

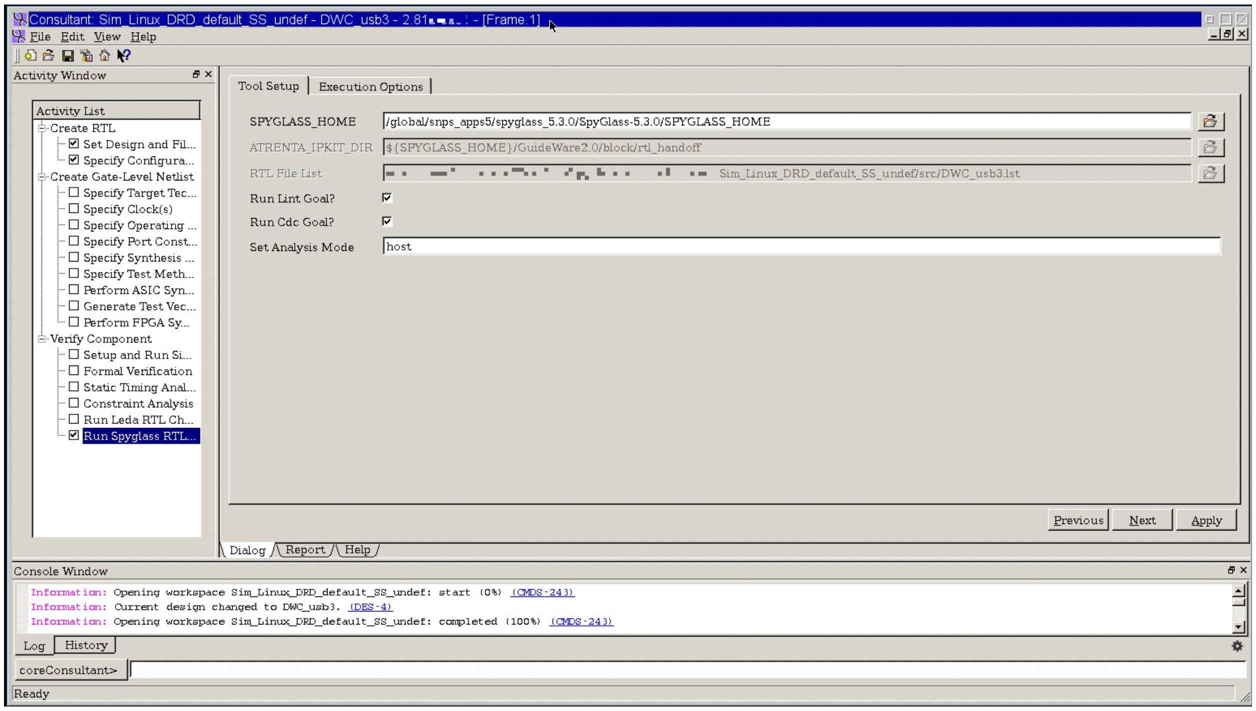
To generate the required view, click the respective **generate view** in the **Report** tab.

Access the generated views from the ***workspace*/export** directory.

2.4 Running Spyglass Lint and CDC

Follow the steps in this section for running Spyglass Lint and CDC.

Figure 2-8 Spyglass Options in coreConsultant



Consultant: Sim_Linux_DRD_default_SS_undef - DWC_usb3 - 2.81[redacted] - [Frame.1]

Activity List
- Create RTL
  - ☑ Set Design and Fil...
  - ☑ Specify Configura...
- Create Gate-Level Netlist
  - ☐ Specify Target Tec...
  - ☐ Specify Clock(s)
  - ☐ Specify Operating ...
  - ☐ Specify Port Const...
  - ☐ Specify Synthesis ...
  - ☐ Specify Test Meth...
  - ☐ Perform ASIC Syn...
  - ☐ Generate Test Vec...
  - ☐ Perform FPGA Sy...
- Verify Component
  - ☐ Setup and Run Si...
  - ☐ Formal Verification
  - ☐ Static Timing Anal...
  - ☐ Constraint Analysis
  - ☐ Run Leda RTL Ch...
  - ☑ Run Spyglass RTL...

SPYGLASS_HOME	/global/snps_apps5/spyglass_5.3.0/SpyGlass-5.3.0/SPYGLASS_HOME
ATRENTA_IPKIT_DIR	${SPYGLASS_HOME}/GuideWare2.0/block/rtl_handoff
RTL File List	[redacted] Sim_Linux_DRD_default_SS_undef/src/DWC_usb3.lst
Run Lint Goal?	☑
Run Cdc Goal?	☑
Set Analysis Mode	host

```
Information: Opening workspace Sim_Linux_DRD_default_SS_undef: start (0%) (CMDS-243)
Information: Current design changed to DWC_usb3. (DES-4)
Information: Opening workspace Sim_Linux_DRD_default_SS_undef: completed (100%) (CMDS-243)
```

As shown in Figure 2-8, you can run Lint and CDC goals in the coreConsultant GUI.

The Spyglass flow in coreConsultant runs Guideware 2.0 rules for block/rtl_handoff. Within the block/rtl_handoff, only lint/lint_rtl and cdc/cdc_verify_struct goals are run.

1. Select either Lint, CDC, or both run goals. By default, both Lint and CDC are enabled.

2. Specify the Set Analysis Mode value. Table 2-3 lists the valid values for Set Analysis Mode and their description.

Table 2-3 Set Analysis Mode for Spyglass Lint and CDC – Valid Values

Set Analysis Mode	Description
host	In this mode, CDC analysis is done for host functionality with pipe3_clk for SS logic and utmi_clk for HS logic. This mode assumes that the controller is operating as a host with SS and HS ports.
host_ulpi	In this mode, CDC analysis is done for host functionality with pipe3_clk for SS logic and ulpi_clk for HS logic. This mode assumes that the controller is operating as a host with SS and HS ports.
dev_usb3	In this mode, CDC analysis is done with pipe3_clk for SS logic only. This mode assumes that the controller is operating as a SS device.

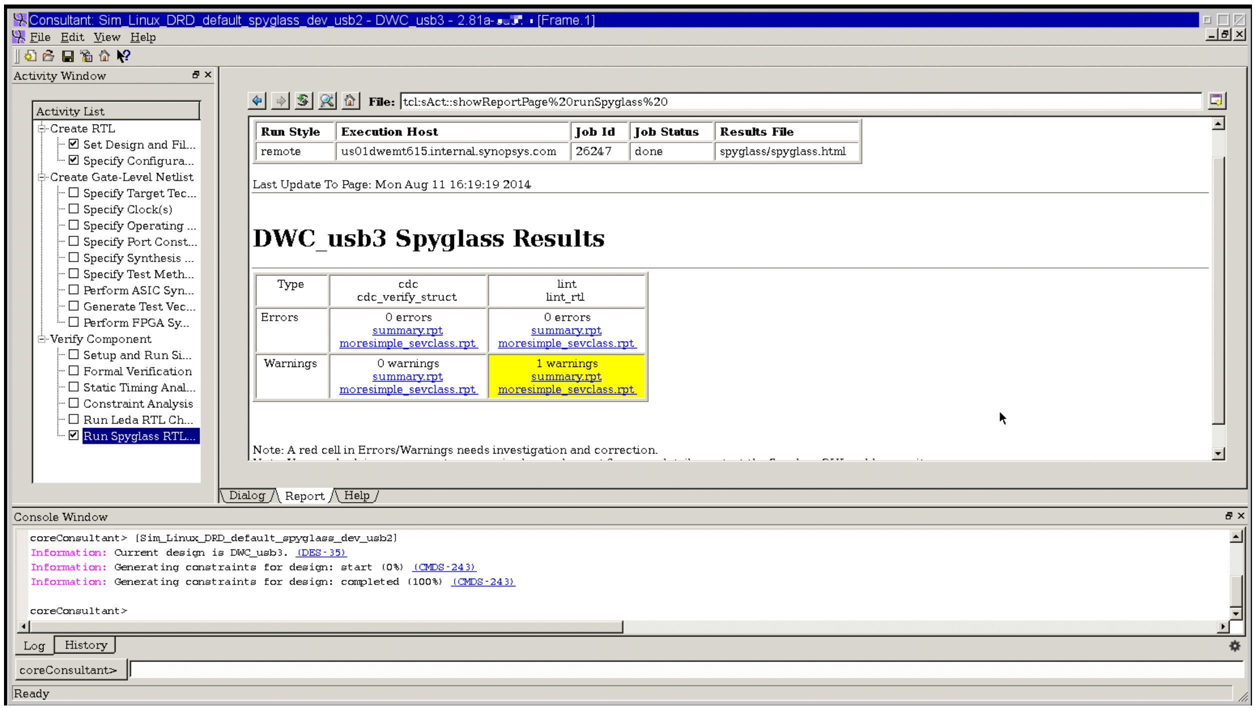
Table 2-3 Set Analysis Mode for Spyglass Lint and CDC – Valid Values

Set Analysis Mode	Description
dev_usb2	In this mode, CDC analysis is done with utmi_clk for HS logic only. This mode assumes that the device controller is operating in HS/FS.
dev_usb2_ulpi	In this mode, CDC analysis is done with ulpi_clk for HS logic only. This mode assumes that the device controller is operating in HS/FS.
hub	In this mode, CDC analysis is done for hub functionality.
dev_usb3_lp	In this mode, CDC analysis is done for low power mode for SS logic.
dev_usb2_usb3_lp	In this mode, CDC analysis is done for low power mode for usb2 logic with utmi_clk.
dev_usb2_ulpi_usb3_lp	In this mode, CDC analysis is done for low power mode for usb2 logic with ulpi_clk.

Note Do not input a value that is irrelevant for a given mode. For instance, if the configuration is device-only (DWC_USB3_MODE=0), do not run with a value of "host."

When the Lint and/or CDC is run, the results are available in the Report tab. Any error is displayed with a red colored cell and any warning is displayed in displayed in a yellow colored cell as shown in Figure 2-9.

Figure 2-9 coreConsultant Spyglass Report Summary



Run Style	Execution Host	Job Id	Job Status	Results File
remote	us01dwemt615.internal.synopsys.com	26247	done	spyglass/spyglass.html

Last Update To Page: Mon Aug 11 16:19:19 2014

DWC_usb3 Spyglass Results

Type	cdc cdc_verify_struct	lint lint_rtl
Errors	0 errors summary.rpt moresimple_sevclass.rpt	0 errors summary.rpt moresimple_sevclass.rpt
Warnings	0 warnings summary.rpt moresimple_sevclass.rpt	1 warnings summary.rpt moresimple_sevclass.rpt

Note: A red cell in Errors/Warnings needs investigation and correction.

```
coreConsultant> [Sim_Linux_DRD_default_spyglass_dev_usb2]
Information: Current design is DWC_usb3. (DES-35)
Information: Generating constraints for design: start (0%) (CMDS-243)
Information: Generating constraints for design: completed (100%) (CMDS-243)

coreConsultant>
```

2.4.1 Fixed Settings

The settings are fixed (hardcoded) when you run Spyglass in coreConsultant.

- USB3 mode analysis is performed with PIPE data width assigned to 32 bits.
- ram_clk is assigned to bus_clk domain.
- suspend_clk is used only in low power modes, and not used in normal modes.

2.4.2 Errors and Warnings

It is possible for errors and/or warnings to occur when you run Spyglass using the coreConsultant flow because of the large number of configurations. If you encounter errors and/or warnings, open a support case.

When the Spyglass run fails with a FATAL error, inspect the following files:

File	Description
configured_workspace/spyglass/work/core/<design_prefix>DWC_usb31/cdc/cdc_verify_struct	
spyglass.log	Log file which records all the run-time information
spyglass_reports/spyglass_violations.rpt	Report for any CDC violations
configured_workspace/spyglass/work/core/<design_prefix>DWC_usb31/lint/lint_rtl	
spyglass.log	Log file which records all the run-time information

File	Description
spyglass_reports/spyglass_violations.rp	Report for any Lint violations

2.4.3 Lint

The coreConsultant tool uses the following waiver files for Spyglass Lint.

FileName	Description
<workspace>/spyglass/spyglass_design_specific_waivers.swl	These are DWC_usb3 design specific rule waivers. There are 2 sections in this waiver file. The first portion is Lint waivers. The second portion is CDC waivers. The reason(s) for each of the the waivers are included as comments in the file.
<workspace>/spyglass/spyglass_engineering_council_waivers.swl	These are rules which Synopsys waives for its IPs.

2.4.4 CDC

To define the Spyglass CDC constraints, it is important to understand the reset and clock logic used in DWC_usb3. For more information, refer to the *DesignWare Cores SuperSpeed USB 3.0 Controller Databook*.

The following figures show the reset and clock logic for various DWC_usb3 modes of operation:

- "Reset Logic"
- "Clock Domains" on page 43
- "USB 2.0 Only Mode" on page 44
- "USB 3.0 Host" on page 45
- "USB 3.0 Device and DRD" on page 46

Reset Logic

- Resets are asserted asynchronously and de-asserted synchronously with regards to respective clock domains.
- The reset generation logic also has soft reset control inputs.

Refer to Figure 2-10 on page 42.

Figure 2-10 Reset Logic

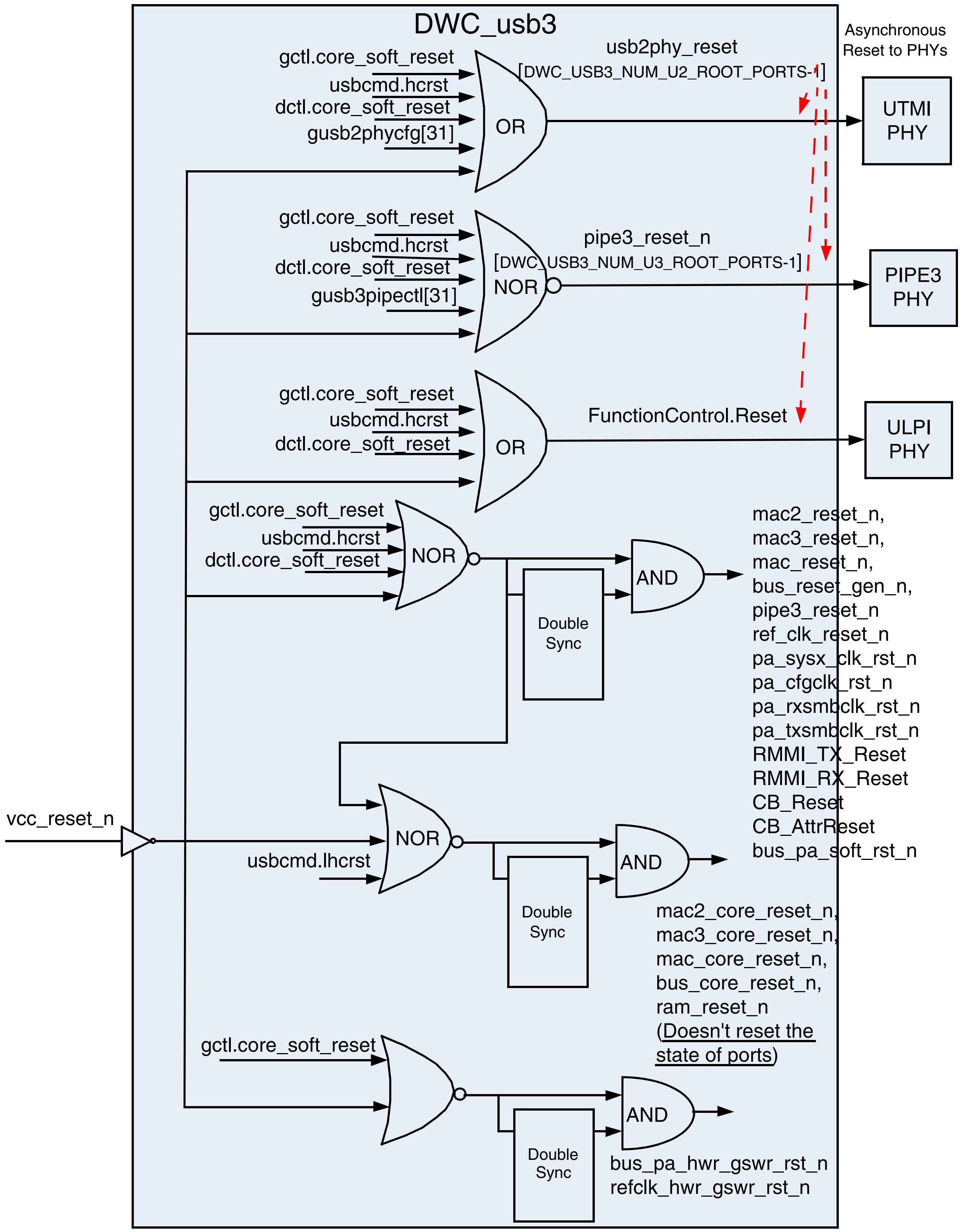


Figure 2-11 Clock Domains

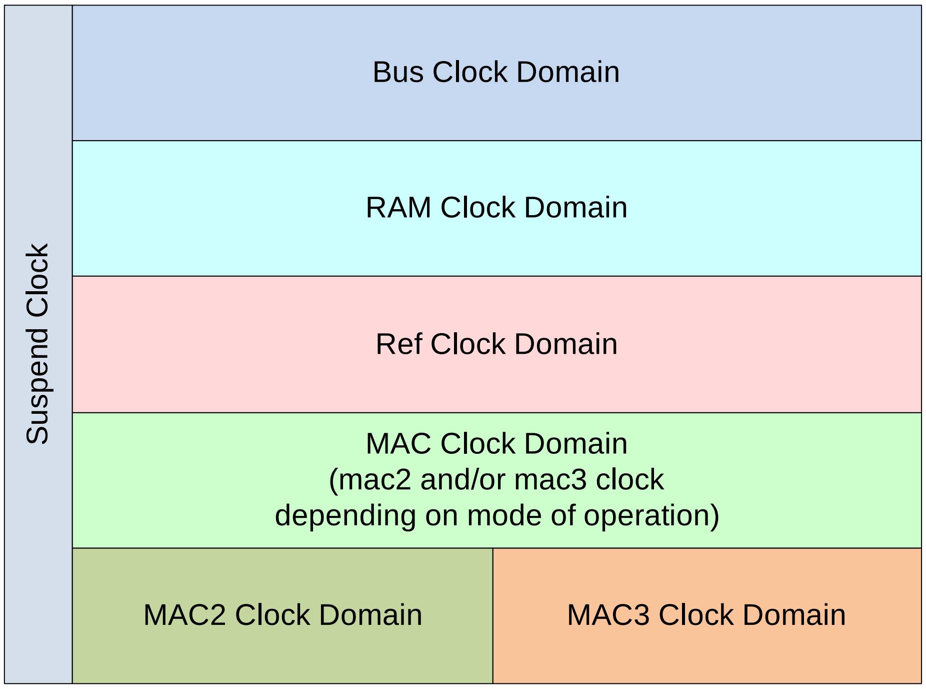


Figure 2-12 USB 2.0 Only Mode

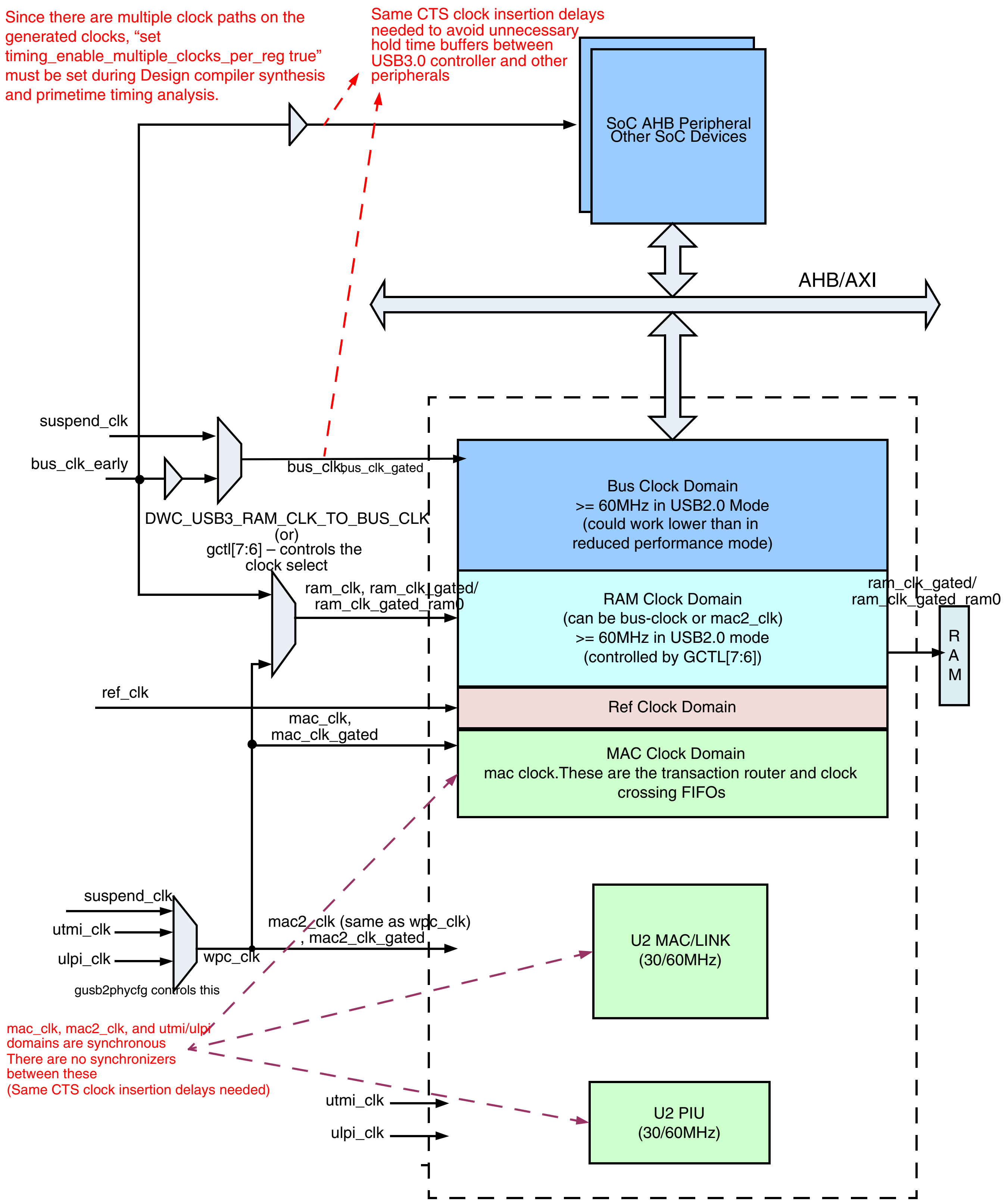

Since there are multiple clock paths on the generated clocks, "set timing_enable_multiple_clocks_per_reg true" must be set during Design compiler synthesis and primetime timing analysis.
Same CTS clock insertion delays needed to avoid unnecessary hold time buffers between USB3.0 controller and other peripherals
SoC AHB Peripheral
Other SoC Devices
AHB/AXI
suspend_clk
bus_clk_early
bus_clk
bus_clk_gated
DWC_USB3_RAM_CLK_TO_BUS_CLK (or) gctl[7:6] – controls the clock select
Bus Clock Domain >= 60MHz in USB2.0 Mode (could work lower than in reduced performance mode)
ram_clk, ram_clk_gated/ ram_clk_gated_ram0
RAM Clock Domain (can be bus-clock or mac2_clk) >= 60MHz in USB2.0 mode (controlled by GCTL[7:6])
ram_clk_gated/ ram_clk_gated_ram0
R A M
ref_clk
Ref Clock Domain
mac_clk, mac_clk_gated
MAC Clock Domain mac clock.These are the transaction router and clock crossing FIFOs
suspend_clk
utmi_clk
ulpi_clk
wpc_clk
gusb2phycfg controls this
mac2_clk (same as wpc_clk) , mac2_clk_gated
U2 MAC/LINK (30/60MHz)
mac_clk, mac2_clk, and utmi/ulpi domains are synchronous There are no synchronizers between these (Same CTS clock insertion delays needed)
utmi_clk
ulpi_clk
U2 PIU (30/60MHz)

Figure 2-13 USB 3.0 Host

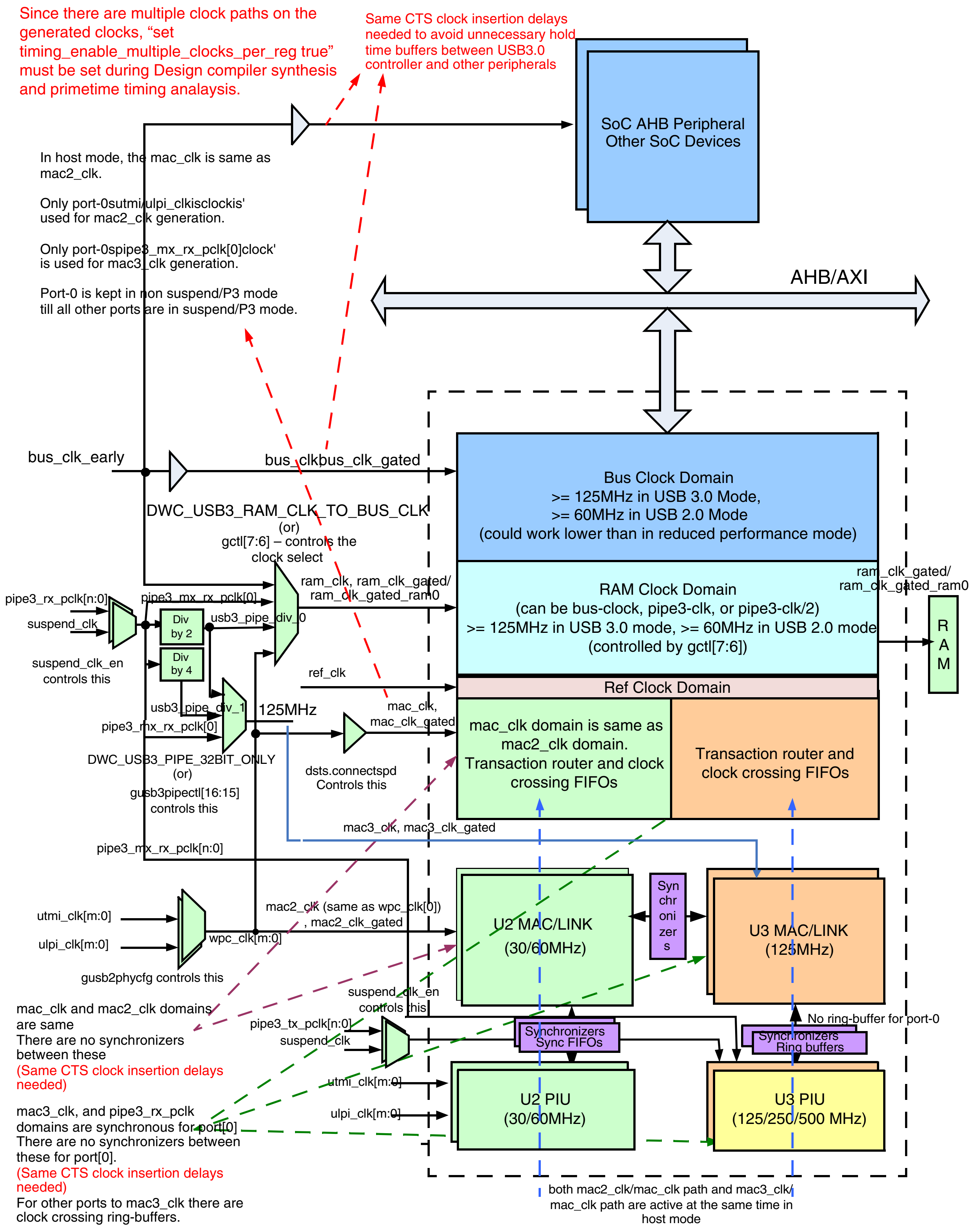
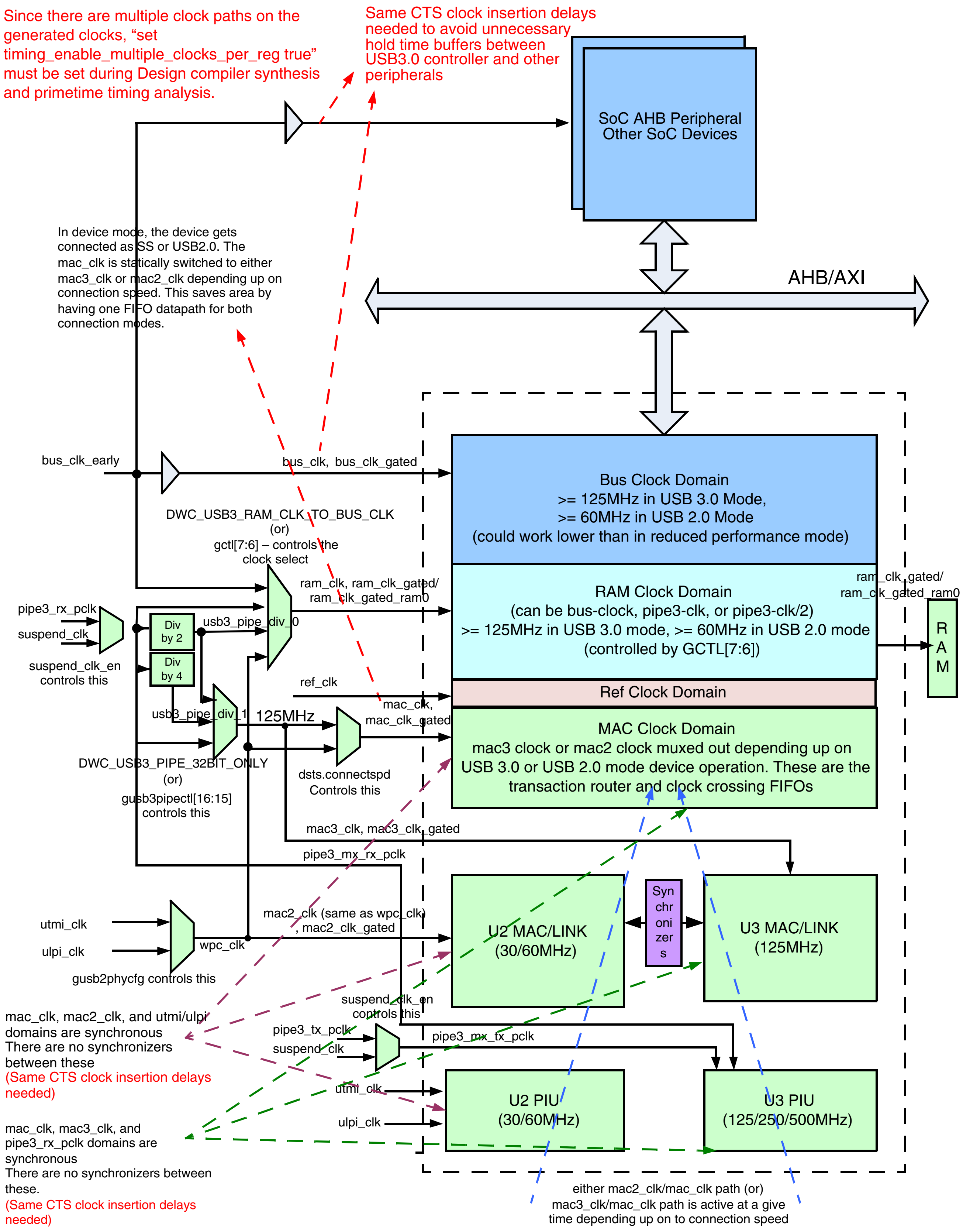

Since there are multiple clock paths on the generated clocks, "set timing_enable_multiple_clocks_per_reg true" must be set during Design compiler synthesis and primetime timing analysis.
Same CTS clock insertion delays needed to avoid unnecessary hold time buffers between USB3.0 controller and other peripherals
SoC AHB Peripheral Other SoC Devices
AHB/AXI
In host mode, the mac_clk is same as mac2_clk.
Only port-0sutmi/ulpi_clkisclockis' used for mac2_clk generation.
Only port-0spipe3_mx_rx_pclk[0]clock' is used for mac3_clk generation.
Port-0 is kept in non suspend/P3 mode till all other ports are in suspend/P3 mode.
bus_clk_early
bus_clk
bus_clk_gated
DWC_USB3_RAM_CLK_TO_BUS_CLK
(or)
gctl[7:6] – controls the clock select
Bus Clock Domain
>= 125MHz in USB 3.0 Mode,
>= 60MHz in USB 2.0 Mode
(could work lower than in reduced performance mode)
ram_clk, ram_clk_gated/ ram_clk_gated_ram0
ram_clk_gated/ ram_clk_gated_ram0
RAM Clock Domain
(can be bus-clock, pipe3-clk, or pipe3-clk/2)
>= 125MHz in USB 3.0 mode, >= 60MHz in USB 2.0 mode
(controlled by gctl[7:6])
R A M
pipe3_rx_pclk[n:0]
suspend_clk
pipe3_mx_rx_pclk[0]
Div by 2
usb3_pipe_div_0
Div by 4
suspend_clk_en controls this
ref_clk
Ref Clock Domain
usb3_pipe_div_1
125MHz
pipe3_mx_rx_pclk[0]
mac_clk, mac_clk_gated
mac_clk domain is same as mac2_clk domain. Transaction router and clock crossing FIFOs
Transaction router and clock crossing FIFOs
DWC_USB3_PIPE_32BIT_ONLY
(or)
gusb3pipectl[16:15] controls this
dsts.connectspd Controls this
mac3_clk, mac3_clk_gated
pipe3_mx_rx_pclk[n:0]
utmi_clk[m:0]
ulpi_clk[m:0]
wpc_clk[m:0]
mac2_clk (same as wpc_clk[0]) , mac2_clk_gated
U2 MAC/LINK (30/60MHz)
Syn chr oni zer s
U3 MAC/LINK (125MHz)
gusb2phycfg controls this
suspend_clk_en controls this
pipe3_tx_pclk[n:0]
suspend_clk
Synchronizers Sync FIFOs
Synchronizers Ring buffers
No ring-buffer for port-0
utmi_clk[m:0]
ulpi_clk[m:0]
U2 PIU (30/60MHz)
U3 PIU (125/250/500 MHz)
mac_clk and mac2_clk domains are same
There are no synchronizers between these
(Same CTS clock insertion delays needed)
mac3_clk, and pipe3_rx_pclk domains are synchronous for port[0]
There are no synchronizers between these for port[0].
(Same CTS clock insertion delays needed)
For other ports to mac3_clk there are clock crossing ring-buffers.
both mac2_clk/mac_clk path and mac3_clk/ mac_clk path are active at the same time in host mode

Figure 2-14 USB 3.0 Device and DRD


Since there are multiple clock paths on the generated clocks, "set timing_enable_multiple_clocks_per_reg true" must be set during Design compiler synthesis and primetime timing analysis.
Same CTS clock insertion delays needed to avoid unnecessary hold time buffers between USB3.0 controller and other peripherals
SoC AHB Peripheral Other SoC Devices
AHB/AXI
In device mode, the device gets connected as SS or USB2.0. The mac_clk is statically switched to either mac3_clk or mac2_clk depending up on connection speed. This saves area by having one FIFO datapath for both connection modes.
bus_clk_early
bus_clk, bus_clk_gated
Bus Clock Domain
>= 125MHz in USB 3.0 Mode,
>= 60MHz in USB 2.0 Mode
(could work lower than in reduced performance mode)
DWC_USB3_RAM_CLK_TO_BUS_CLK (or) gctl[7:6] – controls the clock select
ram_clk, ram_clk_gated/ ram_clk_gated_ram0
ram_clk_gated/ ram_clk_gated_ram0
RAM Clock Domain
(can be bus-clock, pipe3-clk, or pipe3-clk/2)
>= 125MHz in USB 3.0 mode, >= 60MHz in USB 2.0 mode
(controlled by GCTL[7:6])
R A M
pipe3_rx_pclk
suspend_clk
suspend_clk_en controls this
Div by 2
Div by 4
usb3_pipe_div_0
ref_clk
Ref Clock Domain
usb3_pipe_div_1
125MHz
mac_clk, mac_clk_gated
MAC Clock Domain
mac3 clock or mac2 clock muxed out depending up on USB 3.0 or USB 2.0 mode device operation. These are the transaction router and clock crossing FIFOs
DWC_USB3_PIPE_32BIT_ONLY (or) gusb3pipectl[16:15] controls this
dsts.connectspd Controls this
mac3_clk, mac3_clk_gated
pipe3_mx_rx_pclk
utmi_clk
ulpi_clk
wpc_clk
gusb2phycfg controls this
mac2_clk (same as wpc_clk) , mac2_clk_gated
U2 MAC/LINK (30/60MHz)
Synchronizers
U3 MAC/LINK (125MHz)
mac_clk, mac2_clk, and utmi/ulpi domains are synchronous
There are no synchronizers between these
(Same CTS clock insertion delays needed)
suspend_clk_en controls this
pipe3_tx_pclk
suspend_clk
pipe3_mx_tx_pclk
utmi_clk
ulpi_clk
U2 PIU (30/60MHz)
U3 PIU (125/250/500MHz)
mac_clk, mac3_clk, and pipe3_rx_pclk domains are synchronous
There are no synchronizers between these.
(Same CTS clock insertion delays needed)
either mac2_clk/mac_clk path (or) mac3_clk/mac_clk path is active at a give time depending up on to connection speed

2.4.4.1 CDC Files

The coreConsultant tool uses the following files for Spyglass CDC.

FileName	Description
configured_workspace/spyglass/manual.sgdc	These are the constraints pertaining to a given mode.
configured_workspace/spyglass/ports.sgdc	These are the list of I/O signals and their respective clocks.
configured_workspace/spyglass/spyglass_ design_specific_waivers.swl	These are DWC_usb3 design specific rule waivers. There are two sections in this waiver file: ■ Lint waivers ■ CDC waivers The reasons for each of the waivers are included as comments in the file.
configured_workspace/spyglass/spyglass_ engineering_council_waivers.swl	These are rules which Synopsys waives for its IPs.

2.4.4.2 CDC Path Debug Using Spyglass GUI

For CDC, it may become necessary to run Spyglass in Interactive mode in the configured workspace. To invoke the Spyglass GUI and run CDC, complete the following steps:

1. Go to *configured_workspace*/spyglass directory.
2. In run.scr file, replace `./sh.spyglass -batch` with `./sh.spyglass`.
3. Issue `./run.scr` to start the Spyglass GUI.
4. In the Spyglass GUI, the Goal Setup window opens by default.
5. Uncheck the lint_rtl option and click the **Selected Goal (s)** button.
6. After the CDC run is complete, the Analyze Results window displays the results.
7. Navigate to and select the relevant errors to open a schematic for analysis.
8. After analysis, the constraints need to be updated in manual.sgdc.

3 Simulating the Controller

This chapter provides information about the UVM-based Packaged Verification Environment (PVE) Testbench that is provided with the DWC_usb3 controller. The PVE provides a starting point for understanding how to use DesignWare Verification IP (VIP) and the configured DWC_usb3 controller together in your USB 3.0 verification environment.

This section covers the following topics:

- "PVE Testbench" on page 50
- "Running PVE Tests Using coreConsultant" on page 70
- "Running PVE Tests from Command Line" on page 77
- "Link-Up Sequence" on page 81

Note

- For description of various configuration parameters and signals mentioned in this section, refer to *DWC SuperSpeed USB 3.0 Controller Databook*. For information about register, refer to the *DWC SuperSpeed USB 3.0 Controller Programming Guide*.
- For unit-level testing, the coreConsultant tool automatically generates a testbench. You can also perform formal verification with the Synopsys Formality tool using coreConsultant.
- Refer to the .../doc/DWC_usb3_install.pdf document for tool versions and the licenses required for running tests.

3.1 PVE Testbench

The PVE testbench is implemented using UVM methodology. When you configure the DWC_usb3 controller, the PVE also gets configured based on the cC parameter values that you choose and various other derived parameters.

The PVE consists of the following modules:

- VIP test environment that includes USB 3.0 VIP and AMBA VIP
- SoC wrapper that includes configured DWC_usb3 controller (DUT) with instantiated PHY and RAMs
- Testcases with the supporting sequencers and drivers

The SoC wrapper is shipped along with the PVE environment. It has the DUT, Synopsys USB 3.0, and USB 2.0 PHYs and their interconnections.

The VIP test environment, on both the USB and the Application side, communicates to the SoC wrapper through SystemVerilog interfaces. You can use these interfaces to detach the VIPs and attach other applications (for example, a processor, or other third-party VIPs). On the application side, the VIP test environment communicates to the DUT inside the SoC wrapper through the AHB/AXI interfaces.

Similarly, the DWC_usb3 controller provides USB 3.0 (PIPE3) PHY and USB 2.0 (UTMI+ Level 3) PHY interfaces on the USB side. It interfaces to AHB or AXI on the application side.

The PVE demonstrates the following:

- How to connect DesignWare AMBA, USB 3.0 VIP, the SoC wrapper consisting of the DWC_usb3 synthesizable (RTL) component, and PHY in a UVM-based testbench.
- How to initialize and program (using Verilog VIP commands) the DWC_usb3 controller to perform basic operating functions.

3.1.1 PVE Testbench Details

Figure 3-1 on page 52 shows the example SoC test environment provided with the DWC_usb3 controller. The PVE testbench consists of the following blocks:

- SoC wrapper

  This module consists of the device under test – the configured DWC_usb3 controller interconnected with a PHY.

- USB 3.0 VIP

  The DW USB 3.0 VIP is configured to operate in the desired mode depending on the configuration. It is setup in the following different modes:

  - USB 3.0 serial PHY interface
  - USB 2.0 serial PHY interface

- AHB Master VIP

  The PVE uses a DW AMBA VIP as an AHB Bus Master to perform reads and writes to the slaves in the system and access the CSRs of the controller.

- AHB Slave VIP

  The PVE uses a DW AMBA VIP as an AHB Bus Slave that has system memory where data resides. The writes to the memory happen through back-door commands and also through actual AHB bus transactions.

- AXI Master VIP

  The PVE uses a DW AMBA VIP as an AXI Bus Master to perform reads and writes to the slaves present in the system and also access the CSRs of the controller.

- AXI Slave VIP

  The PVE uses a DW AMBA VIP as an AXI Bus Slave that has system memory where data resides. The writes to memory happen through back-door commands and also through actual AXI bus transactions.

- PHY Models

  The PVE uses the following three PHYs models that are provided as deliverables:

  - Synopsys USB 3.0 PHY model

    This PHY runs USB 3.0 serial tests. It provides the pipe3_tx_pclk required for tests. It connects to the PIPE3 interface to the controller on one side and the USB 3.0 serial pins to the VIP on the other side.

    You can link any third-party USB 3.0 PHY into the testbench by using your own USB 3.0 PHY file list (usb3phy_gtech.f). This can be specified in the coreConsultant GUI under "Setup and Run Simulation" --> "Testbench" tab.

  - Synopsys USB 2.0 GTECH PHY

    This PHY runs HS/FS/LS tests in non-HSIC mode. It provides the utmi_clk required for non-HSIC tests at all speeds. It supports both UTMI+ and the ULPI interfaces. The ULPI interface needs an ULPI2UTMI adapter.

    You can link any third-party USB 2.0 PHY into the testbench by using your own USB 2.0 PHY file list (dwc_usb20_phy_1p_ms_otg0_ns_inst.f). This can be specified in the coreConsultant GUI under “Setup and Run Simulation” > “Testbench” tab.

  - Synopsys USB 2.0 HSIC PHY model

    This PHY runs HS tests in HSIC mode. It provides the utmi_clk to the controller when HSIC-enabled HS tests are run. It is instantiated only when HSIC is enabled (that is, when you select “Yes” for the query “Enable HSIC Support for USB 2.0 Ports?” in the “PHY Config” tab of the “Specify Configuration” activity in coreConsultant, (DWC_USB3_ENABLE_HSIC=1)).

    You can link any third-party USB 2.0 PHY into the testbench by using your own USB 2.0 PHY file list (DWC_usb3_hsic_phy_gate.f). This can be specified in the coreConsultant GUI under “Setup and Run Simulation” > “Testbench” tab.

The main testbench control is a simple Verilog structural file. The primary objective of the tests in the PVE is to show end-to-end data flow on different endpoints. The test stimulus is written in the form of single command stream that is controlling the VIP in a sequential fashion.

Figure 3-1 Block Diagram of Top-Level Device/Host/DRD PVE Testbench

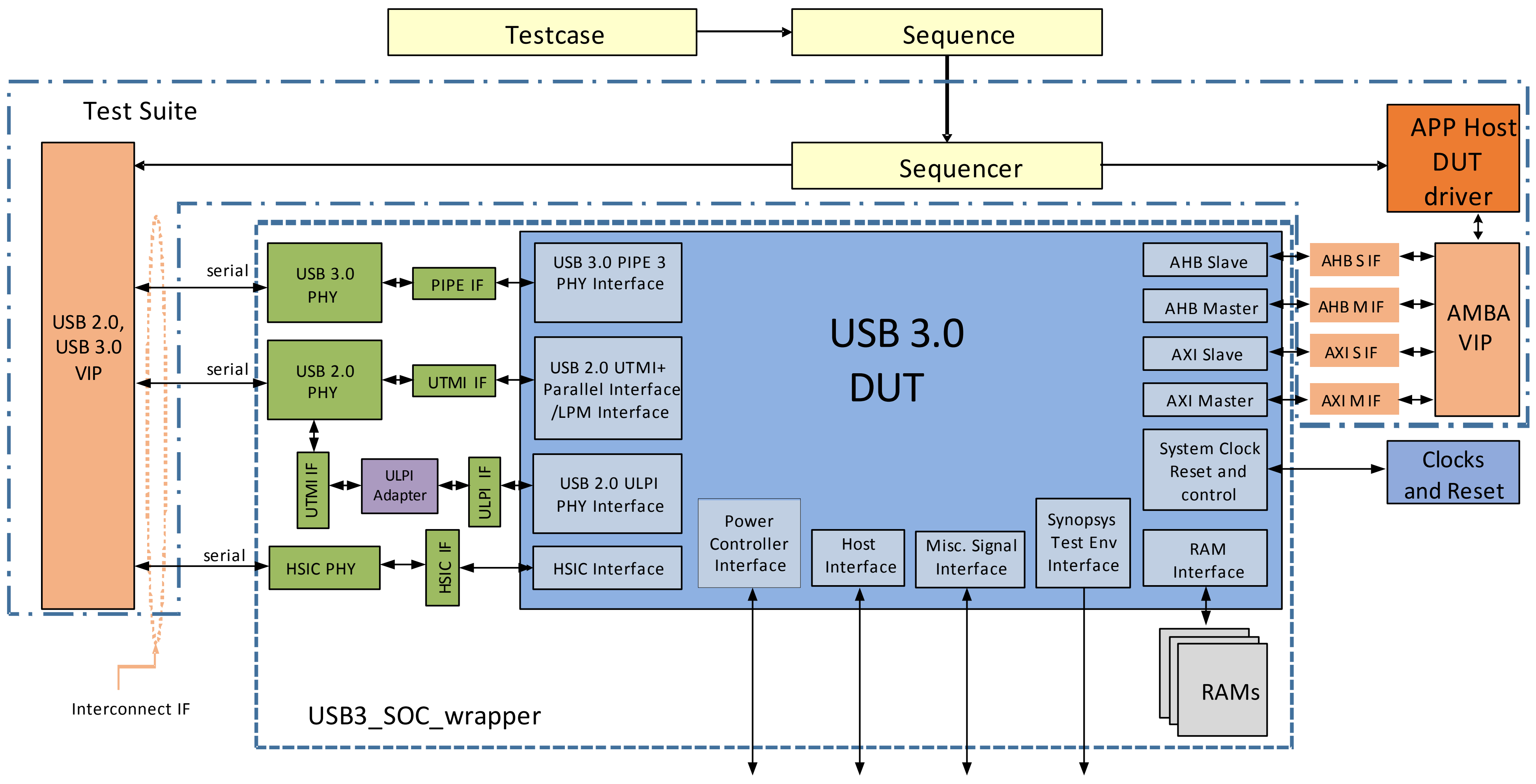
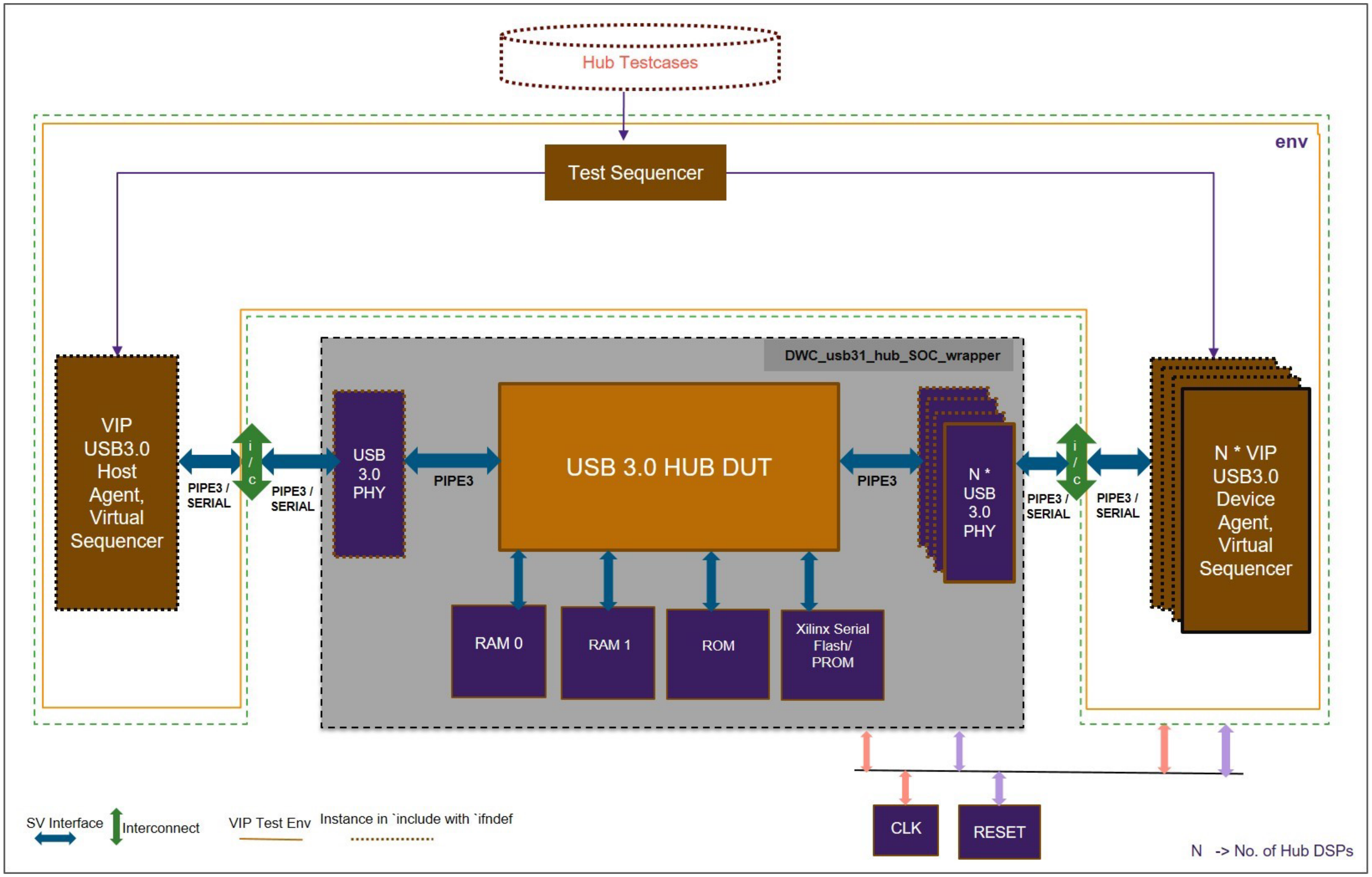


Figure 3-2 Block Diagram of Top-Level Hub PVE Testbench



3.1.2 PVE Testbench Files

PVE testbench files and testcases are located in the following directories:

- *workspace*/sim/Soc_sim/pve/com
- *workspace*/sim/Soc_sim/pve/device
- *workspace*/sim/Soc_sim/pve/device/tests
- *workspace*/sim/Soc_sim/pve/host
- *workspace*/sim/Soc_sim/pve/host/tests
- *workspace*/sim/Soc_sim/pve/hub
- *workspace*/sim/Soc_sim/pve/hub/tests

Table 3-1 describes the various files shipped in the PVE test environment.

Table 3-1 PVE Source Files

File and Locations	Description
.../com/	
test_top.v	Top level of the PVE testbench
tb_variables.v	Declare and initialize global variables and set dump options
tb_defines.v	Defines for the 3.0 PVE
pve_external_testlist.tcl	Contains testcases for external regression
pve_internal_testlist.tcl	Contains testcases for internal regression
pve_iip_dut_dwc_usb30_test_suite_configuration.sv	User test suite configuration file
DWC_usb3_xprop_config_file.cfg	Xprop configuration file
DWC_usb3.upf	UPF Power Intent for the USB3.0 Controller
DWC_usb3_SOC_wrapper.v	SoC wrapper file that interconnects the DUT with the PHY
DWC_usb3_clocks.v	Generation of all clocks required in the testbench
DWC_usb3_resets.v	Generation of resets used in the testbench
DWC_usb3_SOC_wrapper_inst.v	Instantiation of the SoC wrapper into the testbench
DWC_usb3_SOC_wrapper_xz_check.inc	has the x/z check for all the SoC wrapper top level pins
DWC_usb3_TE_components.v	Rest of test environment components and uvm_config_db for various interfaces
DWC_usb3_app_if.v	The interface instantiations to connect to the UVM components to the external HDL wires for the application side
DWC_usb3_ifs.v	interface definitions that are used in the testbench
runsim_vcs	VCS compile/run script
runsim_ncv	NC compile/run script

Table 3-1 PVE Source Files (Continued)

File and Locations	Description
runsim_mti	MTI compile/run script
DWC_usb3_vip_interconnect.v	The feed-through interconnect between the interface and the UVM components on the USB side
DWC_usb3_TE_utils.v	The PVE testbench utilities like timeout logic
DWC_usb3_ulpi_wrapper.v	The ULPI wrapper file
DWC_usb3_ss_gtech_phy.v	Instantiation of the GTECH USB 3.0 PHY
DWC_usb3_snps_ss_phy_init.v	The initialization snippet for the USB 3.0 PHY
USB20_PHY_GTECH_FD3.v	Verilog Simulation model for USB20_PHY_GTECH_FD3
usb3phy_gtech.f	The .f file or the SNPS GTECH PHY
ulpi_phy_files.f	The .f file for ULPI PHY
DWC_usb3_hsic_phy.v	The instantiation of the SNPS HSIC GTECH PHY
DWC_usb3_hsic_phy_gate.f	The .f file for the SNPS HSIC GTECH PHY
dwc_usb20_phy_1p_ms_otg0_ns_inst.f	The .f file for the SNPS 2.0 GTECH PHY
dwc_usb20_phy_1p_ms_otg0_ns_gtech.v	SNPS 2.0 GTECH PHY
dwc_usb20_phy_1p_ms_otg0_ns_inst.v	Instance of SNPS 2.0 GTECH PHY
DWC_usb3_hub_clocks.v	Generation of all clocks required in the hub testbench
DWC_usb3_hub_snps_ss_phy_init.v	The initialization snippet for the USB 3.0 PHY for use with hub
DWC_usb3_hub_SOC_wrapper_inst.v	Instantiation of the hub SoC wrapper into the testbench
DWC_usb3_hub_SOC_wrapper.v	SoC wrapper file that instantiates the DUT as hub and interconnects the PHYs with the DUT
DWC_usb3_hub_TE_components.v	Rest of test environment components and uvm_config_db for various interfaces of hub
DWC_usb3_hub_vip_interconnect.v	The feed-through interconnect between the PIPE/Serial interface and the UVM components on the USB side for hub
DWC_usb3_ss_hub_phy.f	The .f for the HUB USB 3.0 PHY
DWC_usb3_ss_hub_phy_inst.v	Instance of HUB USB 3.0 PHY
svt_usb_20_na_30_ss_pve_ltssm_state_transition_virtual_sequence.sv	PVE LTSSM Transition Sequences
.../device/	
README	README text file to show how to run the scripts
.../device/tests/	The testcases with their class
svt_usb_test_suite_user_device_base_test_build_phase.sv	Configuration of the VIP parameters to match the RTL parameters

Table 3-1 PVE Source Files (Continued)

File and Locations	Description
svt_usb_test_suite_user_device_configuration_settings.sv	Allows you to change VIP timing parameters related to USB
device_external_testlist.inc	Includes all the device tests that are to be the part of external test list
.../host/	
README	README text file to show how to run the scripts
.../host/tests/	The testcases with their class
svt_usb_test_suite_user_host_configuration_settings.sv	Allows you to change VIP timing parameters related to USB
host_external_testlist.inc	Includes all the host tests that are to be the part of external test list
.../hub/	
README	README text file to show how to run the scripts
.../hub/tests/	
svt_usb_test_suite_user_device_configuration_settings.sv	Allows you to change VIP timing parameters related to USB protocol
hub_external_testlist.inc	Includes all the hub tests that are to be the part of external test list

3.1.3 List of PVE Tests

Table 3-2 on page 58, Table 3-3 on page 63, and Table 3-4 on page 68 list the device-mode, host-mode, and hub-mode tests provided with the PVE Testbench.

Note

- You can find the enabling condition for any test by right clicking on the test in the coreConsultant GUI. Based on the cC parameters you chose, the test is enabled if the conditions are met.
- All USB 3.0 tests use the USB 3.0 PHY model. This PHY is always present in all tests to provide pipe_clk.
- In non-HSIC mode, all HS/FS/LS tests use the USB 2.0 GTECH model. This PHY is always present in non-HSIC mode in all tests to provide the utmi_clk.
- In HSIC mode, all HS tests use USB 2.0 HSIC PHY. This PHY is always present in HSIC mode in all tests to provide the utmi_clk.

Device-Mode Tests

Table 3-2 List of Device-Mode Tests Provided in the SoC Integration Example

Tests	Speed	Description	Usage
SuperSpeed Tests			
usb_20_na_30_ss_bulk_in	SS	A single SS BULK_IN_TRANSFER, consisting of one or more IN transactions, is queued to the SS VIP Host. In each IN Transaction, DUT responds to ACK packet received from VIP with either a DATA packet or a NAK packet.	Recommended for SoC porting
usb_20_na_30_ss_bulk_out_with_payload_size_256k	SS	A single SS BULK_OUT_TRANSFER, with 256K payload, is queued to the SS VIP Host. DUT responds to OUT DATA packet received from VIP with either a ACK packet or a NAK packet.	Recommended for SoC porting
usb_20_na_30_ss_control_read	SS	A single SS CONTROL_TRANSFER with DEVICE_TO_HOST bmrequest type, consisting of one or more IN transactions, is queued to the SS VIP Host. In each IN Transaction, DUT responds to ACK packet received from VIP with either a DATA packet or a NAK packet.	Recommended for SoC porting
usb_20_na_30_ss_control_write	SS	A single SS CONTROL_TRANSFER with HOST_TO_DEVICE bmrequest type, consisting of one or more OUT transactions, is queued to the SS VIP Host. In each OUT Transaction, DUT responds to DATA packet received from VIP with either a ACK packet or a NAK packet.	Recommended for SoC porting
usb_20_na_30_ss_interrupt_in	SS	A single SS INTERRUPT_IN_TRANSFER, consisting of one or more IN transactions, is queued to the SS VIP Host. In each IN Transaction, DUT responds to ACK packet received from VIP with either a DATA packet or a NAK packet.	Optional for SoC porting

Table 3-2 List of Device-Mode Tests Provided in the SoC Integration Example (Continued)

Tests	Speed	Description	Usage
usb_20_na_30_ss_interrupt_out	SS	A single SS INTERRUPT_OUT_TRANSFER, consisting of one or more OUT transactions, is queued to the SS VIP Host. In each OUT Transaction, DUT responds to DATA packet received from VIP with either an ACK packet or a NAK packet.	Optional for SoC porting
usb_20_na_30_ss_isochronous_in	SS	A single SS ISOC_IN_TRANSFER with ISOC mult 0, consisting of one or more IN transactions, is queued to the SS VIP Host. In each IN Transaction, DUT responds to ACK packet received from VIP with either a DATA packet or ignores the ACK packet.	Optional for SoC porting
usb_20_na_30_ss_isochronous_out	SS	A single SS ISOC_OUT_TRANSFER with ISOC mult 0, consisting of one or more OUT transactions, is queued to the SS VIP Host.	Optional for SoC porting
usb_20_na_30_ss_ltssm_u0_to_u1_to_u0	SS	Completes the link training in SS, VIP Host Initiates the U0 to U1 to U0 link transition sequence, to which DUT also responds with corresponding link transitions.	Recommended for SoC porting
usb_20_na_30_ss_device_framework_configure_state_set_feature_request_u1enable	SS	Completes the link training in SS, DUT device initiates U0 to U1 to U0, VIP host responds to DUT link transition request	Recommended for SoC porting
usb_20_na_30_ss_ltssm_u0_to_u2_to_u0	SS	Completes the link training in SS, VIP Host Initiates the U0 to U2 to U0 link transition sequence, DUT responds with corresponding link transitions.	Recommended for SoC porting
usb_20_na_30_ss_device_framework_configure_state_set_feature_request_u2enable	SS	Completes the link training in SS, DUT device initiates U0 to U2 to U0, VIP host responds to DUT link transition request	Recommended for SoC porting
usb_20_na_30_ss_ltssm_u0_to_u3_to_u0	SS	Completes the link training in SS, VIP Host Initiates the U0 to U3 to U0 link transition sequence, to which DUT also responds with corresponding link transitions.	Recommended for SoC porting

Table 3-2 List of Device-Mode Tests Provided in the SoC Integration Example (Continued)

Tests	Speed	Description	Usage
usb_20_na_30_ss_bulk_in_len_low_mid_high	SS	Multiple SS BULK_IN_TRANSFERs, consisting of various payload sizes ranging from 0 to maximum possible, are queued to the SS VIP Host. In each IN Transaction, DUT responds to IN Token packet received from VIP with either a DATA packet or a NAK packet.	Recommended for SoC porting
usb_20_na_30_ss_bulk_out_in_simultaneous	SS	VIP USB Host executes Bulk Out and Bulk In simultaneous transfer by sending DPs & TP_ACKs requesting for random number of DPs by the Device DUT.	Optional for SoC porting
usb_20_na_30_ss_power_management_save_and_restore_xfer	SS	After completing Link training in SS, VIP Host initiates Hibernation through U3 state transition, host VIP initiates the Hibernation exit and does an SS transfer.	Optional for SoC porting
High-Speed and Full-Speed Tests			
usb_20_hs_fs_30_na_bulk_in	HS/FS	A single HS/FS BULK In transfer is queued to the HS HOST, DUT responds to the IN token packet received from VIP with a DATA packet or a NAK packet.	Recommended for SoC porting
usb_20_hs_fs_30_na_bulk_out	HS/FS	A single HS/FS BULK_OUT_TRANSFER, with 1 or more OUT transactions. DUT responds to OUT DATA packet received from VIP with either a ACK packet or a NAK packet	Recommended for SoC porting
usb_20_hs_fs_ls_30_na_intr_in	HS/FS	A single HS/FS Interrupt In transfer is queued to the HS HOST, DUT responds to the IN token packet received from VIP with a DATA packet or a NAK packet.	Optional for SoC porting
usb_20_hs_fs_ls_30_na_intr_out	HS/FS	A single HS/FS Interrupt Out transfer is queued to the HS HOST, DUT responds to the OUT DATA packet received from VIP with an ACK packet or NAK packet.	Optional for SoC porting
usb_20_hs_fs_ls_30_na_suspend_resume_via_host	HS/FS	Completes the HS/FS reset, VIP Host keeps the bus idle to enter suspend. Resume is initiated by the host, following which host does a transfer in HS/FS.	Recommended for SoC porting

Table 3-2 List of Device-Mode Tests Provided in the SoC Integration Example (Continued)

Tests	Speed	Description	Usage
usb_20_hs_fs_ls_30_na_suspend_resume_via_remote_wakeup	HS	Complete the HS reset, VIP Host keeps the bus idle to enter suspend. Remote wakeup is initiated by device, host completes the resume, following which host does a transfer in HS/FS.	Optional for SoC porting
usb_20_hs_fs_ls_30_na_l1_exit_via_device_remote_wakeup	HS/FS	Complete the HS/FS reset, VIP Host initiates the L1 suspend. Remote wakeup is initiated by device, host completes the resume, following which host does a transfer in HS/FS.	Optional for SoC porting
usb_20_hs_fs_ls_30_na_lpm_errata_l1_exit_via_device_remote_wakeup	HS/FS	Complete the HS/FS reset, VIP Host initiates the LPM Errata L1 suspend. Remote wakeup is initiated by device, host completes the resume, following which host does a transfer in HS/FS.	Optional for SoC porting
usb_20_hs_fs_ls_30_na_suspend_resume_via_connect_disconnect	FS	Complete the FS reset, VIP Host keeps the bus idle to enter suspend. Device DUT is disconnected and connected, following which host does a transfer in FS.	Optional for SoC porting
usb_20_hs_fs_ls_30_na_ctrl_in	HS/FS	A single HS/FS CONTROL In transfer is queued to the HS HOST, DUT responds to the IN token packet received from VIP with a DATA packet or a NAK packet.	Recommended for SoC porting
usb_20_hs_fs_ls_30_na_ctrl_out	HS/FS	A single HS/FS CONTROL OUT TRANSFER, with 1 or more OUT transactions. DUT responds to OUT DATA packet received from VIP with either an ACK packet or a NAK packet	Recommended for SoC porting
usb_20_hs_fs_30_na_isoc_in	HS	A single HS ISOC In transfer is queued to the HS HOST, DUT responds to the IN token packet received from VIP with a DATA packet.	Optional for SoC porting
usb_20_hs_fs_30_na_isoc_out	HS	A single HS ISOC OUT TRANSFER, with 1 or more OUT transactions, are queued to VIP HOST.	Optional for SoC porting

Table 3-2 List of Device-Mode Tests Provided in the SoC Integration Example (Continued)

Tests	Speed	Description	Usage
usb_20_hs_fs_ls_30_na_device_framework_configure_state_get_config_desc_request	HS/FS	HS VIP Host initiates GET_DESCRIPTOR request on the configured DEVICE by setting Descrpitor type as Configuration Descriptor and wValue, wIndex and wLength fields in HS/FS.	Optional for SoC porting

Host-Mode Tests

Table 3-3 List of Host-Mode Tests Provided in the SoC Integration Example

Tests	Speed	Description	Usage
SuperSpeed Tests			
host_usb_20_na_30_ss_bulk_in	SS	A single SS BULK_IN_TRANSFER, consisting of one or more IN transactions is initiated from DUT host. In each IN Transaction, VIP responds to IN Token packet received from DUT with either a DATA packet or a NAK packet.	Recommended for SoC porting
host_usb_20_na_30_ss_bulk_out	SS	A single SS BULK_OUT_TRANSFER, is initiated from SS DUT Host. VIP responds to OUT DATA packet received from DUT with either a ACK packet or a NAK packet.	Recommended for SoC porting
host_usb_20_na_30_ss_interrupt_in	SS	A single SS INTERRUPT_IN_TRANSFER, consisting of one or more IN transactions is initiated from DUT host. In each IN Transaction, VIP responds to IN Token packet received from DUT with either a DATA packet or a NAK packet.	Optional for SoC porting
host_usb_20_na_30_ss_interrupt_out	SS	A single SS INTERRUPT_OUT_TRANSFER, is initiated from SS DUT Host. VIP responds to OUT DATA packet received from DUT with either a ACK packet or a NAK packet.	Optional for SoC porting
host_usb_20_na_30_ss_control_read	SS	A single SS CONTROL Read Transfer, consisting of one or more IN transactions is initiated from DUT host. In each IN Transaction, VIP responds to IN Token packet received from DUT with either a DATA packet or a NAK packet.	Recommended for SoC porting
host_usb_20_na_30_ss_control_write	SS	A single SS CONTROL Write/OUT Transfer, is initiated from SS DUT Host. VIP responds to OUT DATA packet received from DUT with either a ACK packet or a NAK packet.	Recommended for SoC porting

Table 3-3 List of Host-Mode Tests Provided in the SoC Integration Example (Continued)

Tests	Speed	Description	Usage
host_usb_20_na_30_ss_xfer_pm_sus_res_xfer	SS	Complete the SS Link training and transfer, SS HOST DUT initiates the Hibernation though U3 entry and exit. Initiate a transfer after back in U0.	Optional for SoC porting
host_usb_20_hs_30_ss_concurrent_hs_bulk_in_ss_bulk_in	SS	HOST DUT does the concurrent HS BULK IN and SS BULK IN transfers	Optional for SoC porting
host_usb_20_hs_30_ss_concurrent_hs_control_read_ss_bulk_in	SS	HOST DUT does the concurrent HS CONTROL READ and SS BULK IN transfers	Optional for SoC porting
host_usb_20_hs_30_ss_concurrent_hs_interrupt_in_ss_bulk_out	SS	HOST DUT does the concurrent HS INTERRUPT IN and SS BULK OUT transfers	Optional for SoC porting
host_usb_20_hs_30_ss_concurrent_hs_isochronous_in_ss_bulk_in	SS	HOST DUT does the concurrent HS ISOC IN and SS BULK IN transfers	Optional for SoC porting
host_usb_20_na_30_ss_isochronous_in	SS	A single SS ISOC_IN_TRANSFER with ISOC mult 0, consisting of one or more IN transactions, is initiated from SS DUT host. In each IN Transaction, SS VIP responds to IN Token packet received from DUT with either a DATA packet or ignores the ACK packet.	Optional for SoC porting
host_usb_20_na_30_ss_isochronous_out	SS	A single SS ISOC_OUT_TRANSFER with ISOC mult 0, consisting of one or more OUT transactions, is initiated from SS DUT host.	Optional for SoC porting
host_usb_20_na_30_ss_ltssm_u0_to_u1_to_u0	SS	Complete the link training in SS, VIP Device Initiates the U0 to U1 to U0 link transition sequence, to which DUT also responds with corresponding link transitions.	Recommended for SoC porting
host_usb_20_na_30_ss_ltssm_u0_to_u1_to_u0_dut_initiated_u1entry	SS	Complete the link training in SS, HOST DUT initiates the U0 to U1 to U0 link transition sequence, to which VIP device also responds with corresponding link transitions.	Recommended for SoC porting
host_usb_20_na_30_ss_ltssm_u0_to_u2_to_u0	SS	Complete the link training in SS, VIP Host Initiates the U0 to U2 to U0 link transition sequence, DUT responds with corresponding link transitions.	Recommended for SoC porting

Table 3-3 List of Host-Mode Tests Provided in the SoC Integration Example (Continued)

Tests	Speed	Description	Usage
host_usb_20_na_30_ss_ltssm_u0_to_u2_to_u0_dut_initiated_u2entry	SS	DUT Host initiated entry and VIP Device exit U0-U2-U0 SS test	Recommended for SoC porting
host_usb_20_na_30_ss_ltssm_u0_to_u3_to_u0_dut_initiated_u3entry	SS	Complete the link training in SS, HOST DUT initiates the U0 to U3 to U0 link transition sequence, to which VIP device also responds with corresponding link transitions.	Recommended for SoC porting
host_usb_20_na_30_ss_mult_ep_mix_transfers	SS	Various transfers are queued in Host SS DUT for all available endpoints, VIP device responds to the received transfers	Optional for SoC porting
host_usb_20_na_30_ss_ltssm_ss_inactive_quiet_to_disconnect_detect_to_rx_detect	SS	Complete the link training in SS, VIP Device Initiates a disconnect and connect followed by DUT initiated SS transfer	Recommended for SoC porting
host_usb_20_na_30_ss_ltssm_u0_to_u3_to_u0	SS	Complete the link training in SS, VIP Device Initiates the U0 to U3 to U0 link transition sequence, to which DUT also responds with corresponding link transitions.	Optional for SoC porting
host_usb_20_na_30_ss_bulk_out_in_multi_stream	SS	SS DUT HOST initiates multiple BULK In and OUT transfers simultenously on different eps of the Device, VIP Device responds to the the Multiple Streams on each bulk In and OUT eps as per specification	Optional for SoC porting
host_usb_20_na_30_ss_u0_to_u1_to_u0_ss_bulk_in	SS	HOST DUT working with multi port setup, SS port does BULK IN transfer, on other SS Port VIP Device Initiates the U0 to U1 to U0 link transition sequence, to which DUT also responds with corresponding link transitions.	Optional for SoC porting
host_usb_20_na_30_ss_u0_to_u2_to_u0_ss_control_in	SS	HOST DUT working with multi port setup, SS port does CONTROL IN transfer, on other SS Port VIP Device Initiates the U0 to U2 to U0 link transition sequence, to which DUT also responds with corresponding link transitions.	Optional for SoC porting

Table 3-3 List of Host-Mode Tests Provided in the SoC Integration Example (Continued)

Tests	Speed	Description	Usage
host_usb_20_na_30_ss_disconnect_connect_ss_bulk_in	SS	HOST DUT working with multi port setup, SS port does BULK IN transfer, on other SS Port VIP Device initiates a disconnect and connect and a transfer	Optional for SoC porting
host_usb_20_na_30_ss_disconnect_connect_ss_control_in	SS	HOST DUT working with multi port setup,SS port does CONTROL IN transfer, on other SS Port VIP Device initiates a disconnect and connect and a transfer	Optional for SoC porting
host_usb_20_na_30_ss_bulk_out_in_simultaneous	SS	DUT USB Host executes Bulk Out and Bulk In simultaneous transfer by sending DPs & TP_ACK's requesting for random number of DP's from VIP USB Device	Optional for SoC porting
High-Speed, Full-Speed, and Low-Speed Tests			
host_usb_20_hs_fs_ls_30_na_ctrl_out	HS/FS/ LS	A single HS/FS/LS CONTROL OUT TRANSFER, with 1 or more OUT transactions. DUT sends the OUT DATA packets to which VIP responds with either a ACK packet or a NAK packet	Recommended for SoC porting
host_usb_20_hs_fs_ls_30_na_ctrl_in	HS/FS/ LS	A single HS/FS/LS CONTROL IN TRANSFER, with 1 or more IN transactions. DUT sends the IN token packet to which VIP responds with either a DATA packet or a NAK packet	Recommended for SoC porting
host_usb_20_hs_fs_ls_30_na_intr_in	HS/FS/ LS	A single HS/FS/LS INTERRUPT IN TRANSFER, with 1 or more IN transactions. DUT sends the IN token packet to which VIP responds with either a DATA packet or a NAK packet	Optional for SoC porting
host_usb_20_hs_fs_ls_30_na_mult_ep_mix_transfers	HS/LS	Various transfers are queued in Host HS/LS DUT for all available endpoints, VIP device responds to the received transfers	Optional for SoC porting
host_usb_20_hs_fs_ls_30_na_intr_out	HS/FS/ LS	DUT HS/FS/LS host initiates Interrupt Out transfer is queued to the HS HOST, DUT responds to the OUT DATA packet received from VIP with a ACK packet or NAK packet.	Optional for SoC porting

Table 3-3 List of Host-Mode Tests Provided in the SoC Integration Example (Continued)

Tests	Speed	Description	Usage
host_usb_20_hs_fs_30_na_bulk_in	HS/FS	A single HS/FS BULK In transfer is initiated from DUT HOST, Device VIP responds to the IN token packet received from DUT with a DATA packet or NAK packet.	Recommended for SoC porting
host_usb_20_hs_fs_30_na_bulk_out	HS/FS	A single HS/FS BULK_OUT_TRANSFER, with 1 or more OUT transactions. DUT responds to OUT DATA packet received from VIP with either a ACK packet or a NAK packet	Recommended for SoC porting
host_usb_20_hs_fs_ls_30_na_xfer_pm_sus_res_xfer	HS/FS/ LS	Complete the HS/FS/LS reset, initiate a transfer, enter hibernation through Suspend and exit via resume	Optional for SoC porting
host_usb_20_hs_fs_30_na_isoc_in	HS	A single HS ISOC IN transfer is initiated from DUT HOST, Device VIP responds to the IN token packet received from DUT with a DATA packet.	Optional for SoC porting
host_usb_20_hs_fs_30_na_isoc_out	HS/FS	A single HS/FS ISOCHRONOUS_OUT_TRANSFER, consisting of one or more OUT transactions, is queued to the Host DUT.	Optional for SoC porting
host_usb_20_hs_fs_ls_30_na_suspend_resume_via_remote_wakeup	HS/FS/ LS	Complete the HS/FS/LS reset, DUT host enters the suspend. VIP Device does remote wakeup, host completes the resume, following which host does a transfer in HS/FS/LS.	Recommended for SoC porting
host_usb_20_hs_fs_ls_30_na_suspend_resume_via_host	HS/FS/ LS	Complete the HS/FS/LS reset, DUT host enters the suspend. HOST DUT does the resume and transfer	Recommended for SoC porting
host_usb_20_hs_fs_ls_30_na_l1_exit_via_host	HS/FS/ LS	Complete the HS/FS/LS reset, DUT host enters the L1 suspend. HOST DUT does the L1 resume.	Optional for SoC porting
host_usb_20_hs_fs_ls_30_na_suspend_resume_via_connect_disconnect	HS/FS/ LS	Complete the HS/FS/LS reset, DUT Host enters suspend. VIP Device is disconnected and connected, following which DUT host does a transfer in HS/FS/LS	Optional for SoC porting

Hub-Mode Tests

Table 3-4 List of Hub-Mode Tests Provided in the SoC Integration Example

Tests	Speed	Description	Usage
SuperSpeed Tests			
hub_h_ss_d_ss_bulk_in_bulk_out	SS	Bulk IN and Bulk OUT transfers scheduled by SS Host VIP to multiple SS Device VIPs through the Hub in Serial mode. Bulk transfer payload size is 8k.	Recommended for SoC porting
hub_h_ss_d_ss_interrupt_in_interrupt_out	SS	Interrupt IN and OUT transfers scheduled by SS Host VIP to multiple SS Device VIPs through the Hub in Serial mode. Interrupt transfer payload size is 6k. Also include Interrupt IN transfers from Host VIP to Hub upstream port.	Recommended for SoC porting
hub_h_ss_d_ss_iso_in_iso_out	SS	Isoc IN and OUT transfers scheduled by SS Host VIP to multiple SS Device VIPs through the Hub in Serial mode. Isoc transfer payload size is 8k	Recommended for SoC porting
hub_h_ss_d_ss_ltssm_u0_to_u1_to_u0	SS	Low power U1 entry and exit for all Hub ports in Serial mode. Hub downstream ports enter U1 on inactivity timeout. Hub then initiates U1 entry on upstream port. Host VIP initiated U1 exit	Recommended for SoC porting
hub_h_ss_d_ss_ltssm_u0_to_u2_to_u0	SS	Low power U2 entry and exit for all Hub ports in Serial mode. Hub downstream ports enter U2 on inactivity timeout. Hub then initiates U2 entry on upstream port. Device VIP initiated U2 exit	Recommended for SoC porting
hub_h_ss_d_ss_ltssm_u0_to_u3_to_u0	SS	Low power U3 entry and exit for all Hub ports in Serial mode. Host VIP initiated U3 entry. Device VIP initiated U3 exit	Recommended for SoC porting
hub_h_ss_d_ss_bulk_out_in_concurrent	SS	Bulk IN and OUT concurrent transfers scheduled by SS Host VIP to SS Device VIPs through the Hub in Serial mode. Bulk transfer payload size is 8k	Recommended for SoC porting

Table 3-4 List of Hub-Mode Tests Provided in the SoC Integration Example (Continued)

Tests	Speed	Description	Usage
hub_h_ss_d_ss_iso_out_in_concurrent	SS	Isochronous IN and OUT concurrent transfers scheduled by SS Host VIP to SS Device VIPs through the Hub in Serial mode. Isoc transfer payload size is 8k	Recommended for SoC porting
hub_h_ss_d_ss_interrupt_out_in_concurrent	SS	Interrupt IN and OUT concurrent transfers scheduled by SS Host VIP to SS Device VIPs through the Hub in Serial mode. Interrupt transfer payload size is 6k. Also includes Interrupt IN transfers from Host VIP to Hub upstream port	Recommended for SoC porting
hub_h_ss_d_ss_bulk_intr_iso_out_in_concurrent	SS	Bulk, Interrupt, Isochronous IN and OUT concurrent transfers scheduled by SS Host VIP to SS Device VIPs through the Hub in Serial mode.	Recommended for SoC porting

3.2 Running PVE Tests Using coreConsultant

This section shows you how to simulate the DWC_usb3 controller in the coreConsultant environment. You can simulate the controller RTL in the supplied PVE testbench, which can be used as a starting point for integrating the controller, Synopsys PHY, and Synopsys Verification IP (VIP) into your system-level test environment. The primary objective of the tests in the PVE is to show end-to-end data flow on different endpoints.

Note You can use the PVE and the VIP examples as a basis for a system testbench.

The steps involved in simulating the DWC_usb3 controller are:

- "Running the Simulation"
- "Checking Simulation Status and Results" on page 75

Attention

- To run simulations, you must have installed the Discovery Verification IP for USB 3.0 and AMBA Verification IP.
- If coreConsultant is open when you install the USB 3.0 VIP or AMBA VIP, then to run simulations with the newly installed VIPs, you must exit coreConsultant and open it again.
- For instructions to install the VIPs, see section "Downloading and Installing Verification IP Components" in the *DWC SuperSpeed USB 3.0 Controller Installation Guide.*
- To run simulations, set the VRO_CACHE_DIR environment variable to <workspace>/scratch directory.

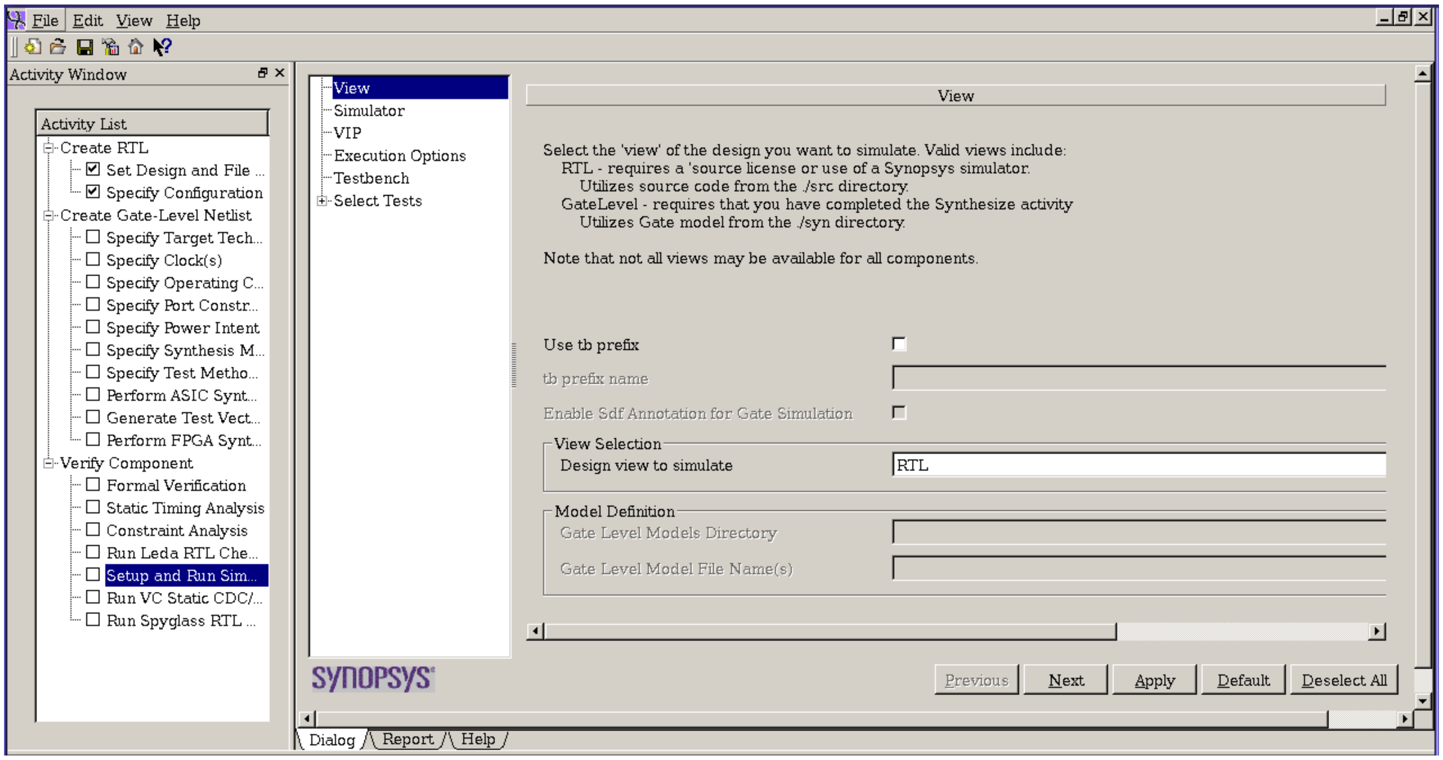
3.2.1 Running the Simulation

The Simulate activity (Figure 3-3 on page 71) 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 box.

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 28). You can edit your environment settings by modifying your UNIX environment setup or through the **Edit > Tool Installation Roots** menu (make sure to set the 64-Bit check box if you are using 64-bit tools).

Figure 3-3 Setup and Run Simulation Activity



View

Select the 'view' of the design you want to simulate. Valid views include:
RTL - requires a 'source license or use of a Synopsys simulator.
Utilizes source code from the ./src directory.
GateLevel - requires that you have completed the Synthesize activity
Utilizes Gate model from the ./syn directory.

Note that not all views may be available for all components.

Use tb prefix

tb prefix name

Enable Sdf Annotation for Gate Simulation

View Selection
Design view to simulate: RTL

Model Definition
Gate Level Models Directory
Gate Level Model File Name(s)

Under “Testbench” option in “Setup and Run Simulation” activity (Figure 3-4 on page 72), specify the following:

1. Specify the location of the file list for the SuperSpeed PHY model. In addition, specify the location of file list for the USB 2.0 PHY that you want to use in simulations. If you want to run high-speed tests in HSIC mode, specify the location of file list for the USB 2.0 HSIC PHY. Default for all file lists points to the Synopsys delivered models.

   - SuperSpeed PHY Verilog File List used for Simulation

     This path points to the location of the file list for the SuperSpeed PHY that is to be used for simulations. When you specify this, all USB 3.0 serial tests use this PHY. By default, this file list points to the encrypted model of the Synopsys USB 3.0 PHY that is provided as a deliverable. If you need the unencrypted model, you need to purchase the Synopsys PHY RTL separately. If you want to make any changes to the PHY file list, or if you are using a third-party PHY, you must modify this usb3phy_gtech.f file to the physical directory of that PHY file list.

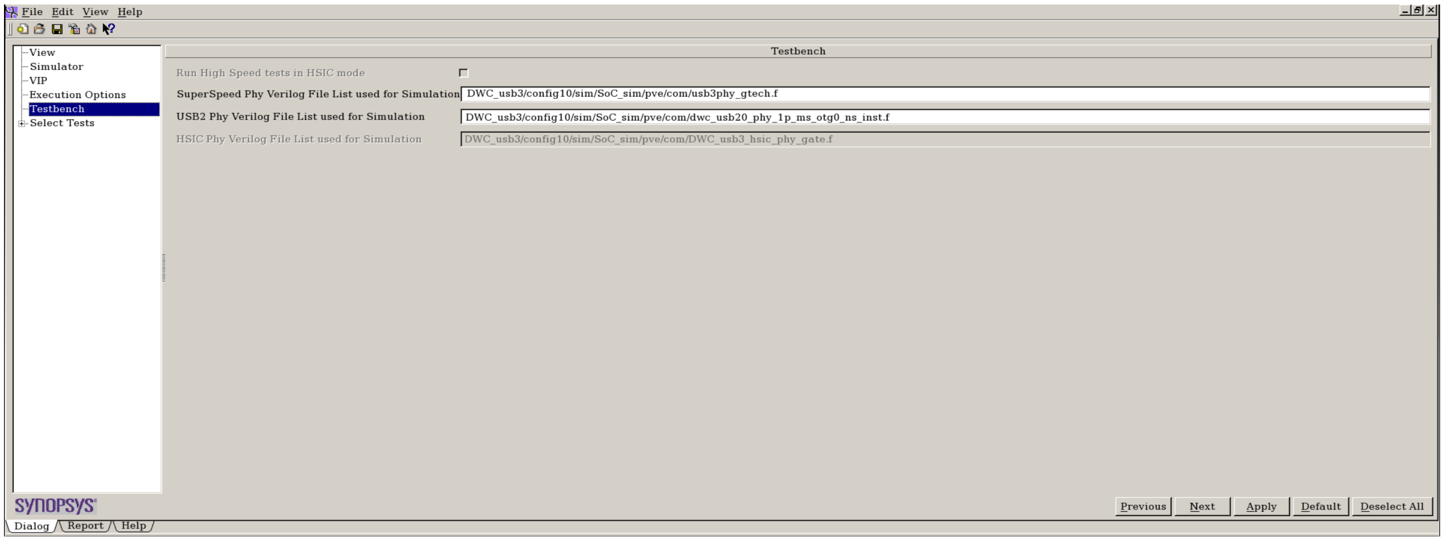
   - USB 2.0 PHY Verilog File List used for Simulation

     This path points to the location of the file list for the USB 2.0 PHY that is to be used for simulations. By default, this file list points to the Synopsys USB 2.0 GTECH PHY model that is provided as a deliverable. Note that the RTL is not part of the deliverables of this product. You need to purchase the Synopsys PHY RTL separately. If you want to make any changes to the installation path of the USB 2.0 PHY, or if you are using a third-party PHY, you must modify the path or content of this dwc_usb20_phy_1p_ms_otg0_ns_inst.f file to the physical directory of the PHY files.

- USB 2.0 HSIC PHY Verilog File List used for Simulation

  This path points to the location of the file list for the USB 2.0 HSIC PHY that is to be used for simulations. By default, this file list points to the Synopsys USB 2.0 HSIC GTECH PHY model that is provided as a deliverable. Note that the RTL is not part of the deliverables of this product. You need to purchase the Synopsys PHY RTL separately. If you want to make any changes to the installation path of the USB 2.0 HSIC PHY, or if you are using a third-party PHY, you must modify the path or content of this DWC_usb3_hsic_phy_gate.f file to the physical directory of the PHY files.

Figure 3-4 Specify Location of File Lists for PHY Models



Testbench

Run High Speed tests in HSIC mode	☐
SuperSpeed Phy Verilog File List used for Simulation	DWC_usb3/config10/sim/SoC_sim/pve/com/usb3phy_gtech.f
USB2 Phy Verilog File List used for Simulation	DWC_usb3/config10/sim/SoC_sim/pve/com/dwc_usb20_phy_1p_ms_otg0_ns_inst.f
HSIC Phy Verilog File List used for Simulation	DWC_usb3/config10/sim/SoC_sim/pve/com/DWC_usb3_hsic_phy_gate.f

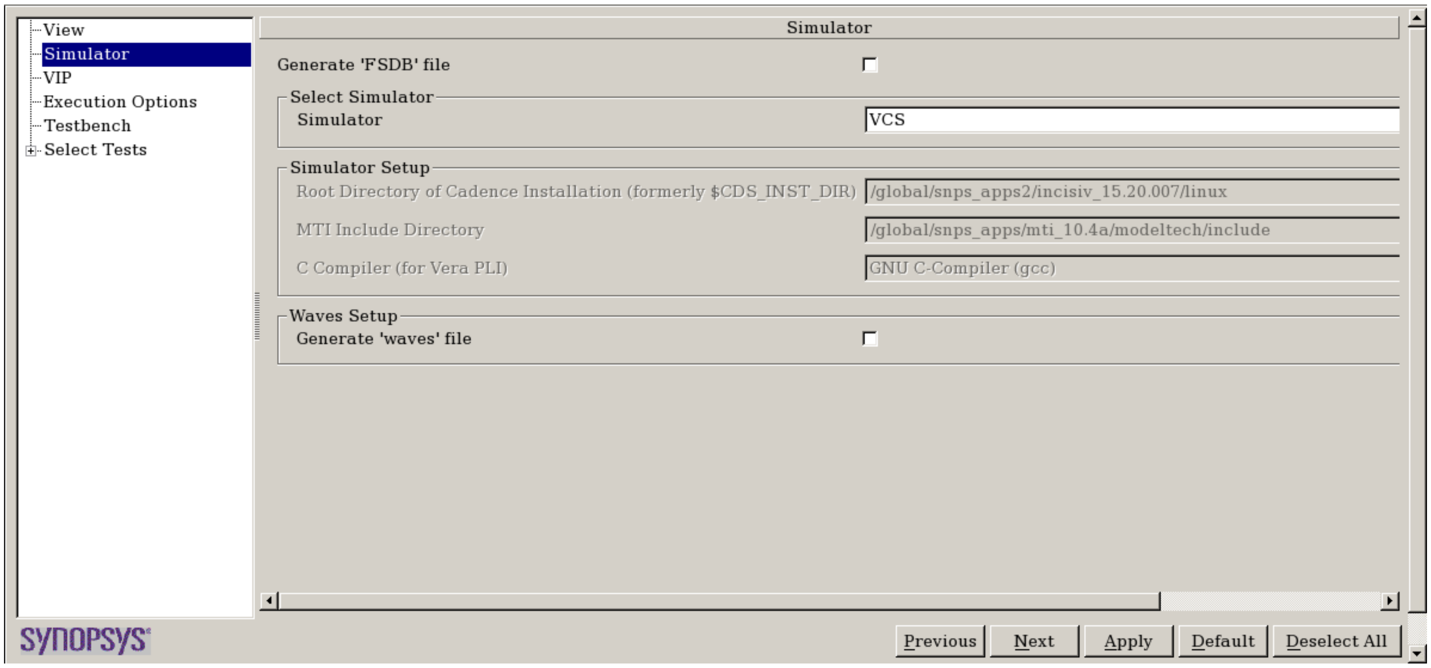
2. Specify whether you want to run high-speed tests in HSIC mode.

   When HSIC is enabled in the controller, the controller supports both HSIC and non-HSIC (standard USB 2.0 PHY) modes of operation. The HSIC mode is port/register selectable.

   When you select this option, all the HS tests run in HSIC mode with an HSIC PHY. The full-speed tests are disabled. When you do not select this option, the HS and FS tests are run in non-HSIC mode using a USB 2.0 PHY. The SS tests are not affected by this option.

Under the "Simulator" option in the "Setup and Run Simulations" activity, select VCS.

Figure 3-5 Supported Simulators



Generate 'FSDB' file ☐

Select Simulator
Simulator: VCS

Simulator Setup
Root Directory of Cadence Installation (formerly $CDS_INST_DIR): /global/snps_apps2/incisiv_15.20.007/linux
MTI Include Directory: /global/snps_apps/mti_10.4a/modeltech/include
C Compiler (for Vera PLI): GNU C-Compiler (gcc)

Waves Setup
Generate 'waves' file ☐

Note

- Clicking Apply in any page of the Simulate dialog box applies all of your simulation options (on all pages) and starts the simulation. Do not click Apply until you have selected all your simulation options on all pages of the dialog box.
- The internally generated MAC clock is a multiplexed version of the UTMI+ and ULPI clocks. The coreConsultant gate simulation might not pass in all configurations; for example, in UTMI+ and ULPI mode, because of a Verilog simulator race condition.

  With a post-layout annotated SDF file and a hold-time-fixed netlist, typically, this is not a problem. Therefore, using Formality to validate RTL-to-gate equivalence is considered best practice. Another option is to remove all library cell delays except the clock-to-output delays of flop cells. This avoids the Verilog simulator race condition. Use the *workspace*/example/rm_non_DFF_timing.pl example script to remove the delays from the file tsmc18.v that is shipped with the controller.

3.2.2 Running VCS Xprop Analyzer

After generating the RTL in the Specify Configuration activity of coreConsultant, you can run the Xprop feature of the VCS simulator in the coreConsultant environment to check for the instrumentation capabilities of Xprop on the generated RTL. The RTL is analyzed with relevant VCS switches. You can check the resulting log file to ensure the RTL is 100% instrumented. For more details on VCS Xprop, see the VCS documentation.

Running VCS Xprop Analyzer

You must configure the controller (see "Configuring the Controller" on page 30) before starting this task. To run the VCS Xprop analyzer on your controller, complete the following steps in the Run VCS Xprop Analyzer activity (see Figure 3-6).

- **Specify Command Line Switches**

  The following command line switches can be passed to VCS. By default, you do not need to specify any argument; you can leave these fields blank.

  - **VCS Xprop Cfg File**: Specify the full path of the Xprop config file that you want to pass to VCS as `-xprop=<CfgFile>`.

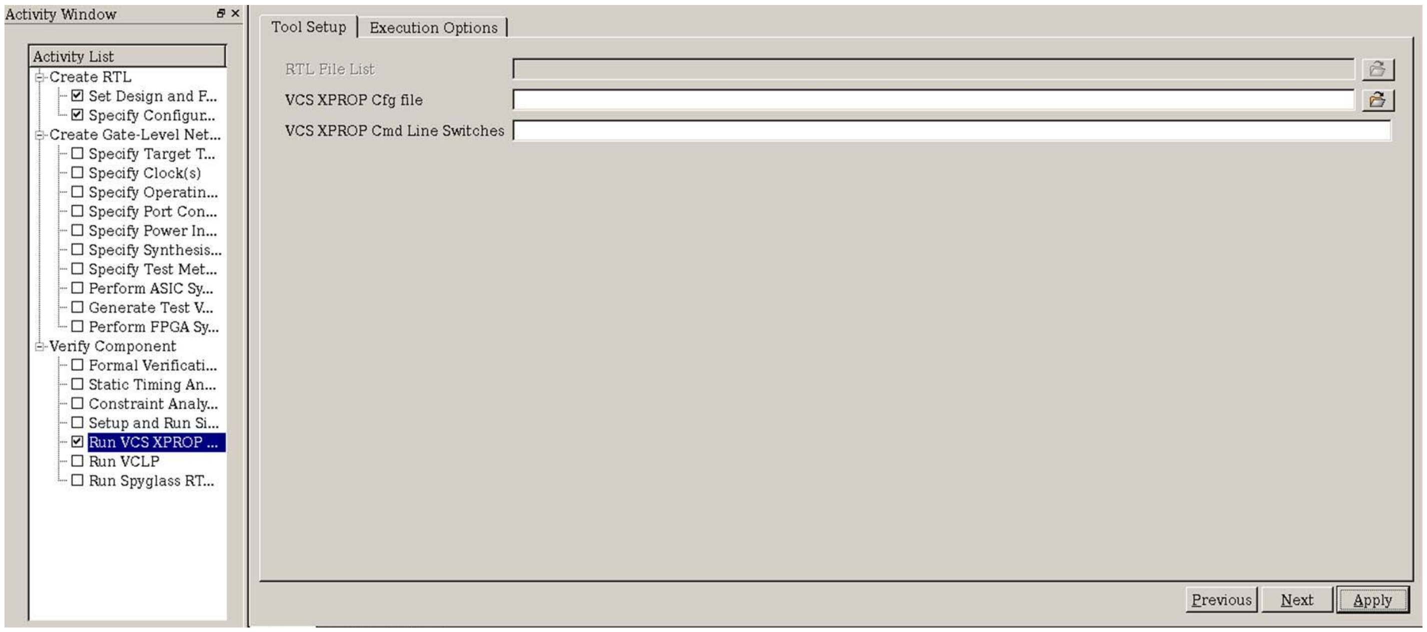
    By default, you do not need to specify any file; you can leave this field blank.

  - **VCS Xprop Cmd Line Switches**: You can specify any additional switches that you want to pass to the VCS command line when running the VCS executable.

    By default, you do not need to specify any switch; you can leave this field blank.

- Click **Apply** to run the VCS Xprop Analyzer activity.

Figure 3-6 Run VCS Xprop Analyzer Activity



Checking VCS Xprop Analyzer Results

- Click the **Report** tab.
- Click the **Run VCS Xprop Analyzer Summary** link.

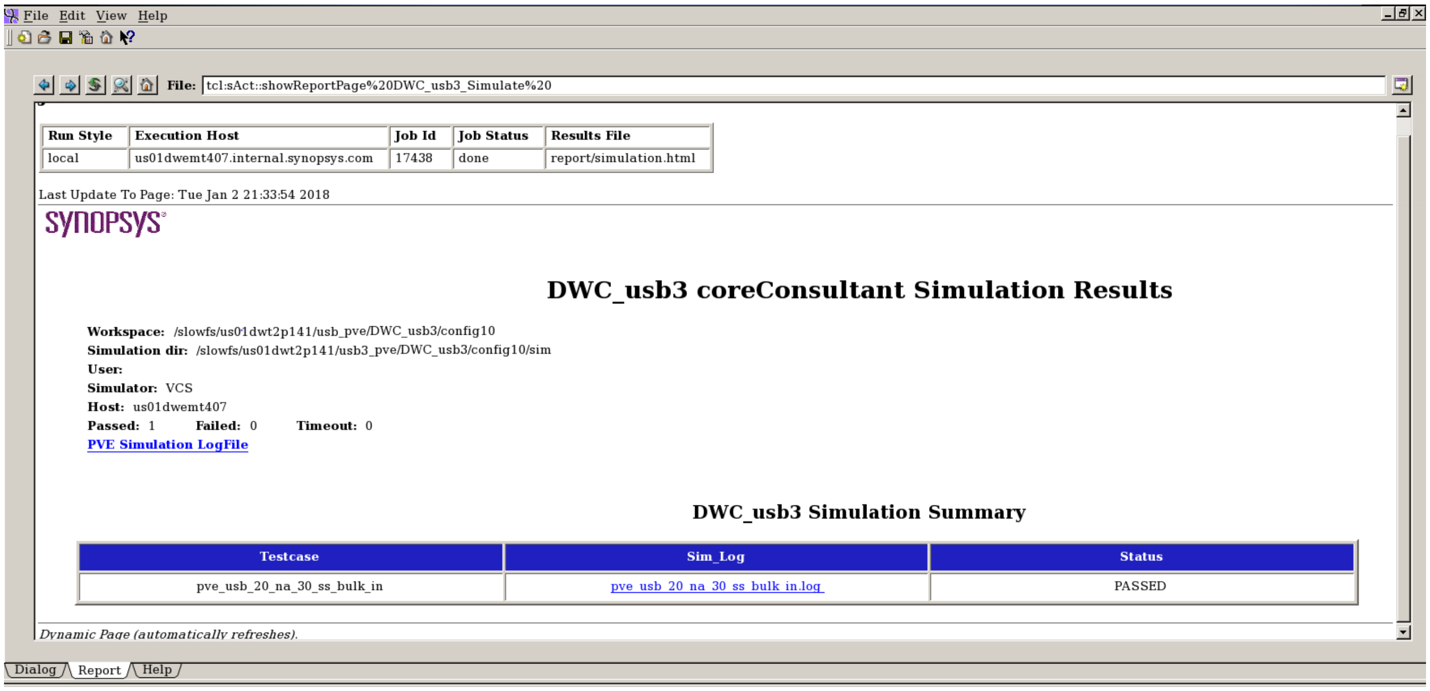
  In the opened spreadsheet, the **Instrumentation** row should show 100%. If it is not 100%, then additional information will be provided for the non-instrumented statements.

3.2.3 Checking Simulation Status and Results

To check simulation status and results:

1. Click the **Report** tab.
2. Click the **Simulate Summary** link.

Figure 3-7 Example Simulate Summary Report



Run Style	Execution Host	Job Id	Job Status	Results File
local	us01dwemt407.internal.synopsys.com	17438	done	report/simulation.html

Last Update To Page: Tue Jan 2 21:33:54 2018

DWC_usb3 coreConsultant Simulation Results

Workspace: /slowfs/us01dwt2p141/usb_pve/DWC_usb3/config10
Simulation dir: /slowfs/us01dwt2p141/usb3_pve/DWC_usb3/config10/sim
User:
Simulator: VCS
Host: us01dwemt407
Passed: 1 **Failed:** 0 **Timeout:** 0
PVE Simulation LogFile

DWC_usb3 Simulation Summary

Testcase	Sim_Log	Status
pve_usb_20_na_30_ss_bulk_in	pve_usb_20_na_30_ss_bulk_in.log	PASSED

Dynamic Page (automatically refreshes).

When you select the “LSF/GRD” option for the Run Style in the ‘Execution Options’ of the Simulate dialog box in Figure 3-3 on page 71, the status of the simulation jobs (running or complete) is incorrect. After all of 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.

After Running at least One Simulation

After you run at least one simulation from coreConsultant (preferably using VCS), the file runsim_vcs is created in the *workspace*/`sim/SoC_sim/pve/com` directory.

Table 3-5 Files (of Interest) Created After Specify Configuration and Simulation Activity

File Name	Description
workspace/sim/SoC_sim/pve/com/runsim_vcs	Command script to compile controller in VCS.
workspace/sim/SoC_sim/pve/com/runsim_vcs	Simulator options and include paths.
workspace/export/DWC_usb3.lst	List of source files in proper analysis order

By inspecting the files, you can determine how to access your configured controller from your chip design database and use the simulator of your choice.

The README.cmds.<*workspace_name*> File

After you select the testcases and the testbench options, when you click Apply in any page of the Simulate dialog box, coreConsultant creates a README.cmds.<*workspace_name*> file in the .../device, .../host, and .../hub directories. This file contains the launching commands with the RTL and testbench options for your configuration. Each gmake line in the README gives the launching command for each test that you selected in the coreConsultant GUI. You can use these commands as a reference to launch the testcases from the command line in the …/pve directory. For details on options used with gmake, see "Running PVE Tests from Command Line" on page 77.

Note Each time you click Apply in any page of the Simulate dialog box, the coreConsultant tool regenerates the README.cmds.<*workspace_name*> file for the testcases that you have currently selected.

3.3 Running PVE Tests from Command Line

To run a test in the command line, you must have run atleast one test through the coreConsultant GUI. This pre-requisite ensures that all the right files are installed correctly based on the configuration parameters chosen.

To run a test command line from the …/pve directory, issue the following command:

```
gmake usb_20_na_30_ss_bulk_in
```

The first argument to gmake is always the testcase name. The testcase names can be deduced from the …/device/tests/ts.*.sv file, where "*" indicates various testcase names.

This command creates the log file in .../device/output/usb_20_na_30_ss_bulkin.log and the result file in .../device/output/usb_20_na_30_ss_bulkin.result.

This section describes the following command line options that are available:

- MODE Option to run tests in host mode
- WAVES Option to generate waveforms
- USB_SPEED Option to select speed for SS/HS/FS tests
- USB20_IF_SEL Option to select HSIC or UTMI interface for HSIC tests
- DWC_USB30_PVE_UTMI_ULPI_SEL Option to select UTMI or ULPI interface when DWC_USB3_HSPHY_INTERFACE=3
- XPROP_EN option to select VCS Xprop option
- NUM_DEVICES Option to select number of Devices connected to Hub downstream ports

You can run tests with these options based on your configuration. To use the correct options/choices, refer to the following files the .../device, .../host, and .../hub directories:

- README file

  This README file lists the generic examples for default configuration. It also summarizes the definition of various command line options that are described in this section. Depending upon the choice of HSIC or ULPI, some additional command line options may be required for your configuration.

- README.cmds.<*workspace_name*> file

  This README file is specific to your configuration. It gives the launching commands for all tests that you previously ran using coreConsultant GUI. For details, refer to "The README.cmds.<workspace_name> File" on page 76

Each argument is independent of the others. Because these arguments are plusargs, avoid any whitespaces before and after "=" .

MODE Option

To run host-mode tests, use MODE=host option. To run hub-mode tests, use MODE=hub option. If you do not specify this option, it defaults to device mode and does not launch the host and hub tests.

Example:

```
gmake host_usb_20_na_30_ss_bulk_in MODE=host
```

WAVES Option

To generate waveforms, use the WAVES option. Table 3-6 lists the possible values available for this option. By default, no waveform is generated.

Table 3-6 WAVES Option

Option	Description
`WAVES=VPD`	Creates vcdplus.vpd file
`WAVES=DUMP`	Creates the tb.dump dumpfile
`WAVES=FSDB`	Creates the Verdi FSDB dumpfile

Example:

```
gmake usb_20_na_30_ss_bulk_in WAVES=VPD
```

USB_SPEED Option

To select the speed for SuperSpeed/high-speed/full-speed tests, use the USB_SPEED option. Table 3-7 lists the possible values available for this option.

Table 3-7 USB_SPEED Option

Option	Description
`USB_SPEED=HOST_SS_DEV_SS`[1]	Applicable for "*30_ss_*" tests in SuperSpeed
`USB_SPEED=HOST_HS_DEV_HS`	Runs the chosen *20_hs_fs* test in High Speed
`USB_SPEED=HOST_FS_DEV_FS`	Runs the chosen *20_hs_fs* test in Full Speed
`USB_SPEED=HOST_LS_DEV_LS`	Runs the chosen *20_hs_fs_ls* test in Low Speed

1. HOST_SS_DEV_SS is the default value for USB_SPEED option.

Example:

```
gmake usb_20_hs_fs_30_na_bulk_in USB_SPEED=HOST_HS_DEV_HS WAVES=VPD
```

Note

- Use USB_SPEED option only for *usb_20_hs_fs_ls* testcases.
- When parameter DWC_USB3_ENABLE_HSIC=1, only HS tests are enabled.

USB20_IF_SEL Option

This option is applicable only when you select “Yes” for the query "Enable HSIC Support for USB 2.0 Ports?" in the “PHY Config” tab of the “Specify Configuration” activity in coreConsultant (that is, DWC_USB3_ENABLE_HSIC=1).

You can run a HSIC-enabled configuration (DWC_USB3_ENABLE_HSIC=1) in both HSIC mode and UTMI mode.

- While running tests in coreConsultant, you can make this selection by answering the query "Run High Speed tests in HSIC mode?" in the “Testbench” tab of "Setup and Run Simulation" activity in coreConsultant.
- While running tests in command line, you can select HSIC or UTMI interface for HSIC tests using USB20_IF_SEL option. Table 3-8 lists the possible values available for this option.

Table 3-8 USB20_IF_SEL Option

Option	Description
`USB20_IF_SEL=HSIC`	■ Makes the HSIC-enabled configuration to run tests in HSIC mode. ■ The HSIC interface of the HSIC PHY is chosen in high-speed tests.
`USB20_IF_SEL=UTMI`	■ Makes the HSIC-enabled configuration to run tests in UTMI mode, that is, go through full enumeration. ■ The UTMI interface of the USB 3.0 PHY is chosen for high-speed tests.t

Note When parameter DWC_USB3_ENABLE_HSIC=1, only HS tests are enabled.

DWC_USB30_PVE_UTMI_ULPI_SEL Option

This option is applicable only when you select "UTMI+ and ULPI" for the query "High-Speed PHY Interface(s)?" in the “PHY Config” tab of the "Specify Configuraton" activity in coreConsultant (that is, DWC_USB3_HSPHY_INTERFACE=3).

While running tests in command line, you can select either UTMI or ULPI interface using UTMI_ULPI_SEL option. Table 3-9 lists the possible values available for this option.

Table 3-9 UTMI_ULPI_SEL Option

Option	Description
`UTMI_ULPI_SEL=ULPI_SEL`	The ULPI interface is used and GUSB2PHYCFG[4] will be set to 1.
`UTMI_ULPI_SEL=UTMI_SEL`	The UTMI interface is used and GUSB2PHYCFG[4] will be set to 0

XPROP_EN option

This option determines whether to enable or disable VCS Xprop option. If you do not specify this option, Xprop is disabled by default.

Note You need a VCS Add-on license to use this option.

NUM_DEVICES Option

This option determines the number of devices connected to the Hub downstream ports. By default, a value of 1 is selected.

3.4 Link-Up Sequence

This section describes USB 3.0 link-up sequence from reset removal to connect event for Host and Device mode. Timing diagrams presented in this section are not clock accurate. Red arrow indicates PHY action and black arrow indicates controller action.

3.4.1 Important Debug Signals

This section describes the important debug signals that are required to debug the link-related failures, i.e., link not entering U0, link Polling fail, link receiver detection fail, link stuck in SS.Disabled state, etc.

Note

- Signals with v_ prefix are ASCII signals.
- For debugging multi-port configuration, inst_port[*n*] can be replaced by appropriate port number, where *n* is the port number ranging from 0 to *n*-1.

- U_DWC_usb3.U_DWC_usb3_noclkrst.U_DWC_usb3_pwrm.inst_port[0].U_DWC_usb3_pwrm_prt.U_DWC_usb3_pwrm_u3piu.mac3_clk
  - In non-P3 state, mac3_clk is generated internally from PCLK. It must be 125MHz/8ns for all configurations.
  - In P3 state, mac3_clk is replaced by suspend_clk.
- U_DWC_usb3.U_DWC_usb3_noclkrst.U_DWC_usb3_pwrm.inst_port[0].U_DWC_usb3_pwrm_prt.U_DWC_usb3_pwrm_u3piu.pipe3_tx_pclk
  - In non-P3 state, pipe3_tx_pclk is PCLK.
  - In P3 state, pipe3_tx_pclk is replaced by suspend_clk.
- U_DWC_usb3.U_DWC_usb3_noclkrst.U_DWC_usb3_pwrm.inst_port[0].U_DWC_usb3_pwrm_prt.U_DWC_usb3_pwrm_u3ltssm.v_link_state

  ASCII representation of link state. For link states, refer to Chapter 'Signals' in the *DWC SuperSpeed USB 3.0 Controller Databook*.
- U_DWC_usb3.U_DWC_usb3_noclkrst.U_DWC_usb3_pwrdwn.U_DWC_usb3_u3pwrdwn.U_DWC_usb3_u3rhb.inst_port[0].U_DWC_usb3_u3rhb_prt.U_DWC_usb3_u3link.U_DWC_usb3_u3link_tctrl.v_tx_lc_cmd

  ASCII representation of transmitted link commands.
- U_DWC_usb3.U_DWC_usb3_noclkrst.U_DWC_usb3_pwrdwn.U_DWC_usb3_u3pwrdwn.U_DWC_usb3_u3rhb.inst_port[0].U_DWC_usb3_u3rhb_prt.U_DWC_usb3_u3link.v_rx_link_command

  ASCII representation of received link commands.
- U_DWC_usb3.U_DWC_usb3_noclkrst.U_DWC_usb3_pwrdwn.U_DWC_usb3_u3pwrdwn.U_DWC_usb3_u3rhb.inst_port[0].U_DWC_usb3_u3rhb_prt.U_DWC_usb3_u3link.U_DWC_usb3_u3link_rpkt.v_rx_pkt_info

  U_DWC_usb3.U_DWC_usb3_noclkrst.U_DWC_usb3_pwrdwn.U_DWC_usb3_u3pwrdwn.U_DWC_usb3_u3rhb.inst_port[0].U_DWC_usb3_u3rhb_prt.U_DWC_usb3_u3link.U_DWC_usb3_u3link_rpkt.v_rx_pkt_data

U_DWC_usb3.U_DWC_usb3_noclkrst.U_DWC_usb3_pwrdwn.U_DWC_usb3_u3pwrdwn.U_DWC_usb3_u3rhb.inst_port[0].U_DWC_usb3_u3rhb_prt.U_DWC_usb3_u3link.U_DWC_usb3_u3link_rpkt.v_rx_pkt_c5_c16_hseq

Above signals are ASCII representations of received Header packet.

- v_rx_pkt_info: This signal indicates sub-type, delayed bit and Header sequence number.
- v_rx_pkt_data: This signal indicates content of header packet in Hex.
- v_rx_pkt_c5_c16_hseq: This signal indicates correctness of CRC5, CRC16, and Header sequence number (1: correct, 0: wrong).

- U_DWC_usb3.U_DWC_usb3_noclkrst.U_DWC_usb3_pwrdwn.U_DWC_usb3_u3pwrdwn.U_DWC_usb3_u3rhb.inst_port[0].U_DWC_usb3_u3rhb_prt.U_DWC_usb3_u3link.U_DWC_usb3_u3link_tctrl.v_tx_mpkt_info

  U_DWC_usb3.U_DWC_usb3_noclkrst.U_DWC_usb3_pwrdwn.U_DWC_usb3_u3pwrdwn.U_DWC_usb3_u3rhb.inst_port[0].U_DWC_usb3_u3rhb_prt.U_DWC_usb3_u3link.U_DWC_usb3_u3link_tctrl.v_tx_mpkt_data

  Above signals are ASCII representations of transmitted Header packet.

  - v_tx_mpkt_info: This signal indicates sub-type, delayed bit and Header sequence.
  - v_tx_mpkt_data: This signal indicates content of header packet in Hex.

- U_DWC_usb3.U_DWC_usb3_noclkrst.U_DWC_usb3_pwrm.U_DWC_usb3_pwrm_csr.b2md_csr_gctl

  Global Controller Control Register (GCTL). For more information, refer to GCTL register definition in the “Registers” chapter of the *DWC SuperSpeed USB 3.0 Controller Programming Guide*.

- U_DWC_usb3.U_DWC_usb3_noclkrst.U_DWC_usb3_pwrm.U_DWC_usb3_pwrm_csr.b2md_csr_dctl

  Device Control Register (DCTL). For more information, refer to DCTL register definition in the “Registers” chapter of the *DWC SuperSpeed USB 3.0 Controller Programming Guide*.

- U_DWC_usb3.U_DWC_usb3_noclkrst.U_DWC_usb3_pwrm.U_DWC_usb3_pwrm_csr.gusb3pipectl

  Global USB3 PIPE Control Register (GUSB3PIPECTLn). For more information, refer to GUSB3PIPECTLn register definition in the “Registers” chapter of the *DWC SuperSpeed USB 3.0 Controller Programming Guide*.

- U_DWC_usb3.U_DWC_usb3_noclkrst.U_DWC_usb3_pwrm.U_DWC_usb3_pwrm_csr.b2md_portsc.

  Port Status and Control Register (PORTSC). For more information, refer to xHCI Specification.

- U_DWC_usb3.U_DWC_usb3_noclkrst.U_DWC_usb3_pwrm.inst_port[0].U_DWC_usb3_pwrm_prt.U_DWC_usb3_pwrm_u3ltssm.v_phy_cmd_state

  PHY command state machine sub state machine. For link sub-states, refer to Table ‘Link Sub-states’ in the *DWC SuperSpeed USB 3.0 Controller Databook*.

- U_DWC_usb3.U_DWC_usb3_noclkrst.U_DWC_usb3_pwrm.inst_port[0].U_DWC_usb3_pwrm_prt.U_DWC_usb3_pwrm_u3ltssm.v_clk_state

  PCLK state machine sub state. For link sub-states, refer to Table ‘Link Sub-states’ in the *DWC SuperSpeed USB 3.0 Controller Databook*.

- U_DWC_usb3.U_DWC_usb3_noclkrst.U_DWC_usb3_pwrm.inst_port[0].U_DWC_usb3_pwrm_prt.U_DWC_usb3_pwrm_u3ltssm.v_rx_det_state

  Rx.Detect sub-state. For link sub-states, refer to Table 'Link Sub-states' in the *DWC SuperSpeed USB 3.0 Controller Databook*.

- U_DWC_usb3.U_DWC_usb3_noclkrst.U_DWC_usb3_pwrm.inst_port[0].U_DWC_usb3_pwrm_prt.U_DWC_usb3_pwrm_u3ltssm.v_poll_state

  Polling sub-state. For link sub-states, refer to Table 'Link Sub-states' in the *DWC SuperSpeed USB 3.0 Controller Databook*.

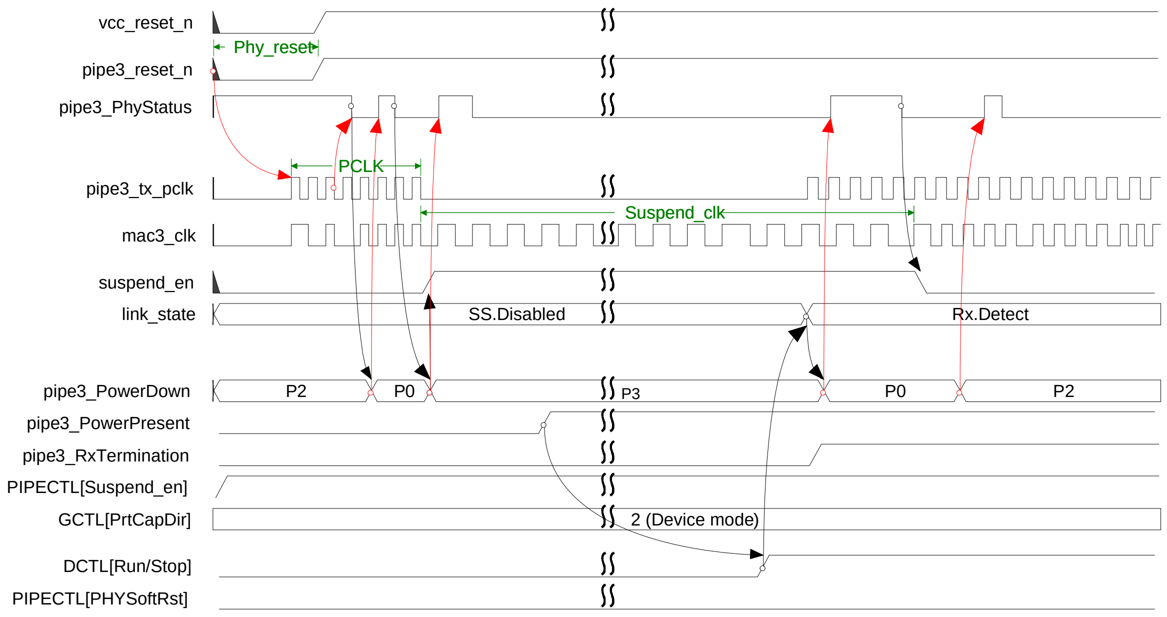
3.4.2 Device Link-Up Sequence

This section discusses the link-up sequence for a Device.

3.4.2.1 Device Rx.Detect Entry

This section discusses the Rx.Detect entry sequence.

Figure 3-8 Device Rx.Detect Entry



1. If DWC_USB3_PIPE_RXTERM_RESET_VAL is 1, the controller starts with Rx.Detect link state. If DWC_USB3_PIPE_RXTERM_RESET_VAL is 0, the controller starts with SS.Disabled link state.
2. When vcc_reset_n transitions from 0 to 1 and GUSB3PIPECTL[PHYSoftRst] is 0, the controller de-asserts pipe3_reset_n. The pipe3_PhyStatus must be high at this time.
3. After the removal of pipe3_reset_n, the SS PHY starts PCLK at a required frequency and de-asserts pipe3_PhyStatus. The controller considers PCLK is running at the correct frequency when pipe3_PhyStatus transitions from 1 to 0. If DWC_USB3_PIPE_RXTERM_RESET_VAL is 1, the link state transitions from Rx.Detect to SS.Disabled state.
   - For 32-bit PHY: PCLK = 125 MHz, mac3_clk = PCLK = 125 MHz
   - For 16-bit PHY: PCLK = 250 MHz, mac3_clk = PCLK/2 = 125 MHz

- For 8-bit PHY: PCLK = 500 MHz, mac3_clk = PCLK/4 = 125MHz

4. If GUSB3PIPECTL[Suspend_en] is high, and DCTL[Run/Stop] bit is not set, the controller puts the PHY into P3 power state. The PIPE3 Specification does not allow direct transition from P2 to P3. So the controller will go through P2 -> P0 -> P3 transitions. When pipe3_PowerDown goes from P0 -> P3, the internal mac3_clk/PCLK switches to suspend_clk.

   If GUSB3PIPECTL[Suspend_en] is low, then the controller does not request the PHY to enter P3. It keeps the PHY in P2 power state.

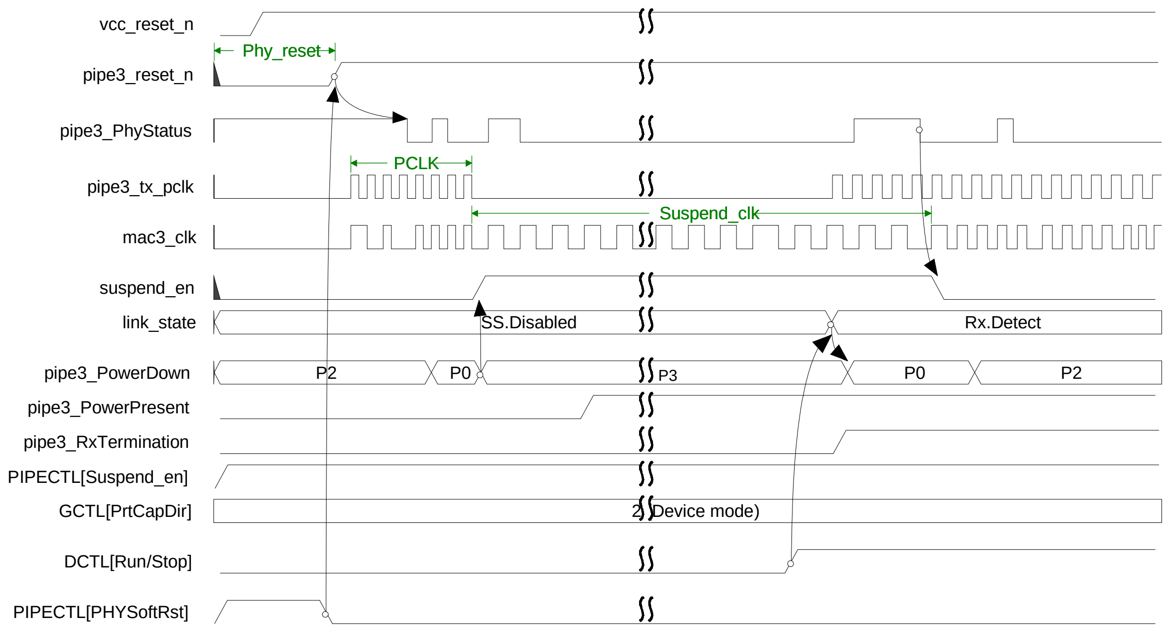
5. When pipe3_PowerPresent is 1 and DCTL[Run/Stop] is 1, the controller transitions from SS.Disabled(P2/P3) to Rx.Detect(P2) state (if controller is not in Rx.Detect state) and generates link state change event.

Attention

- GCTL[PrtCapDir] must be set to 2 for device configurations.
- GUSB3PIPECTL[PHYSoftRst] bit is 0.

Note that GUSB3PIPECTL[PHYSoftRst] can be used when vcc_reset_n duration is smaller than the minimum reset duration required for the SS PHY. In this case, GUSB3PIPECTL[PHYSoftRst] power on value should be 1. If GUSB3PIPECTL[PHYSoftRst] is 1, then the controller will keep the PHY in reset state even if vcc_reset_n is removed. Software is responsible for de-asserting this bit. The controller will proceed to connection process when this bit is 0. Figure 3-9 shows how to use GUSB3PIPECTL[PHYSoftRst].

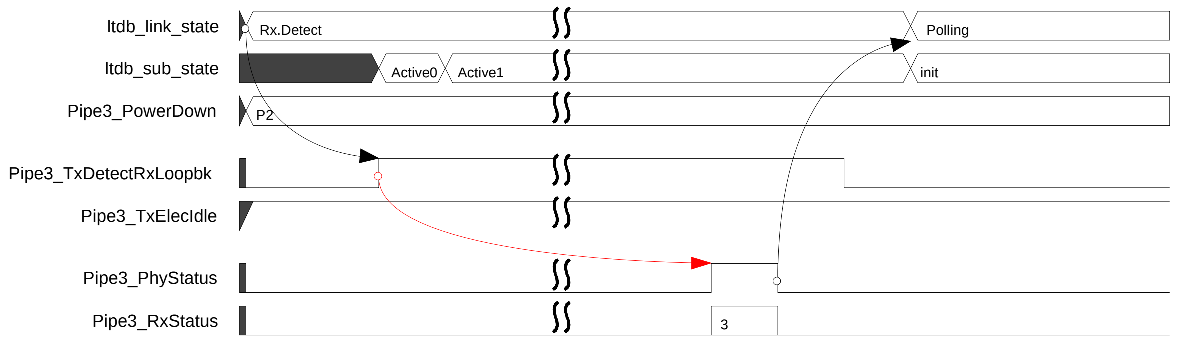
Figure 3-9 How to Use GUSB3PIPECTL[PHYSoftRst]



3.4.2.2 Device Rx.Detect to Polling Entry

This section discusses the Rx.Detect to Polling entry sequence.

Figure 3-10 Device Rx.Detect to Polling Entry



1. After entry into Rx.Detect state, the controller starts receiver detection by asserting pipe3_TxDetectRxLoopbk and pipe3_TxElecIdle.
2. If the remote receiver termination is detected, the PHY asserts pipe3_PhyStatus and pipe3_RxStatus as 3. If the remote receiver termination is not detected, the PHY asserts pipe3_PhyStatus and pipe3_RxStatus as 0.
3. Upon receiving pipe3_PhyStatus, the controller de-asserts pipe3_TxDetectRxLoopbk.
4. If the remote receiver is detected, the controller transitions from Rx.Detect to Polling state and generates link state change event.

   If the remote receiver is not detected, the controller stays in Rx.Detect.Quiet sub-state for 12ms. After 12ms, controller starts receiver detection process again. In this way, if the receiver detection fails eight times, the controller disables SS connection (link state = SS.Disabled) and starts connection sequence in non-SS mode.

Attention If pipe3_RxStatus stays 0, then the controller will never enter into the Polling state. The pipe3_RxStatus must be 3 for before the controller can transition from Rx.Detect to Polling.

Table 3-10 lists the Rx.Detect Timer/Counter values.

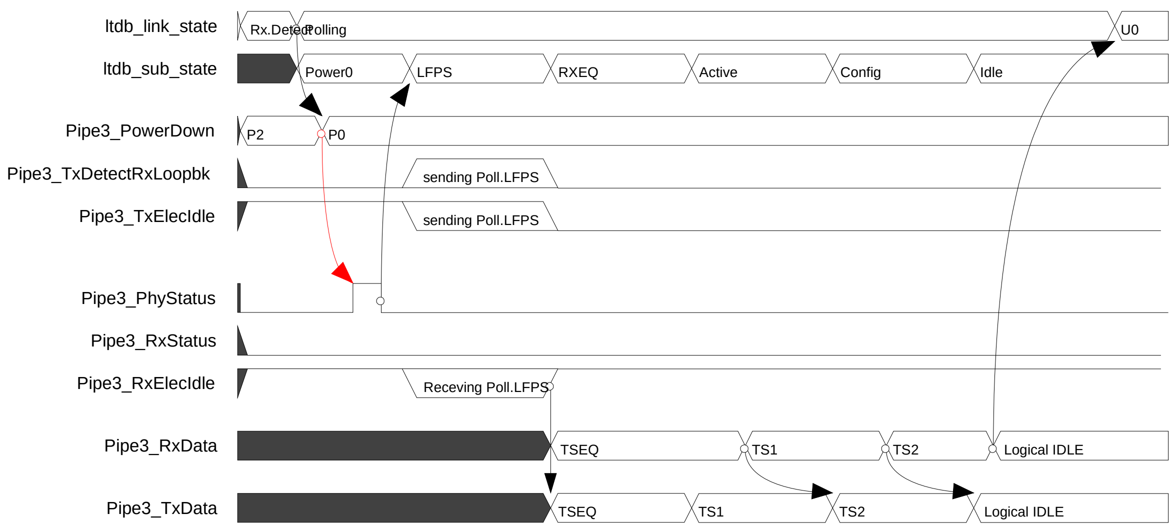
Table 3-10 Rx.Detect Timer/Counter

Parameter	Non-ScaleDown Value	ScaleDown Value
tRxDetectQuietTimeout	12ms	10us

3.4.2.3 Device Polling to U0 Entry

This section discusses the Polling to U0 entry sequence.

Figure 3-11 Device Polling to U0 Entry



1. After entry into Polling state, the controller puts the PHY in to P0 and starts the training sequence. First, it does polling.LFPS handshake. .

Note If there is a PHY data width mismatch between the controller and the PHY, then PCLK and mac3_clk will not be running at the correct frequency and polling.LFPS may fail.

2. After successful Poll.LFPS handshake, the controller transitions from Poll.LFPS to Poll.RxEq sub-state.
3. There is no handshake required in Poll.RxEq sub-state. The controller transitions from Poll.RxEq to Poll.Active sub-state upon sending the required TSEQ ordered sets.
4. After Poll.Active entry, the controller starts sending TS1 ordered set and waits for TS1/TS2 to be received. Upon successful TS1 handshake, the controller transitions from Poll.Active to Poll.Configuration sub-state.
5. After Poll.Configuration entry, the controller starts sending TS2 ordered set and waits for TS2 to be received. Upon successful TS2 handshake, the controller transitions from Poll.Configuration to Poll.Idle sub-state.
6. After Poll.Idle entry, the controller starts sending Logical IDLE symbols and waits for Logical IDLE symbols to be received. Upon successful LIDLE handshake, the controller transitions from Polling to U0 state and generates link state change event.

Table 3-11 lists the Polling Timer/Counter values.

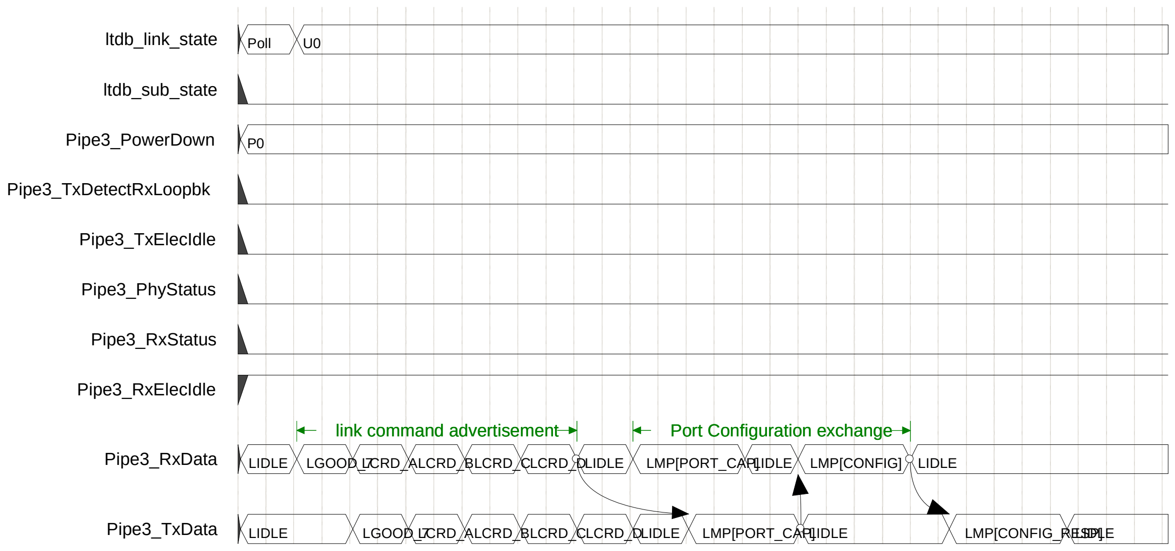
Table 3-11 Polling Timer/Counter

Parameter	Non-ScaleDown Value	ScaleDown Value	Notes
LFPS_POLL_BURST_CLKS	1us	104ns	Tx Poll.LFPS burst duration
LFPS_POLL_SPACE_US	9us	2us	Tx Poll.LFPS space duration
LFPS_POLL_MIN_CLKS	488ns	80ns	Rx Poll.LFPS Min duration
LFPS_POLL_MAX_CLKS	2us	160ns	Rx Poll.LFPS Max duration
LGEN_POLL_NUM	16	2	Min total Poll.LFPS to send
LGEN_POLL_SENT	4	2	Min Poll.LFPS to send after receiving one
RX_EQ_COUNT	66191	8	Num TSEQ to send

3.4.2.4 Device in U0

This section discusses the device in U0 state.

Figure 3-12 Device in U0



1. After U0 entry, the controller starts the link initialization process. The initialization includes the Header Sequence Number Advertisement and the Rx Header Buffer Credit Advertisement between the two ports before a header packet can be transmitted.

Note The controller will not send any header packet until it receives LCRD_x from the remote link.

2. After finishing the link initialization process, the controller starts port configuration process by sending LMP[CAP] (Upstream port capable). After receiving LMP[Config], the controller sends LMP[Config_response] and generates connect event to the software.

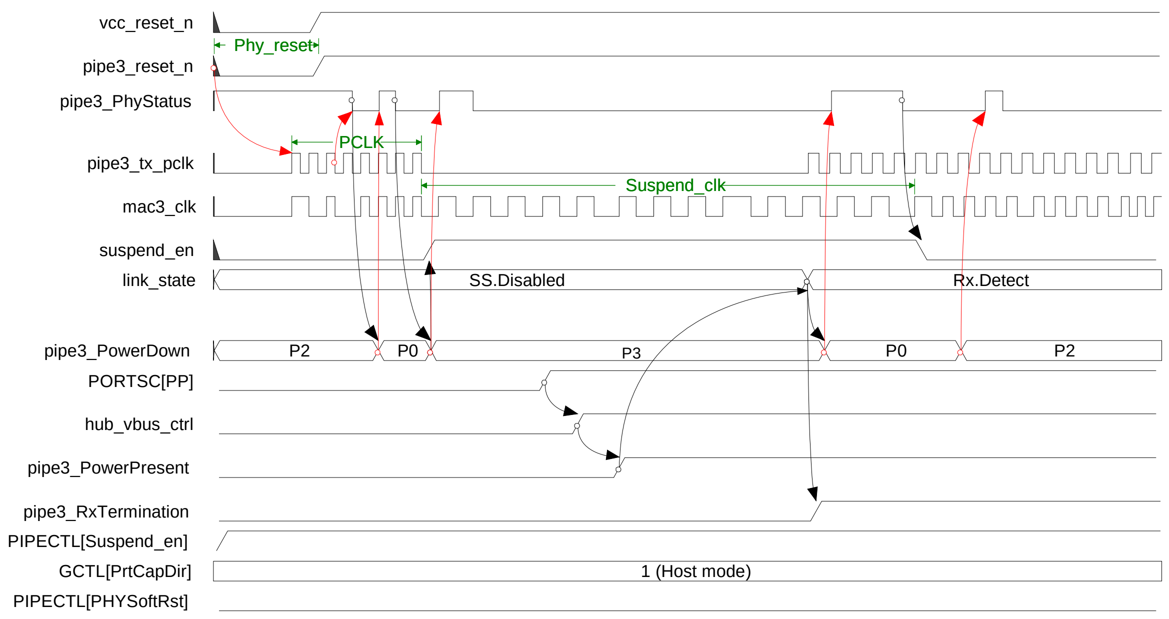
3.4.3 Host Link-Up Sequence

This section discusses the link-up sequence for a Host.

3.4.3.1 Host Rx.Detect Entry

This section discusses the Rx.Detect entry sequence for a Host.

Figure 3-13 Host Rx.Detect Entry



1. If DWC_USB3_PIPE_RXTERM_RESET_VAL is 1, the controller starts with Rx.Detect link state. If DWC_USB3_PIPE_RXTERM_RESET_VAL is 0, the controller starts with SS.Disabled link state.
2. The vcc_reset_n and pipe3_reset_n is removed after the minimum PHY reset duration. The pipe3_PhyStatus must be high at this time.
3. After removal of pipe3_reset_n, the SS PHY starts PCLK at a required frequency and de-asserts pipe3_PhyStatus. The controller considers PCLK is running at the correct frequency when pipe3_PhyStatus transitions from 1 to 0. If DWC_USB3_PIPE_RXTERM_RESET_VAL is 1, the link state transitions from Rx.Detect to SS.Disabled state
    - For 32-bit PHY: PCLK = 125MHz, mac3_clk = PCLK = 125MHz
    - For 16-bit PHY: PCLK = 250MHz, mac3_clk = PCLK/2 = 125MHz
    - For 8 bit-PHY: PCLK = 500MHz, mac3_clk = PCLK/4 = 125MHz
4. If GUSB3PIPECTL[Suspend_en] is high, and PORTSC[PP] bit is not set, the controller puts the PHY in P3 power state. The PIPE3 Specification does not allow direct transition from P2 to P3. So the controller goes through P2 -> P0 -> P3 transitions. When pipe3_PowerDown goes from P0 -> P3, the internal mac3_clk/PCLK switches to suspend_clk.

    If GUSB3PIPECTL[Suspend_en] is low, the controller does not request the PHY to enter P3. It keeps the PHY in P2 power state.

5. When xHCI driver writes PORTSC[PP] as 1, the controller asserts hub_vbus_ctrl to the PHY. Upon detecting hub_vbus_cntl as 1, the PHY asserts pipe3_PowerPresent. On pipe3_PowerPresent = 1, the controller transitions from SS.Disabled(P2/P3) to Rx.Detect(P2) state (if controller is not in Rx.Detect state).

Note PORTSC[PP] in Figure 3-13 on page 88 is shown as '0' after vcc_reset_n de-assertion to illustrate the LTSSM transitions and to capture the trigger points for the transitions. In hardware, PORTSC[PP] will be '1' after vcc_reset_n de-assertion.

3.4.3.2 Host Rx.Detect to Polling Entry

Host Rx.Detect to Polling entry sequence is same as the Device Rx.Detect to Polling entry sequence (refer to "Device Rx.Detect to Polling Entry" on page 85) except that the Host controller will not generate the link state change event.

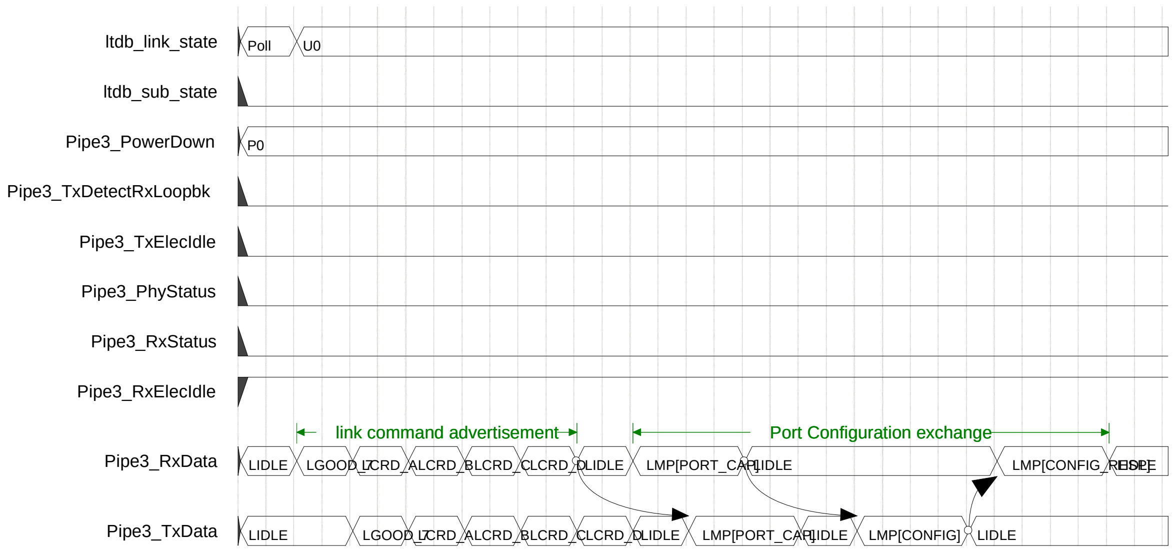
3.4.3.3 Host Polling to U0 Entry

Host Polling to U0 entry sequence is same as Device Polling to U0 entry sequence (refer to "Device Polling to U0 Entry" on page 86) except that the Host controller will not generate the link state change event.

3.4.3.4 Host in U0

This section discusses the host in U0 state.

Figure 3-14 Host in U0



1. After U0 entry, the controller starts the link initialization process. The initialization includes the Header Sequence Number Advertisement and the Rx Header Buffer Credit Advertisement between the two ports before a header packet can be transmitted.

Note The controller will not send any header packet until it receives LCRD_x from the remote link.

---

2. After finishing the link initialization process, the controller starts port configuration process by sending LMP[CAP] (Downstream port capable). After receiving the LMP[CAP], the controller sends LMP[Config]. Upon receiving the LMP[Config], the remote link (Upstream port) sends LMP[config_resp]. The controller generates connect event to the software when it receives LMP[config_resp].

4
Performing Synthesis and Post-Synthesis Activities

This chapter describes the synthesis flow for the DWC_usb3 controller. It also shows you how to run formal verification, use the coreConsultant tool to insert DFT, run automatic test pattern generation (ATPG), and perform static timing analysis (STA).

The chapter discusses the following topics:

4.1 Synthesizing the Controller

The coreConsultant tool is your interface to Synopsys synthesis tools for the DWC_usb3 controller.. The default set of synthesis attributes provides acceptable results in most libraries. When you want to access the synthesis scripts for exporting to your chip database, or when you are using non-Synopsys synthesis tools, then refer to "Synthesizing Outside of coreConsultant" on page 100.

The topics in this section are as follows:

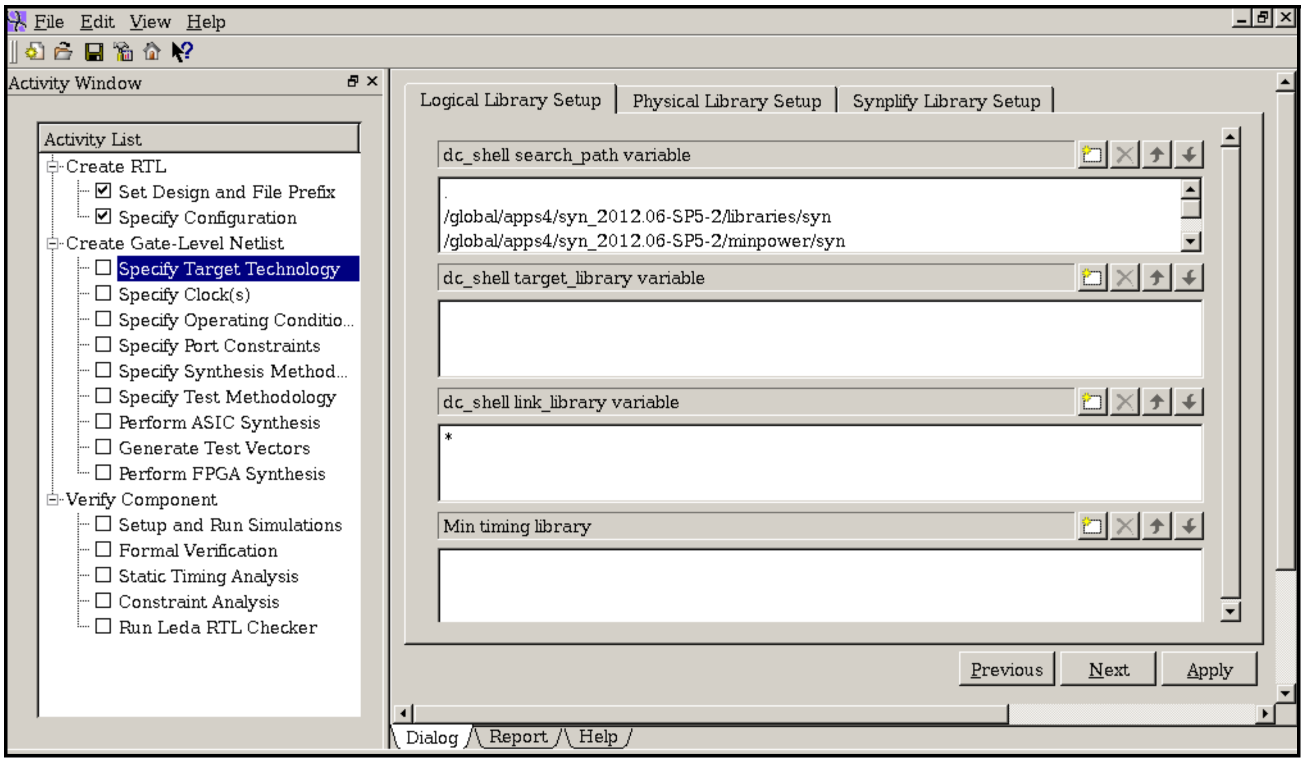
- "Synthesizing the Controller for an ASIC" on page 93
- "Synthesizing the Controller for an FPGA" on page 100
- "Synthesizing Outside of coreConsultant" on page 100
- "Synthesis Constraints for DWC_usb3 Controller" on page 102

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 28). To check your tools, click the **Help > Check Environment** menu item. You can edit your environment settings by modifying your Unix environment setup or through the **Edit > Tool Installation Roots** menu (make sure to set the 64-Bit checkbox if you are using 64-bit tools).

To synthesize your controller, select the 'Create Gate-Level Netlist' activity as shown in Figure 4-1.

Figure 4-1 Create Gate-Level Netlist (Synthesis) Activity



dc_shell search_path variable
```
.
/global/apps4/syn_2012.06-SP5-2/libraries/syn
/global/apps4/syn_2012.06-SP5-2/minpower/syn
```
dc_shell target_library variable

dc_shell link_library variable
```
*
```
Min timing library

4.1.1 Synthesizing the Controller for an ASIC

When you want to map and synthesize the controller to an ASIC using the Synopsys DC tool within coreConsultant, follow the process outlined in this section. Otherwise, you must export the synthesis scripts from coreConsultant as outlined in "Synthesizing Outside of coreConsultant" on page 100 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 93
- "Checking ASIC Synthesis Results" on page 98

4.1.1.1 Performing ASIC Synthesis

You must configure the controller (see "Configuring the Controller" on page 27) before starting this task. To synthesize your controller, complete the following steps in the Create Gate-Level Netlist activity (see Figure 4-1 on page 92).

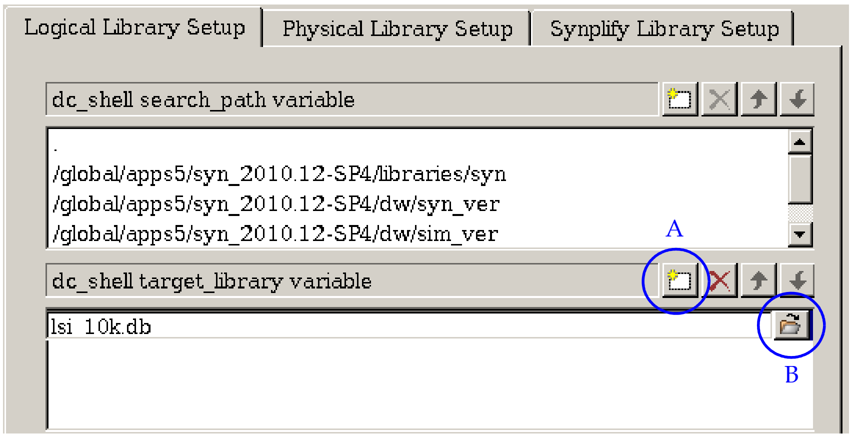
1. **Specify Target Technology.**

   A target library must be specified, otherwise, errors occur in coreConsultant. For ASIC synthesis, use a target technology of .18 microns or less. To add a target technology library, click the folder icon (A in Figure 4-2 on page 93) and use the navigation tool (B in the figure).

   When you are running Physical Synthesis, you must specify the physical library under the "Physical Library Setup" tab.

   The default values for the search_path and link library do not normally need to be changed.

Figure 4-2 Specify Target Technology – Logical Library Setup



```
.
/global/apps5/syn_2010.12-SP4/libraries/syn
/global/apps5/syn_2010.12-SP4/dw/syn_ver
/global/apps5/syn_2010.12-SP4/dw/sim_ver
```

```
lsi_10k.db
```

2. **Specify Clock(s).**

   The default CycleTime for the input clocks gets populated in the initial screen. These times match the DWC_usb3 default clock-rate configuration (125 MHz). The default values for clock-related synthesis attributes provide acceptable synthesis results in most libraries.

   For different clocks in the design, see "System Clock, Reset, and Control Signals" section. The internal clocks are derived from multiplexing multiple sources (ulpi_clk and utmi_clk). Software controls the selection of the source clock to phy_clk.

Note

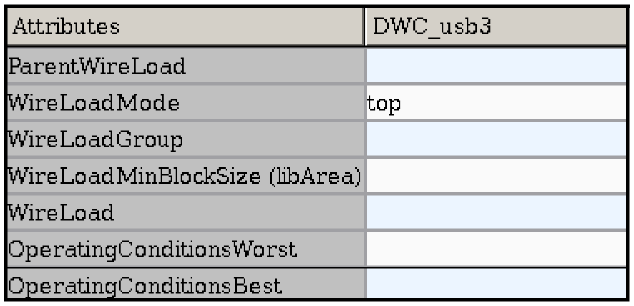
- Even though the utmi_clk and ulpi_clk could be different frequencies, during synthesis it is recommended to set the same clock frequency for all of these clocks to avoid hold time fixing.
- <*workspace*/syn/auxScripts/clk_div_postelab.tcl File:
  - This file is created when the DWC_USB3_EN_USB2_ONLY=0.
  - The DWC_usb3_clk_div_2_4 module needs to be pre-compiled and used to avoid a known issue with PrimeTime unable to apply generated clock constraint.

3. **Specify Operating Conditions and Wireloads.**

   When you do not see a value beside "OperatingConditionsWorst", select an appropriate value from the drop-down list. You may want to list some example values for your controller. When there is no value for this attribute, you may get an error message.

   You should 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** and look at the report, which gives the operating conditions and wireload information.

Figure 4-3 Specify Operating Conditions and Wireloads Window



Attributes	DWC_usb3
ParentWireLoad	
WireLoadMode	top
WireLoadGroup	
WireLoadMinBlockSize (libArea)	
WireLoad	
OperatingConditionsWorst	
OperatingConditionsBest	

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.

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

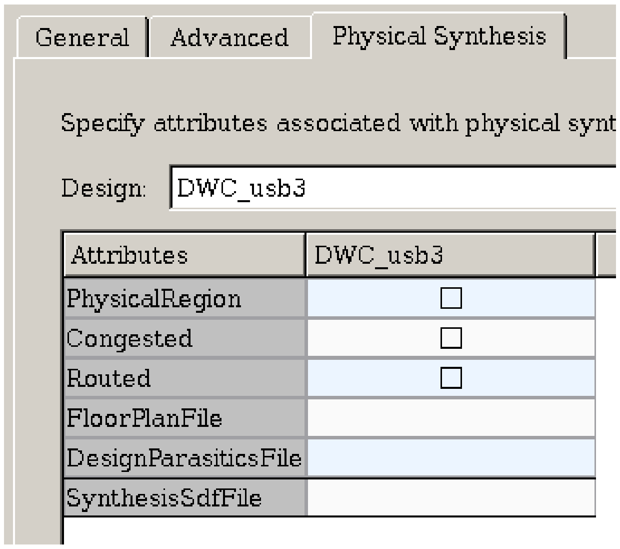
5. **Specify Synthesis Methodology.**

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

Note <*workspace*>/syn/auxScripts/constraint.tcl File:
You can specify DC commands to be executed prior to a specific optimization phase in the **Specify Synthesis Methodology** > **Advanced** > **TclAuxSynthesisScript[Constrain]** field. By default, this field points to the constraint.tcl file.

To run Physical Synthesis, click on the "Physical Synthesis" tab (Figure 4-4) and specify information in the related fields. The physical synthesis flow uses the Synopsys DC-Topographical engine, which is part of Design Compiler (dc_shell).

Figure 4-4 Specify Synthesis Methodology – Physical Synthesis Tab



Specify attributes associated with physical synt

Design: DWC_usb3

Attributes	DWC_usb3
PhysicalRegion	☐
Congested	☐
Routed	☐
FloorPlanFile	
DesignParasiticsFile	
SynthesisSdfFile	

6. **Specify Test Methodology.**

   These attributes are set by the controller developer and can be optionally modified by you. For more details, refer to "Inserting Design For Test" on page 109.

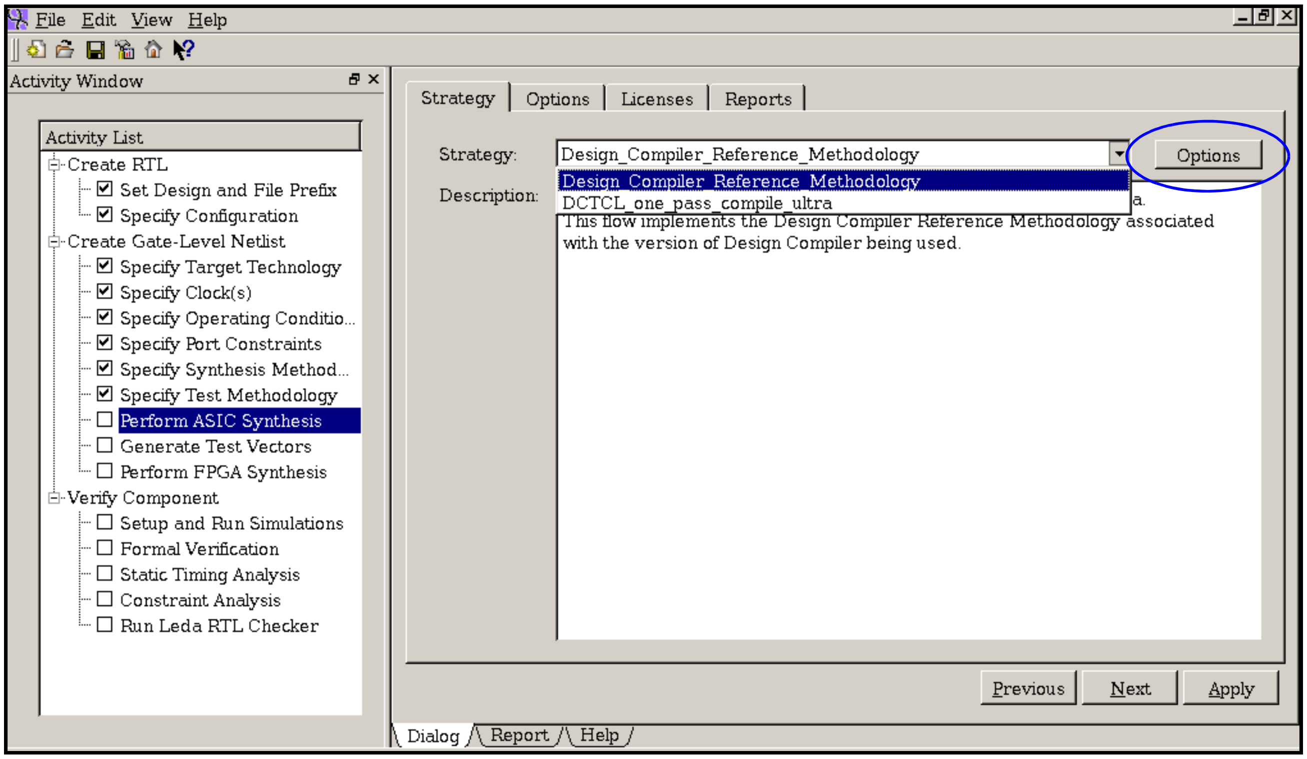
Hint You can access global synthesis options through **Edit -> Preferences -> Synthesis** menu.

7. Perform ASIC Synthesis.

You can select the synthesis strategy on the Strategy tab shown in Figure 4-5 on page 96. The default flow is "DCTL_opto_strategy". A detailed explanation of each strategy is provided in the GUI.

Hint One other 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://solvnet.synopsys.com/retrieve/021023.html. This synthesis strategy is quicker and does not generate intermediate stages. Therefore, it is much easier to trace the steps in the flow.

Figure 4-5 Synthesize Activity -> Strategy Tab



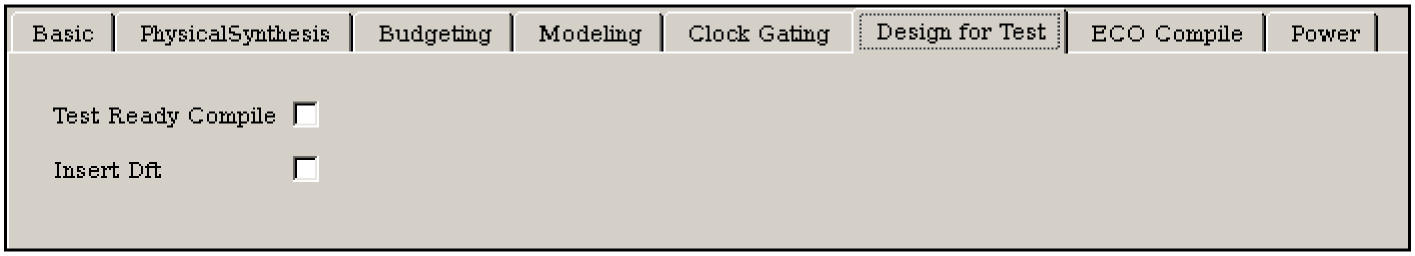
Strategy: Design_Compiler_Reference_Methodology

Design_Compiler_Reference_Methodology
DCTCL_one_pass_compile_ultra

Description: This flow implements the Design Compiler Reference Methodology associated with the version of Design Compiler being used.

Clicking on the **Options** button (highlighted in Figure 4-5) beside the selected strategy lists a series of other options to specify other options for that particular strategy. More information for all these options is available in the *coreConsultant User Guide* (also see "Help Information" on page 200) or by right-clicking on any dialog text.

Figure 4-6 Synthesize Activity -> Options Button -> Design for Test Tab



Test Ready Compile ☐

Insert Dft ☐

These are the most important options.

- **Design for Test**

  Here you specify whether to add the -scan option to the initial compile call (“Test Ready Compile”) and/or insert DFT circuitry (“Insert DFT”). See “Inserting Design For Test” on page 109.

- **Physical Synthesis**

  When you are running Physical Synthesis, tick the “Physical Synthesis” box and complete the other relevant fields. Setting this parameter causes the physical synthesis placement engine to be used during optimization. Net loads and delays are estimated during placement. The flow may 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.

Note In the SoC synthesis, pipe3_ElasBufferMode output is a static signal. Consider the path involved with this signal as a false path.

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

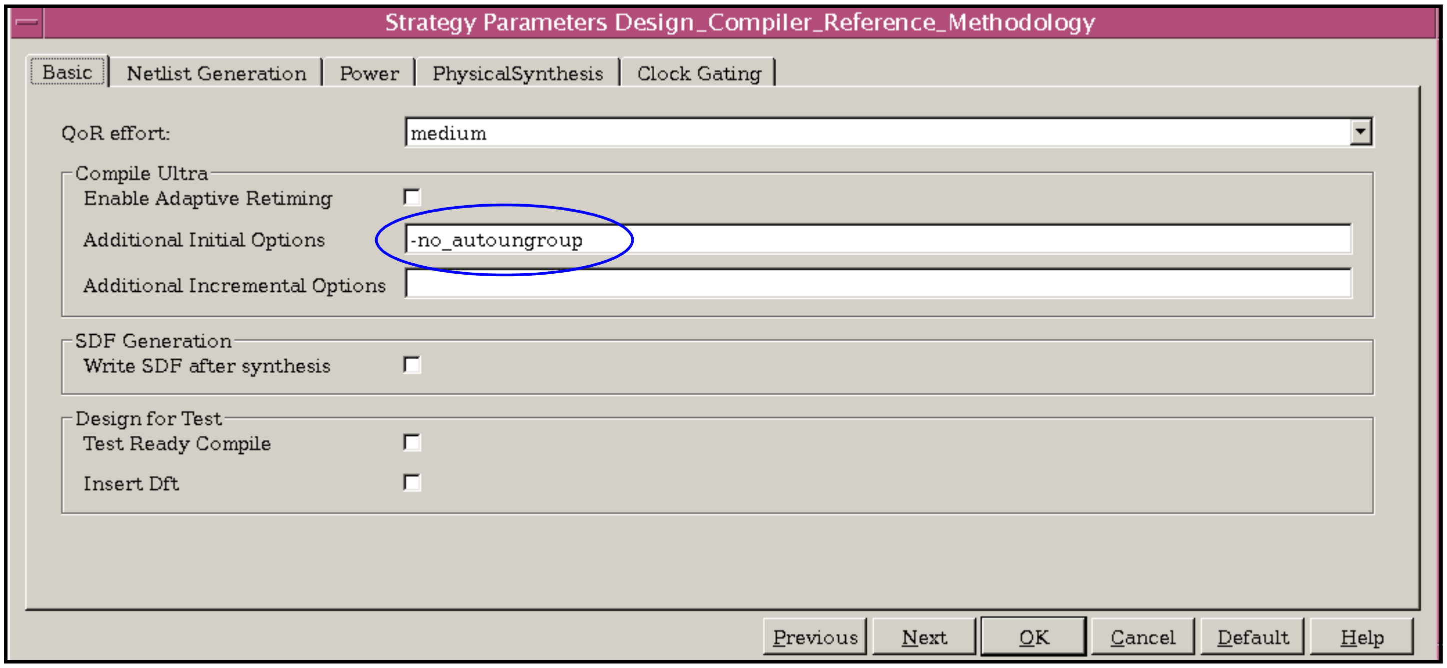
Note If you experience any problems during synthesis, refer to “Troubleshooting” on page 198.

9. If you need to run gate-level simulation, include the following option before running Synthesis:

   Under Perform ASIC Synthesis activity, Strategy tab, click on **Options** button (highlighted in Figure 4-5). Under the Basic tab, for Additional Initial Options, enter -no_autoungroup as shown in Figure 4-7.

   This ensures that the netlist is not flattened. A flattened netlist may result in compilation issues while running gate-level simulation.

Figure 4-7 Synthesize Activity -> Options Button -> Basic Tab


Strategy Parameters Design_Compiler_Reference_Methodology

QoR effort: medium

Compile Ultra
Enable Adaptive Retiming ☐
Additional Initial Options: -no_autoungroup
Additional Incremental Options:

SDF Generation
Write SDF after synthesis ☐

Design for Test
Test Ready Compile ☐
Insert Dft ☐

4.1.1.2 Checking ASIC Synthesis Results

After you have run synthesis, coreConsultant generates several reports. You can access these by clicking the Report tab in Figure 4-1 on page 92.

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.

Except in the case of the DCTCL_one_pass_compile_ultra strategy, your "final" netlist and report directories (initial, incr2, or incr2) depend on the QoR effort that you chose for your synthesis (default is Medium) or whether you chose to insert DFT (see "Inserting Design For Test" on page 109):

- Low effort – initial
- Medium effort – incr1
- High effort – 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 4-1 on page 99 includes the other files that are generated after synthesis.

1. BUTTON as circled in Figure 4-5 on page 96 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.

Table 4-1 ASIC Synthesis Output Files

Files	Purpose
./syn/final/db/DWC_usb3.ddc ./syn/final/db/DWC_usb3_top.ddc	Synopsys database files (gate level) that can be read into dc_shell for further synthesis, if desired.
./syn/final/db/DWC_usb3.sdc	Synopsys Design Constraints file.
./syn/final/db/DWC_usb3.v	Gate-level netlist that is mapped to technology libraries that you specify.
./syn/final/report/*.*	Synthesis report files.

The posedge_ffs.rpt, negedge_ffs.rpt and latches.rpt files under the <workspace>/syn/final/report/directory contain the number and names of the posedge flip-flops, negedge flip-flops, and latches, respectively. When you run Physical Synthesis, the final .ddc file is stored in the <workspace>/syn/final/db directory.

Note 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.

4.1.1.3 Troubleshooting Common Problems in This Step

- 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.
- A cell name was specified that 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.
- USB 3.0 controller cannot be synthesized with PHY. When you synthesize the USB 3.0 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 USB 3.0 Controller correctly. Therefore, the error message can be ignored.
- If you cannot achieve timing closure with the default settings, then send your configuration along with the syn/final/DWC_usb3.log file to *Synopsys Support*.
- For more troubleshooting help, see the “Troubleshooting and Support” appendix in the *Installation Guide.*

4.1.2 Synthesizing the Controller for an FPGA

Unit-level FPGA synthesis using coreConsultant is not recommended for the DWC_usb3 controller. For system-level FPGA synthesis and implementation, refer to Appendix B, "FPGA Implementation of DWC_usb3 Controller".

4.1.3 Synthesizing 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 for an ASIC" on page 93. 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, you can use either method. If you do not have access to Synopsys Design Compiler, then 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 write_sdc Command

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.

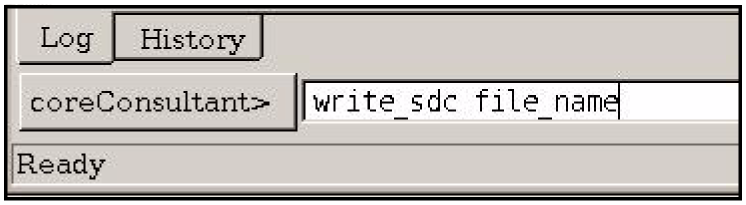
Attention When a technology library is not loaded in the **Specify Target Technology -> dc_shell target_library variable** (or **Synplicity Library Setup** in the case of an FPGA), the generated SDC file contains constraints relative to a generic technology library. You need to modify these constraints to match your target library cell names.

You invoke this command on the coreConsultant command line as follows:

```
write_sdc [-force] file_name
```

If you use the `-force` option to force the writing of the SDC file even if the **Specify Target Technology** activity (or **Synplicity Library Setup** in the case of FPGA) is not complete.

Figure 4-8 Using write_sdc command on coreConsultant Command Line



```
coreConsultant> write_sdc file_name
Ready
```

You can also generate the SDC contraints file using the Report tab. For more details refer to "Creating Optional Reports and Views" on page 33.

Method 2: Access Synopsys Design Compiler Scripts

You must first complete ALL the set-up steps in "Performing ASIC Synthesis" on page 93, and then select the **Synthesize -> Options -> Execution Options -> Generate scripts only?** option before clicking **Apply**.

Table 4-2 ASIC Synthesis Flow Files in *workspace*/syn

Synthesis Type	File	Description
ASIC	constrain/script/DWC_usb3.cscr	Primary constraints
	run.scr	Runs synthesis
	Makefile	Creates synthesis scripts

Synthesizing with Xilinx Tools

You must have access to Synopsys Synplify to synthesize and map the DWC_usb3 controller to an FPGA device using coreConsultant. Otherwise, you must run your third-party tools outside of coreConsultant.

Synthesizing with Altera Tools

You must have access to Synopsys Synplify to synthesize and map the DWC_usb3 controller to an FPGA device using coreConsultant. Otherwise you must run your third party tools outside of coreConsultant. Synopsys provides Altera-specific scripts and constraint files for use with Altera-provided synthesis and place and route tools as examples. You may need to modify them to meet the needs of your design.

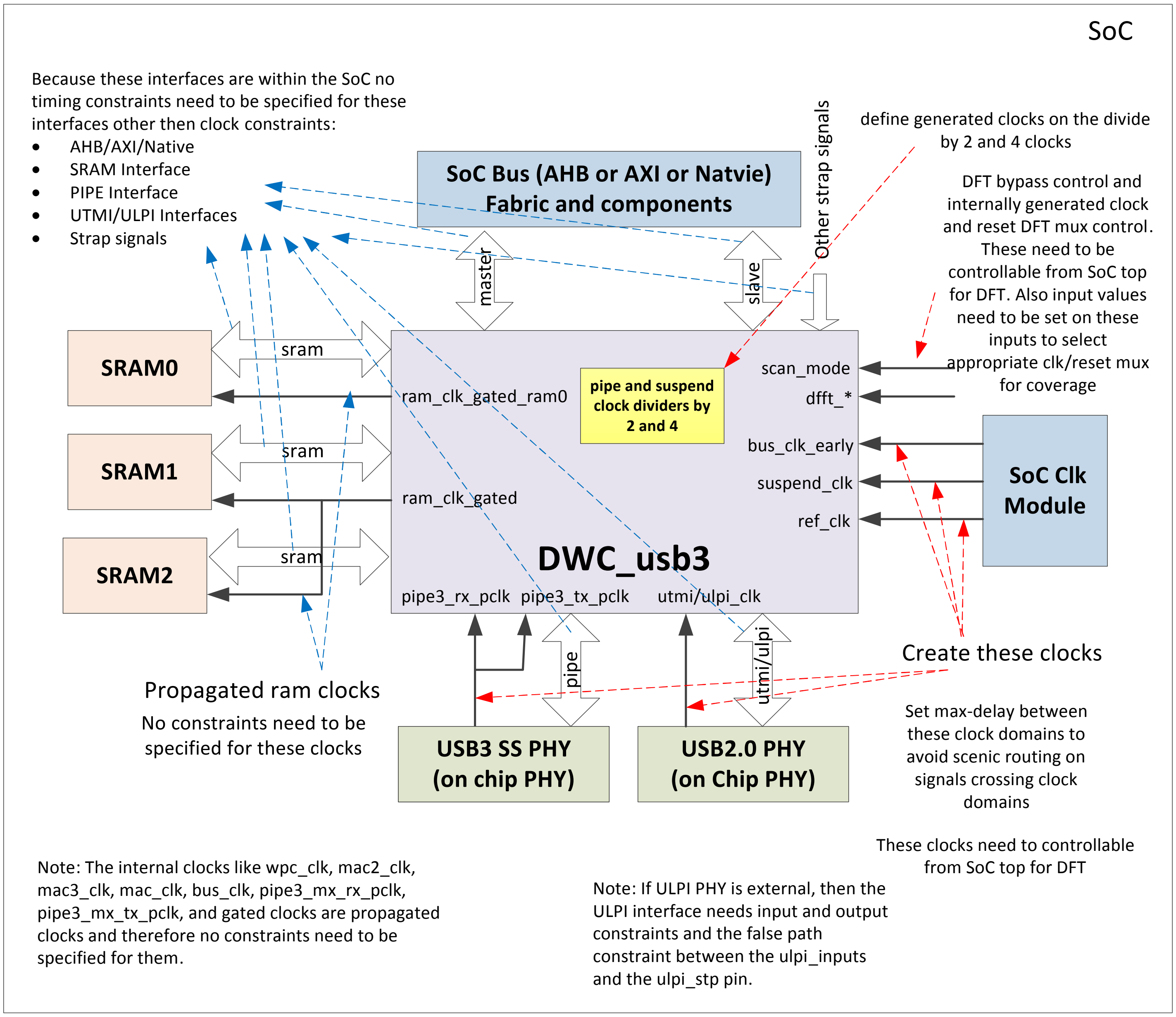
4.1.4 Synthesis Constraints for DWC_usb3 Controller

The DWC_usb3 coreConsultant environment comes with standalone synthesis scripts. When integrating the DWC_usb3 module in your SoC, you have the option of using a coreConsultant-synthesized netlist in your design or your SoC top-down synthesis with the DWC_usb3.

Because the system bus (AHB, AXI, and Native), USB PHY (PIPE3, UTMI, and ULPI), and RAM are already in the SoC, these I/O constraints are not needed. You only have to include your SoC clock constraints.

For SoC integration, refer to *workspac*>/example/DWC_usb3_sync.tcl and timing_exception.tcl scripts.

Figure 4-9 Synthesis Constraints


SoC
Because these interfaces are within the SoC no timing constraints need to be specified for these interfaces other then clock constraints:
• AHB/AXI/Native
• SRAM Interface
• PIPE Interface
• UTMI/ULPI Interfaces
• Strap signals
SoC Bus (AHB or AXI or Natvie) Fabric and components
Other strap signals
define generated clocks on the divide by 2 and 4 clocks
DFT bypass control and internally generated clock and reset DFT mux control. These need to be controllable from SoC top for DFT. Also input values need to be set on these inputs to select appropriate clk/reset mux for coverage
master
slave
SRAM0
SRAM1
SRAM2
sram
sram
sram
ram_clk_gated_ram0
ram_clk_gated
pipe and suspend clock dividers by 2 and 4
scan_mode
dfft_*
bus_clk_early
suspend_clk
ref_clk
SoC Clk Module
DWC_usb3
pipe3_rx_pclk
pipe3_tx_pclk
utmi/ulpi_clk
pipe
utmi/ulpi
Create these clocks
Propagated ram clocks
No constraints need to be specified for these clocks
USB3 SS PHY (on chip PHY)
USB2.0 PHY (on Chip PHY)
Set max-delay between these clock domains to avoid scenic routing on signals crossing clock domains
These clocks need to controllable from SoC top for DFT
Note: The internal clocks like wpc_clk, mac2_clk, mac3_clk, mac_clk, bus_clk, pipe3_mx_rx_pclk, pipe3_mx_tx_pclk, and gated clocks are propagated clocks and therefore no constraints need to be specified for them.
Note: If ULPI PHY is external, then the ULPI interface needs input and output constraints and the false path constraint between the ulpi_inputs and the ulpi_stp pin.

Example 4-1 Reference Script – DRD Configuration With ULPI and UTMI

The following is a reference script for a DRD configuration with both ULPI and UTMI interfaces:

```
#---------------------------------
# Multiple Clock Analysis
#---------------------------------
set timing_enable_multiple_clocks_per_reg true
set timing_input_port_default_clock false
set timing_all_clocks_propagated true
array set clkarr {}

#---------------------------------
# Clock Frequency specification
#---------------------------------
set bus_clk_pe 5
set bus_clk_uncertainty [expr 0.025 * $bus_clk_pe]
set bus_clk_high [expr 0.5 * $bus_clk_pe]

set utmi_clk_pe 16.66
set utmi_clk_uncertainty [expr 0.025 * $utmi_clk_pe]
set utmi_clk_high [expr 0.5 * $utmi_clk_pe]

set ulpi_clk_pe 16.66
set ulpi_clk_uncertainty [expr 0.025 * $ulpi_clk_pe]
set ulpi_clk_high [expr 0.5 * $ulpi_clk_pe]

set pipe3_pclk_pe 4.0
set pipe3_pclk_uncertainty [expr 0.025 * $pipe3_pclk_pe]
set pipe3_pclk_high [expr 0.5 * $pipe3_pclk_pe]

set suspend_clk_pe 4.0
set suspend_clk_uncertainty [expr 0.025 * $suspend_clk_pe]
set suspend_clk_high [expr 0.5 * $suspend_clk_pe]

set ref_clk_pe 4.0
set ref_clk_uncertainty [expr 0.025 * $ref_clk_pe]
set ref_clk_high [expr 0.5 * $ref_clk_pe]

#---------------------------------
# Clock specification
#---------------------------------
# bus_clk_early clock
create_clock -period $bus_clk_pe bus_clk_early -wave [list 0 $bus_clk_high]
set_clock_uncertainty $bus_clk_uncertainty bus_clk_early
set clkarr(bus_clks)  [list bus_clk_early]

# suspend_clk
create_clock -period $suspend_clk_pe suspend_clk -wave [list 0 $suspend_clk_high]
set_clock_uncertainty $suspend_clk_uncertainty suspend_clk
set clkarr(suspend_clks) [list suspend_clk]

# pipe3 clk
create_clock -period $pipe3_pclk_pe -name pipe3_rx_pclk[0] pipe3_rx_pclk[0] -wave [list
0 $pipe3_pclk_high]
create_clock -period $pipe3_pclk_pe -name pipe3_tx_pclk[0] pipe3_tx_pclk[0] -wave [list
0 $pipe3_pclk_high]
```

```
 # UTMI clk
 create_clock -period $utmi_clk_pe utmi_clk[0] -wave [list 0 $utmi_clk_high]
 set_clock_uncertainty $utmi_clk_uncertainty utmi_clk[0]
 set clkarr(utmi_clks)  [list utmi_clk[0]]

 #ULPI clk
 create_clock -period $ulpi_clk_pe ulpi_clk[0] -wave [list 0 $ulpi_clk_high]
 set_clock_uncertainty $ulpi_clk_uncertainty ulpi_clk[0]
 set clkarr(ulpi_clks)  [list ulpi_clk[0]]

 # ref_clk
 create_clock -period $pipe3_pclk_pe ref_clk -wave [list 0 $pipe3_pclk_high]
 set_clock_uncertainty $pipe3_pclk_uncertainty ref_clk
 set clkarr(ref_clks)  [list ref_clk]

 # Divide by 2 and 4 clocks with respect to pipe3_rx_pclk and suspend_clk. Need to be
specified when USB 2.0-only mode is not selected (DWC_USB3_EN_USB2_ONLY==0)
 create_generated_clock -name pipe3_div2 -source [get_attribute [get_clocks
pipe3_rx_pclk[0]] sources] -divide_by 2 [get_pins
U_DWC_usb3_clk/U_DWC_usb3_clk_div_2_4/clk_div_2] -master_clock pipe3_rx_pclk[0] -add
 create_generated_clock -name pipe3_div4 -source [get_attribute [get_clocks
pipe3_rx_pclk[0]] sources] -divide_by 4 [get_pins
U_DWC_usb3_clk/U_DWC_usb3_clk_div_2_4/clk_div_4] -master_clock pipe3_rx_pclk[0] -add
 set clkarr(pipe3_pclks)  [list pipe3_rx_pclk[0] pipe3_tx_pclk[0] pipe3_div2 pipe3_div4]

 create_generated_clock -name suspend_div2 -source [get_attribute [get_clocks
suspend_clk] sources] -divide_by 2 [get_pins
U_DWC_usb3_clk/U_DWC_usb3_clk_div_2_4/clk_div_2] -master_clock suspend_clk -add
 create_generated_clock -name suspend_div4 -source [get_attribute [get_clocks
suspend_clk] sources] -divide_by 4 [get_pins
U_DWC_usb3_clk/U_DWC_usb3_clk_div_2_4/clk_div_4] -master_clock suspend_clk -add
 set clkarr(suspend_clks)  [list suspend_clk suspend_div2 suspend_div4]

 # Set Exclusive clocks - Instead use the max-delay based constrains as shown next
 #set_clock_groups -name clk_all -logically_exclusive -group [get_clocks bus_clk_early] -
group [get_clocks [list pipe3_rx_pclk[0] pipe3_tx_pclk[0] pipe3_div2 pipe3_div4]] -group
[get_clocks [list suspend_clk suspend_div2 suspend_div4]] -group [get_clocks utmi_clk[0]]
-group [get_clocks ulpi_clk[0]] -group [get_clocks ref_clk]

 # Set Max-delay Constrain between clock-domains
 # "get_attribute [get_clocks $clk1] period" does not work for generated clock in a
script but work in a command line only and have to seperate them
 set clklst [array names clkarr]
 for {set i 0} {$i < [llength $clklst]} {incr i} {
   for {set j 0} {$j < [llength $clklst]} {incr j} {
      foreach clk1 $clkarr([lindex $clklst $i]) {
         foreach clk2 $clkarr([lindex $clklst $j]) {
          if {($clk1 != $clk2) && (!([regexp "pipe3_" $clk1] && [regexp "pipe3_" $clk2]))
&& (!([regexp "suspend" $clk1] && [regexp "suspend" $clk2]))} {
           if {[regexp "pipe3_div2" $clk1]} {
           echo "set_max_delay \[expr 2.0 * $pipe3_pclk_pe\] -from \[get_clocks $clk2\] -
to \[get_clocks $clk1\]"
           set_max_delay [expr 2.0 * $pipe3_pclk_pe] -from [get_clocks $clk2] -to
[get_clocks $clk1]
         } elseif {[regexp "pipe3_div4" $clk1]} {
           echo "set_max_delay \[expr 4.0 * $pipe3_pclk_pe\] -from \[get_clocks $clk2\] -
to \[get_clocks $clk1\]"
```

```
              set_max_delay [expr 4.0 * $pipe3_pclk_pe] -from [get_clocks $clk2] -to
[get_clocks $clk1]
            } elseif {[regexp "suspend_div2" $clk1]} {
              echo "set_max_delay \[expr 2.0 * $suspend_clk_pe\] -from \[get_clocks $clk2\]
-to \[get_clocks $clk1\]"
              set_max_delay [expr 2.0 * $suspend_clk_pe] -from [get_clocks $clk2] -to
[get_clocks $clk1]
            } elseif {[regexp "suspend_div4" $clk1]} {
              echo "set_max_delay \[expr 4.0 * $suspend_clk_pe\] -from \[get_clocks $clk2\]
-to \[get_clocks $clk1\]"
              set_max_delay [expr 4.0 * $suspend_clk_pe] -from [get_clocks $clk2] -to
[get_clocks $clk1]
            } else {
              echo "set_max_delay [expr 1.0 * [get_attribute [get_clocks $clk1] period]] -
from \[get_clocks $clk2]\ -to \[get_clocks $clk1\]"
              set_max_delay [expr 1.0 * [get_attribute [get_clocks $clk1] period]] -from
[get_clocks $clk2] -to [get_clocks $clk1]
            }
            set_false_path -from [get_clocks $clk2] -to [get_clocks $clk1] -hold
            }
          }
        }
      }
}
 # report
 #report_transitive_fanout -from [get_pins
U_DWC_usb3_clk/U_DWC_usb3_clk_div_2_4/clk_div_2] > div0_rep
 #report_transitive_fanout -from [get_pins
U_DWC_usb3_clk/U_DWC_usb3_clk_div_2_4/clk_div_4] > div1_rep
 report_transitive_fanout -clock_tree > all_rep
```

4.1.4.1 Clock Gating Cell

DWC_usb3 controller uses a clock gating cell which could be mapped to a library specific clock gating cell. The clock gating cell is specified by the module DWC_usb3_clkgate_leaf_cell.v and can be found in the src/com directory.

4.1.4.2 Timing Checks for Reset Signals

During reset path timing analysis, ensure that reset timing checks for the following paths are not ignored:

- Start Point(s):
  - U_DWC_usb3_clk/U_DWC_usb3_sync_ctl_mac3_reset/in_p_2d
  - U_DWC_usb3_clk/U_DWC_usb3_sync_ctl_mac2_reset/in_p_2d

  End Point: U_DWC_usb3_clk/mac2_clk_reset_n
- Start Point(s): U_DWC_usb3_clk/U_DWC_usb3_sync_ctl_pipe3_common_reset/in_p_2d

  End Point: U_DWC_usb3_clk/pipe3_common_clk_reset_n
- Start Point(s): U_DWC_usb3_clk/U_DWC_usb3_sync_ctl_suspend_reset/in_p_2d

  End Point: U_DWC_usb3_clk/suspend_clk_reset_n
- Start Point(s): U_DWC_usb3_rst/U_DWC_usb3_sync_ctl_mhgswr/in_p_2d

  End Point: U_DWC_usb3_rst/mac_hwr_gswr_rst_n
- Start Point(s): U_DWC_usb3_rst/U_DWC_usb3_sync_ctl_mac/in_p_2d

  End Point: U_DWC_usb3_rst/mac_reset_n
- Start Point(s): U_DWC_usb3_rst/U_DWC_usb3_sync_ctl_mac_core/in_p_2d

  End Point: U_DWC_usb3_rst/mac_core_reset_n
- Start Point(s): U_DWC_usb3_rst/U_DWC_usb3_sync_ctl_mac2/in_p_2d

  End Point: U_DWC_usb3_rst/mac2_reset_n
- Start Point(s): U_DWC_usb3_rst/U_DWC_usb3_sync_ctl_mac2_core/in_p_2d

  End Point: U_DWC_usb3_rst/mac2_core_reset_n
- Start Point(s): U_DWC_usb3_rst/U_DWC_usb3_sync_ctl_ram/in_p_2d

  End Point: U_DWC_usb3_rst/ram_reset_n
- Start Point(s): U_DWC_usb3_rst/U_DWC_usb3_sync_ctl_ram_hw/in_p_2d

  End Point: U_DWC_usb3_rst/ram_reset_hw_n
- Start Point(s): U_DWC_usb3_rst/U_DWC_usb3_sync_ctl_wpc/in_p_2d

  End Point: U_DWC_usb3_rst/wpc_rst_n
- Start Point(s): U_DWC_usb3_rst/U_DWC_usb3_sync_ctl_whgswr/in_p_2d

  End Point: U_DWC_usb3_rst/wpc_hwr_gswr_rst_n
- Start Point(s): U_DWC_usb3_rst/U_DWC_usb3_sync_ctl_mac3/in_p_2d

  End Point: U_DWC_usb3_rst/mac3_reset_n
- Start Point(s): U_DWC_usb3_rst/U_DWC_usb3_sync_ctl_mac3_core/in_p_2d

  End Point: U_DWC_usb3_rst/mac3_core_reset_n

- Start Point(s): U_DWC_usb3_rst/U_DWC_usb3_sync_ctl_ulpi/in_p_2d

  End Point: U_DWC_usb3_rst/ulpi_rst_n
- Start Point(s): U_DWC_usb3_rst/U_DWC_usb3_sync_ctl_utmi/in_p_2d

  End Point: U_DWC_usb3_rst/utmi_rst_n
- Start Point(s): U_DWC_usb3_rst/U_DWC_usb3_sync_ctl_bus_vcc/in_p_2d

  End Point: U_DWC_usb3_rst/bus_reset_n
- Start Point(s): U_DWC_usb3_rst/U_DWC_usb3_sync_ctl_bus_vcc_soft/in_p_2d

  End Point: U_DWC_usb3_rst/bus_reset_gen_n
- Start Point(s):
  - U_DWC_usb3_noclkrst/U_DWC_usb3_pwrm/U_DWC_usb3_pwrm_csr/dctl_reg[30]
  - U_DWC_usb3_noclkrst/U_DWC_usb3_pwrm/U_DWC_usb3_pwrm_csr/gctl_reg[11]
  - U_DWC_usb3_noclkrst/U_DWC_usb3_pwrm/U_DWC_usb3_pwrm_csr/hc_software_rst_st[3:0]

  End Point: U_DWC_usb3_rst/bus_hwr_gswr_rst_int_n
- Start Point(s): U_DWC_usb3_rst/U_DWC_usb3_sync_ctl_bhgswr/in_p_2d

  End Point: U_DWC_usb3_rst/bus_hwr_gswr_rst_n
- Start Point(s):
  - U_DWC_usb3_noclkrst/U_DWC_usb3_pwrm/U_DWC_usb3_pwrm_csr/dctl_reg[30]
  - U_DWC_usb3_noclkrst/U_DWC_usb3_pwrm/U_DWC_usb3_pwrm_csr/gctl_reg[11]
  - U_DWC_usb3_noclkrst/U_DWC_usb3_pwrm/U_DWC_usb3_pwrm_csr/hc_software_rst_st[3:0]
  - U_DWC_usb3_noclkrst/U_DWC_usb3_pwrm/U_DWC_usb3_pwrm_csr/pmgt_light_soft_reset_n

  End Point: U_DWC_usb3_rst/bus_core_reset_byp_n
- Start Point(s): U_DWC_usb3_rst/U_DWC_usb3_sync_ctl_bus_core/in_p_2d

  End Point: U_DWC_usb3_rst/bus_core_reset_n
- Start Point(s):
  - U_DWC_usb3_noclkrst/U_DWC_usb3_pwrm/U_DWC_usb3_pwrm_csr/dctl_reg[30]
  - U_DWC_usb3_noclkrst/U_DWC_usb3_pwrm/U_DWC_usb3_pwrm_csr/gctl_reg[11]
  - U_DWC_usb3_noclkrst/U_DWC_usb3_pwrm/U_DWC_usb3_pwrm_csr/hc_software_rst_st[3:0]

  End Point: U_DWC_usb3_rst/bus_reset_gen_byp_n
- Start Point(s):
  - U_DWC_usb3_noclkrst/U_DWC_usb3_pwrm/U_DWC_usb3_pwrm_csr/dctl_reg[30]
  - U_DWC_usb3_noclkrst/U_DWC_usb3_pwrm/U_DWC_usb3_pwrm_csr/gctl_reg[11]
  - U_DWC_usb3_noclkrst/U_DWC_usb3_pwrm/U_DWC_usb3_pwrm_csr/hc_software_rst_st[3:0]

  End Point: U_DWC_usb3_rst/bypass_bus_pmgt_soft_reset_n

- Start Point(s):
  - U_DWC_usb3_noclkrst/U_DWC_usb3_pwrm/U_DWC_usb3_pwrm_csr/dctl_reg[30]
  - U_DWC_usb3_noclkrst/U_DWC_usb3_pwrm/U_DWC_usb3_pwrm_csr/gctl_reg[11]
  - U_DWC_usb3_noclkrst/U_DWC_usb3_pwrm/U_DWC_usb3_pwrm_csr/hc_software_rst_st[3:0]
  - U_DWC_usb3_noclkrst/U_DWC_usb3_pwrm/U_DWC_usb3_pwrm_csr/pmgt_light_soft_reset_n

  End Point: U_DWC_usb3_rst/core_pmgt_light_soft_reset_n
- Start Point(s):
  - U_DWC_usb3_noclkrst/U_DWC_usb3_pwrm/U_DWC_usb3_pwrm_csr/dctl_reg[30]
  - U_DWC_usb3_noclkrst/U_DWC_usb3_pwrm/U_DWC_usb3_pwrm_csr/gctl_reg[11]
  - U_DWC_usb3_noclkrst/U_DWC_usb3_pwrm/U_DWC_usb3_pwrm_csr/hc_software_rst_st[3:0]
  - U_DWC_usb3_noclkrst/U_DWC_usb3_pwrm/U_DWC_usb3_pwrm_csr/gusb3pipectl_reg[31]

  End Point: U_DWC_usb3_rst/pipe3_reset_n
- Start Point(s): U_DWC_usb3_rst/U_DWC_usb3_sync_ctl_pipe3/in_p_2d

  End Point: U_DWC_usb3_rst/pipe3_int_reset_n
- Start Point(s): U_DWC_usb3_rst/U_DWC_usb3_sync_ctl_sof_clk/in_p_2d

  End Point: U_DWC_usb3_rst/ref_clk_reset_n
- Start Point(s): U_DWC_usb3_rst/U_DWC_usb3_sync_ctl_suspend/in_p_2d

  End Point: U_DWC_usb3_rst/suspend_reset_n

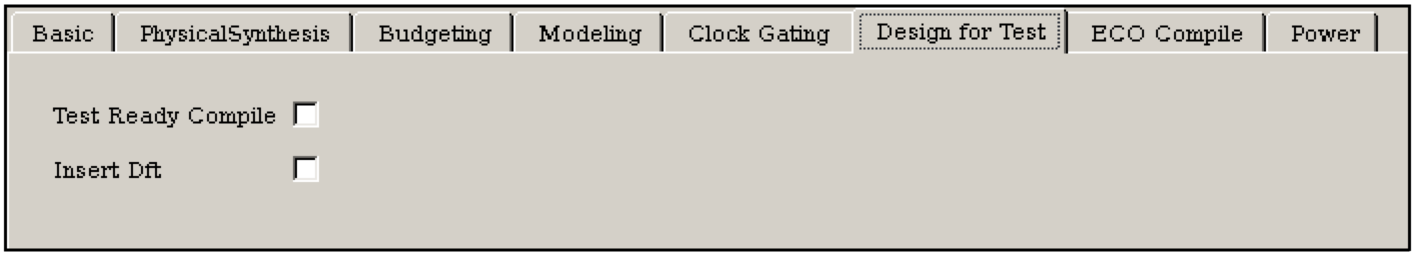
4.2 Inserting Design For Test

This activity is not applicable to FPGA implementations.

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. See page 94.
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 4-5 on page 96). 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.

Figure 4-10 Design For Test Tab



Test Ready Compile ☐

Insert Dft ☐

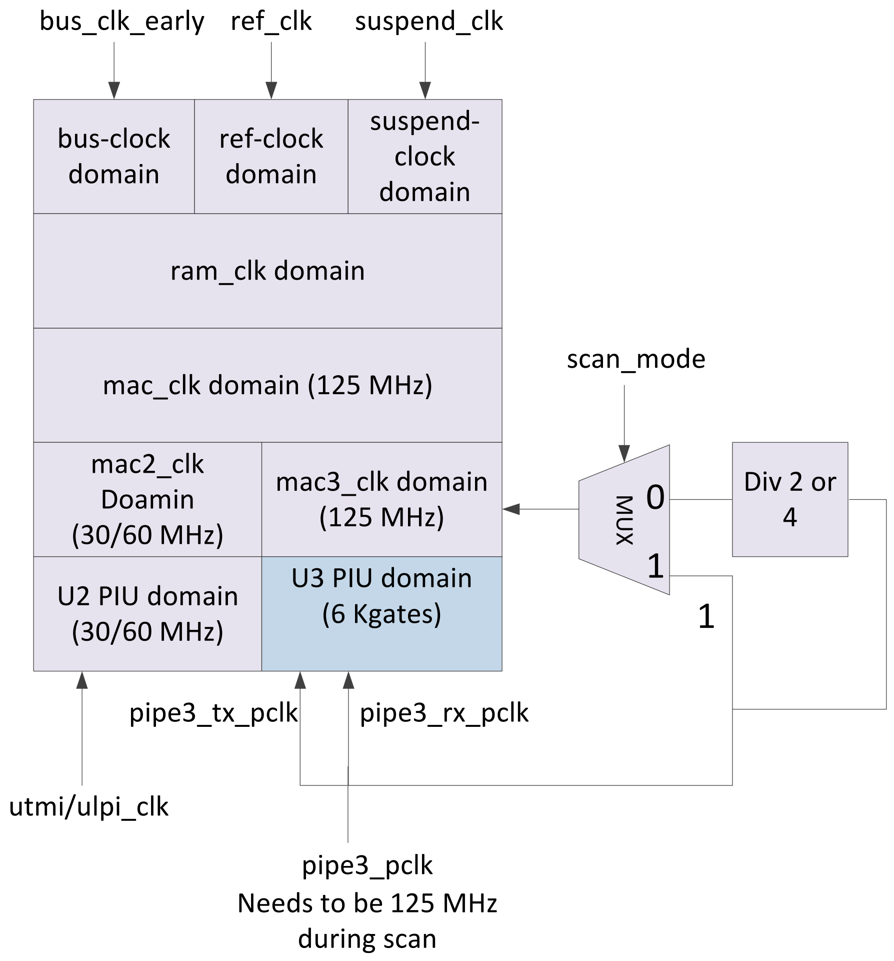
The DFT results are saved in the <workspace>/syn/dft/ directory.

4.2.1 Clock Frequency Requirements for DFT

By default, the DWC_usb3_controller supports at-speed Design for Test (DFT) by driving the appropriate clock frequency for each clock domain. All clocks used in DWC_usb3 modules are visible at the top so that you can apply appropriate clock frequencies to these clocks during at-speed testing. The only exception is the DWC_usb3_u3piu module (6 KGates) if your PIPE interface runs in 8-bit/16-bit mode where the pipe3_pclk is at 500 MHz/250 MHz.

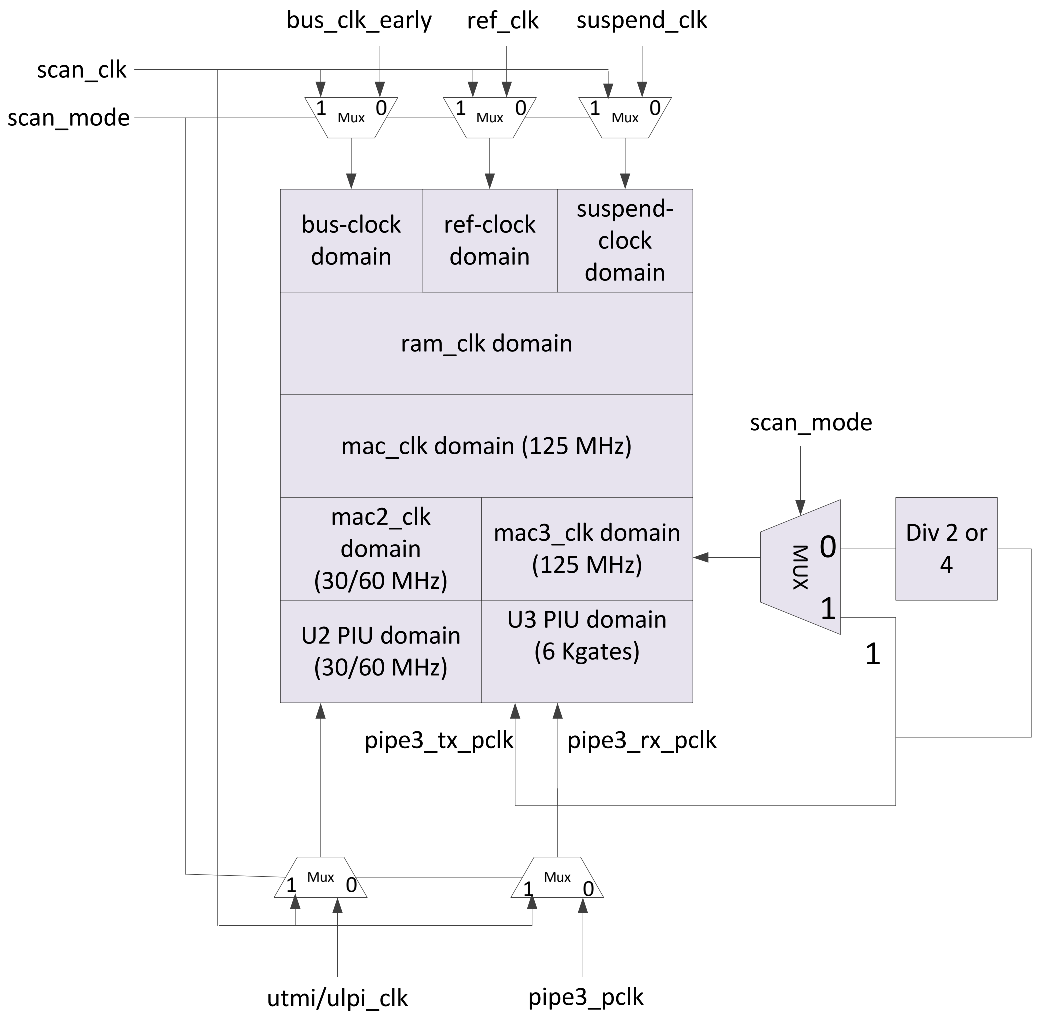
Because a divided pipe3_pclk is used for mac3_domain (125 MHz), which is bypassed during scan, the DWC_usb3_u3piu also runs at 125 MHz, and the pipe3_pclk to the controller needs to be 125 MHz during at-speed scan testing. If you need to run DWC_usb3_u3piu module at the required clock, you can synthesize mac3_domain at the pipe3_pclk frequency instead at 125 MHz and run both the DWC_usb3_u3piu and mac3_clock domains at pipe3_pclk frequency, or you can use OCC flow defined in the chapter "On-Chip Clocking Support" of the *Synopsys DFT Compiler Scan User Guide*. If you are using a 32-bit PIPE interface, the PHY's pipe3_clk is always 125 MHz and there is no exception in this configuration.

Figure 4-11 At-Speed DFT



If you do not need at-speed DFT, you can multiplex the required scan clock into all of the clock inputs of the DWC_usb3 module. This needs to be done outside the DWC_usb3_module.

Figure 4-12 Non-At-Speed DFT



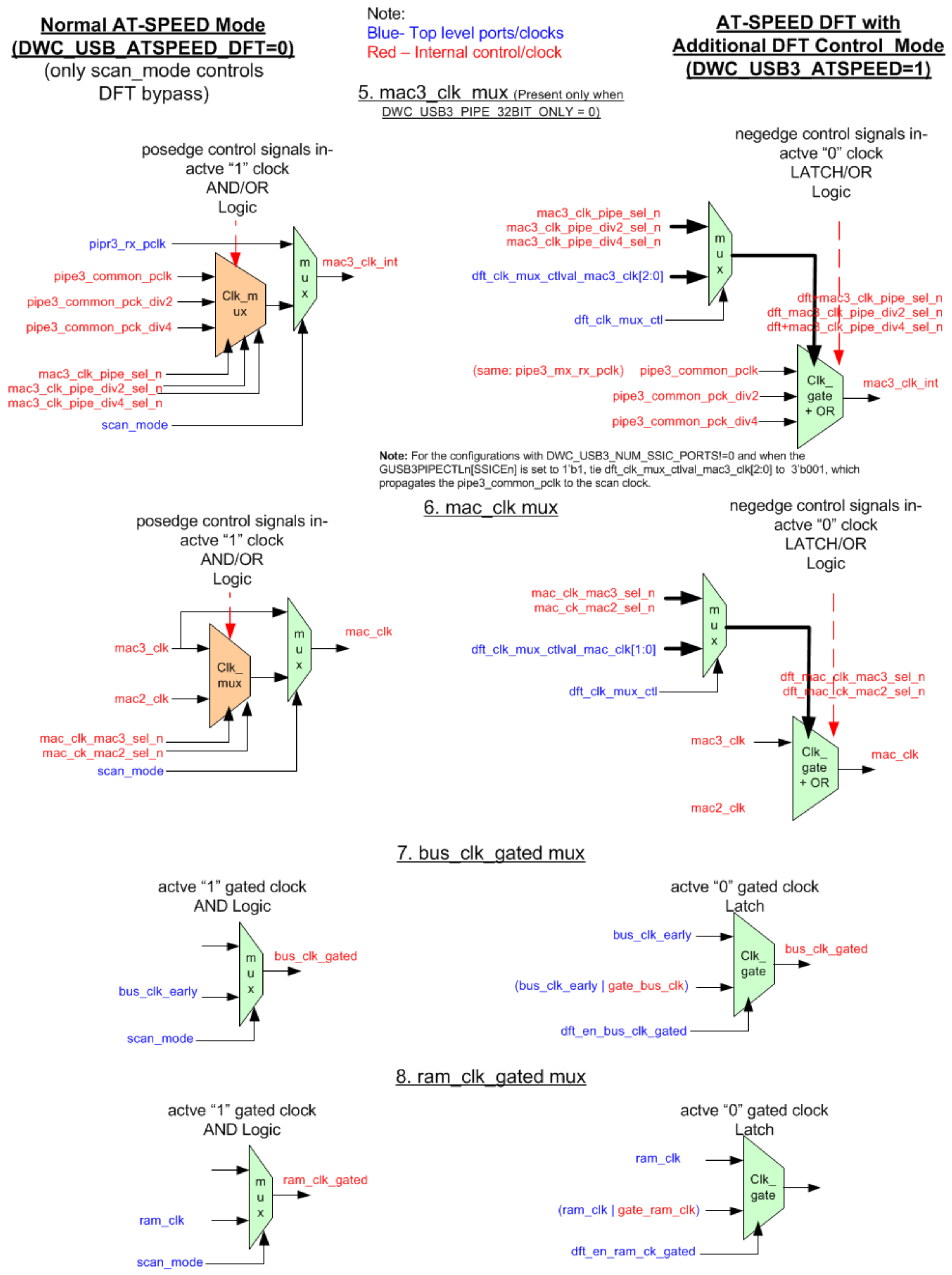
The DWC_usb3 module has internally generated clock and reset signals. All of these clocks and resets can be bypassed/controlled by the top-level ports in the DWC_usb3.v module. These ports must be controllable from your top-level design to get DFT coverage, otherwise, uncontrollable warnings and reduced coverage may occur.

When the “Enable Additional DFT Control Ports” coreConsulant parameter is set to “No” (DWC_USB3_ATSPEED_DFT = 0), the “scan_mode” pin is used to by-pass all the internally generated clocks and resets during scan mode. The “scan_mode” must be set to ‘1” during scan.

When the “Enable Additional DFT Control Ports” coreConsulant parameter is set to “Yes” (DWC_USB3_ATSPEED_DFT = 1), the scan_mode, dft_reset_mux_ctl, dft_clk_mux_static_ctl, and dft_clk_mux_ctl control signals are used to control the bypass. These must be set to “1” during scan. The dft_clk_mux_ctlval* signals control the MUX-select of the internally generated clocks.

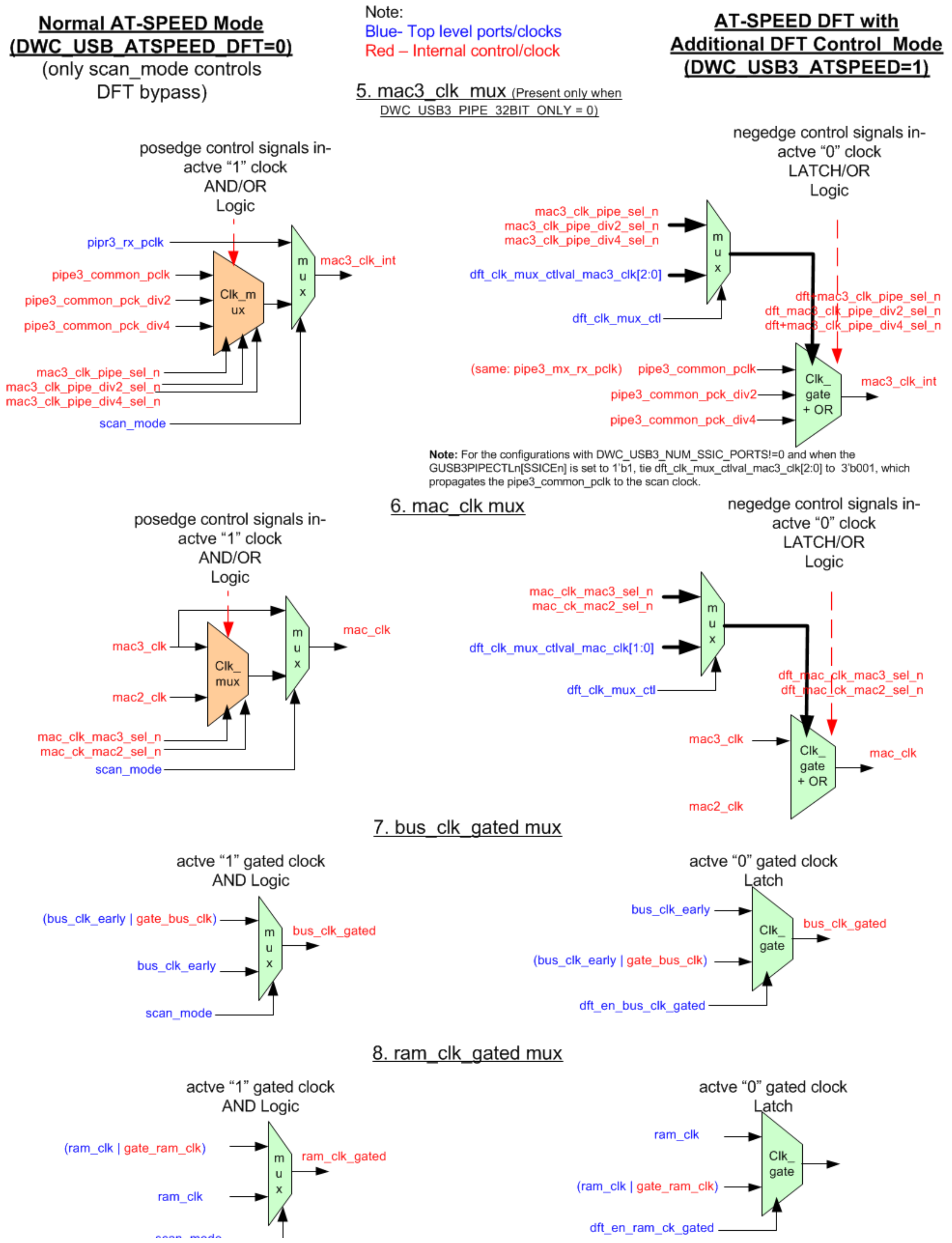
Figure 4-13 on page 112, and Figure 4-14 on page 113 show the DFT bypass/control details when the “Enable Additional DFT Control Ports” configuration parameter is set to both “No” and “Yes”. Figure 4-15 on page 114 shows the bus_clk_mux, and wpc_clk_mux specific to the hibernation mode (DWC_USB3_EN_PWROPT=2).

Figure 4-13 DFT Bypass/Control Details when “Enable Additional DFT Control Ports” Set to No



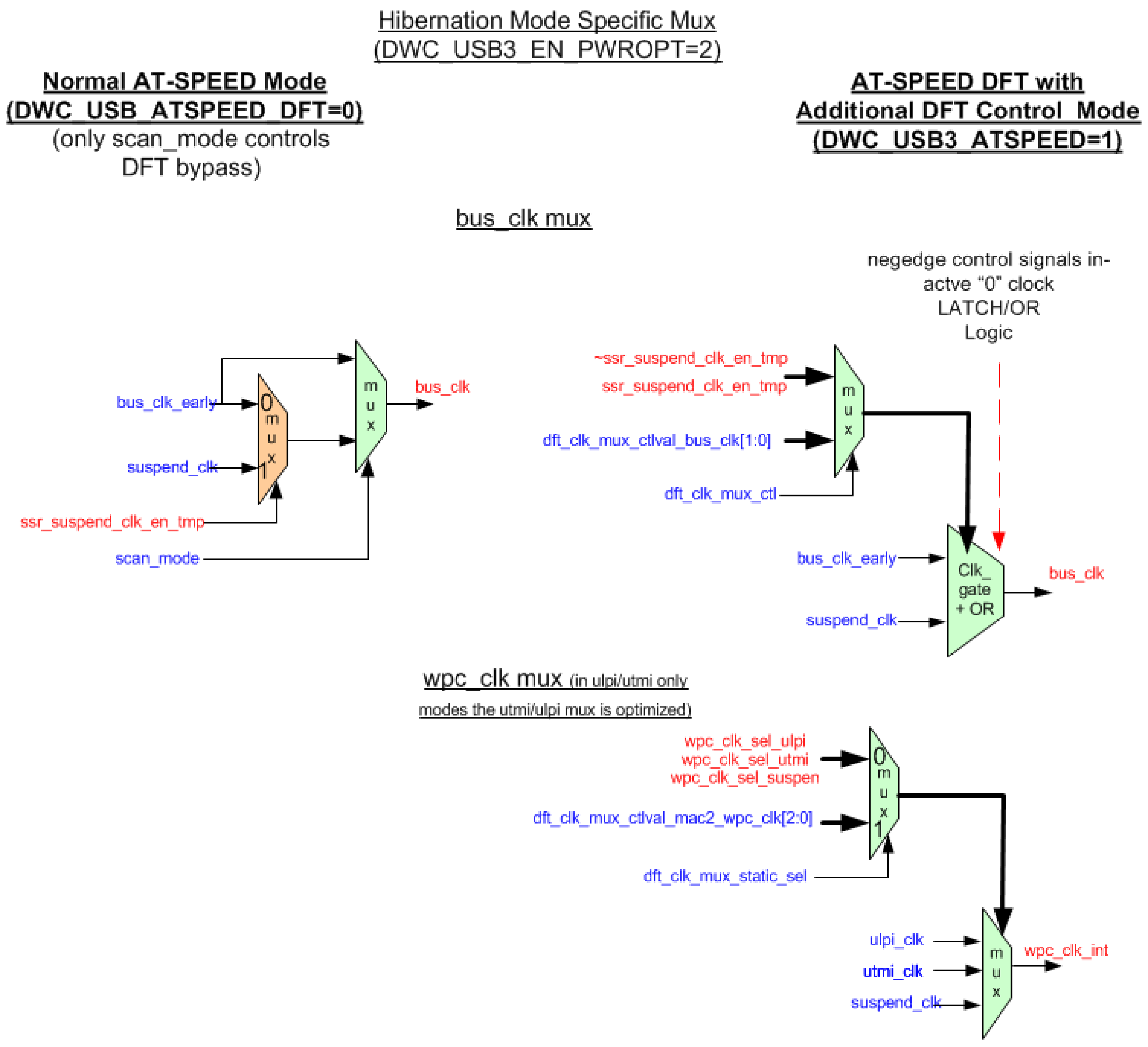
Note: For the configurations with DWC_USB3_NUM_SSIC_PORTS!=0 and when the GUSB3PIPECTLn[SSICEn] is set to 1'b1, tie dft_clk_mux_ctlval_mac3_clk[2:0] to 3'b001, which propagates the pipe3_common_pclk to the scan clock.

Figure 4-14 DFT Bypass/Control Details when “Enable Additional DFT Control Ports” Set to Yes



Note: For the configurations with DWC_USB3_NUM_SSIC_PORTS!=0 and when the GUSB3PIPECTLn[SSICEn] is set to 1'b1, tie dft_clk_mux_ctlval_mac3_clk[2:0] to 3'b001, which propagates the pipe3_common_pclk to the scan clock.

Figure 4-15 Hibernation Mode MUX when DWC_USB3_EN_PWROPT Set to 2



Example 4-2 Recommended SoC-Level DFT Script

The following is a recommended SoC-level DFT script.

```
#-------------
 # Scan Insertion
 #----------------
 if {  $scan_insertion == 1 } {
 set test_enable_dft_drc true
 set save_tmax_logs true
 sh rm -rf new_cache
 sh mkdir new_cache
 set test_dft_drc_cache ./new_cache
 write -f ddc -hier -o [format "%s%s"  $db_name "_predft.ddc"]

 # define clocks 45 55 instead 45 55 since 4 flops in DWC_usb3_demo.v_clk.v sample on
negedge (posedge generated)
 if { $DWC_USB3_MODE != 3 && $DWC_USB3_MBUS_TYPE != 4 } {
 set_dft_signal -view existing_dft -type ScanClock -timing [list 45 55] -port
bus_clk_early
 }

 if { $DWC_USB3_EN_PWROPT == 2 || $DWC_USB3_PIPE3 != 0 } {
 set_dft_signal -view existing_dft -type ScanClock -timing [list 45 55] -port suspend_clk
 }

 if { $DWC_USB3_MODE != 0 && $DWC_USB3_MODE != 3 || $DWC_USB3_NUM_SSIC_PORTS != 0} {
 set_dft_signal -view existing_dft -type ScanClock -timing [list 45 55] -port ref_clk
 }

 if { $DWC_USB3_JTAG == 1 } {
 set_dft_signal -view existing_dft -type ScanClock -timing [list 45 55] -port jtag_tclk
 }

 if { $DWC_USB3_PIPE3 != 0  } {
     for { set i 0 } { $i < $DWC_USB3_NUM_U3_ROOT_PORTS}  { incr i } {
       set_dft_signal -view existing_dft -type ScanClock -timing [list 45 55] -port
pipe3_rx_pclk[$i]
       set_dft_signal -view existing_dft -type ScanClock -timing [list 45 55] -port
pipe3_tx_pclk[$i]
     }
   }
   if { $DWC_USB3_UTMI == 1 } {
     for { set i 0 } { $i < $DWC_USB3_NUM_U2_ROOT_PORTS}  { incr i } {
       set_dft_signal -view existing_dft -type ScanClock -timing [list 45 55] -port
utmi_clk[$i]
     }
   }
   if { $DWC_USB3_ULPI == 1 } {
     for { set i 0 } { $i < $DWC_USB3_NUM_U2_ROOT_PORTS}  { incr i } {
       set_dft_signal -view existing_dft -type ScanClock -timing [list 45 55] -port
ulpi_clk[$i]
     }
   }

if { $DWC_USB3_SSIC_EN == 1 } {
       set_dft_signal -view existing_dft -type ScanClock -timing [list 45 55] -port
ssic_soc_pa_clk
```

```
   for { set i 0 } { $i < $DWC_USB3_NUM_SSIC_PORTS}  { incr i } {
       set_dft_signal -view existing_dft -type ScanClock -timing [list 45 55] -port 
RMMI_TX_SymbolClk[$i]
   }
   for { set i 0 } { $i < $DWC_USB3_SSIC_NUM_LANE}  { incr i } {
       set_dft_signal -view existing_dft -type ScanClock -timing [list 45 55] -port 
RMMI_RX_SymbolClk[$i]
   }
 }


if { $DWC_USB3_JTAG == 1 } {
set_dft_signal -view existing_dft -type ScanClock -timing [list 45 55] -port jtag_tclk
}


create_port -direction "in" [list scan_en]
set_dft_signal -view spec -port [find port "scan_en"] -type ScanEnable

if {  $numberScanChains > 1 } {
 set count 0
 while {  $count < $numberScanChains } {
  create_port [format "%s%s"  [format "%s%s"  {scan_in[} $count] "\]"] -direction "in"
  create_port [format "%s%s"  [format "%s%s"  {scan_out[} $count] "\]"] -direction "out"
 set count [expr $count + 1]
     }
     create_bus {scan_in[*} scan_in
     create_bus {scan_out[*} scan_out
   } else {
      create_port scan_in -direction "in"
     create_port scan_out -direction "out"
 }
   set_dft_signal -view spec -port [find port "scan_in*"] -type ScanDataIn
   set_dft_signal -view spec -port [find port "scan_out*"] -type ScanDataOut
   set_drive 0 [list scan_en]
   set_max_delay 16 -from [list scan_en]
   set_max_delay 16 -from [list scan_in]
   set_max_delay 16 -to [list scan_out]
   set_dft_signal -view existing_dft -type Constant -active_state 1 -port [list 
scan_mode]
if { $DWC_USB3_ATSPEED_DFT == 1 } {
    set_dft_signal -view existing_dft -type Constant -active_state 1 -port [list 
dft_reset_mux_ctl]
   set_dft_signal -view existing_dft -type Constant -active_state 1 -port [list 
dft_clk_mux_static_ctl]
   set_dft_signal -view existing_dft -type Constant -active_state 1 -port [list 
dft_clk_mux_ctl]

# Select ulpi clock if enabled or else select utmi clock
   if { $DWC_USB3_ULPI == 1 } {
      set_dft_signal -view existing_dft -type Constant -active_state 1 -port 
   [list dft_clk_mux_ctlval_mac2_wpc_clk]
   } else {
   set_dft_signal -view existing_dft -type Constant -active_state 0 -port [list 
   dft_clk_mux_ctlval_mac2_wpc_clk]
   }

# Select pipe3_rx_pclk
```

```
    set_dft_signal -view existing_dft -type Constant -active_state 1 -port [list
dft_clk_mux_ctlval_rx_mx_pclk[0]]
   set_dft_signal -view existing_dft -type Constant -active_state 0 -port [list
dft_clk_mux_ctlval_rx_mx_pclk[1]]

# Select pipe3_tx_pclk
   set_dft_signal -view existing_dft -type Constant -active_state 1 -port [list
dft_clk_mux_ctlval_tx_mx_pclk[0]]
   set_dft_signal -view existing_dft -type Constant -active_state 0 -port [list
dft_clk_mux_ctlval_tx_mx_pclk[1]]

# Select bus_clk_early
   set_dft_signal -view existing_dft -type Constant -active_state 0 -port [list
dft_clk_mux_ctlval_ram_clk[0]]
   set_dft_signal -view existing_dft -type Constant -active_state 0 -port [list
dft_clk_mux_ctlval_ram_clk[1]]
   set_dft_signal -view existing_dft -type Constant -active_state 0 -port [list
dft_clk_mux_ctlval_ram_clk[2]]
   set_dft_signal -view existing_dft -type Constant -active_state 1 -port [list
dft_clk_mux_ctlval_ram_clk[3]]

# Select pipe3_rx_pclk[0]
   set_dft_signal -view existing_dft -type Constant -active_state 1 -port [list
dft_clk_mux_ctlval_mac3_clk[0]]
   set_dft_signal -view existing_dft -type Constant -active_state 0 -port [list
dft_clk_mux_ctlval_mac3_clk[1]]
   set_dft_signal -view existing_dft -type Constant -active_state 0 -port [list
dft_clk_mux_ctlval_mac3_clk[2]]

# select mac3_clk for mac_clk
   set_dft_signal -view existing_dft -type Constant -active_state 1 -port [list
dft_clk_mux_ctlval_mac_clk[0]]
   set_dft_signal -view existing_dft -type Constant -active_state 0 -port [list
dft_clk_mux_ctlval_mac_clk[1]]
   set_dft_signal -view existing_dft -type Constant -active_state 0 -port [list
dft_clk_mux_ctlval_mac_clk[2]]

# Always enable gated-clocks
   set_dft_signal -view existing_dft -type Constant -active_state 1 -port [list
dft_en_bus_clk_gated]
   set_dft_signal -view existing_dft -type Constant -active_state 1 -port [list
dft_en_ram_clk_gated]
 }

   set test_dont_fix_constraint_violations true
   create_test_protocol -infer_async
   dft_drc -verbose
   preview_dft > scan_path_info
   preview_dft -show all
   set_dft_insertion_configuration -synthesis_optimization none
   set_dft_insertion_configuration -map_effort high
   insert_dft
   dft_drc -verbose
   dft_drc -verbose -coverage_estimate
```

4.3 Synthesizing Using UPF Power Intent

When hibernation mode is enabled, the DWC_usb3 can be powered down leaving only the PMU modules enabled (ON). An example UPF flow is described in this section.

- "Power Intent Specification"
- "Power Intent Specification for DWC_usb3"
- "Running Synthesis using UPF Power Intent" on page 120
- "Formality Checks using UPF Power Intent" on page 124
- "Gate-level Simulations Using UPF Power Intent" on page 124
- "PVETests"

4.3.1 Power Intent Specification

The power intent for DWC_usb3 is described in the UPF file (<workspace>/upf/DWC_usb3.upf) with the following information:

- Power domains
- Power switch
- Power states
- Level shifters
- List of inputs to PMU that need to be isolated

  Based on configuration, the corresponding interface and the signals are isolated.
- Outputs of non-PMU modules that need to be isolated

4.3.2 Power Intent Specification for DWC_usb3

The characteristics of Vcc domain is as follows:

- DWC_usb3_filter, DWC_usb3_noclkrst, DWC_usb3_clk DWC_usb3_rst and their respective sub modules are placed in this power domain.
- Primary supply voltage (Vcc_net) is supplied as power net for the isolation power cells.
- Primary supply voltage (Vcc_GATED) is gated.
- No retention registers.
- Inputs from Vaux may be level shifted.
- Outputs to DWC_usb3_top_pd need isolation.

The characteristics of Vaux power domain is as follows:

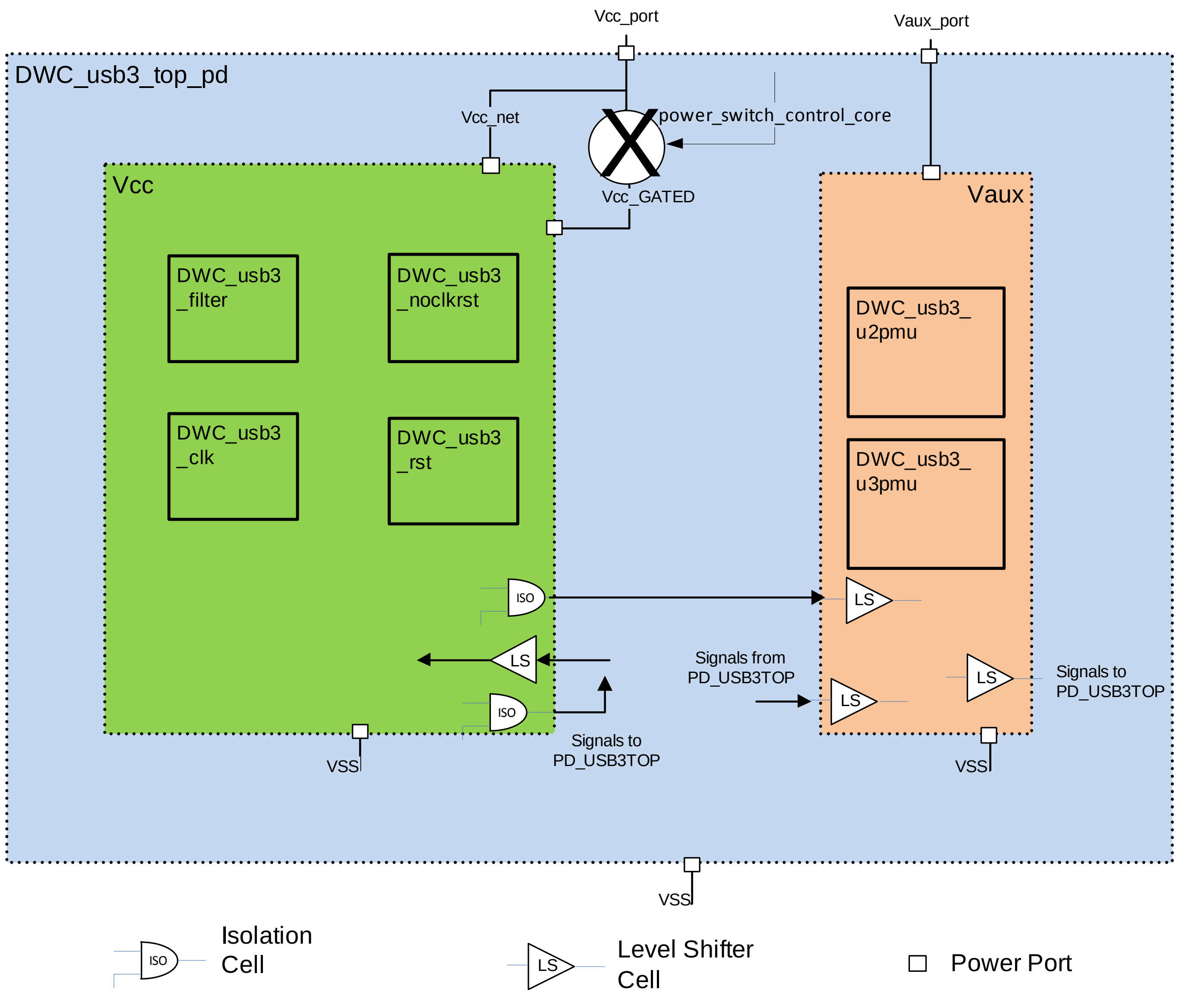
- DWC_usb3_u2pmu, DWC_usb3_u3pmu and their respective sub modules are placed in this power domain.
- Primary supply voltage (Vaux_port) is not gated.

- No retention registers.
- Inputs from Vccpower domain and DWC_usb3_top_pd power domain may be level shifted.
- Inputs from Vcc need isolation.
- Outputs to DWC_usb3_top_pd may need to be level shifted.

The characteristics of DWC_usb3_top_pd power domain is as follows:

- Primary supply voltage (Vcc_port) is not gated.
- Two individual power domains in which one of them can be turned off.
  - Vcc can be turned off.
  - Vaux is always on.

Figure 4-16 DWC_usb3 Power Intent Diagram



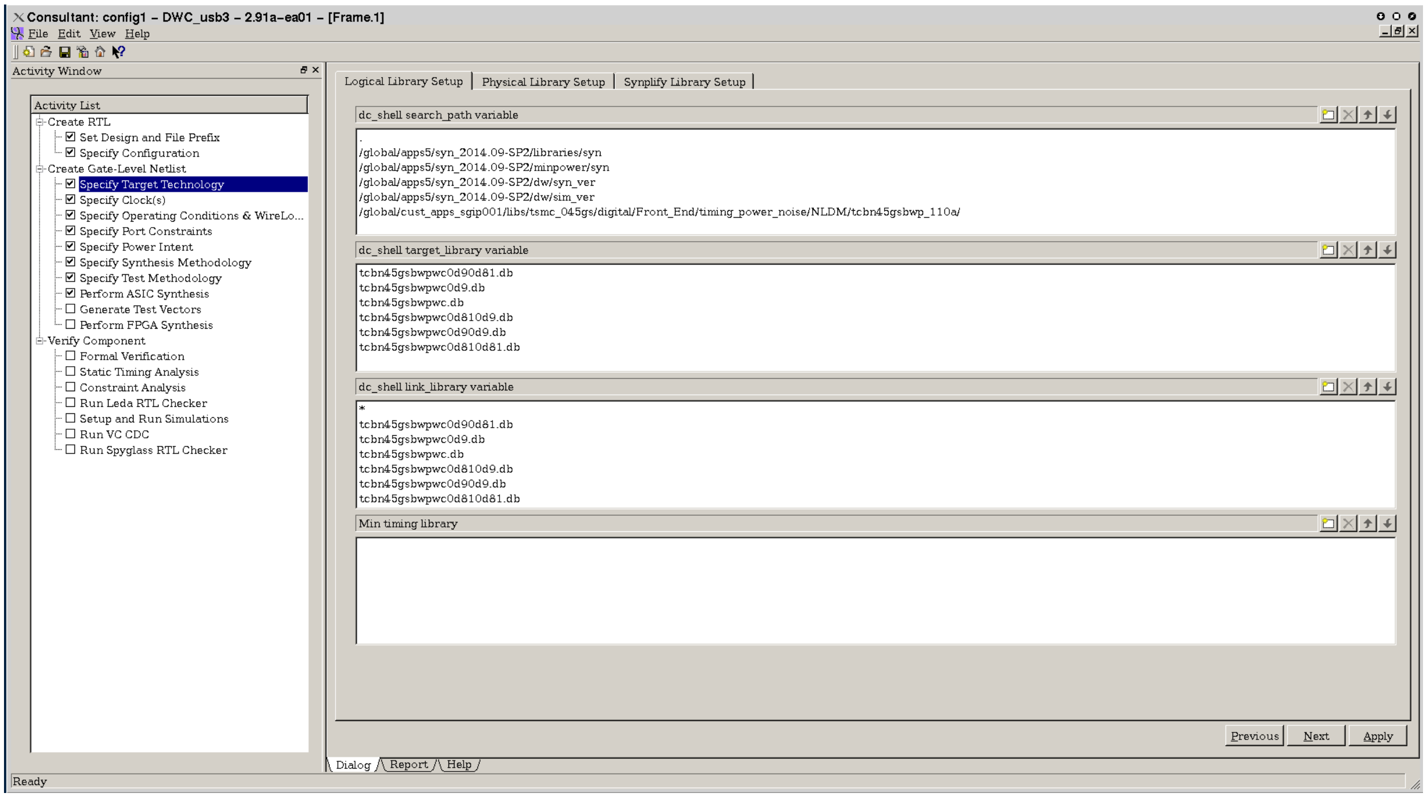
4.3.3 Running Synthesis using UPF Power Intent

The coreConsultant tool currently supports synthesis using UPF with some limitations. However, to run synthesis along with the UPF power intent, follow these steps. The following example assumes an industry-standard 45nm library and lists the steps.

1. Specify the target_library and link_library of the cells that correspond to the appropriate voltage domains. 0.81v and 0.9v are chosen.

```
set ::target_library [list tcbn45gsbwpwc0d90d81.db tcbn45gsbwpwc0d9.db tcbn45gsbwpwc.db
tcbn45gsbwpwc0d810d9.db tcbn45gsbwpwc0d90d9.db tcbn45gsbwpwc0d810d81.db]
set ::link_library [list * tcbn45gsbwpwc0d90d81.db tcbn45gsbwpwc0d9.db tcbn45gsbwpwc.db
tcbn45gsbwpwc0d810d9.db tcbn45gsbwpwc0d90d9.db tcbn45gsbwpwc0d810d81.db]
```

Figure 4-17 Specify Target Technology



Activity List
- Create RTL
  - ☑ Set Design and File Prefix
  - ☑ Specify Configuration
- Create Gate-Level Netlist
  - ☑ Specify Target Technology
  - ☑ Specify Clock(s)
  - ☑ Specify Operating Conditions & WireLo...
  - ☑ Specify Port Constraints
  - ☑ Specify Power Intent
  - ☑ Specify Synthesis Methodology
  - ☑ Specify Test Methodology
  - ☑ Perform ASIC Synthesis
  - ☐ Generate Test Vectors
  - ☐ Perform FPGA Synthesis
- Verify Component
  - ☐ Formal Verification
  - ☐ Static Timing Analysis
  - ☐ Constraint Analysis
  - ☐ Run Leda RTL Checker
  - ☐ Setup and Run Simulations
  - ☐ Run VC CDC
  - ☐ Run Spyglass RTL Checker

dc_shell search_path variable

```
.
/global/apps5/syn_2014.09-SP2/libraries/syn
/global/apps5/syn_2014.09-SP2/minpower/syn
/global/apps5/syn_2014.09-SP2/dw/syn_ver
/global/apps5/syn_2014.09-SP2/dw/sim_ver
/global/cust_apps_sgip001/libs/tsmc_045gs/digital/Front_End/timing_power_noise/NLDM/tcbn45gsbwp_110a/
```

dc_shell target_library variable

```
tcbn45gsbwpwc0d90d81.db
tcbn45gsbwpwc0d9.db
tcbn45gsbwpwc.db
tcbn45gsbwpwc0d810d9.db
tcbn45gsbwpwc0d90d9.db
tcbn45gsbwpwc0d810d81.db
```

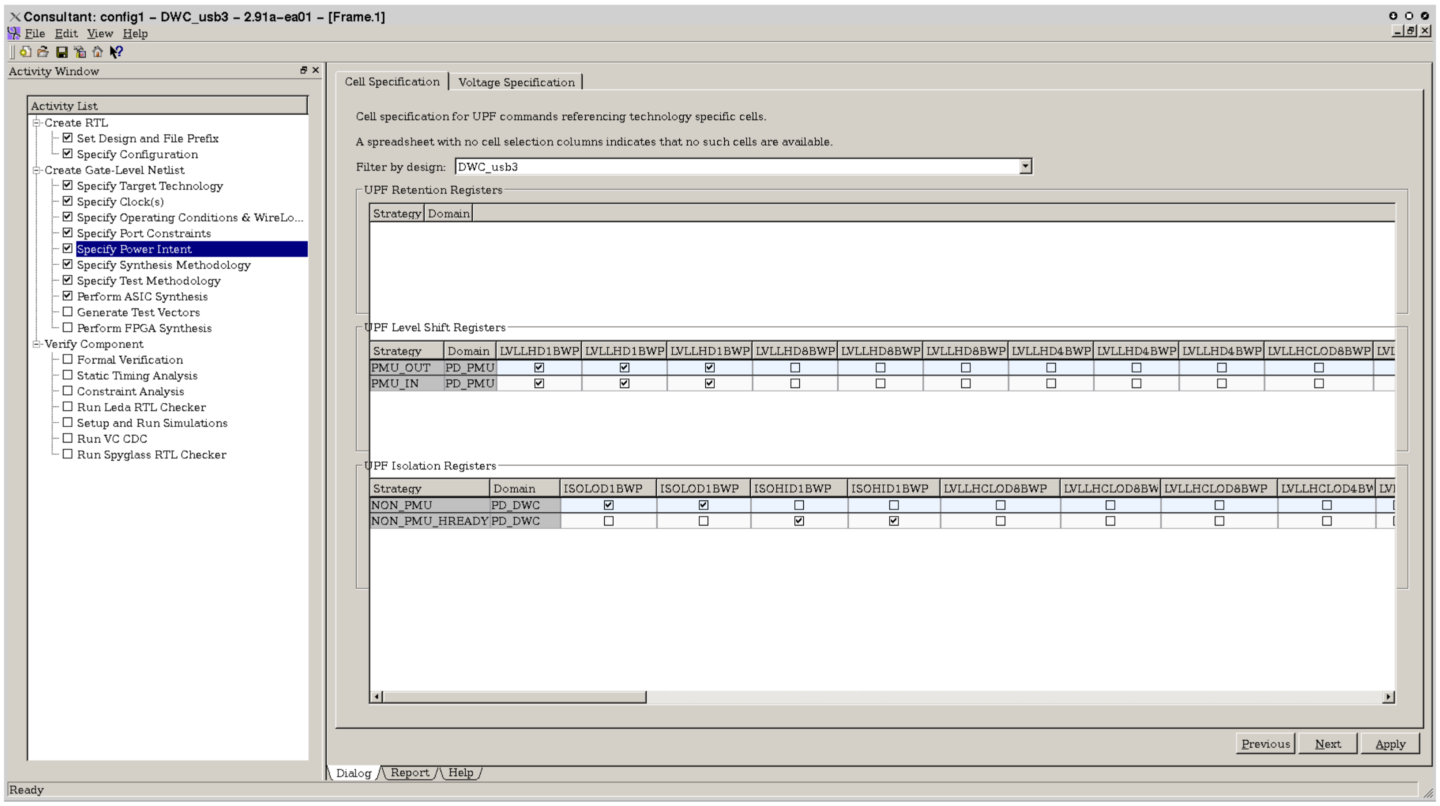
dc_shell link_library variable

```
*
tcbn45gsbwpwc0d90d81.db
tcbn45gsbwpwc0d9.db
tcbn45gsbwpwc.db
tcbn45gsbwpwc0d810d9.db
tcbn45gsbwpwc0d90d9.db
tcbn45gsbwpwc0d810d81.db
```

Min timing library

2. Under the Specify Power Intent, Cell Specification activity, choose the required Level shifter registers and Isolation Registers. The level shifters and isolation cells are automatically inferred from the technology library.
   - For level shifter registers, select LVLLHD1BWP for both the strategies.
   - For Isolation registers, select ISOLOD1BWP and ISOHID1BWP strategy.

Figure 4-18 Specify Power Intent – Cell Specification Activity



Activity List
- Create RTL
  - ☑ Set Design and File Prefix
  - ☑ Specify Configuration
- Create Gate-Level Netlist
  - ☑ Specify Target Technology
  - ☑ Specify Clock(s)
  - ☑ Specify Operating Conditions & WireLo...
  - ☑ Specify Port Constraints
  - ☑ Specify Power Intent
  - ☑ Specify Synthesis Methodology
  - ☑ Specify Test Methodology
  - ☑ Perform ASIC Synthesis
  - ☐ Generate Test Vectors
  - ☐ Perform FPGA Synthesis
- Verify Component
  - ☐ Formal Verification
  - ☐ Static Timing Analysis
  - ☐ Constraint Analysis
  - ☐ Run Leda RTL Checker
  - ☐ Setup and Run Simulations
  - ☐ Run VC CDC
  - ☐ Run Spyglass RTL Checker

Cell specification for UPF commands referencing technology specific cells.

A spreadsheet with no cell selection columns indicates that no such cells are available.

Filter by design: DWC_usb3

UPF Retention Registers

Strategy	Domain

UPF Level Shift Registers

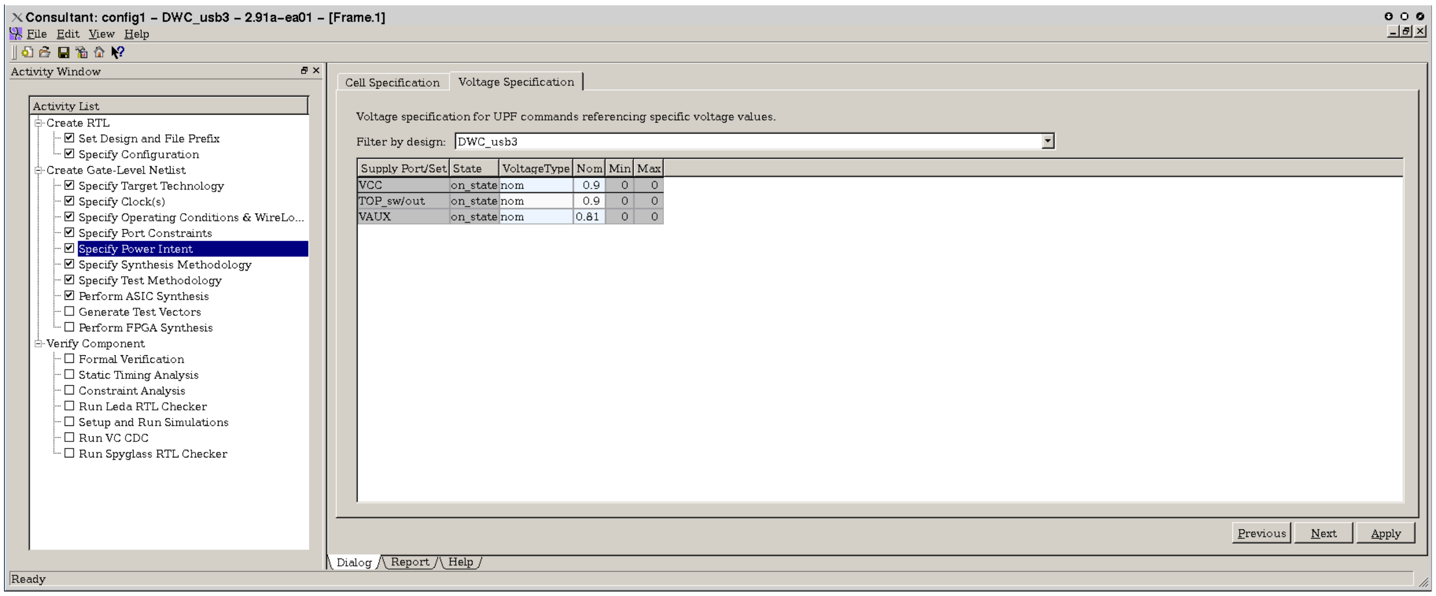
Strategy	Domain	LVLLHD1BWP	LVLLHD1BWP	LVLLHD1BWP	LVLLHD8BWP	LVLLHD8BWP	LVLLHD8BWP	LVLLHD4BWP	LVLLHD4BWP	LVLLHD4BWP	LVLLHCLOD8BWP	LVL
PMU_OUT	PD_PMU	☑	☑	☑	☐	☐	☐	☐	☐	☐	☐	
PMU_IN	PD_PMU	☑	☑	☑	☐	☐	☐	☐	☐	☐	☐	

UPF Isolation Registers

Strategy	Domain	ISOLOD1BWP	ISOLOD1BWP	ISOHID1BWP	ISOHID1BWP	LVLLHCLOD8BWP	LVLLHCLOD8BW	LVLLHCLOD8BWP	LVLLHCLOD4BW	LVI
NON_PMU	PD_DWC	☑	☑	☐	☐	☐	☐	☐	☐	☐
NON_PMU_HREADY	PD_DWC	☐	☐	☑	☑	☐	☐	☐	☐	☐

3. Under the Specify Power Intent, Voltage Specification activity, choose the required voltage values for each supply port. In this example it is 0.81v and 0.9v. Hit "Apply" on the bottom right of the window. This generates 2 UPF power intent files under <workspace>/upf directory to be used for synthesis. These files are DWC_usb3_common.upf and DWC_usb3_syn.upf.

Figure 4-19 Specify Power Intent – Voltage Specification Activity



Activity List
- Create RTL
  - ☑ Set Design and File Prefix
  - ☑ Specify Configuration
- Create Gate-Level Netlist
  - ☑ Specify Target Technology
  - ☑ Specify Clock(s)
  - ☑ Specify Operating Conditions & WireLo...
  - ☑ Specify Port Constraints
  - ☑ Specify Power Intent
  - ☑ Specify Synthesis Methodology
  - ☑ Specify Test Methodology
  - ☑ Perform ASIC Synthesis
  - ☐ Generate Test Vectors
  - ☐ Perform FPGA Synthesis
- Verify Component
  - ☐ Formal Verification
  - ☐ Static Timing Analysis
  - ☐ Constraint Analysis
  - ☐ Run Leda RTL Checker
  - ☐ Setup and Run Simulations
  - ☐ Run VC CDC
  - ☐ Run Spyglass RTL Checker

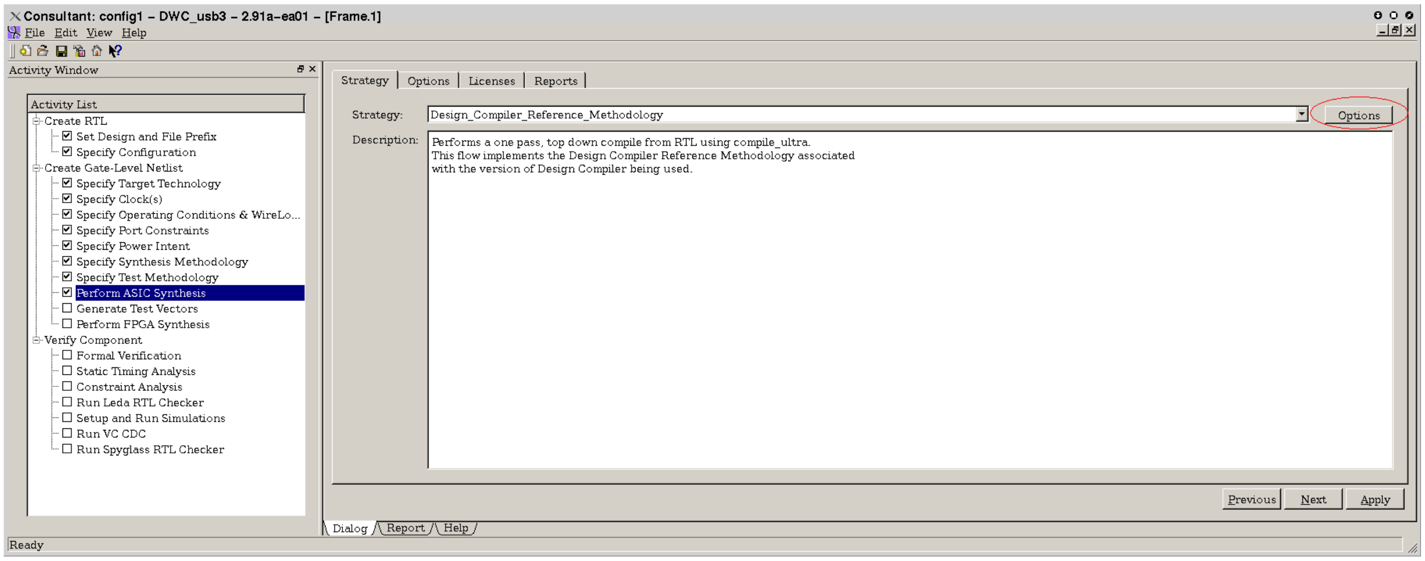
Voltage specification for UPF commands referencing specific voltage values.

Filter by design: DWC_usb3

Supply Port/Set	State	VoltageType	Nom	Min	Max
VCC	on_state	nom	0.9	0	0
TOP_sw/out	on_state	nom	0.9	0	0
VAUX	on_state	nom	0.81	0	0

4. Proceed to Perform ASIC Synthesis activity. Hit "Options" near the top right of the window.

Figure 4-20 Perform ASIC Synthesis



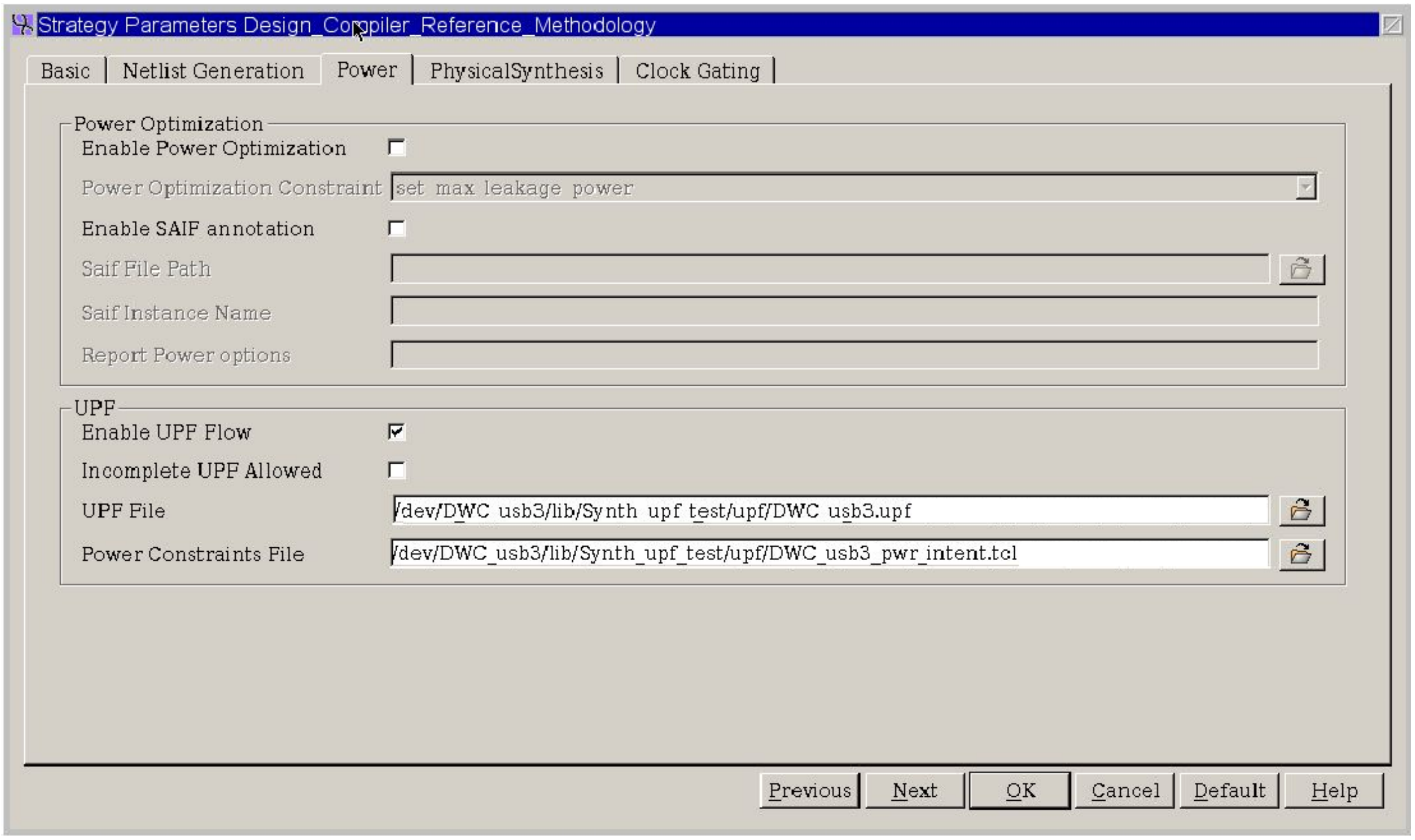
Activity List
- Create RTL
  - ☑ Set Design and File Prefix
  - ☑ Specify Configuration
- Create Gate-Level Netlist
  - ☑ Specify Target Technology
  - ☑ Specify Clock(s)
  - ☑ Specify Operating Conditions & WireLo...
  - ☑ Specify Port Constraints
  - ☑ Specify Power Intent
  - ☑ Specify Synthesis Methodology
  - ☑ Specify Test Methodology
  - ☑ Perform ASIC Synthesis
  - ☐ Generate Test Vectors
  - ☐ Perform FPGA Synthesis
- Verify Component
  - ☐ Formal Verification
  - ☐ Static Timing Analysis
  - ☐ Constraint Analysis
  - ☐ Run Leda RTL Checker
  - ☐ Setup and Run Simulations
  - ☐ Run VC CDC
  - ☐ Run Spyglass RTL Checker

Strategy: Design_Compiler_Reference_Methodology

Description: Performs a one pass, top down compile from RTL using compile_ultra.
This flow implements the Design Compiler Reference Methodology associated
with the version of Design Compiler being used.

5. Under "Power" tab, select UPF.
   - ❑ Enter the path of the UPF file. The path of the UPF file is <workspace>/upf/DWC_usb3_common.upf.
   - ❑ Enter the path of the power intent file which specifies the voltages. The path of the power intent file is <workspace>/upf/DWC_usb3_pwr_intent.tcl. .

Figure 4-21 Perform ASIC Synthesis – Specify UPF Options



Enable Power Optimization	☐
Power Optimization Constraint	set_max_leakage_power
Enable SAIF annotation	☐
Saif File Path	
Saif Instance Name	
Report Power options	
Enable UPF Flow	☑
Incomplete UPF Allowed	☐
UPF File	/dev/DWC_usb3/lib/Synth_upf_test/upf/DWC_usb3.upf
Power Constraints File	/dev/DWC_usb3/lib/Synth_upf_test/upf/DWC_usb3_pwr_intent.tcl

6. Perform ASIC Synthesis using the existing steps.

4.3.4 Formality Checks using UPF Power Intent

The coreConsultant automatically copies the UPF file supplied for the synthesis stage in the directory <workspace>/syn/elab/db/DWC_usb3.upf. Note that this is the same UPF file which is used for simulation with some modifications which are synthesis specific.

The synthesis activity writes out its generated UPF in the directory <workspace>/syn/initial/db/DWC_usb3.upf

You need not supply any additional inputs to the formality activity in cC.

4.3.5 Gate-level Simulations Using UPF Power Intent

The generated netlist does not have VDD and VSS pins present on the instantiated technology library cells.

If the gate-level simulations are run with the library models which have VDD and VSS pins, then 'x' propagation is observed since the VDD and VSS pins remain unconnected to power sources. However, if the gate-level simulations are run with the library models which don't have VDD and VSS pins, no 'x' propagation is observed.

4.3.6 PVETests

Only VCS simulator flow is currently supported to run with UPF. The runsim_vcs script is modified for supporting UPF (See $upfopt variable under <workspace>/sim/SoC_sim/pve/com/runsim_vcs).

The list of PVE hibernation tests that support UPF is as follows:

- Test595, Test595_ulpi
- Test596, Test596_ulpi
- Test495
- Test496
- Test900, Test900_ulpi
- Test911, Test911_ulpi
- Test90, Test90_ulpi
- Test91, Test91_ulpi

4.4 Running Automatic Test Pattern Generation

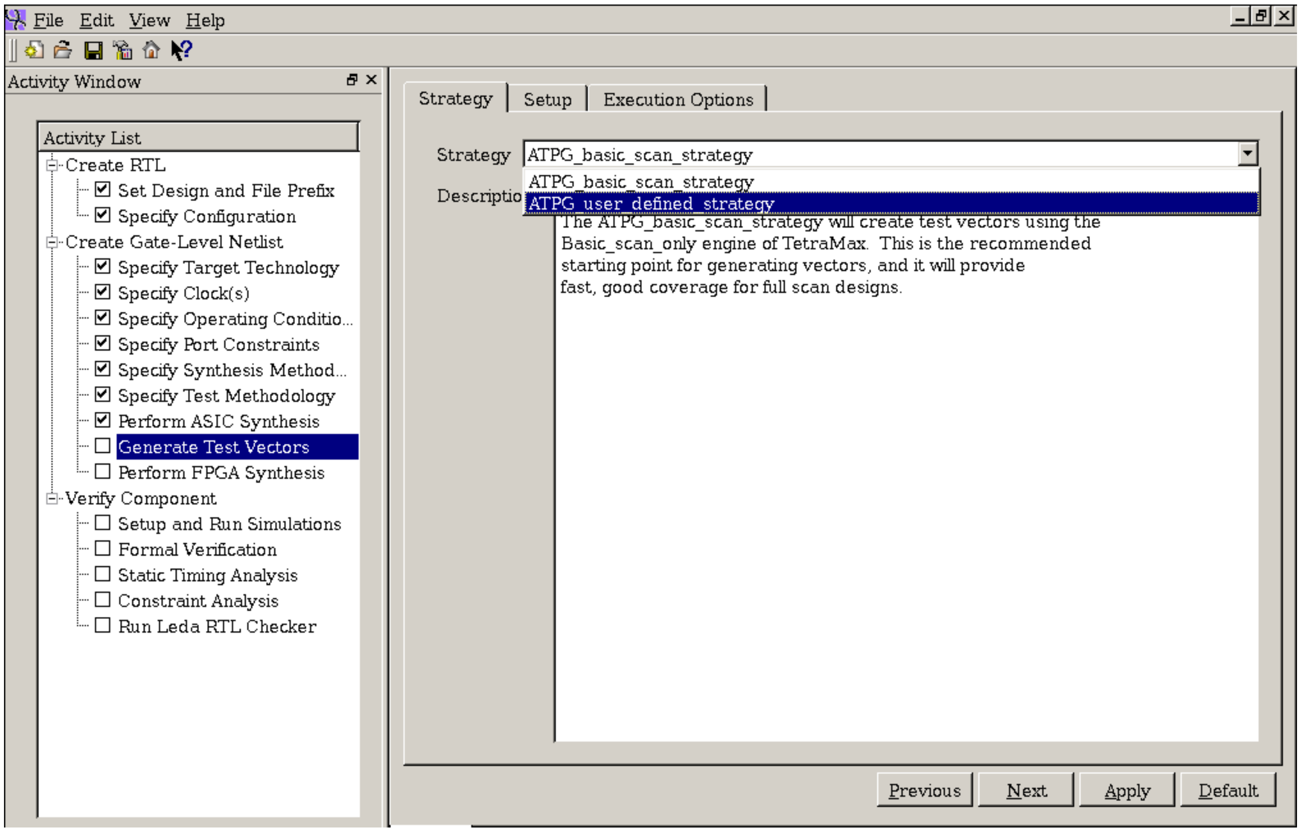
.You perform automatic test pattern generation (ATPG) by using the Synopsys TetraMax tool. This activity is not applicable to FPGA implementations.

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" on page 109) before a meaningful setup can exist in the ATPG window.

To run ATPG, you must complete the Generate Test Vectors activity (Figure 4-22) in coreConsultant.

Figure 4-22 Generate Test Vectors Activity



Strategy: ATPG_basic_scan_strategy
ATPG_basic_scan_strategy
ATPG_user_defined_strategy
Descriptio

The ATPG_basic_scan_strategy will create test vectors using the Basic_scan_only engine of TetraMax. This is the recommended starting point for generating vectors, and it will provide fast, good coverage for full scan designs.

After you have synthesized the controller with the Insert DFT option (see "Inserting Design For Test" on page 109), select the **Generate Test Vectors -> Setup** tab and complete the following fields in the dialog box.

- **Input stage**: Specify the last synthesis stage from the pull-down menu.
- **Output stage**: Set to any name that you want to use identify the output files that will be created in your workspace directory. For example, *atpg*.
- **Test Libraries**: Specify the library Verilog file name(s).

  These 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 be each standard cell instantiated.

- **Test Pattern Format**: Specify your test format.

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

The following files in <workspace>`/syn/<Output stage>` can help you understand the ATPG flow:

- `script/DWC_usb3.tcl`
- `Makefile`

For more information about running ATPG, refer to the *coreConsultant User Guide* (also see "Help Information" on page 200).

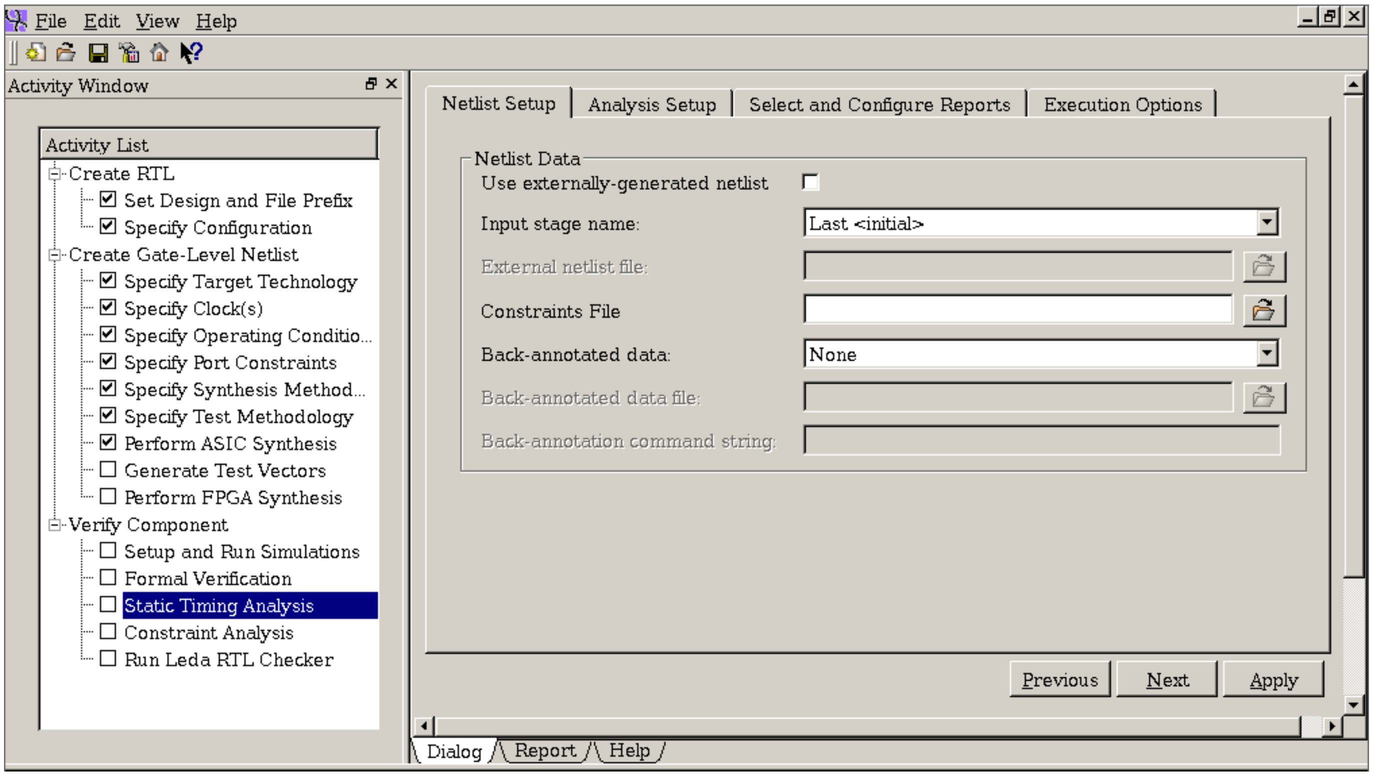
4.5 Running Static Timing Analysis

You perform static timing analysis (STA) by using the Synopsys PrimeTime tool. This activity is not applicable to FPGA implementations.

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 for back-annotation.

You must have completed the "Performing ASIC Synthesis" on page 93 step or have an externally generated netlist available before starting this task. To perform STA, select the **Static Timing Analysis** activity.

Figure 4-23 Static Timing Analysis Activity



Netlist Data
Use externally-generated netlist ☐
Input stage name: Last <initial>
External netlist file:
Constraints File
Back-annotated data: None
Back-annotated data file:
Back-annotation command string:

To run STA, complete the following procedure:

1. **Netlist Setup**

   Click the Netlist Setup tab and complete 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 pulldown 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.

   To perform power analysis, see "Performing Power Analysis" on page 129.

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 perform power analysis, see "Performing Power Analysis" on page 129.

4. Click **Apply**.

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

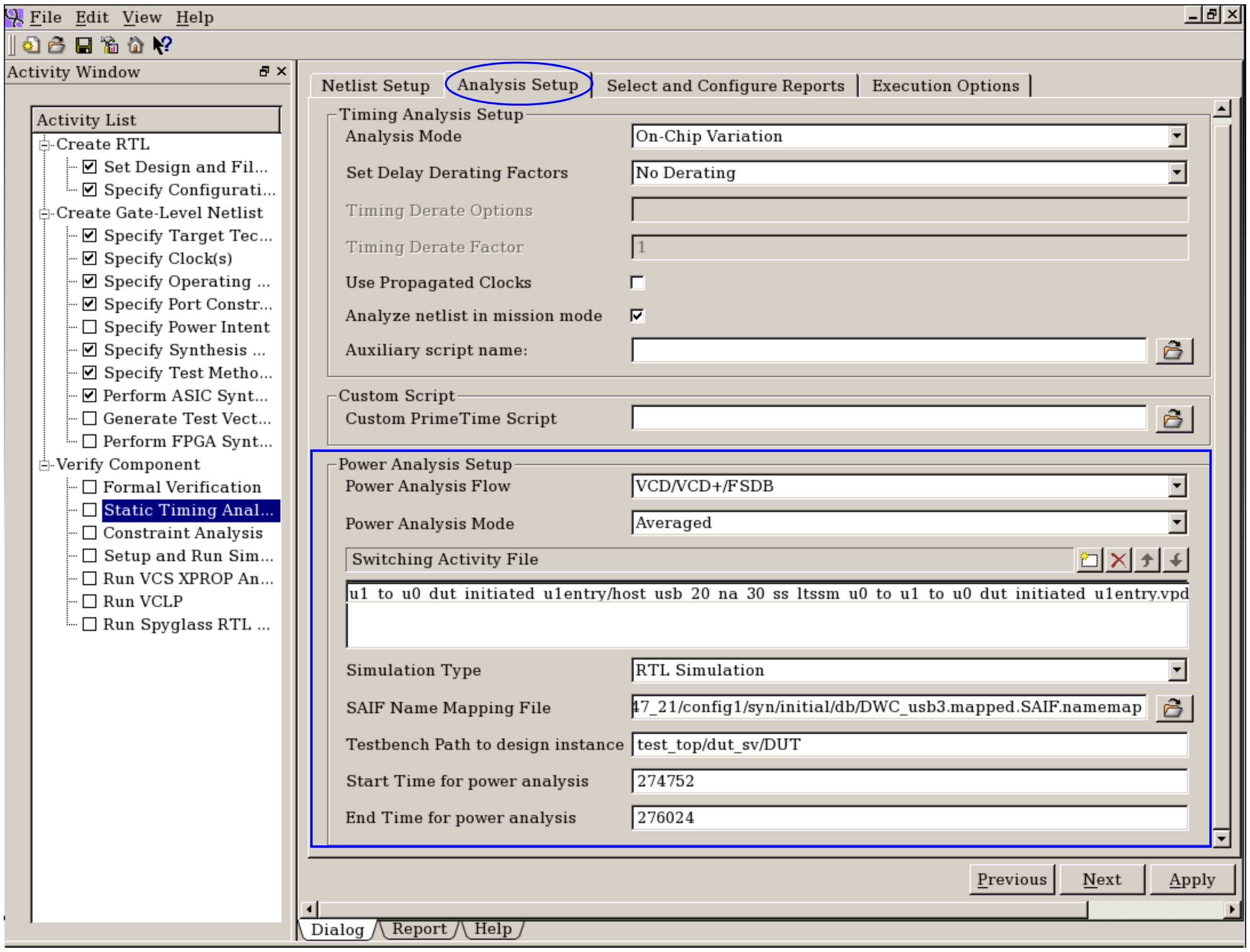
4.6 Performing Power Analysis

This section shows you how to perform power analysis on your synthesized DWC_usb3 controller.

You perform power analysis during Static Timing Analysis activity in the coreConsultant GUI. While performing the steps for STA ("Running Static Timing Analysis" on page 127), you must select/enter the following options to run power analysis:

- **Analysis Setup > Power Analysis Setup**
  - **Switching Activity File**: The switching activity file could be the gate-level netlist or RTL simulation-based VPD/VCD/FSDB file for your configuration.
  - **Testbench Path to design instantiation**: Enter the top-level DUT name.
  - **Start Time for power analysis**: Enter the start time (in ns).
  - **End Time for power analysis**: Enter the end time (in ns).
- **Select and Configure Reports > Power Analysis**
  - **Run Power Analysis**: Select this option to enable power analysis.

Figure 4-24 Power Analysis Setup



Timing Analysis Setup

Analysis Mode	On-Chip Variation
Set Delay Derating Factors	No Derating
Timing Derate Options	
Timing Derate Factor	1
Use Propagated Clocks	☐
Analyze netlist in mission mode	☑
Auxiliary script name:	

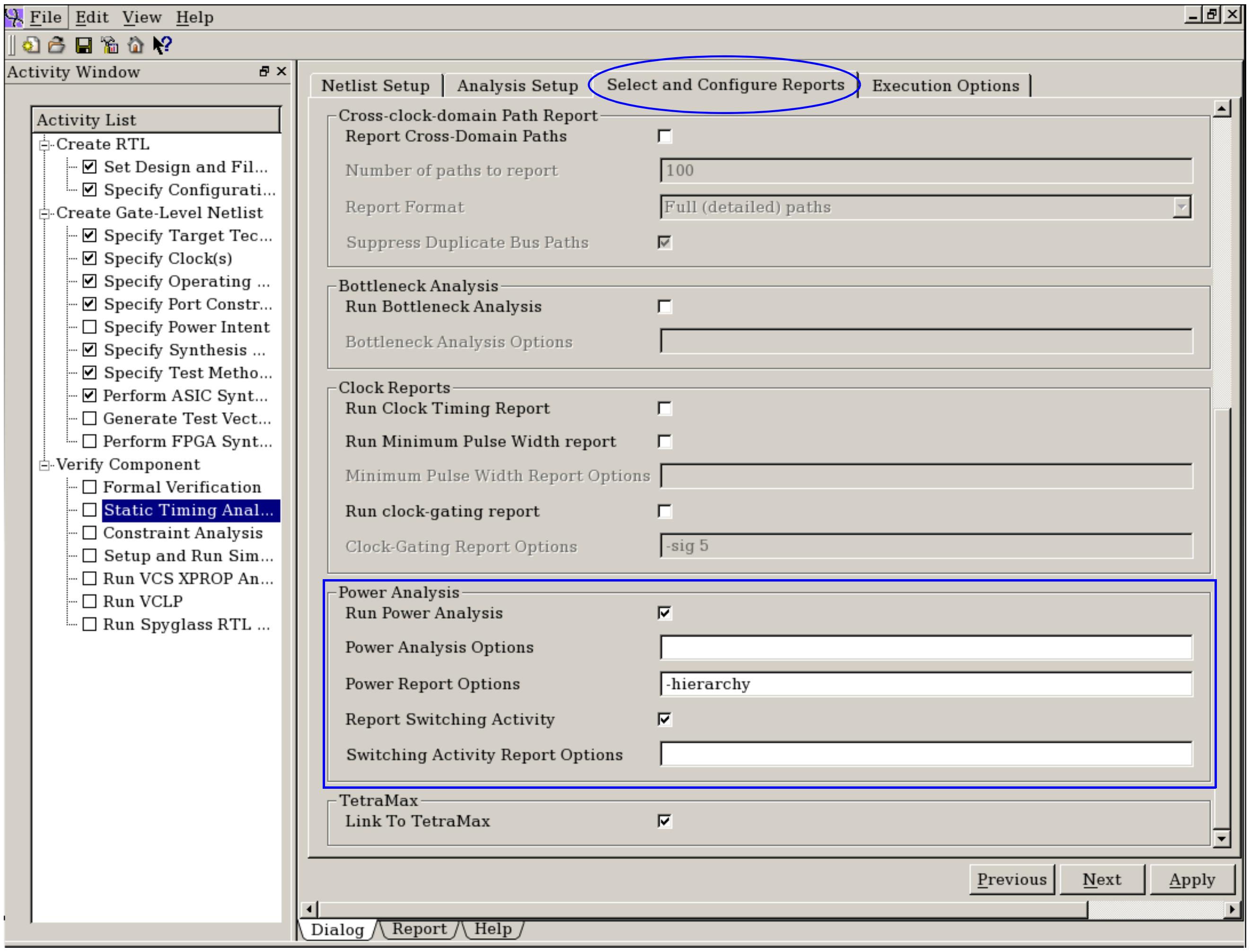
Custom Script

Custom PrimeTime Script	

Power Analysis Setup

Power Analysis Flow	VCD/VCD+/FSDB
Power Analysis Mode	Averaged
Switching Activity File	u1 to u0 dut initiated u1entry/host usb 20 na 30 ss ltssm u0 to u1 to u0 dut initiated u1entry.vpd
Simulation Type	RTL Simulation
SAIF Name Mapping File	47_21/config1/syn/initial/db/DWC_usb3.mapped.SAIF.namemap
Testbench Path to design instance	test_top/dut_sv/DUT
Start Time for power analysis	274752
End Time for power analysis	276024

Figure 4-25 Power Analysis Options



Cross-clock-domain Path Report

Report Cross-Domain Paths	☐
Number of paths to report	100
Report Format	Full (detailed) paths
Suppress Duplicate Bus Paths	✓

Bottleneck Analysis

Run Bottleneck Analysis	☐
Bottleneck Analysis Options	

Clock Reports

Run Clock Timing Report	☐
Run Minimum Pulse Width report	☐
Minimum Pulse Width Report Options	
Run clock-gating report	☐
Clock-Gating Report Options	-sig 5

Power Analysis

Run Power Analysis	✓
Power Analysis Options	
Power Report Options	-hierarchy
Report Switching Activity	✓
Switching Activity Report Options	

TetraMax

Link To TetraMax	✓

The power analysis results are saved in the <workspace>/syn/final/report/sta/DWC_usb3_power.rpt

4.7 Performing Formal Verification

This section shows you how to run formal verification on the DWC_usb3 controller. The topics in this section are as follows:

- "Performing Formal Verification in coreConsultant"
- "Formal Verification Activity" on page 132

4.7.1 Performing Formal Verification in coreConsultant

.You use the Formal Verification activity (in) for RTL-to-gate comparisons in coreConsultant. This activity is not applicable to FPGA implementations.

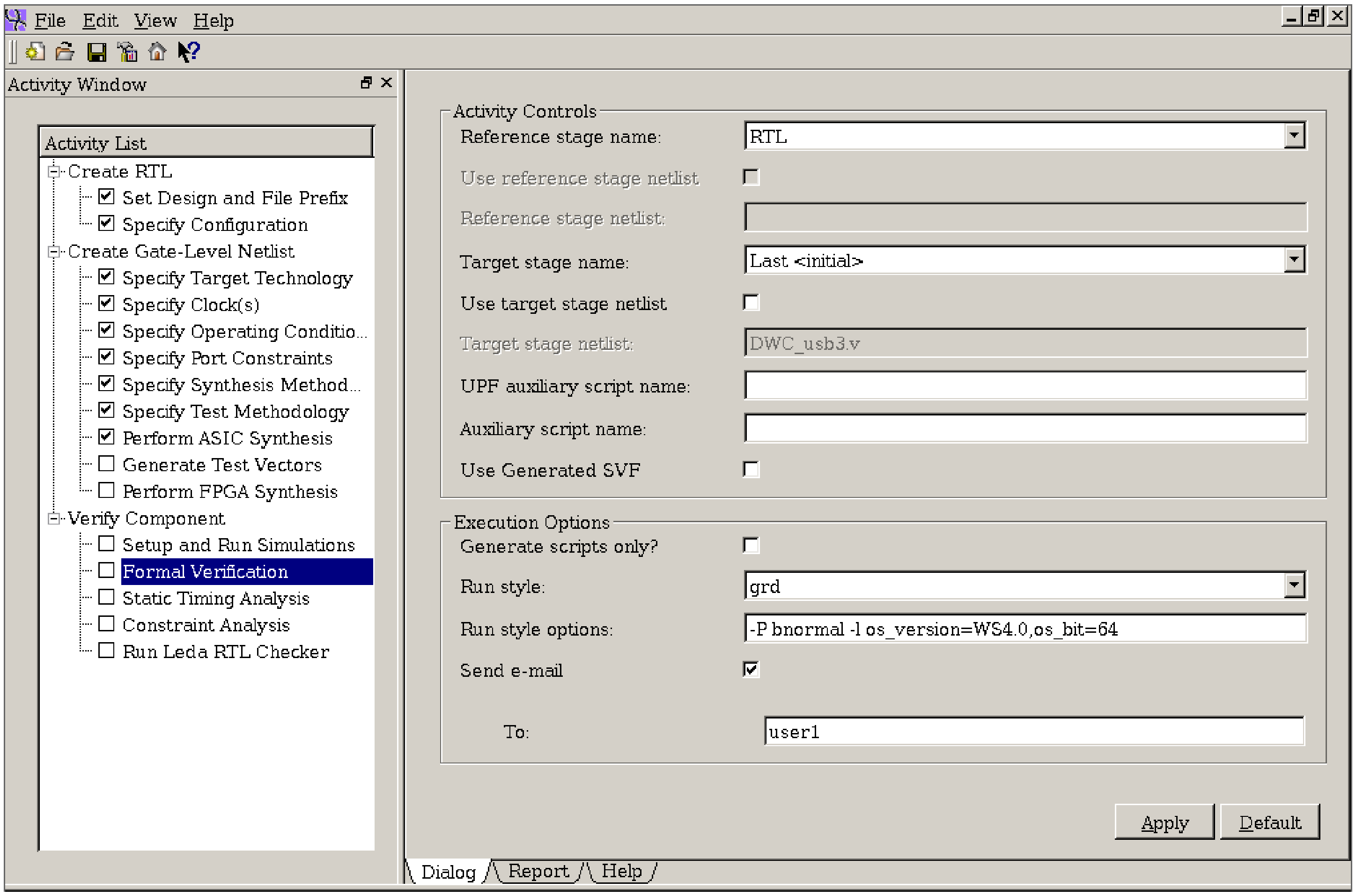
Attention Formal verification, also referred to as formal equivalence checking, is an activity which takes the output of synthesis as input. Synthesis must be completed before a meaningful set up can exist in the Formality window. For more information, refer to the following site:
http://www.synopsys.com/Tools/Verification/FormalEquivalence/Pages/default.aspx

Choose **Formal Verification** from the Activity List and complete the following fields:

- **Reference stage name**: RTL.
- **Target stage name**: Select last synthesis stage from the pull-down menu.
- **Output stage**: Specify a name of your choice to 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. The file <workspace>/syn/<Target stage name>.tcl may be inspected to help you understand the Formal Verification flow.

Figure 4-26 Formal Verification Activity



Activity Controls

Reference stage name:	RTL
Use reference stage netlist	☐
Reference stage netlist:	
Target stage name:	Last <initial>
Use target stage netlist	☐
Target stage netlist:	DWC_usb3.v
UPF auxiliary script name:	
Auxiliary script name:	
Use Generated SVF	☐

Execution Options

Generate scripts only?	☐
Run style:	grd
Run style options:	-P bnormal -l os_version=WS4.0,os_bit=64
Send e-mail	☑
To:	user1

4.7.2 Performing Formal Verification Outside coreConsultant

If you want to run Formality outside coreConsultant, you must export the relevant scripts from coreConsultant as outlined here so that you can run Formality on the controller in your own design flow. This section shows you how to access these scripts and information for your configured controller. You can then transfer these to your chip design database. You must have access to Formality to generate and export the relevant scripts. These activities are not applicable to FPGA implementations.

To access the Formality scripts you must first complete synthesis in coreConsultant. For details on generating and accessing the Formality scripts, see "Performing Formal Verification" on page 131. When you select the DCTCL_one_pass_compile_ultra flow (under **Synthesis -> Strategy**), synthesis runs quicker as it does not generate intermediate stages. The generated Formality script does not contain any controller-specific directives. It is only useful to keep track of the .svf files (for all synthesis stages) and pass them all to Formality.

A template for that script may be inspected at <workspace>/export/fm.tcl. You must use the Synopsys 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}
```

5 Integrating the Controller

The DWC_usb3 configuration options provide flexibility for integrating the controller into your SoC design. These options also provide trade-offs between performance and area, depending upon your application requirements.

Attention	■ For description of various configuration parameters and signals mentioned in this section, refer to the *DWC SuperSpeed USB 3.0 Controller Databook*. ■ For description of various registers mentioned in this section, refer to the *DWC SuperSpeed USB 3.0 Controller Programming Guide*.

This chapter provides integration examples and details on how to integrate the DWC_usb3 controller with your PHY. The topics are covered in this section are:

5.1 Exporting a Controller to Your Chip Design Database

This activity is not applicable to FPGA implementations. At a certain point you may want to transfer (export) a configured controller into your own custom design flow. The coreConsultant tool has a number of features to accomplish this task. 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.

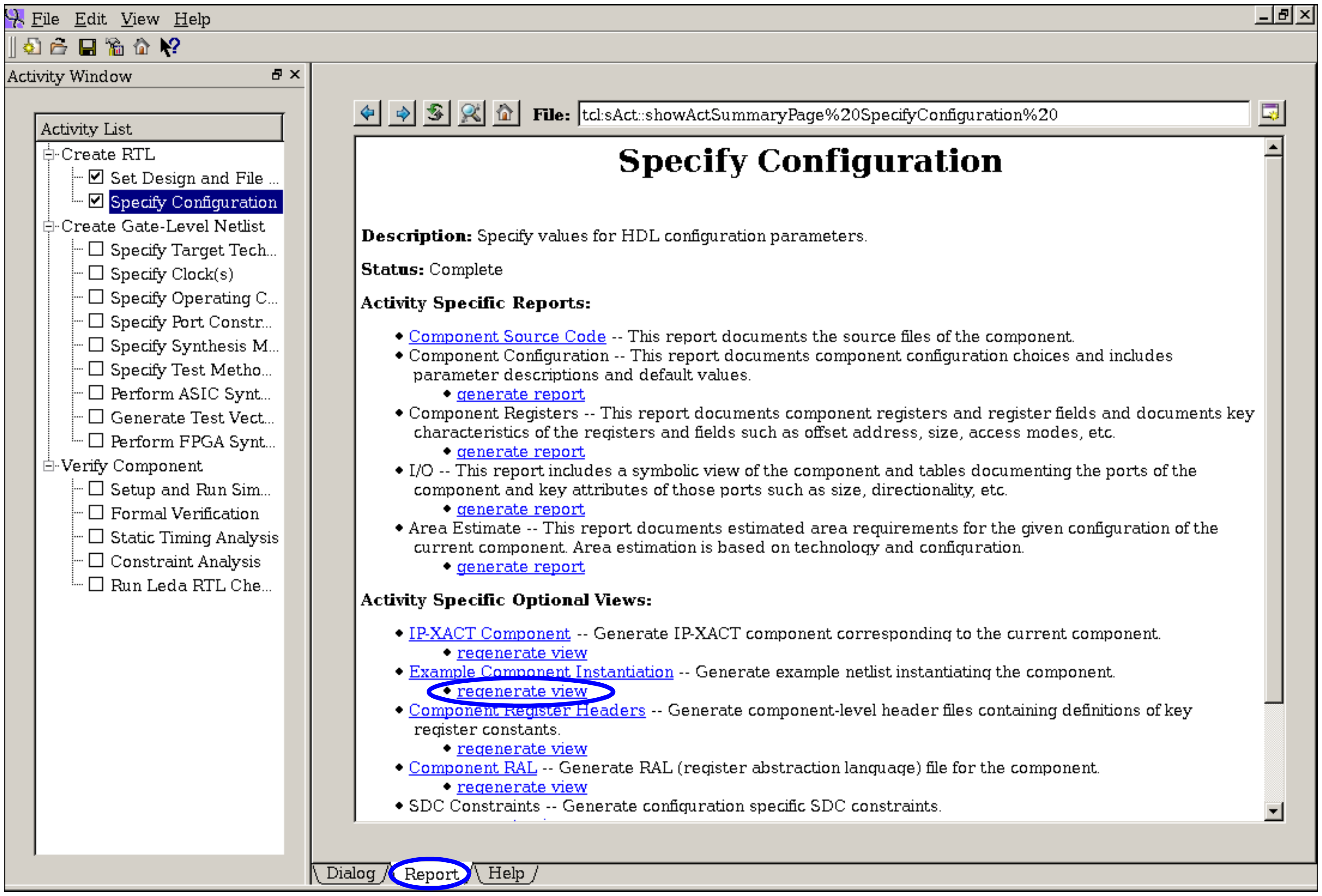
The topics in this section are as follows:

- "Generating an Example Controller Instantiation"
- "Exporting RTL Code"
- "Synthesizing Outside of coreConsultant" on page 100
- "Exporting Formality, DFT, and ATPG Scripts" on page 135

5.1.1 Generating an Example Controller Instantiation

After you have configured your USB 3.0 Controller, you can generate an example Component Instantiation.

Figure 5-1 Generating the Controller Instantiation



tcl:sAct::showActSummaryPage%20SpecifyConfiguration%20

Activity List
- Create RTL
  - ☑ Set Design and File ...
  - ☑ Specify Configuration
- Create Gate-Level Netlist
  - ☐ Specify Target Tech...
  - ☐ Specify Clock(s)
  - ☐ Specify Operating C...
  - ☐ Specify Port Constr...
  - ☐ Specify Synthesis M...
  - ☐ Specify Test Metho...
  - ☐ Perform ASIC Synt...
  - ☐ Generate Test Vect...
  - ☐ Perform FPGA Synt...
- Verify Component
  - ☐ Setup and Run Sim...
  - ☐ Formal Verification
  - ☐ Static Timing Analysis
  - ☐ Constraint Analysis
  - ☐ Run Leda RTL Che...

Specify Configuration

Description: Specify values for HDL configuration parameters.

Status: Complete

Activity Specific Reports:

- Component Source Code -- This report documents the source files of the component.
- Component Configuration -- This report documents component configuration choices and includes parameter descriptions and default values.
  - generate report
- Component Registers -- This report documents component registers and register fields and documents key characteristics of the registers and fields such as offset address, size, access modes, etc.
  - generate report
- I/O -- This report includes a symbolic view of the component and tables documenting the ports of the component and key attributes of those ports such as size, directionality, etc.
  - generate report
- Area Estimate -- This report documents estimated area requirements for the given configuration of the current component. Area estimation is based on technology and configuration.
  - generate report

Activity Specific Optional Views:

- IP-XACT Component -- Generate IP-XACT component corresponding to the current component.
  - regenerate view
- Example Component Instantiation -- Generate example netlist instantiating the component.
  - regenerate view
- Component Register Headers -- Generate component-level header files containing definitions of key register constants.
  - regenerate view
- Component RAL -- Generate RAL (register abstraction language) file for the component.
  - regenerate view
- SDC Constraints -- Generate configuration specific SDC constraints.

To instantiate the controller, follow these steps:

1. Select the Report tab in the Specify Configuration activity.
2. Click the **generate view** link as indicated in Figure 5-1 on page 134.

The controller instantiation is now available in `export/USB 3.0 Controller_inst.v`. If you want coreConsultant to always generate this Component Instantiation, you must set the relevant option as described in "Creating Optional Reports and Views" on page 33.

5.1.2 Exporting RTL Code

There is an `export` directory in your workspace. The coreConsultant tool populates this directory with various links and files for export use. After you have configured the DWC_usb3 controller, the file DWC_usb3.lst is created in the <workspace>/export directory. After you run at least one simulation from coreConsultant (preferably using VCS), the file runsim_vcs is created in the <workspace>/sim/SoC_sim/pve/com directory.

Table 5-1 Files (of interest) Created After Specify Configuration and Simulation Activity

File Name	Description
workspace/sim/SoC_sim/pve/com/runsim_vcs	Command script to compile controller in VCS.
workspace/sim/SoC_sim/pve/com/runsim_vcs	Simulator options and include paths.
workspace/export/DWC_usb3.lst	List of source files in proper analysis order

By inspecting the files, you can determine how to access your configured controller from your chip design database and use the simulator of your choice.

5.1.3 Exporting Formality, DFT, and ATPG Scripts

When you want run Formality and Tetramax (for ATPG) using the Synopsys synthesis tools within coreConsultant, then follow the process as outlined in "Performing Formal Verification" on page 131 or "Running Automatic Test Pattern Generation" on page 125. Otherwise, you must export the relevant scripts from coreConsultant as outlined here so that you can run Formality or TetraMax on the controller in your own design flow.

This section shows you how to access the relevant scripts and information for your configured controller. You can then transfer these to your chip design database. You must have access to Synopsys Design Compiler and Formality to generate and export the relevant scripts. These activities are not applicable to FPGA implementations. The topics in this section are as follows:

- "Exporting DFT Scripts"
- "Exporting ATPG Scripts" on page 136
- "Exporting Views and Reports" on page 136

5.1.3.1 Exporting DFT Scripts

See "Inserting Design For Test" on page 109.

5.1.3.2 Exporting ATPG Scripts

To access the ATPG scripts you must first complete synthesis in coreConsultant. For details on generating and accessing the TetraMax scripts, see "Running Automatic Test Pattern Generation" on page 125. When you select the DCTCL_one_pass_compile_ultra flow (under **Synthesis -> Strategy**), synthesis runs quicker as it does not generate intermediate stages. The generated ATPG script does not contain any controller-specific directives. It is only useful to keep track of the following and to pass them to TetraMax:

- .spf file (from the synthesis DFT stage)
- Technology library netlist
- Design netlist
- snps_phy_atpg_aux.tcl ("Running Automatic Test Pattern Generation" on page 125)

You may inspect a template for that script at <workspace>/export/atpg.tcl. You may inspect a Makefile at <workspace>/export/Makefile.atpg.

5.1.3.3 Exporting Views and Reports

As part of the export process, you may want to generate documentation for your configured controller to be used as part of the documentation for an entire chip. The report and view generation process is based on DocBook instead of HTML. This allows documentation for I/O definitions, parameter definitions, and memory maps to be generated in HTML, RTF, and MIF formats from the DocBook source.

For more information, see "Creating Optional Reports and Views" on page 33. All reports can be accessed in the <workspace>`/report` directory. The optional views are located in the <workspace>`/export` directory.

Attention The Exporting View and Reports feature is still under development within coreConsultant and may not be fully functional with your controller. For assistance, contact Synopsys Customer Support (see "Help Information" on page 200).

5.2 Integration Examples

Figure 5-2 shows how the DWC_usb3 controller can be integrated into an SoC. In this example, the USB 3.0 Device Controller accesses the SoC and other peripherals over a system bus. The PHYs are accessed over a PIPE3 or UTMI/ULPI interface.

Figure 5-2 SoC Integration Example

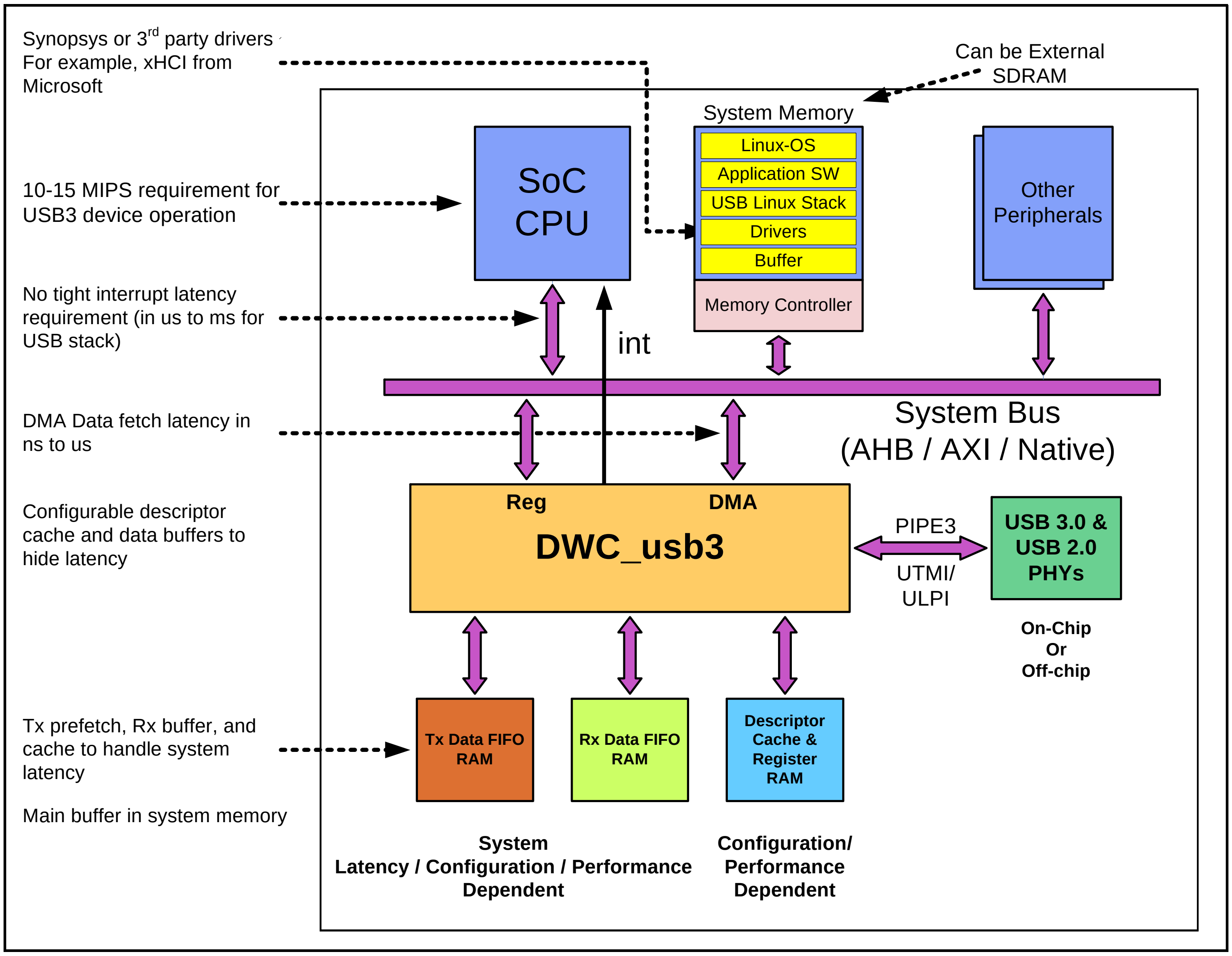
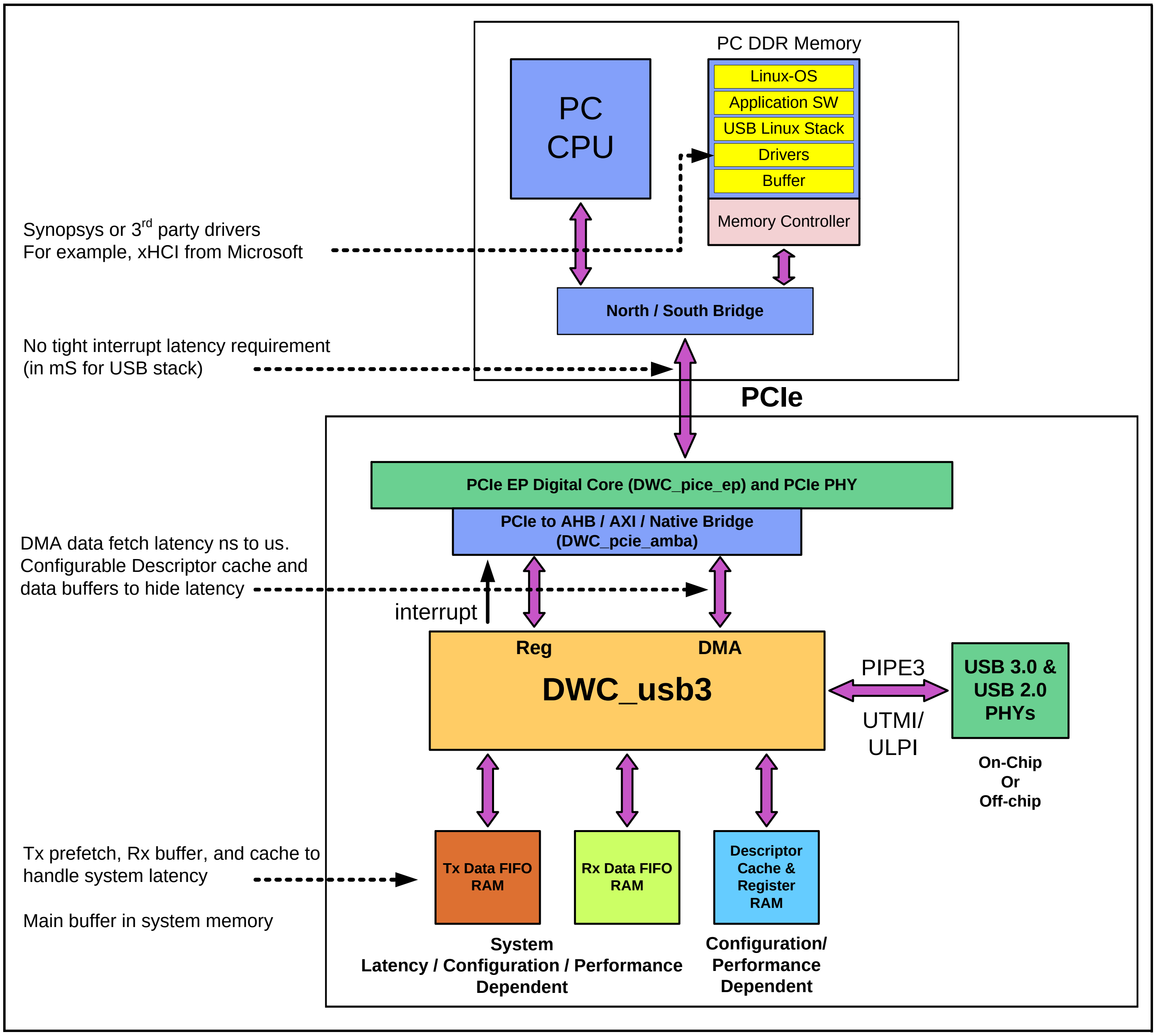

Synopsys or 3rd party drivers
For example, xHCI from Microsoft
10-15 MIPS requirement for USB3 device operation
No tight interrupt latency requirement (in us to ms for USB stack)
DMA Data fetch latency in ns to us
Configurable descriptor cache and data buffers to hide latency
Tx prefetch, Rx buffer, and cache to handle system latency
Main buffer in system memory
Can be External SDRAM
System Memory
Linux-OS
Application SW
USB Linux Stack
Drivers
Buffer
Memory Controller
SoC CPU
Other Peripherals
int
System Bus (AHB / AXI / Native)
Reg
DMA
DWC_usb3
PIPE3
UTMI/ ULPI
USB 3.0 & USB 2.0 PHYs
On-Chip Or Off-chip
Tx Data FIFO RAM
Rx Data FIFO RAM
Descriptor Cache & Register RAM
System Latency / Configuration / Performance Dependent
Configuration/ Performance Dependent

Figure 5-3 shows how the DWC_usb3 controller can be integrated with PCI Express (PCIe).

Figure 5-3 PCIe Integration Example


PC DDR Memory
PC CPU
Linux-OS
Application SW
USB Linux Stack
Drivers
Buffer
Memory Controller
North / South Bridge
Synopsys or 3rd party drivers
For example, xHCI from Microsoft
No tight interrupt latency requirement
(in mS for USB stack)
PCIe
PCIe EP Digital Core (DWC_pice_ep) and PCIe PHY
PCIe to AHB / AXI / Native Bridge
(DWC_pcie_amba)
DMA data fetch latency ns to us.
Configurable Descriptor cache and
data buffers to hide latency
interrupt
Reg
DMA
DWC_usb3
PIPE3
UTMI/
ULPI
USB 3.0 & USB 2.0 PHYs
On-Chip Or Off-chip
Tx prefetch, Rx buffer, and cache to
handle system latency
Main buffer in system memory
Tx Data FIFO RAM
Rx Data FIFO RAM
Descriptor Cache & Register RAM
System
Latency / Configuration / Performance
Dependent
Configuration/
Performance
Dependent

5.3 Integrating the Controller with the PHY

The topics covered in this section are:

5.3.1 Integrating with SuperSpeed PHY

This section describes how to connect your SS PHY to the DWC_usb3 interface. The PIPE3 interface is selected for SuperSpeed PHY Interface (DWC_USB3_SSPHY_INTERFACE) during coreConsultant configuration.

The following subsections explain the steps for integrating the SuperSpeed PHY.

5.3.1.1 Clocks

As described in sections 'Clock Generation and Clock Tree Synthesis (CTS) Requirements' and 'Block Descriptions' in the *DWC SuperSpeed USB 3.0 Controller Databook*, each port of the controller has two clock inputs for the PIPE3 PHY clock. Pipe3_tx_pclk is used for signals driven to the PHY, and pipe3_rx_pclk is used for signals received from the PHY. This allows you to optimize each clock tree individually.

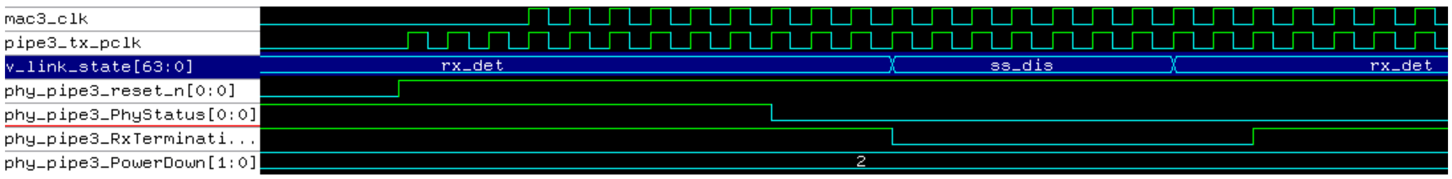
The mac3_clk is derived by dividing down the pipe3_rx_pclk, according to the PHY data width. In USB3 devices, very low latency synchronous data transfers are performed from the pipe3_rx_pclk to mac3_clk and from mac3_clk to pipe3_tx_clk domains. In Hubs and multi-port Hosts, pipe3 clock from port-0 is divided to generate the mac3 clock used throughout the controller. In this case, non-0 ports include ring buffers to accommodate the asynchronous pipe3 and mac3 clocks.

5.3.1.2 Reset

The controller drives reset to the PHY (as described in section 'Reset Generation' in the *DWC SuperSpeed USB 3.0 Controller Databook*) and observes the PhyStatus signal from the PHY to determine when internal controller resets may be deasserted. The PhyStatus signal from the PHY must be valid before pipe3_reset_n is deasserted for correct operation of the controller.

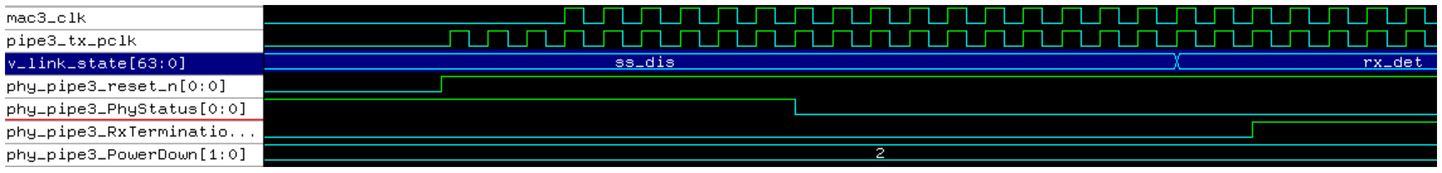
According to section 6.2 of the PIPE3 Specification, the reset value of pipe3_RxTerminatoin is '1'. Consider the case where DWC_USB3_PIPE_RXTERM_RESET_VAL is 1 (See Figure 5-4). The controller asserts pipe3_RxTerminatoin during pipe3_reset_n and de-asserts after removal of pipe3_reset_n in order to perform controller initialization. If the remote link performs receiver detection (while the controller is performing reset), it may enter into Polling state and eventually into compliance or High Speed mode.

Figure 5-4 DWC_USB3_PIPE_RXTERM_RESET_VAL Set to 1



In Figure 5-5, DWC_USB3_PIPE_RXTERM_RESET_VAL is 0. The controller does not assert pipe3_RxTerminatoin until the controller initialization is done and it is ready to connect. This prevents the remote link from entering into Polling when the controller is not ready for connect.

Figure 5-5 DWC_USB3_PIPE_RXTERM_RESET_VAL Set to 0



5.3.1.3 Outputs to PHY

All outputs to the PHY are either synchronous to pipe3_tx_pclk, or hard wired as described in Chapter 'Signals' of the *DWC SuperSpeed USB 3.0 Controller Databook*. The controller is designed so that IO ring flops may be inferred for the synchronous signals. While pipe3_reset_n is asserted, outputs to the PHY are driven to the values required by the PIPE3 specification.

5.3.1.4 Inputs from PHY

All inputs to the PHY are registered synchronously to pipe3_rx_pclk. IO ring flops may be inferred for these signals. For more information, see Chapter 'Signals' in the *DWC SuperSpeed USB 3.0 Controller Databook*.

One input from the PHY, pipe3_PhyStatus is connected to two controller inputs:

- pipe3_PhyStatus
- pipe3_PhyStatus_async

This allows you to infer an IO ring flop for pipe3_PhyStatus.

5.3.1.5 Pipelining on the PIPE Interface

You can insert additional pipelines in the Rx and Tx of the PIPE interface by setting the configuration parameter DWC_USB3_SSPHY_INTERFACE_NUM_PIPE to 1 or 2. Figure 5-6 on page 142 and Figure 5-7 on page 143 show the pipeline stages when DWC_USB3_SSPHY_INTERFACE_NUM_PIPE is set to 1 and 2, respectively.

For more details, refer to the description of DWC_USB3_SSPHY_INTERFACE_NUM_PIPE configuration parameter in Chapter 'Parameters' of the *DWC SuperSpeed USB 3.0 Controller Databook*.

Figure 5-6 Stage 1 of Additional Pipeline

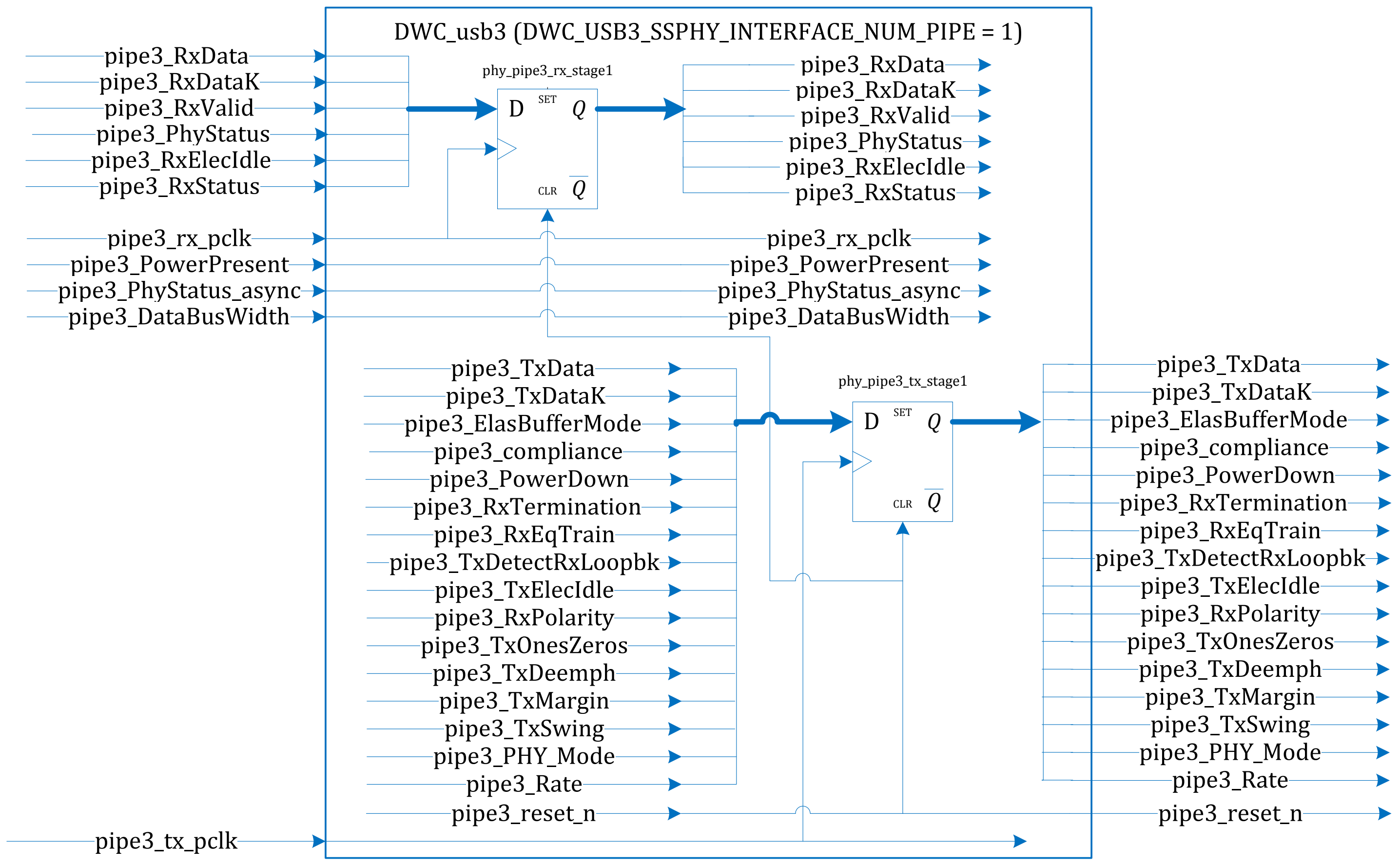
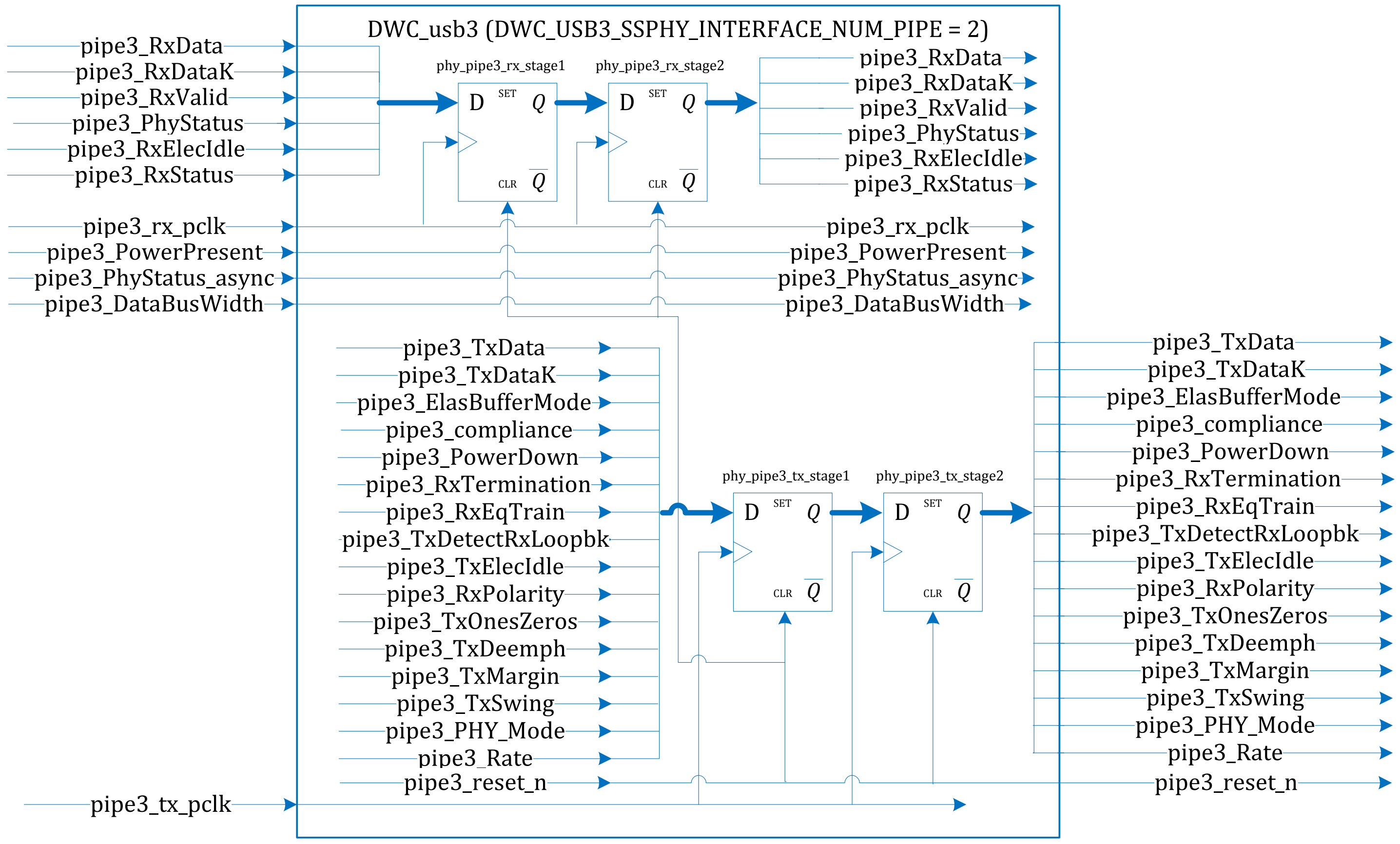


Figure 5-7 Stage 2 of Additional Pipeline



5.3.1.6 PHY Power Down

The controller controls the PHY power down states according to the states of the Link Training State Machine (LTSSM) in the U3LINK module. The power states used are as recommended in the "PHY Interface for the PCI Express and USB 3.0 Architecture" specification.

Note When the PHY is placed in the P3 power down state, the PHY no longer drives PCLK. A slow input clock, suspend_clk drives the small part of the controller that has to operate during PHY P3.

During P3, some of the PHY signals are asynchronous. This is taken into account in the controller logic. One of these signals, pipe3_PhyStatus, may have a pulse width narrower than the suspend_clk period. To reliably capture this signal during P3, the controller includes logic to asynchronously capture the falling edge of this signal. Noise capture is minimized by enabling this capture only when it is necessary.

Before the LTSSM switches the PHY to P3 during the U3 power down state, it checks that any internal activity that requires the PHY clock is complete. If not, the controller uses P2 instead of P3, until the internal activity is complete.

5.3.1.7 Controller Latency for Remote Initiated Low Power Exit

During U1/U2/U3 exit, maximum time gap after an LFPS transmitter stops transmission and before a SuperSpeed transmitter starts transmission is 20 ns. Refer to USB3 specification (Section 6.9.2) for details. The intention of adding this exit condition is to prevent a port from electrical idle before transitioning to Recovery. Otherwise, a port may potentially train its receiver using its own TS1 Ordered Sets due to Near End Cross Talk (NEXT). To meet this requirement, controller will put PHY in P0 before starting LFPS transmitter. An example U1/U2/U3 exit sequence waveform is shown in Figure 5-8.

Figure 5-8 An Example of U1/U2/U3 Exit Sequence

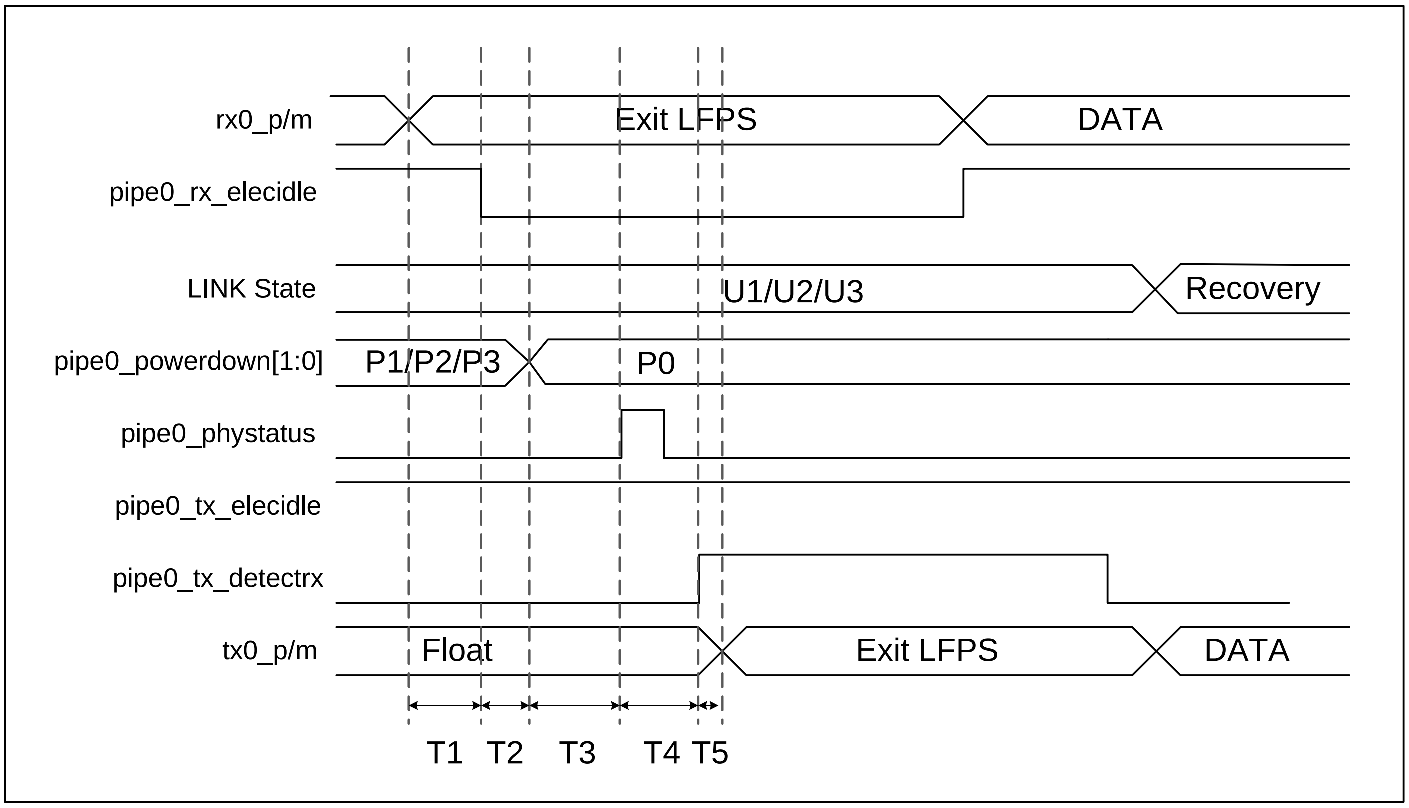


Table 5-2 shows the controller latency for remote initiated low power exit.

Table 5-2 Controller Latency for Remote Initiated Low Power Exit

		8-bit PHY			16-bit PHY			32-bit PHY		
		U1 exit	**U2 exit**	**U3 exit**	**U1 exit**	**U2 exit**	**U3 exit**	**U1 exit**	**U2 exit**	**U3 exit**
LFPS on Rx serial line to de-assertion of pipe3_RxElecIdle	T1	X	X	X	X	X	X	X	X	X
PHY power down request on pipe3_ PowerDown	T2	294ns	310ns	90us	292ns	308ns	90us	304ns	320ns	90us
PHY status at input of PIU	T3	X	X	X	X	X	X	X	X	X
LFPS request from Controller	T4	104ns	104ns	344ns	120ns	120ns	344ns	144ns	160ns	344ns
Assertion of pipe3_TxElecIdle to LFPS on TX serial line	T5	X	X	X	X	X	X	X	X	X
Total Controller time	T6 =T2+T4	398ns	414ns	90.34us	412ns	428ns	90.34us	448ns	480ns	90.34us
USB3.0 Spec Timeout for Ux Exit	T7	900ns	2ms	10ms	900ns	2ms	10ms	900ns	2ms	10ms
Total Time available for PHY	T8 =T7-T6	502ns	1.9ms	9.9ms	488ms	1.9ms	9.9ms	452ns	1.9ms	9.9ms

Note:
1) X is PHY Latency.
2) T1+T3+T5 must be less than T8.
3) In U3, T4 timing depends on width of pipe3_PhyStatus.

5.3.1.8 PHY and Link-Related Controls and Synthesis Options

5.3.1.8.1 PHY Loopback

Both Host and Device-only modes support PHY loopback in slave mode. They do not support PHY loopback in master mode.

When the Host/Device with PHY connects to a Tester or VIP, during the polling state, the Tester/VIP sends TS2 with loopback bit enabled. The Host/Device changes the link state from Polling to Loopback and asserts pipe3_TxDetectRxLoopbk. Then, the Tester/VIP sends TX data to the Host/Device, and the Host/Device PHY decodes the data and sends it back. There is no programming involved. When the Tester is done with the loopback testing and wants to exit loopback mode, it does U2 or Loopback (LFPS handshake) exit.

SS Loopback operation is described in Section 6.8.4 of the *USB 3.0 Specification*. The loopback pattern consists of one BRST set, and a repeating sequence of TB BDAT sets (scrambled idles) and a BERC set. You can estimate the Link Error Rate from this count, and the length of time that the test runs.

The master and slave will exit Loopback under the conditions described in the USB 3.0 specification Section 7.5.11.3.2.

5.3.1.8.2 PHY Data Width

PHY data width is a two bit input signal to the controller as described in Chapter 'Signals' of the *DWC SuperSpeed USB 3.0 Controller Databook*. This signal is sampled once immediately after resets are deasserted.

5.3.1.8.3 GUSB3PIPECTL*n* Register

The LFPS P0 Align, P3 P2 Transitions OK, and P3 Exit Signal in P2 bits allow you to minimize the time between the end of LFPS transmission and the start of symbol transmission while exiting the U1, U2, and U3 power states. See the USB 3.0 specification, Section 6.9.2 for requirements.

LFPS Filter provides an additional condition for the controller to detect reception of LFPS from the PHY.

Skip Rx Detect suppresses Rx Detection under certain conditions that may be required for some PHYs.

TxSwing, TxMargin, TxDemphasis, Elastic Buffer Mode controls are set directly in the GUSB3PIPECTLn register. These controls are driven directly to the PHY, except for TxDemphasis which is overridden during Compliance mode as described in the USB 3.0 specification.

In the following timing diagrams in this section, the red arrow indicates PHY-related actions and the black arrow indicates Controller actions.

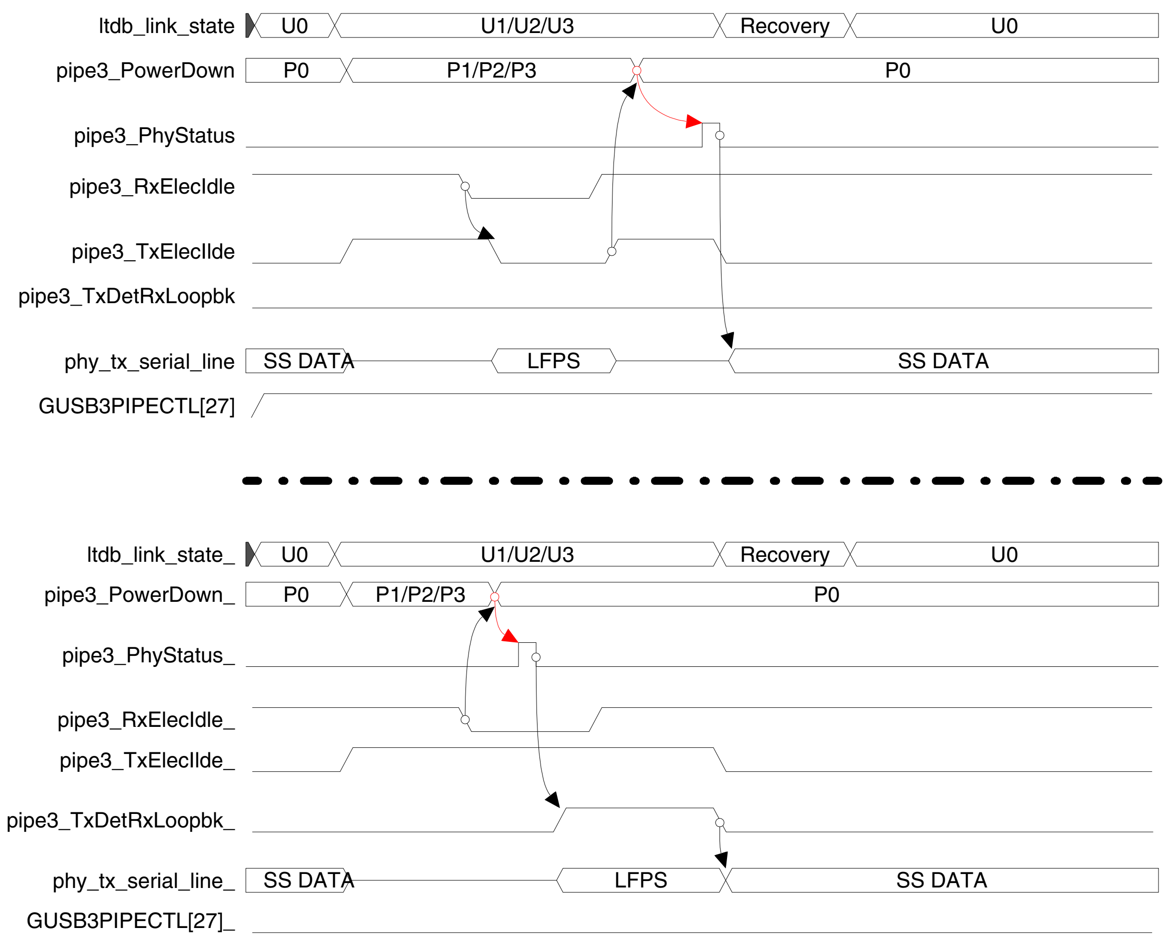
GUSB3PIPECTL[27] (U*x* Exit IN P*x*)

- 0: The controller performs a U1/U2/U3 exit in P0 (default behavior).
- 1: The controller performs a U1/U2/U3 exit in P1/P2/P3, respectively.

Setting this bit to "1" violates the USB 3.0 Specification, Section 6.9: "For U1 exit, U2 exit, U3 Wakeup and not Loopback exit, link partner 1 is ready to transmit the training sequences and the maximum time gap after an LFPS transmitter stops transmission and before a Super Speed transmitter starts transmission is 20 ns." Controller will take more than 20ns to switch to SS transmission.

Figure 5-9 on page 147 shows a timing diagram of this behavior.

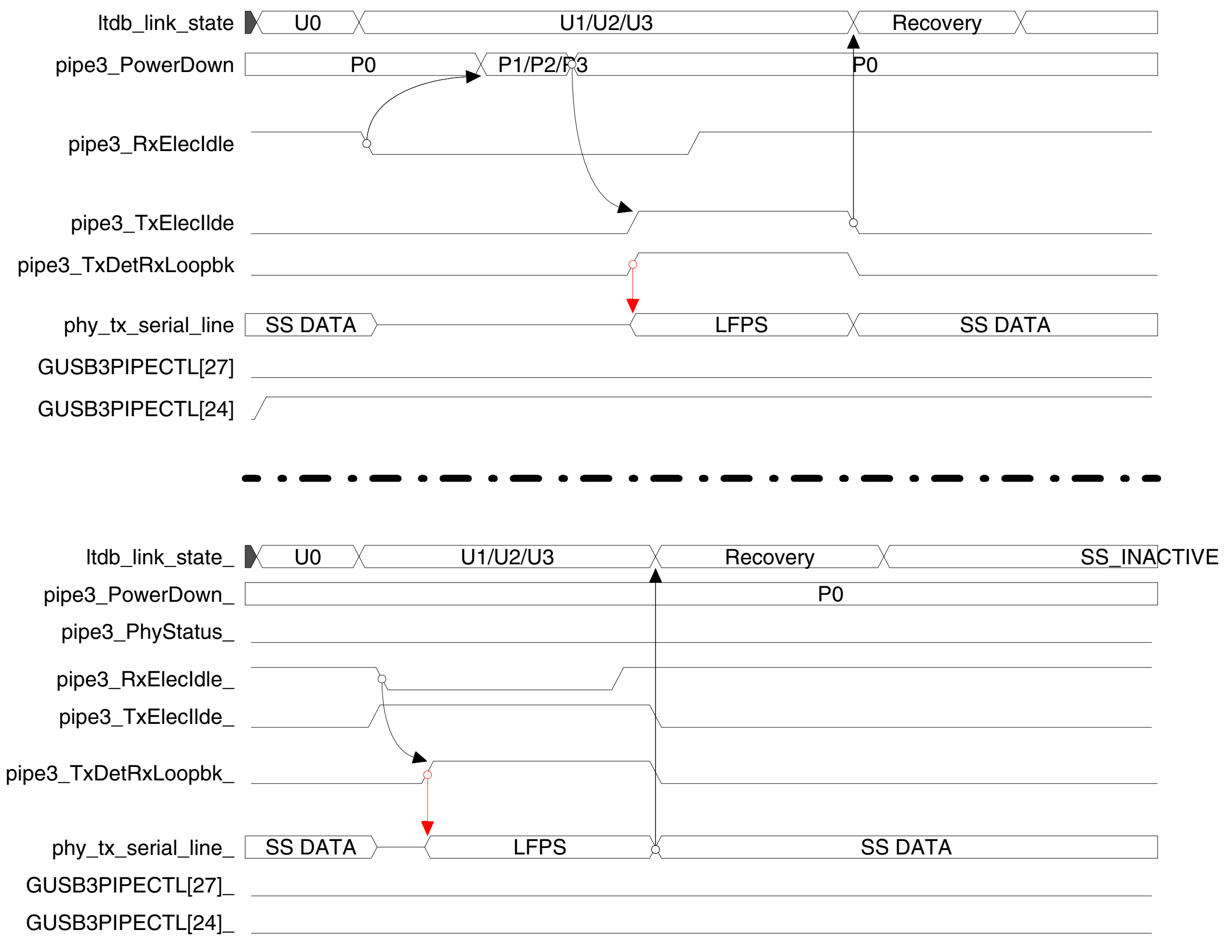
Figure 5-9 GUSB3PIPECTL[27] Timing Diagram (U*x* Exit in P*x*)



GUSB3PIPECTL[24]

When set, the controller always requests a PHY power change from a P0 to P1/P2/P3 during U0 to U1/U2/U3 transition. If this bit is 0 and an immediate U*x* exit (remote initiated or locally initiated) happens, the controller may not request a P1/P2/P3 power state change. This should be "1" for a Synopsys PHY. For a non-Synopsys PHY, check with your PHY vendor.

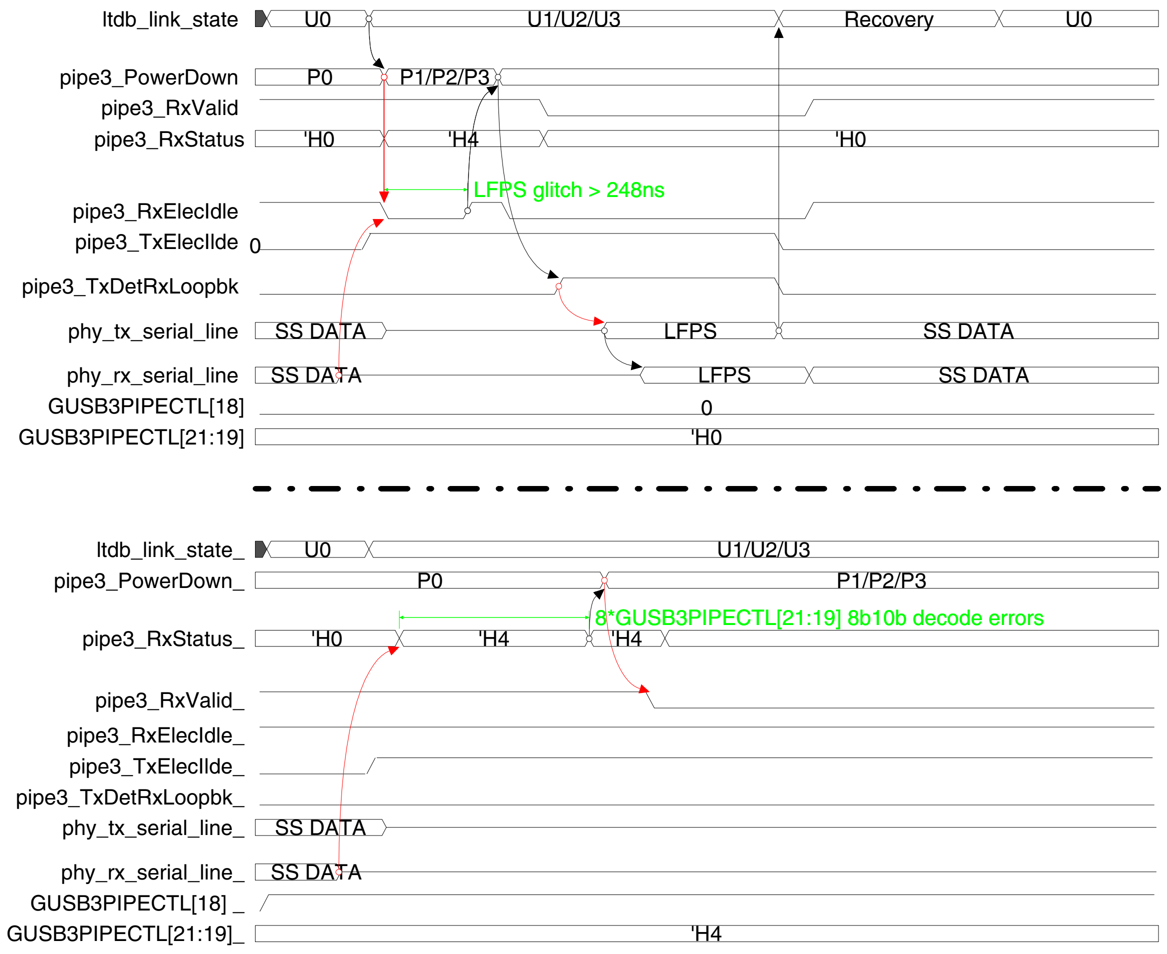
Figure 5-10 GUSB3PIPECTL[24] Timing Diagram



GUSB3PIPECTL[21:19]

These register bits delay P0 to P1/P2/P3 request when entering U1/U2/U3 state until a (DWC_USB3_GUSB3PIPECTL_INIT[21:19]*8) number of 8B10B error happens or when the pipe3_RxValid signal drops. To enable the functionality of these register bits, the DWC_USB3_GUSB3PIPECTL_INIT[18] bit must be set to '1'. For Synopsys PHY, GUSB3PIPECTL[21:19] must be set to 3'b001.

Figure 5-11 GUSB3PIPECTL[21:19] Timing Diagram



GUSB3PIPECTL[18]

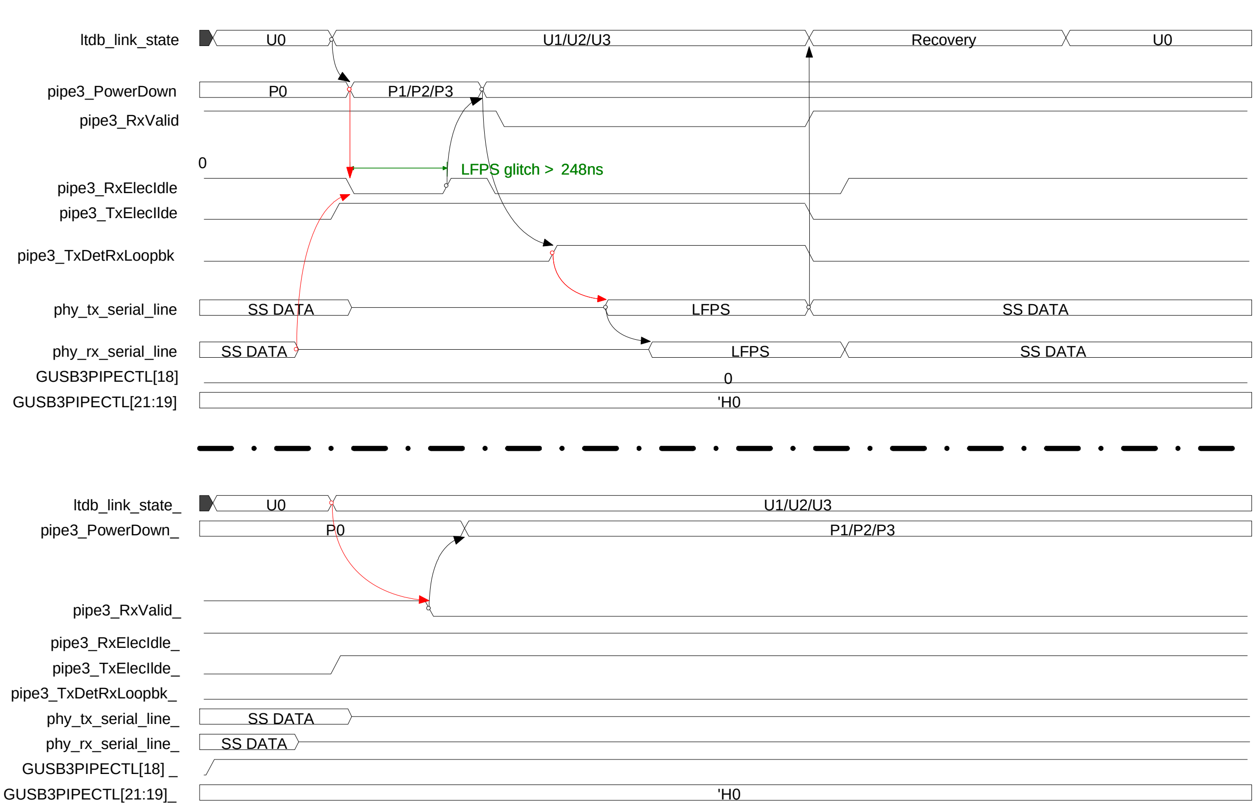
This register bit delays a P1/P2/P3 transition when entering U1/U2/U3 until pipe3_RxElecIdle is 1 and pipe3_Rxvalid is 0.

- 1: Delay P1/P2/P3 transition.
- 0: Do not delay P1/P2/P3 based on pipe3_RxElecIdle and pipe3_Rxvalid.

If you are using Synopsys PHY, contact Synopsys Customer Support for recommendation on setting this bit because it is node dependent.

If you are using a third-party SS PHY, check with your PHY vendor for recommendation on setting this bit.

Figure 5-12 GUSB3PIPECTL[18] Timing Diagram

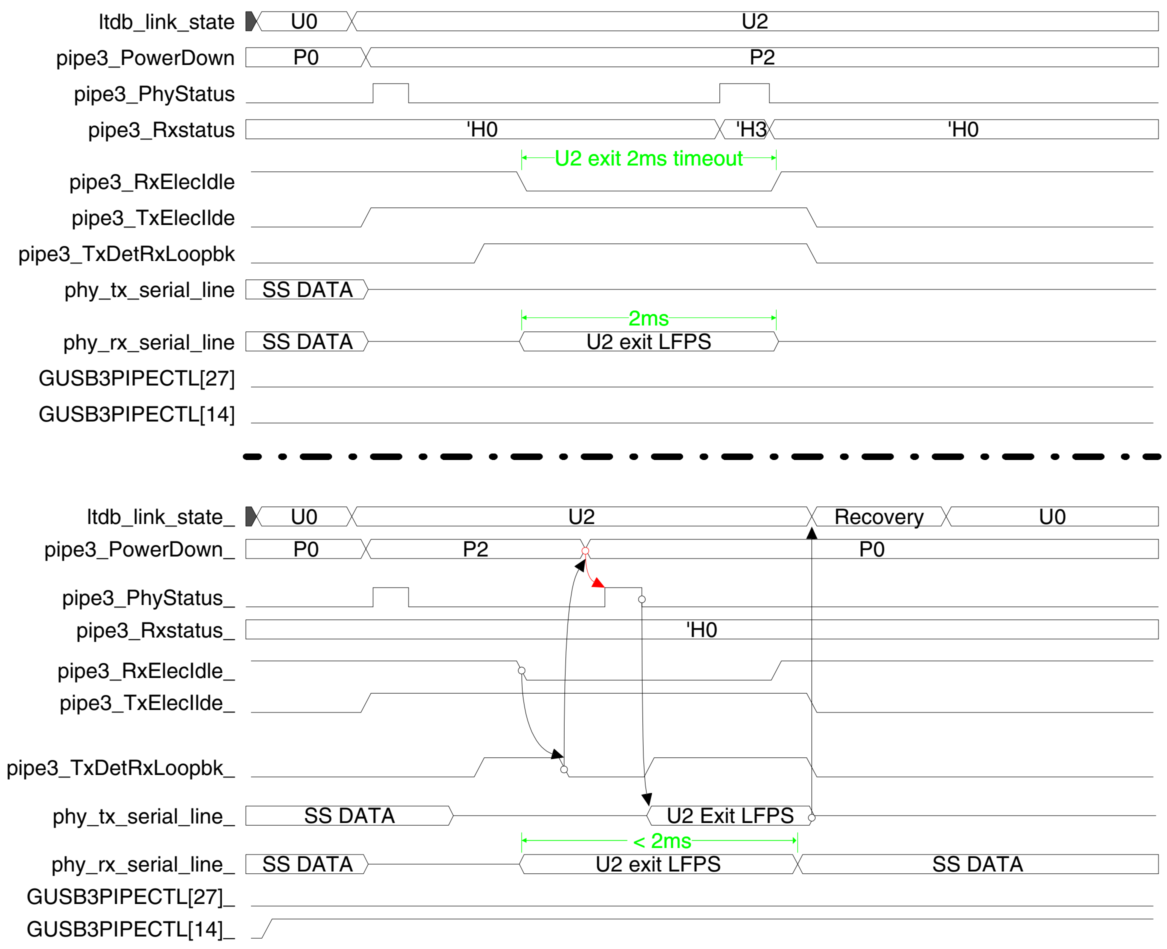


GUSB3PIPECTL[14]

In the link state U2, this bit aborts receiver detection if the remote partner starts U2 exit. This bit should be used only if the SS PHY takes more than 2ms for receiver detection in P2. The Synopsys PHY should be ‘0’. It is used by the third-party SS PHY.

Setting this bit to “1” violates the PIPE3 Specification, Section 6.8, “Receiver Detection”.

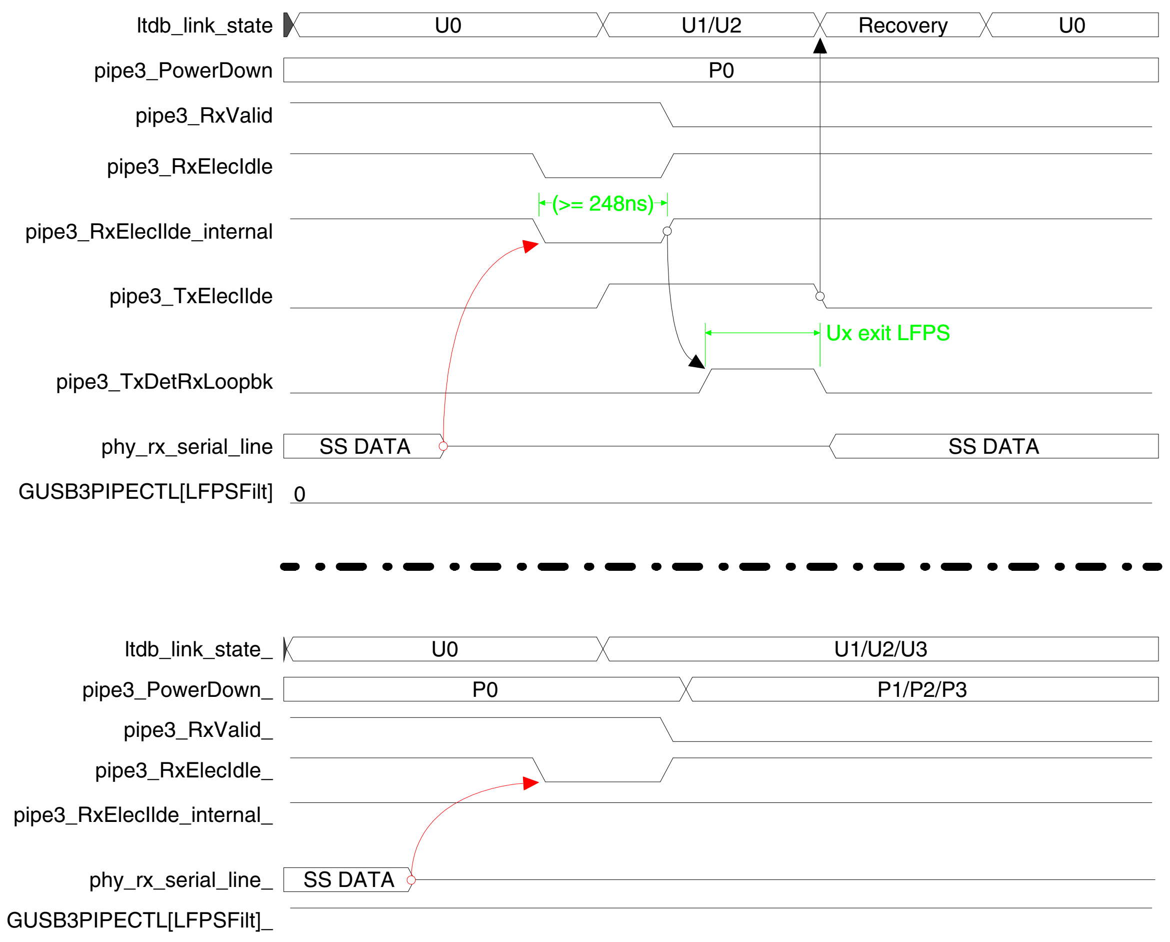
Figure 5-13 GUSB3PIPECTL[14] Timing Diagram



GUSB3PIPECTL[9]

LFPS Filter. When set, filter LFPS reception with pipe3_RxValid in PHY power state P0, that is, ignore LFPS reception from the PHY unless both pipe3_Rxelecidle and pipe3_RxValid are de-asserted.

Figure 5-14 GUSB3PIPECTL[9] Timing Diagram



5.3.1.8.4 GCTL

For information on the Power Down Scale, Debug Attach, Scale Down Mode, and Disable Scrambling register fields, see GCTL register definition in the "Registers" chapter of the *DWC SuperSpeed USB 3.0 Controller Programming Guide*.

Bit 2 (U2EXIT_LFPS) of the GCTL register controls the duration of LFPS detected to be treated as a valid LFPS exit from the link partner. If this bit is set to '1', the controller detects a 8µs LFPS as a valid LFPS for U2 exit. When it is not set, then the controller detects a 248ns LFPS as a valid LFPS exit for U2 exit. For Synopsys PHY, this bit must be set to '1'.

5.3.1.8.5 Global Debug LTSSM Register

For information on the Global Debug LTSSM register, see GDBGLTSSM register definition in the "Registers" chapter of the *DWC SuperSpeed USB 3.0 Controller Programming Guide*.

Note These signals are not synchronized to the bus clock, and are therefore only valid when controller activity is stable.

5.3.1.8.6 Device Control Register

For information on the Run/Stop, Target USB/Link State, U1/U2 Controls, and USB/Link State Change Requests, see DCTL register definition in the "Registers" chapter of the *DWC SuperSpeed USB 3.0 Controller Programming Guide*.

Note that the Link State Change Request field may be used to request that the SS Link LTSSM should transition to the SS.Disabled state. In addition, for a peripheral device, the controller will force the LTSSM into the SS.Disabled state if VBUS is absent (pipe_PowerPresent is not asserted)

5.3.1.8.7 Global User Control Register (GUCTL)

For information on the Enable Check for LFPS Overlap During Remote Ux Exit (EnOverlapChk) field, see GUCTL register definition in the "Registers" chapter of the *DWC SuperSpeed USB 3.0 Controller Programming Guide*.

If EnOverlapChk bit is set to 1'b1, the SuperSpeed link when exiting U1/U2/U3 waits for either the remote link LFPS or TS1/TS2 training symbols before it confirms that the LFPS handshake is complete. .

5.3.2 Integrating with USB 2.0 PHY

Note IC-USB, HSIC, and 3-pin/6-pin USB 1.1 Serial Interface are currently not supported.

This section explains the different USB 2.0 PHY interfaces supported by the DWC_usb3 controller. The topics in this section are:

- "Connecting UTMI+ PHY" on page 154
- "Connecting ULPI PHY Interface" on page 160

The PHY interface(s) available on the controller depends on the value specified for High-Speed PHY interface(s) during coreConsultant configuration. After power-on, you can select either the UTMI+ or ULPI using software.

5.3.2.1 Connecting UTMI+ PHY

This section describes how to connect your UTMI PHY to the DWC_usb3 interface. This section also discusses the Vendor Control interface.

5.3.2.1.1 Connecting UTMI+ PHY Dedicated Interface

The UTMI+ PHY interface is selected for High-Speed PHY Interface(s) during coreConsultant configuration.

Figures 5-15 and 5-16 show unidirectional and bidirectional connections to a dedicated UTMI+ PHY interface.

Figures 5-17, 5-19, 5-21 and 5-22 show the utmi_suspend_n, utmi_suspend_com_n and utmi_clk signal connections to the UTMI+ PHY.

Figure 5-15 UTMI+ PHY Dedicated Interface (Unidirectional)

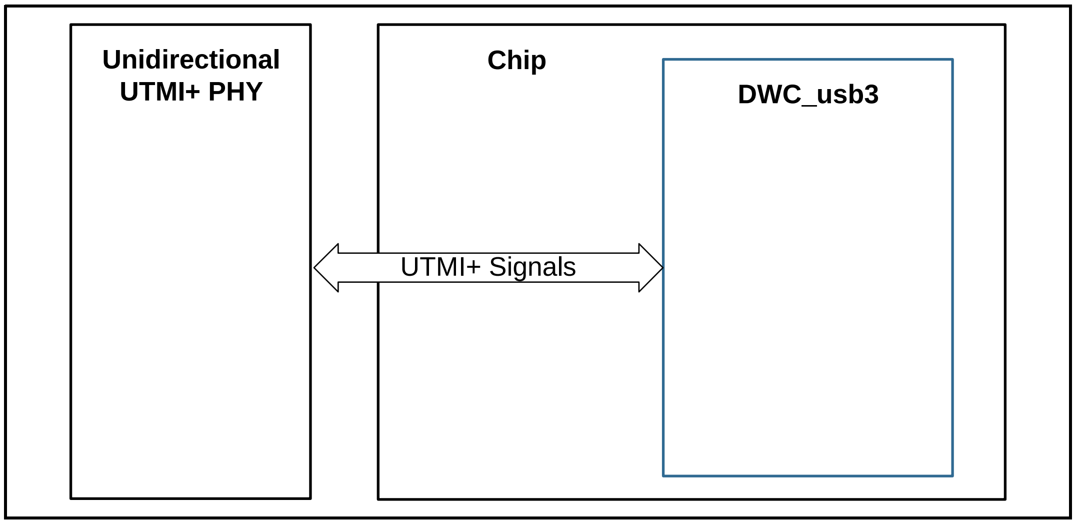


Figure 5-16 UTMI+ PHY Dedicated Interface (Bidirectional)

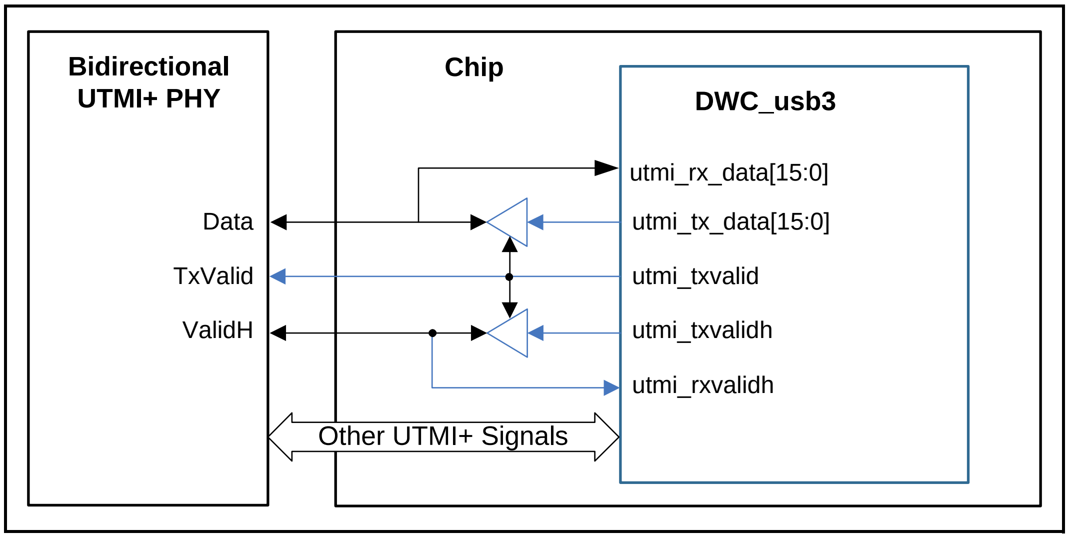


Figure 5-17 Connecting DWC_usb3 Device to Synopsys Single-Port UTMI+ PHY

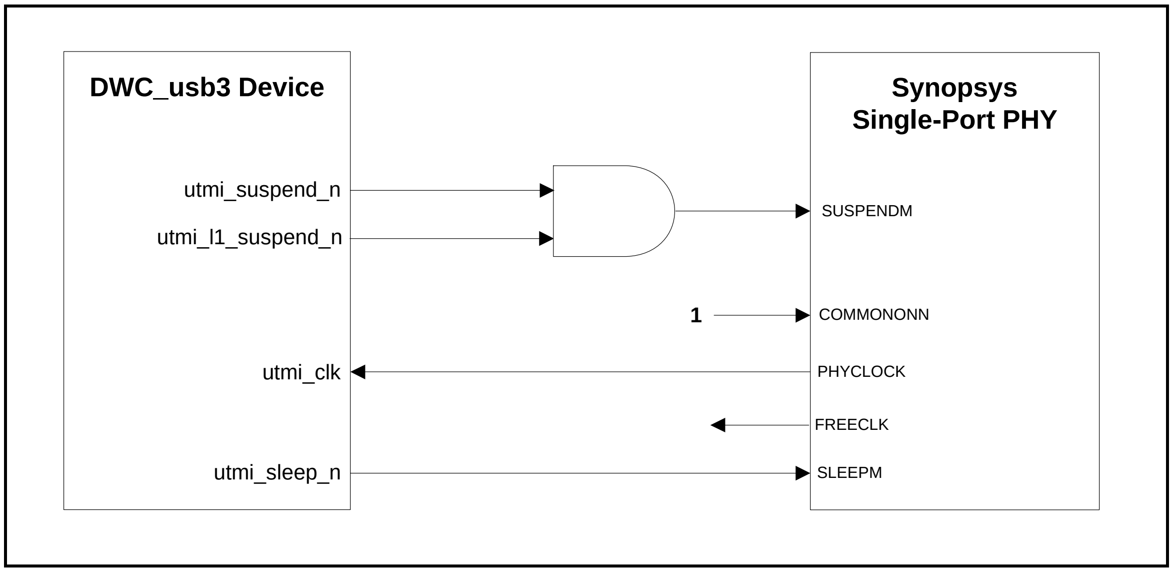


Figure 5-18 Connecting DWC_usb3 Host/DRD to Synopsys Multi-Port UTMI+ PHY

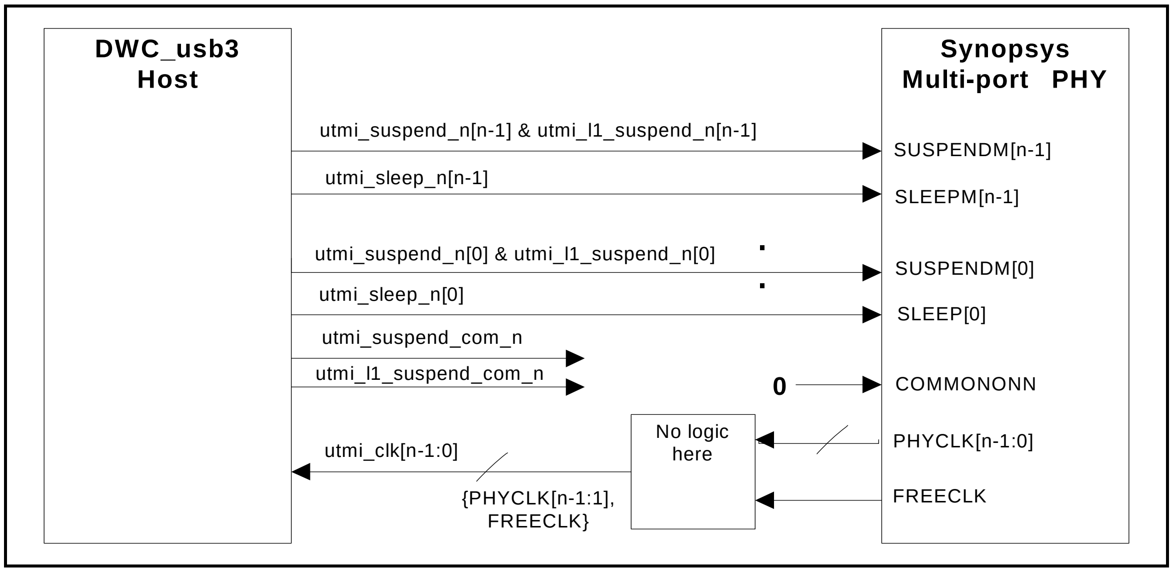


Figure 5-19 Connecting DWC_usb3 Host/DRD to Synopsys Single-Port UTMI+ PHY

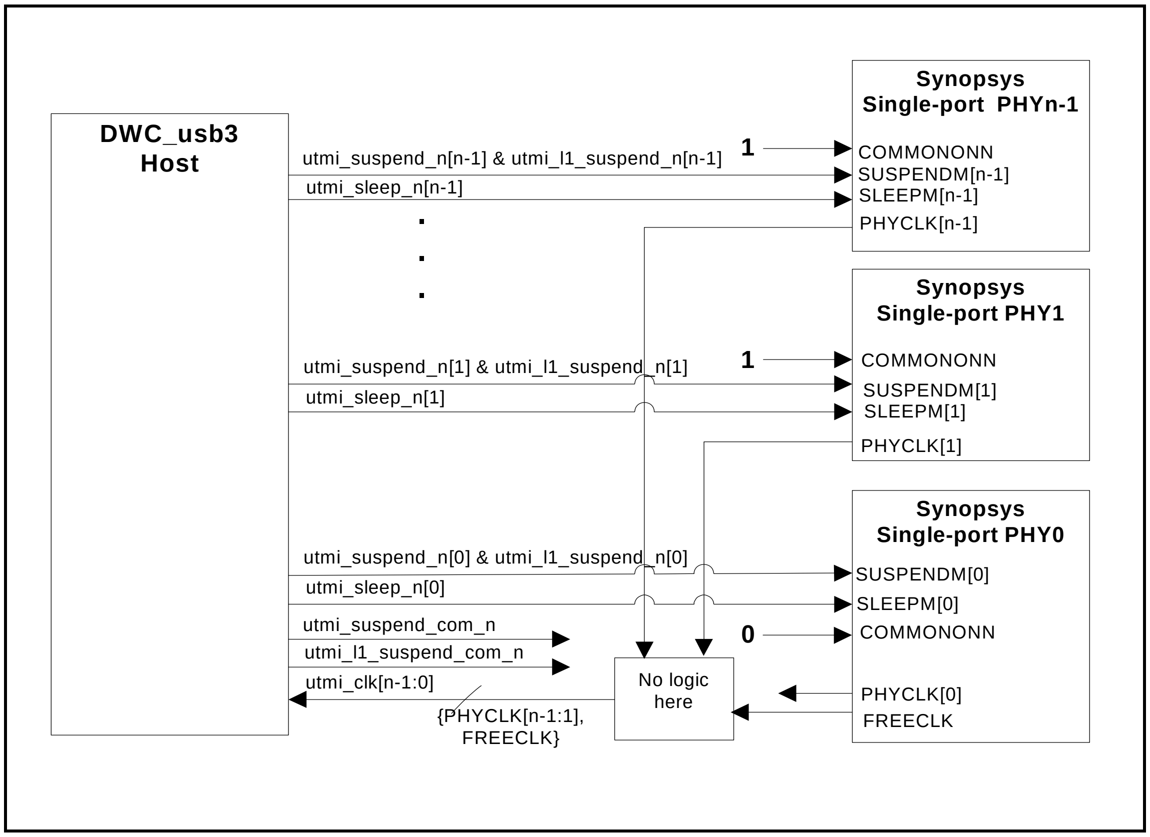
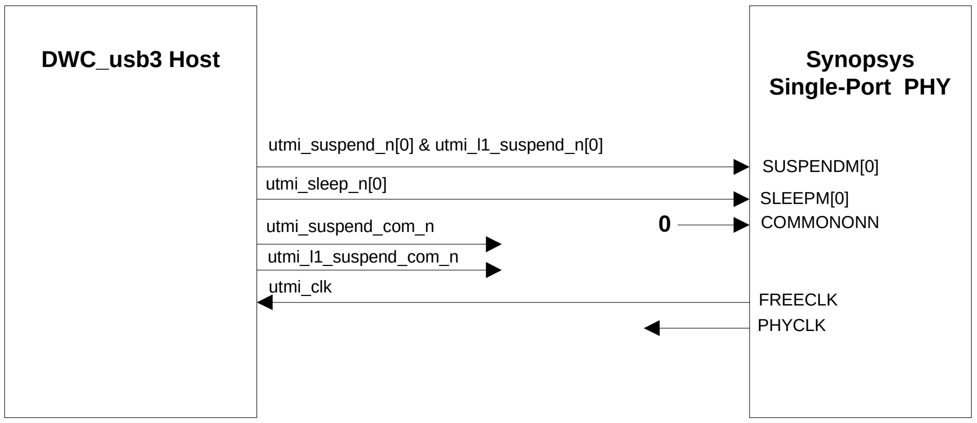


Figure 5-20 Connecting Single-port DWC_usb3 Host/DRD to Synopsys Single-Port UTMI+ PHY



Note

- For Synopsys PHY, if the COMMONONN is tied low, the FREECLK runs all the time that is, the internal PLL is active. If the driver can ensure that the USB2.0 Port0 is suspended only when all the USB2.0 and USB3.0 ports are in suspended/disconnected state, then the COMMONONN can be tied high. This enables complete shutdown of the PHY during suspend operation. If the controller needs the hardware LPM to be enabled, then COMMONONN should be tied low to ensure the utmi_clk[0] is present all the time.
- If you plan to tie COMMONONN low, check with the PHY vendor if this meets the requirements for the suspend current. With this connection, the free-running clock always runs even if the controller is L1/L2 suspended.
- By default, the utmi_suspend_n, utmi_l1_suspend_n, and utmi_sleep_n are enabled (GUSB2PHYCFG[8]=1, GUSB2PHYCFG[6] =1) when the controller is operating in host mode. The software should not disable these bits unless the PHY can handle the remote wakeup if the suspend input is not asserted.
- The utmi_clk[0] input of the controller is to be connected to the free-running clock output of the PHY. Other utmi_clk[n] inputs of the controller are to be connected to respective port clocks of the PHY.
- Each port clock can be stopped/started based on its own utmi_suspend_n, utmi_l1_suspend_n or utmi_sleep_n, except Port0 clock which is a free running clock.
- If you select GCTL.SOFITPSYNC feature and not plan on supporting xHCI Hardware LPM feature, then you can connect COMMONON to '1'. This allows the UTMI PHY PLL to be turned off if there is no active USB 2.0 port (irrespective of the USB 3.0 port states). If you select GFLADJ.GFLADJ_REFCLK_LPM_SEL feature, you can connect COMMONON to '1'. This allows the UTMI PHY PLL to be turned off if there is no active USB 2.0 port (irrespective of the USB 3.0 port states) even with xHCI Hardware LPM feature enabled.

Figure 5-21 Connecting DWC_usb3 Host/DRD to Third-Party Multi-Port UTMI+ PHY (Without Free-Running Clock)

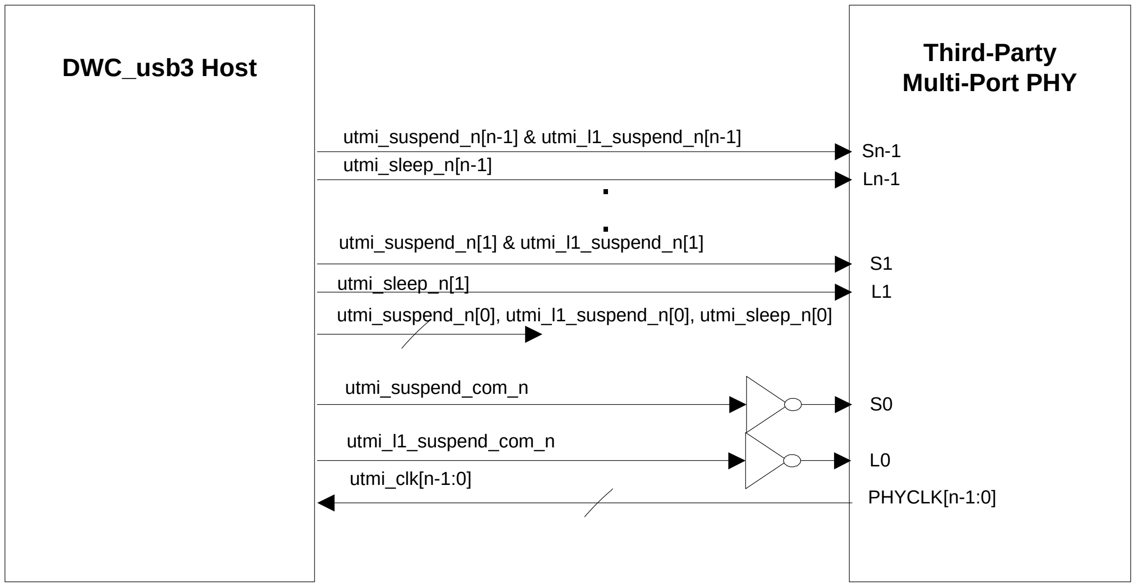
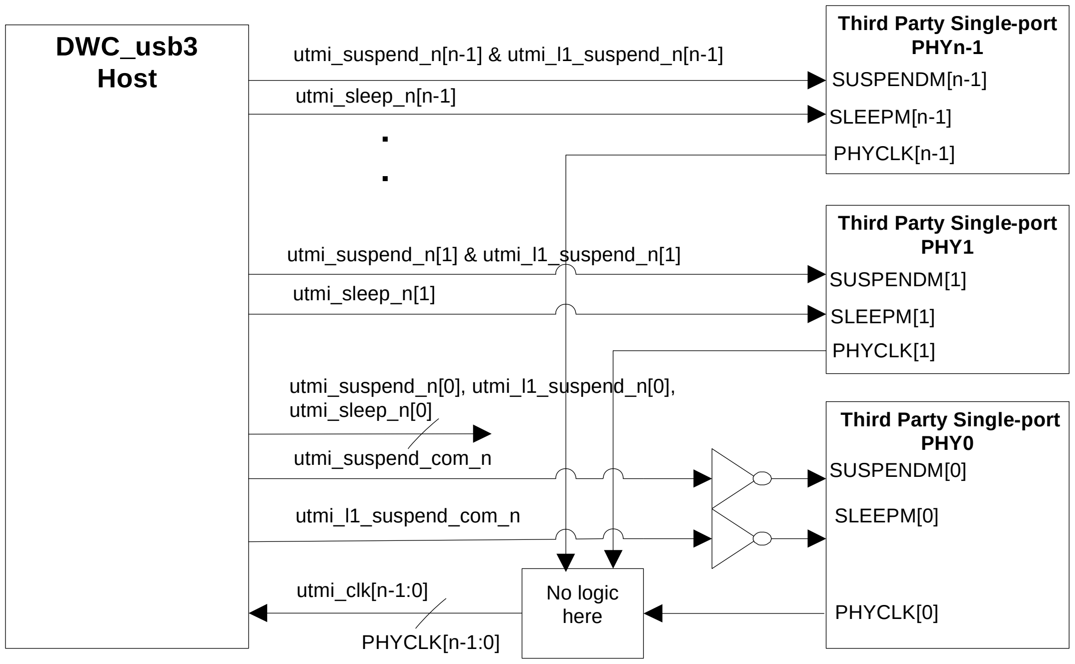

DWC_usb3 Host
Third-Party Multi-Port PHY
utmi_suspend_n[n-1] & utmi_l1_suspend_n[n-1]
utmi_sleep_n[n-1]
Sn-1
Ln-1
utmi_suspend_n[1] & utmi_l1_suspend_n[1]
utmi_sleep_n[1]
S1
L1
utmi_suspend_n[0], utmi_l1_suspend_n[0], utmi_sleep_n[0]
utmi_suspend_com_n
S0
utmi_l1_suspend_com_n
L0
utmi_clk[n-1:0]
PHYCLK[n-1:0]

Figure 5-22 Connecting DWC_usb3 Host/DRD to Third-Party Single-Port UTMI+ PHY Without Free-Running Clock



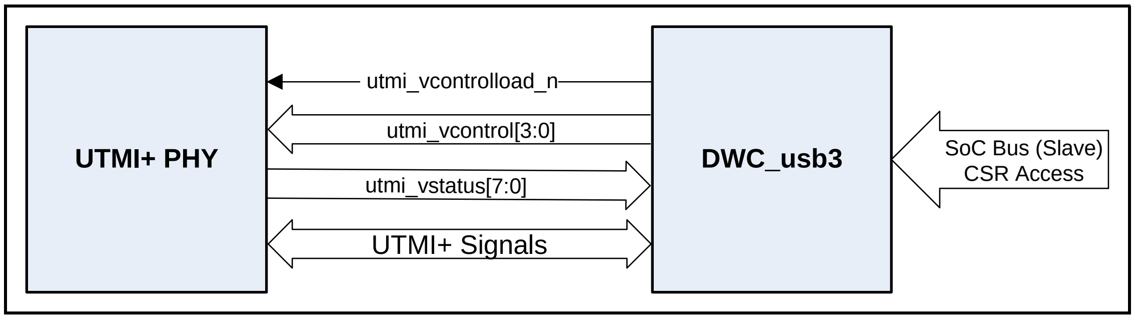
Note

- Third-party PHYs should ensure that they handle the suspend/resume and remote wakeup conditions properly when the SuspendM (S0 in Figure 5-22) input is not active.
- The first port clock from the PHY is used as a free-running clock for the controller. It is controlled by the controller using the control signals utmi_suspend_n and utmi_l1_suspend_n.
- By default, the utmi_suspend_n, utmi_l1_suspend_n, and utmi_sleep_n are enabled (GUSB2PHYCFG[6] =1, GUSB2PHYCFG[8]=1) when the controller is operating in host mode. The software should not disable these bits to ensure that the Port0 clock is stopped when it is no longer required.
- The utmi_clk[0] input of the controller is to be connected to the first port clock output of the PHY. Other utmi_clk[n] inputs of the controller are to be connected to the respective port clocks of the PHY.
- Each port clock (except Port0 clock) can be stopped or started based on its own utmi_suspend_n, utmi_l1_suspend_n or utmi_sleep_n.
- If you connect the utmi_suspend_com_n signal to Port0 of the PHY, Port0 clock continues to run until all other ports (SS ports and USB 2.0 ports) are in suspend state.

5.3.2.1.2 Connecting UTMI+ Vendor Control Interface

Figure 5-23 shows the UTMI+ PHY Vendor Control interface. This feature must be enabled during coreConsultant configuration. The software programs the PHY Vendor Control register for PHY register access.

Figure 5-23 UTMI+ Vendor Control Interface



5.3.2.2 Connecting ULPI PHY Interface

This section describes how to connect your ULPI PHY to the DWC_usb3 interface. This section also discusses pin sharing on this interface with the (internal) Vendor Control interface.

5.3.2.2.1 Connecting ULPI Clock

The ULPI PHY interface must be selected for High-Speed PHY Interface(s) during coreConsultant configuration. Figure 5-24 and Figure 5-25 shows how to connect ULPI PHY with and without free-running clocks.

Figure 5-24 Connecting DWC_usb3 Host/DRD to ULPI PHY with Free-Running Clock

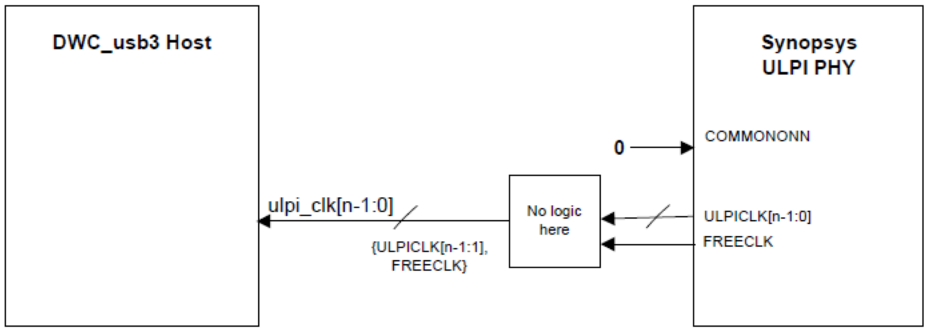
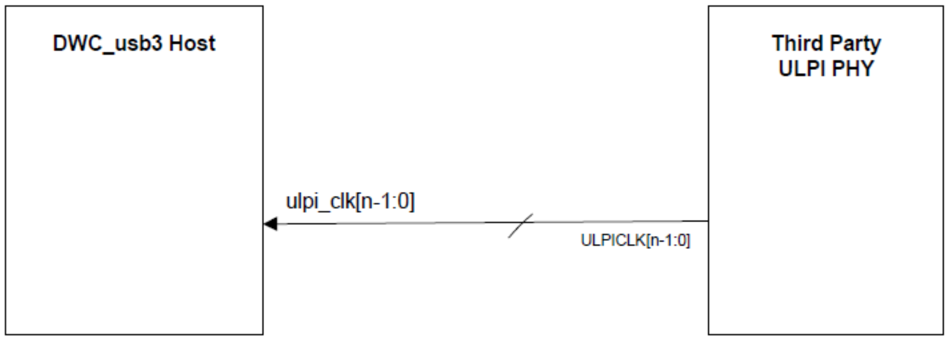


Figure 5-25 Connecting DWC_usb3 Host/DRD to ULPI PHY Without Free-Running Clock

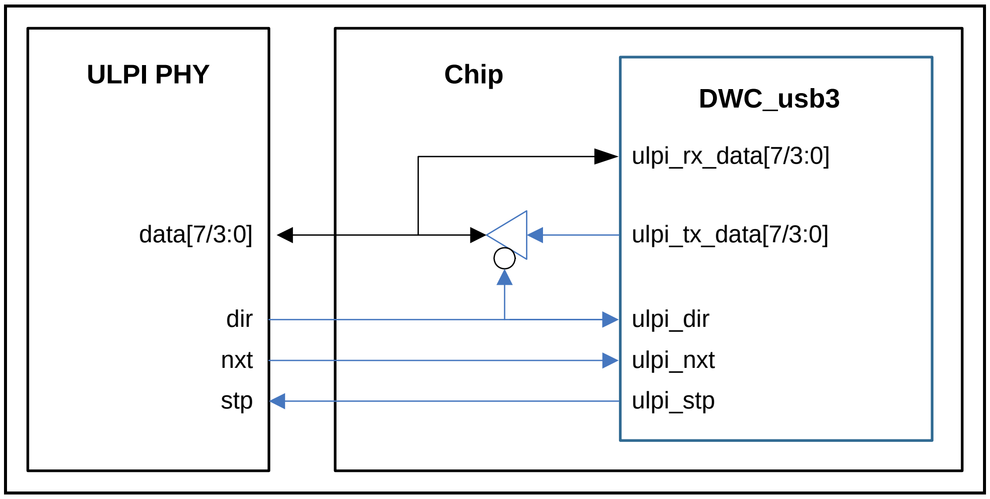


5.3.2.2.2 Connecting ULPI PHY Dedicated Interface

The ULPI PHY interface must be selected for High-Speed PHY Interface(s) during coreConsultant configuration.

Figure 5-26 shows standard connections to a dedicated ULPI PHY interface.

Figure 5-26 ULPI PHY Dedicated Interface



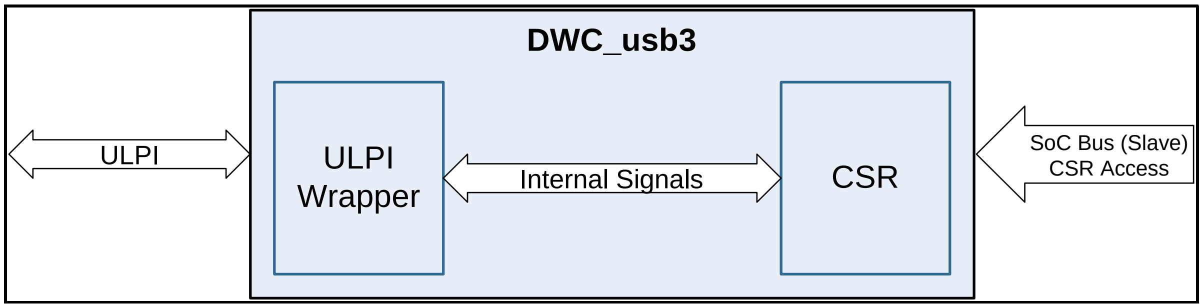
Note The ulpi_tx_data_en signal can also be used to control the tri-state buffers as shown in Figure 5-26.

5.3.2.2.3 Implementing ULPI Vendor Control Access (Internal)

For ULPI, the Vendor Control interface is internal to the DWC_usb3 controller, but must be enabled during coreConsultant configuration. The software programs the PHY Vendor Control register for PHY register access. The PHY register access is translated as a register read/write command in the ULPI.

Figure 5-27 shows the Vendor Control interface.

Figure 5-27 ULPI Vendor Control Interface (Internal)



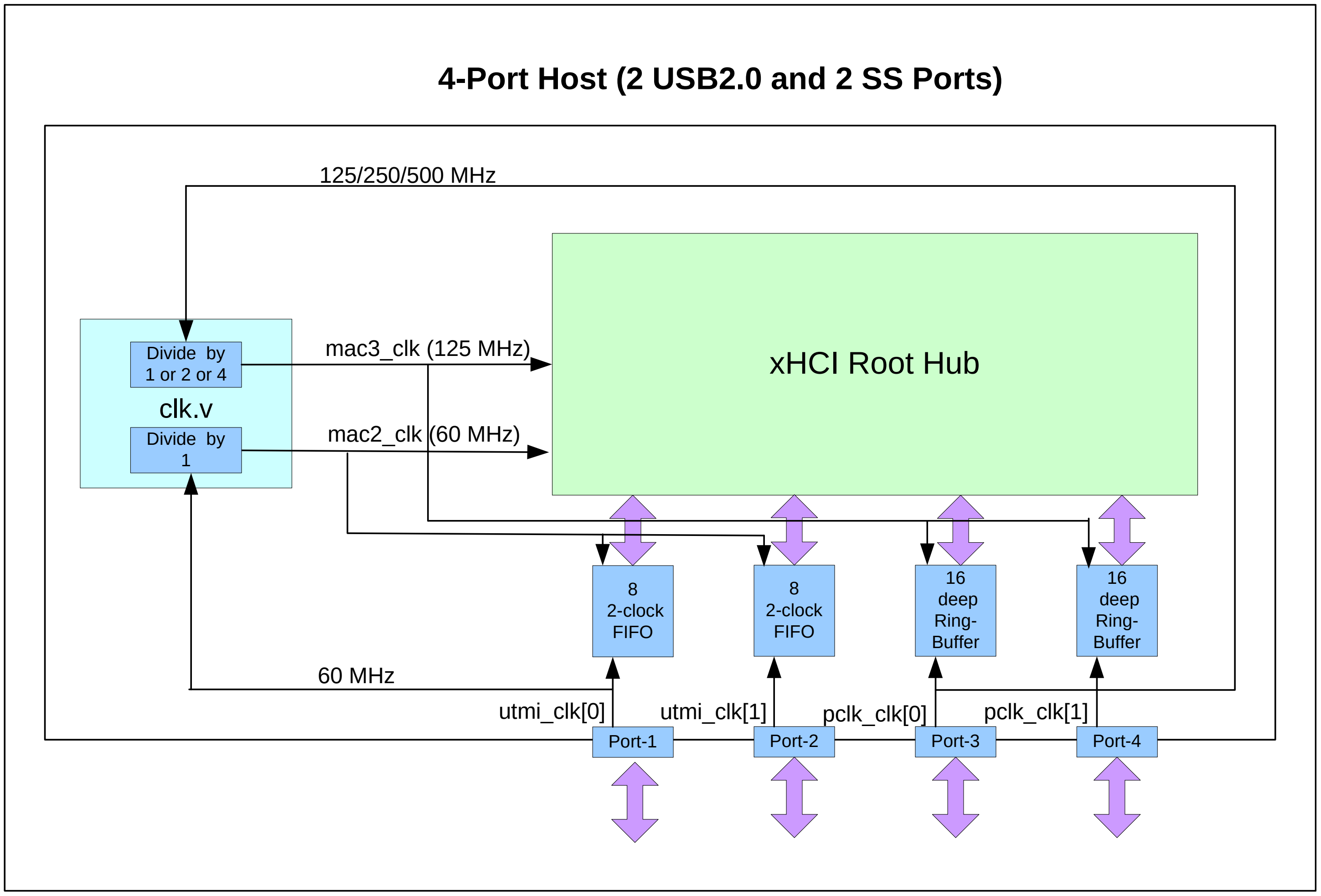
5.3.3 Connecting Multiple Ports in Host and DRD Modes

Figure 5-28 on page 164 shows multi-port PHY interface and the clocks involved.

- The minimum number of SS ports is one and the minimum number of USB2.0 ports is also one. The number of SS ports and the USB2.0 ports can be different. For example, you can have two SS ports and four HS ports.
- In DRD mode, multiple host ports are supported.
- You can plan to use multiple single-port SS and USB2.0 PHYs. The Synopsys xHCI controller does not require you to design a multi-port PHY with all the ports having phase-aligned port clocks.
- The controller has built-in ring-buffers to adjust for the clock PPM difference between the port clocks and the internal mac3_clk/mac_clk. For example, in the SS mode, each port can have its own spread-spectrum clock.
- The first SS port's PHY clock and the first USB2.0 port's PHY clocks are used by the Synopsys xHCI controller to generate the internal common mac3_clk and mac2_clk. The controller keeps the first SS port in P2 state until all the SS ports are in P3 to make sure mac3_clk is available. Similarly, the controller also keeps the first USB2.0 port in non-suspend mode until all the USB2.0 ports are in suspend state and all the SS ports are in P3 state. The mac2_clk is used for SOF and ITP generation.

  If the GCTL.SOFITPSYNC feature is enabled, then the controller uses the ref_clk for ITP generation. In this case, the first USB 2.0 port can be suspended if all the USB 2.0 ports are in a suspend state (independent of SS port's P3 state). For more information regarding the GCTL.SOFITPSYNC bit description, refer to the *DesignWare Cores SuperSpeed USB 3.0 Controller Databook*.
- In DRD configuration:
  - Host role: The controller generates mac_clk from mac2_clk.
  - Device role: The controller generates mac_clk from mac3_clk or mac2_clk depending on whether it is operating in SS mode or not.
- The controller meets turnaround time with the ring-buffers.

Figure 5-28 Connecting Multiple Ports in Host/DRD Mode



5.3.4 Filtering VBUSValid Signal in Host and DRD-Host Modes

The output of an external voltage comparator should be input to utmiotg_vbusvalid/pipe3_PowerPresent. For threshold value of this voltage comparator refer to the description of utmiotg_vbusvalid in *DWC SuperSpeed USB 3.0 Controller Databook*.

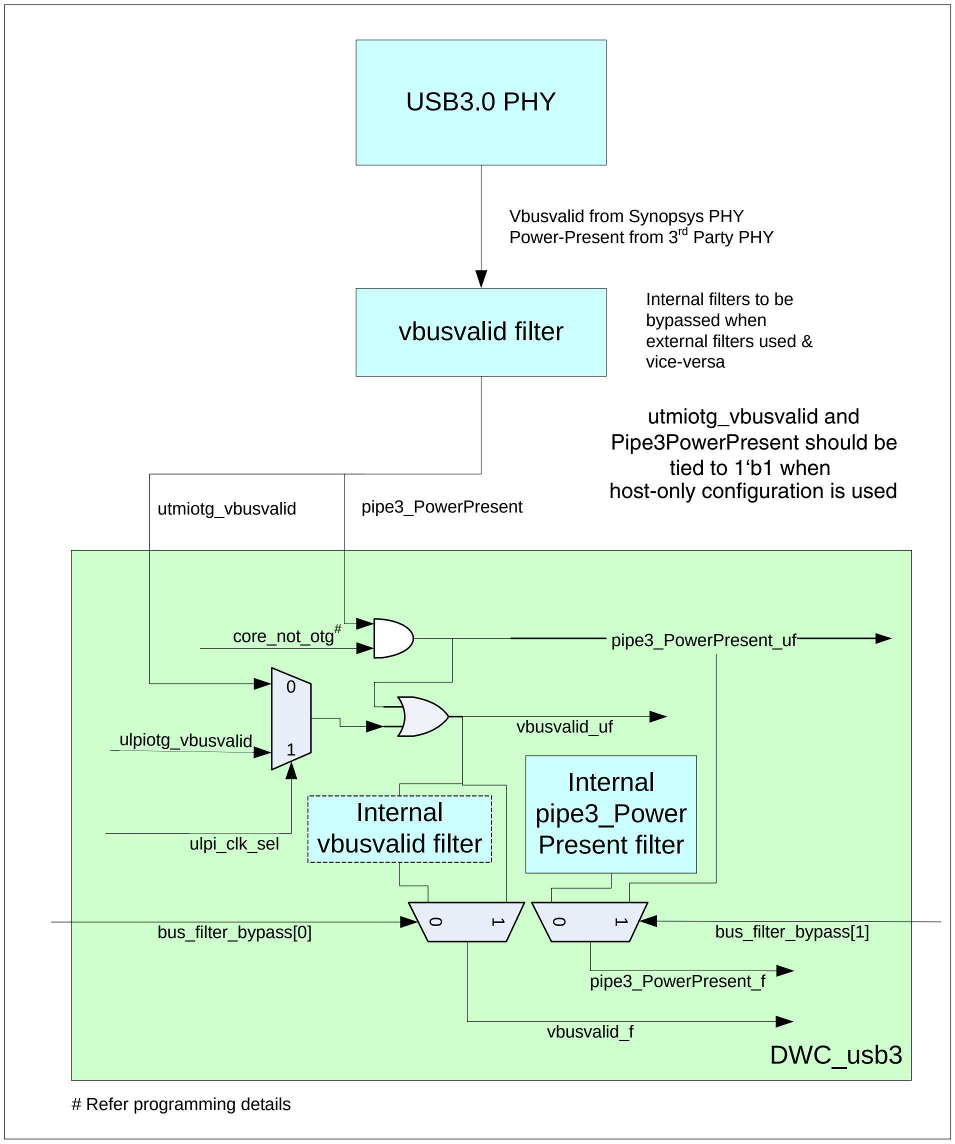
When using Synopsys PHY, use vbusvalid output from the PHY. The vbusvalid and power_present signals from Synopsys PHY are valid only when internal V_{BUS} comparators are used. Check with the PHY vendor whether the power-present/vbusvalid outputs are valid in these modes.

If the coreConsultant parameter DWC_USB3_EN_BUS_FILTERS is chosen as 0 or bus_filter_bypass[2] is set to 1’b1, this should be a filtered signal. In this case, the power-present signal from the PHY needs to be filtered (~5-ms debounce filter) outside the controller before connecting to the utmiotg_vbusvalid and pipe3_PowerPresent signals. Once filtered, vbusvalid (Voltage BUS Valid) signal should be connected to both the ports of the DWC_usb3 controller. If DWC_USB3_EN_BUS_FILTERS is 1 and bus_filter_bypass[2] is 1’b0, the external filter is not required.

The utmiotg_vbusvalid input to the controller should be driven when DWC_USB3_HSPHY_INTERFACE is UTMI. For ULPI mode of DWC_USB3_HSPHY_INTERFACE, the controller extracts vbusvalid signal (ulpi_vbusvalid) from ulpi_datain through RxCmd as per ULPI specifications. The pipe3_PowerPresent is ORed with either utmiotg_vbusvalid or ulpi_vbusvalid internally to generate the final power-present. Figure 5-29 shows external vbusvalid filter. Internal vbusvalid filter is implemented as two separate filters, one for vbusvalid and another for pipe3_PowerPresent. Refer to “Filtering BValid and VBUSValid Signals in Device and DRD-Device Modes” on page 167 also for DRD mode.

The VBUSVALID filter needs to be bypassed in Host only configurations because the inputs utmiotg_vbusvalid and pipe3_PowerPresent are set to high. Therefore, the bus_filter_bypass[3:0] signal should be tied to logical high value (4'b1111) if the DWC_USB3_EN_BUS_FILTERS hardware parameter is set to 1.

Figure 5-29 Filtering VBUSValid Signal


USB3.0 PHY
Vbusvalid from Synopsys PHY
Power-Present from 3rd Party PHY
vbusvalid filter
Internal filters to be bypassed when external filters used & vice-versa
utmiotg_vbusvalid and Pipe3PowerPresent should be tied to 1'b1 when host-only configuration is used
utmiotg_vbusvalid
pipe3_PowerPresent
core_not_otg#
pipe3_PowerPresent_uf
0
1
ulpiotg_vbusvalid
vbusvalid_uf
ulpi_clk_sel
Internal vbusvalid filter
Internal pipe3_Power Present filter
bus_filter_bypass[0]
bus_filter_bypass[1]
pipe3_PowerPresent_f
vbusvalid_f
DWC_usb3
# Refer programming details

Note

- For Synopsys PHY, if you are not using the filter inside the DWC_usb3 controller (DWC_USB3_EN_BUS_FILTERS = 0), use an external vbusvalid filter.
- For other Third-party PHYs, check if the PHY already has a de-bounce filter. If it does not have a filter, and you are not using the filter inside the DWC_usb3 controller (DWC_USB3_EN_BUS_FILTERS = 0), then use an external vbusvalid filter.

5.3.5 Filtering BValid and VBUSValid Signals in Device and DRD-Device Modes

The output of an external voltage comparator should be input to utmisrp_bvalid/pipe3_PowerPresent. For threshold value of this voltage comparator please refer to the description of utmisrp_bvalid in DWC SuperSpeed USB 3.0 Controller Databook.

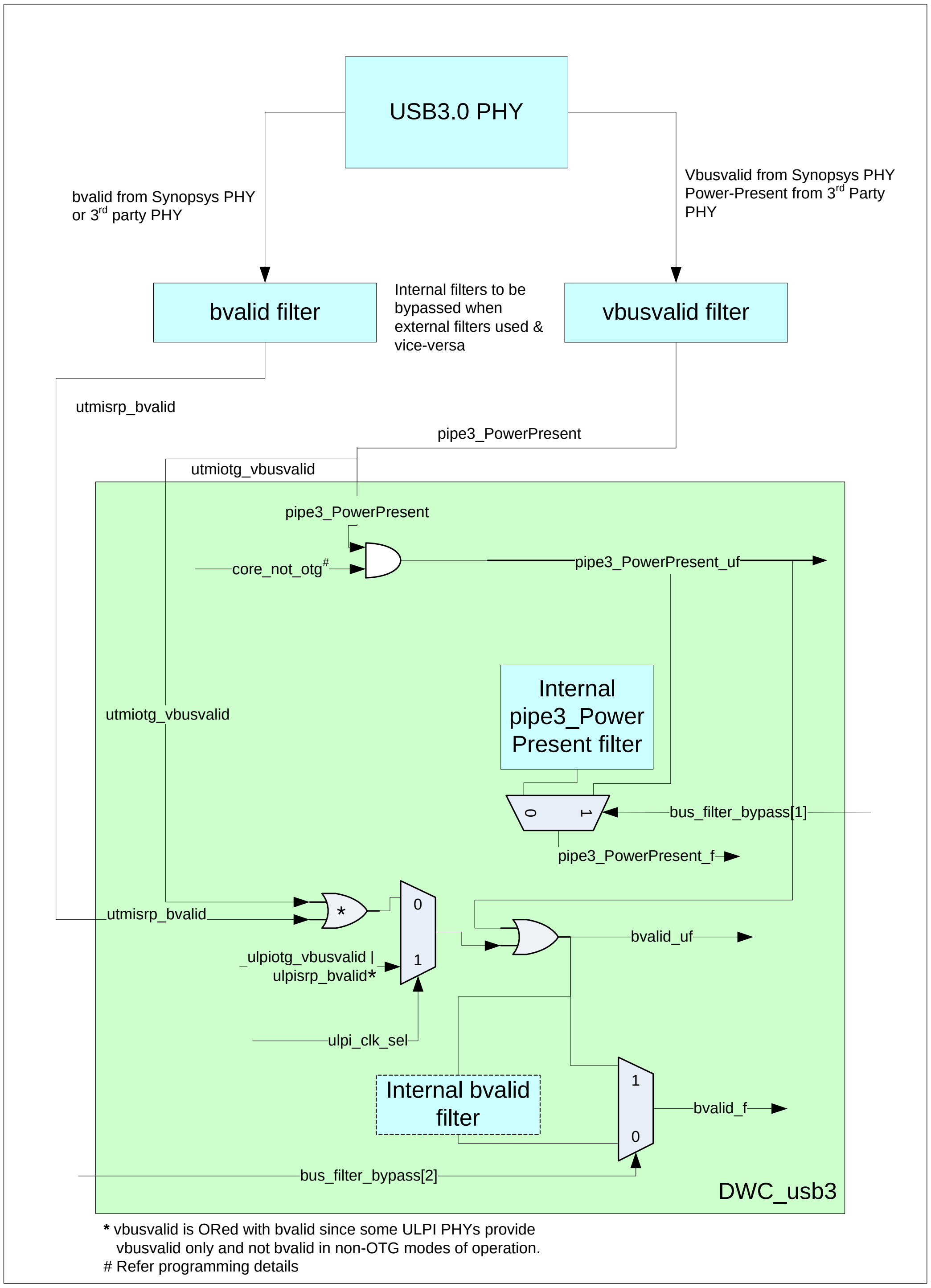
When using Synopsys PHY, use "bvalid" output from the PHY. This is valid in all modes. The vbusvalid and power_present signals from Synopsys PHY are valid only when internal V_{BUS} comparators are used. Check with the PHY vendor whether the Power-Present/bvalid outputs are valid in these modes.

If the coreConsultant parameter DWC_USB3_EN_BUS_FILTERS is chosen as 0 or bus_filter_bypass[2] is set to 1’b1, this should be a filtered signal. In this case, the power-present signal from the PHY needs to be filtered (~5-ms debounce filter) outside the controller before connecting to the utmisrp_bvalid and pipe3_PowerPresent signals. Once filtered, bvalid (session valid) signal should be connected to both the ports of the DWC_usb3 controller. If DWC_USB3_EN_BUS_FILTERS is 1 and bus_filter_bypass[2] is 1’b0, the external filter is not required.

The utmisrp_bvalid input to the controller should be driven when DWC_USB3_HSPHY_INTERFACE is UTMI . For ULPI mode of DWC_USB3_HSPHY_INTERFACE, the controller extracts bvalid signal (ulpi_bvalid) from ulpi_datain through RxCmd as per ULPI specifications. The pipe3_PowerPresent is ORed with either utmisrp_bvalid or ulpi_bvalid internally to generate the final power-present. Figure 5-30 shows external bvalid filter. Internal bvalid filter is implemented as two separate filters, one for bvalid and another for pipe3_PowerPresent.

For DRD mode, ideally pipe3_PowerPresent input should have the same threshold as utmisrp_bvalid for device operation. For host operation, the threshold should be utmiotg_vbusvalid. Use your discretion based on what is available from the PHY vendor and appropriately connect pipe3_PowerPresent. For example, one option is to set pipe3_PowerPresent to 1'b0 and use only utmisrp_bvalid with voltage comparator thresholds defined in chapter ‘Signals’ of the *DWC SuperSpeed USB 3.0 Controller Databook*. The next option is to always feed pipe3_PowerPresent with utmiotg_vbusvalid comparator output. Third option is to have these two comparators and an external MUX before feeding it to all the three ports but this requires knowledge of whether the controller is acting as device or host. So this option is not recommended.

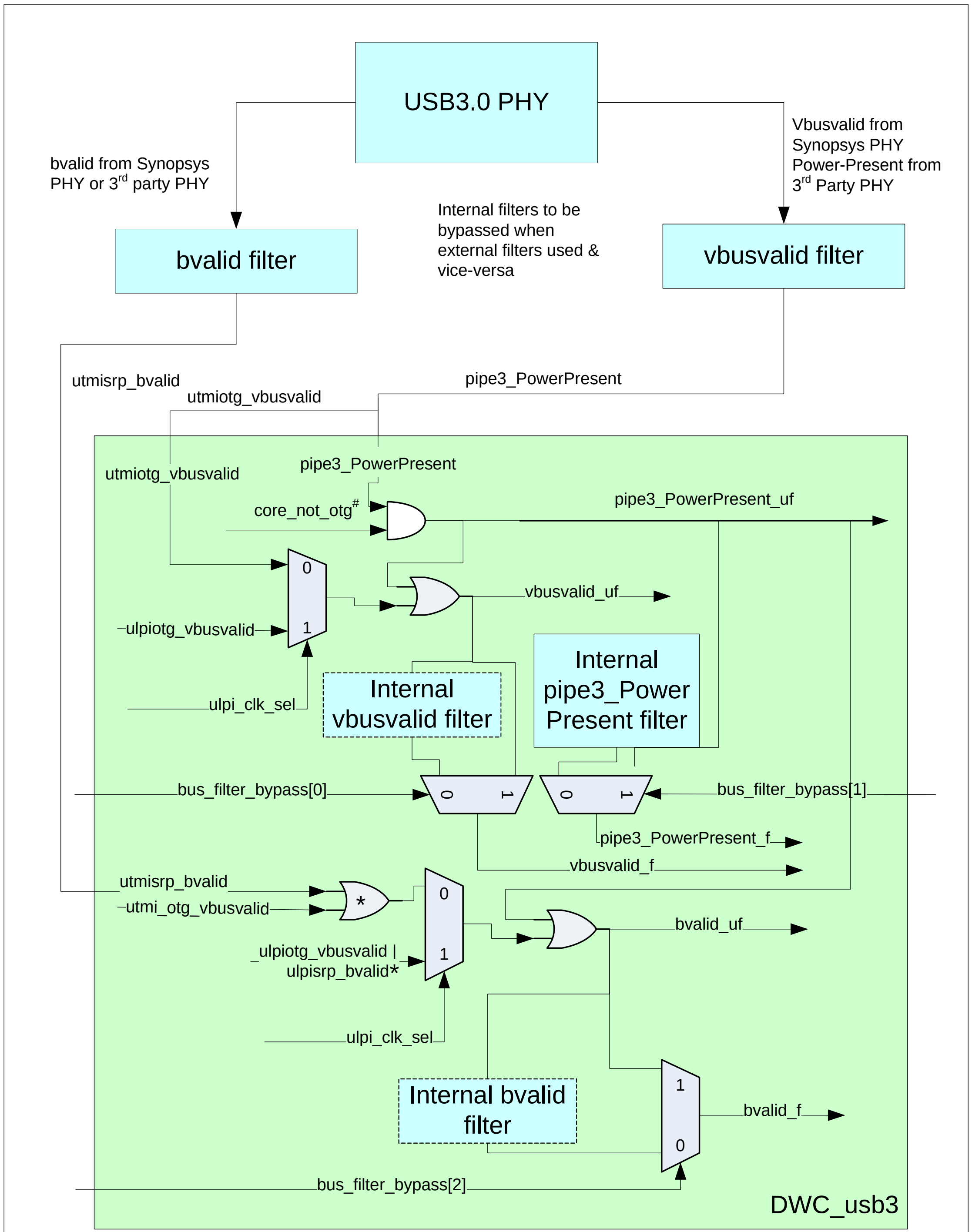
Figure 5-30 Filtering BValid and VBUSValid Signals in Device-Only Mode


USB3.0 PHY
bvalid from Synopsys PHY or 3rd party PHY
Vbusvalid from Synopsys PHY Power-Present from 3rd Party PHY
bvalid filter
Internal filters to be bypassed when external filters used & vice-versa
vbusvalid filter
utmisrp_bvalid
pipe3_PowerPresent
utmiotg_vbusvalid
pipe3_PowerPresent
core_not_otg#
pipe3_PowerPresent_uf
Internal pipe3_Power Present filter
utmiotg_vbusvalid
0
1
bus_filter_bypass[1]
pipe3_PowerPresent_f
utmisrp_bvalid
*
0
1
ulpiotg_vbusvalid | ulpisrp_bvalid*
bvalid_uf
ulpi_clk_sel
Internal bvalid filter
1
0
bvalid_f
bus_filter_bypass[2]
DWC_usb3
* vbusvalid is ORed with bvalid since some ULPI PHYs provide vbusvalid only and not bvalid in non-OTG modes of operation.
# Refer programming details

Note

- For Synopsys PHY, if you are not using the filter inside the DWC_usb3 controller (DWC_USB3_EN_BUS_FILTERS = 0), use an external bvalid filter.
- For other Third-party PHYs, check if the PHY already has a de-bounce filter. If it does not have a filter, and you are not using the filter inside the DWC_usb3 controller (DWC_USB3_EN_BUS_FILTERS = 0), then use an external bvalid filter.

Figure 5-31 Filtering BValid and VBUSValid Signals in DRD Mode



***** vbusvalid is ORed with bvalid since some ULPI PHYs provide vbusvalid only and not bvalid in non-OTG modes of operation.
# Refer programming details

5.3.6 Implementing Internal Filters

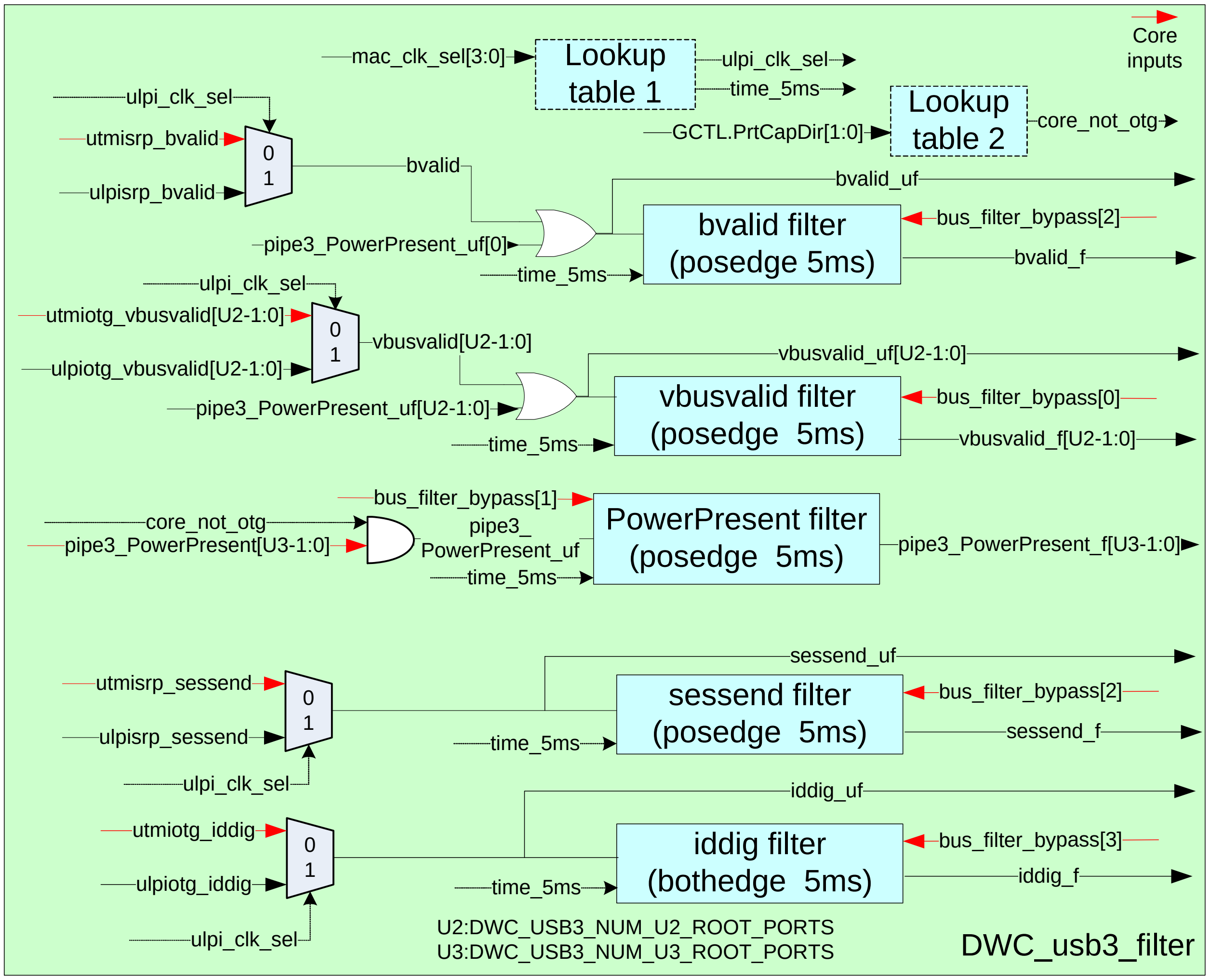
While configuring the DWC_usb3 controller, if you select DWC_USB3_EN_BUS_FILTERS as 1, filters are automatically implemented inside DWC_usb3 for the following signals:

- utmiotg_iddig
- utmiotg_vbusvalid
- utmisrp_bvalid
- utmisrp_sessend
- pipe3_PowerPresent

Figure 5-32 provides the illustration of overall filter implementation within the controller (DWC_usb3_filter module) including IDDIG and SESSEND.

- The filter debounce timers are 5 ms.
- All debounce filters except IDDIG filter posedge only.
- Signal names ending with *_uf are used for generation of suspend only.
- Signal names ending with *_f are used everywhere except suspend generation.
- All debounce filter outputs (*_f) are always synchronous to mac_clk even if bus_filter_bypass input is set and DWC_USB3_EN_BUS_FILTERS is chosen as 0.
- DWC_usb3_u3hub gets unfiltered PowerPresent only.
- The vbusvalid filter is bypassed only when bus_filter_bypass[0] is 1‘b1 irrespective of bus_filter_bypass[1].
- The bus_filter_bypass signals are not synchronized, assumed to be quasi-static, one time programmed at power-on.

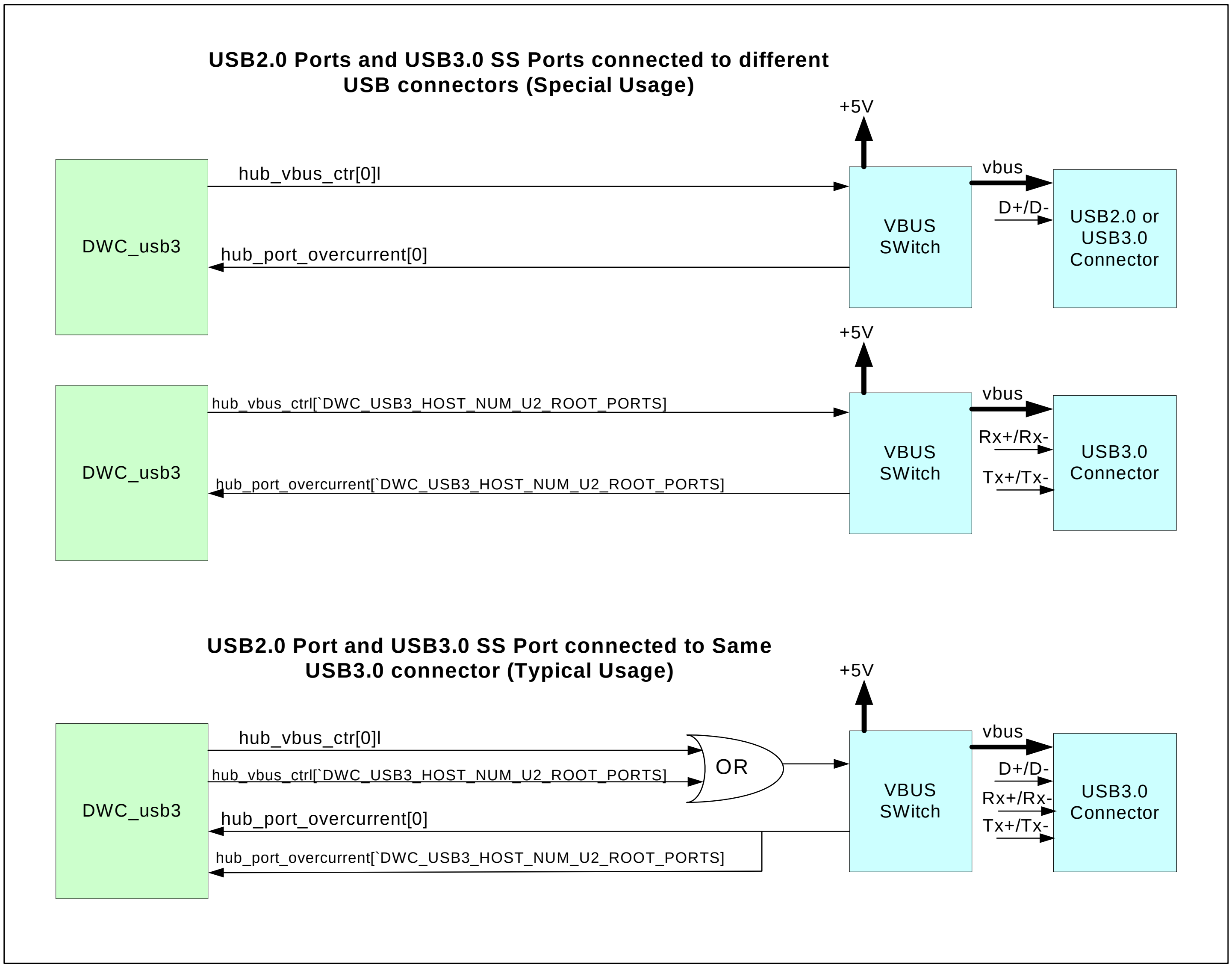
Figure 5-32 Implementing Internal Filters



5.3.7 Connecting Port VBUS and Over-Current Signals in Host and DRD Modes

Figure 5-33 shows how to handle Port VBUS-Control, Port Over-Current, and USB connector routing in Host/DRD mode. For host-only configurations, the filter should be bypassed by setting bus_filter_bypass[3:0] = 4'b1111 because the inputs utmiotg_vbusvalid and pipe3_PowerPresent are tied high.

Figure 5-33 Connecting Port VBUS and Over-Current Signals in Host/DRD Mode

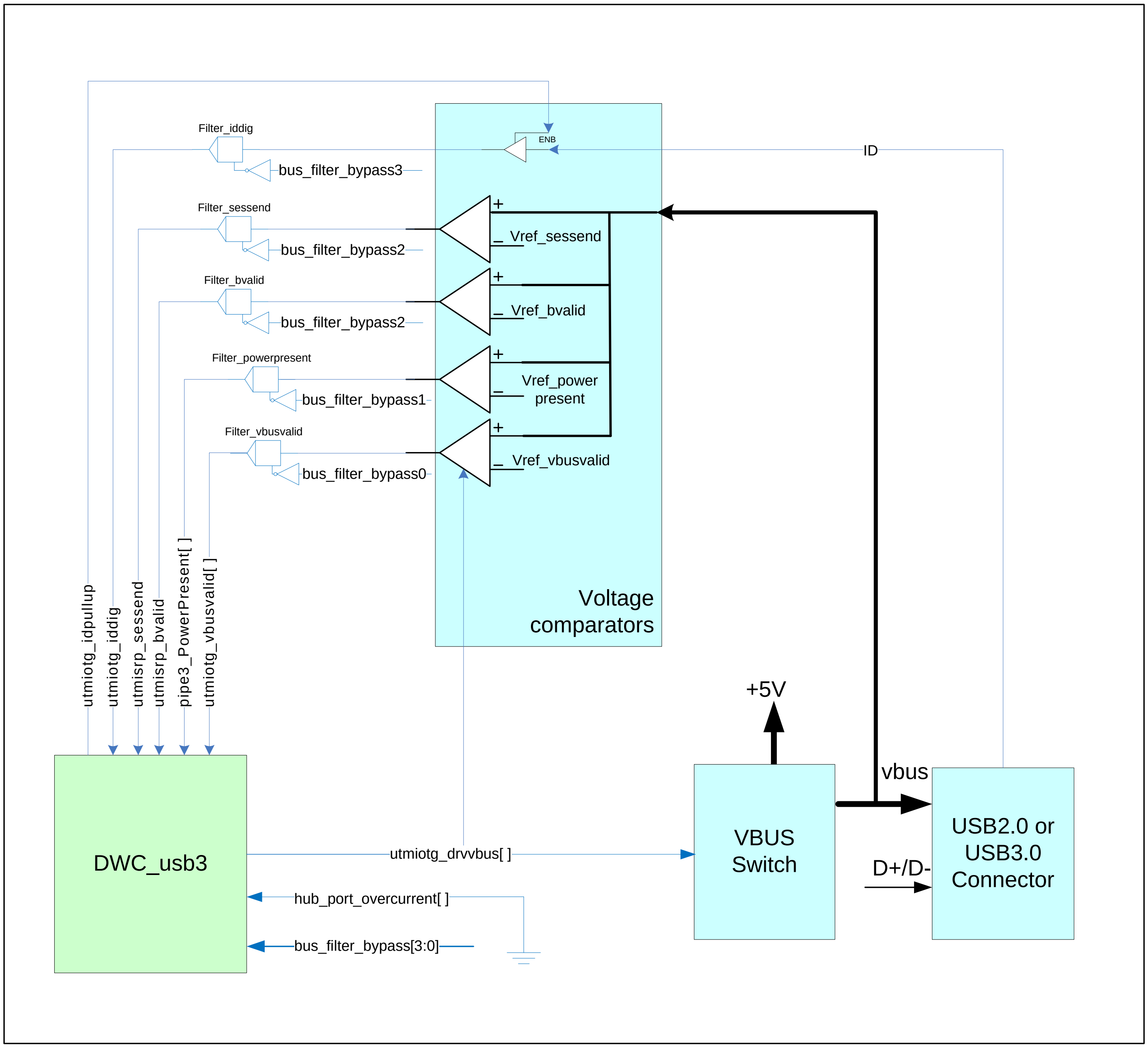

USB2.0 Ports and USB3.0 SS Ports connected to different USB connectors (Special Usage)
+5V
DWC_usb3
hub_vbus_ctr[0]l
hub_port_overcurrent[0]
VBUS SWitch
vbus
D+/D-
USB2.0 or USB3.0 Connector
+5V
DWC_usb3
hub_vbus_ctrl[`DWC_USB3_HOST_NUM_U2_ROOT_PORTS]
hub_port_overcurrent[`DWC_USB3_HOST_NUM_U2_ROOT_PORTS]
VBUS SWitch
vbus
Rx+/Rx-
Tx+/Tx-
USB3.0 Connector
USB2.0 Port and USB3.0 SS Port connected to Same USB3.0 connector (Typical Usage)
+5V
DWC_usb3
hub_vbus_ctr[0]l
hub_vbus_ctrl[`DWC_USB3_HOST_NUM_U2_ROOT_PORTS]
OR
hub_port_overcurrent[0]
hub_port_overcurrent[`DWC_USB3_HOST_NUM_U2_ROOT_PORTS]
VBUS SWitch
vbus
D+/D-
Rx+/Rx-
Tx+/Tx-
USB3.0 Connector

5.3.8 Filtering and Connecting Port VBUS and Over-Current Signals

Figure 5-34 on page 174 shows how to connect Port Over-Current, various VBUS signals and filters.

- The VBUS Voltage comparators and the ID comparators are typically a part of the PHY.
- For reference voltage (Vref) values, refer to the UTMI+ specification.
- If all the filters are implemented external to the controller as shown in Figure 5-34, then the coreConsultant parameter DWC_USB3_EN_BUS_FILTERS is set to 0.
- If any or all of the filters are implemented internal to the controller, then DWC_USB3_EN_BUS_FILTERS is set to 1. The corresponding external filters are not required or can be bypassed. When external filters are bypassed, the internal filters should not be bypassed. You can choose a filter to be internal or external to the controller, by using the bus_filter_bypass[3:0] control signal.
- To implement these filters for all the ports (in particular, vbusvalid[ ] and pipe3_PowerPresent[ ]), enable DWC_USB3_EN_BUS_FILTERS. To control the VBUS switch for all ports except port0, use hub_vbus_ctrl[ ] instead of utmiotg_drvvbus. For port0, use either hub_vbus_ctrl[0] or utmiotg_drvvbus. Further, generate hub_port_overcurrent appropriately. In this case, the bus_filter_bypass0 signal is common to all the vbusvalid[ ] signals and the bus_filter_bypass1 signal is common to all the pipe3_PowerPresent[ ] signals.

Figure 5-34 Filtering and Connecting Port VBUS and Over-Current Signals



5.3.9 Usage of IDDIG and VBUS-Related Sideband Signals in UTMI and ULPI Modes

Table 5-3 gives a summary of the signals usage in UTMI mode of operation.

Table 5-3 Usage of IDDIG and VBUS-Related Signals in UTMI Interface

UTMI Signals	Operational Mode	
	Host	**Device**
Inputs (For usage of vbusvalid and PowerPresent signals within the controller, refer to Figure 5-32)		
utmiotg_vbusvalid	Used as power present	Not used, set to 1'b0
utmiotg_avalid	Not used	Not used
utmisrp_bvalid	Not used	Used as power present, OR'ed with vbusvalid
utmisrp_sessend	Not used	Not used
hub_port_overcurrent	Used	Not used
pipe3_PowerPresent	Used as power-present	Used as power-present
Outputs		
utmiotg_drvvbus	Not used	Not used
hub_vbus_ctrl	Used as drvvbus	Not used
utmiotg_dppulldown	Used and driven high	Used and driven low
utmiotg_dmpulldown	Used and driven high	Used and driven low

For ULPI mode, the utmi* signals are not present and are internally generated during ULPI RxCMD. In ULPI mode, the hub_port_overcurrent and pipe3_PowerPresent signals are used in conjunction with internally derived signals from ULPI RxCMD.

When the DWC_usb3 is configured for Host-only or Device-only modes, certain UTMI+ (DRD related) signals are not a part of the controller I/O. They are usually static signals that need to be tied to some fixed values outside the controller. The UTMI+ PHY input signals for utmiotg_dppulldown and utmiotg_dmpulldown need to driven high (1'b1) for host mode and driven low (1'b0) for device mode.

5.4 Integrating the Dual Power Rail (Hibernation) Feature

When the USB 3.0 controller is configured with the hibernation feature enabled (DWC_USB3_PWROPT=2), dual power rails are supported. In this configuration, the main USB 3.0 controller may be powered by Vcc while the two PMUs (which are instantiated in the DWC_usb3 hierarchy) may be powered by a different power supply (Vaux). Special considerations need to be made when integrating the controller in this fashion. The naming convention for the signals crossing between the Vcc and Vaux power domains use a core_i and core_o prefix:

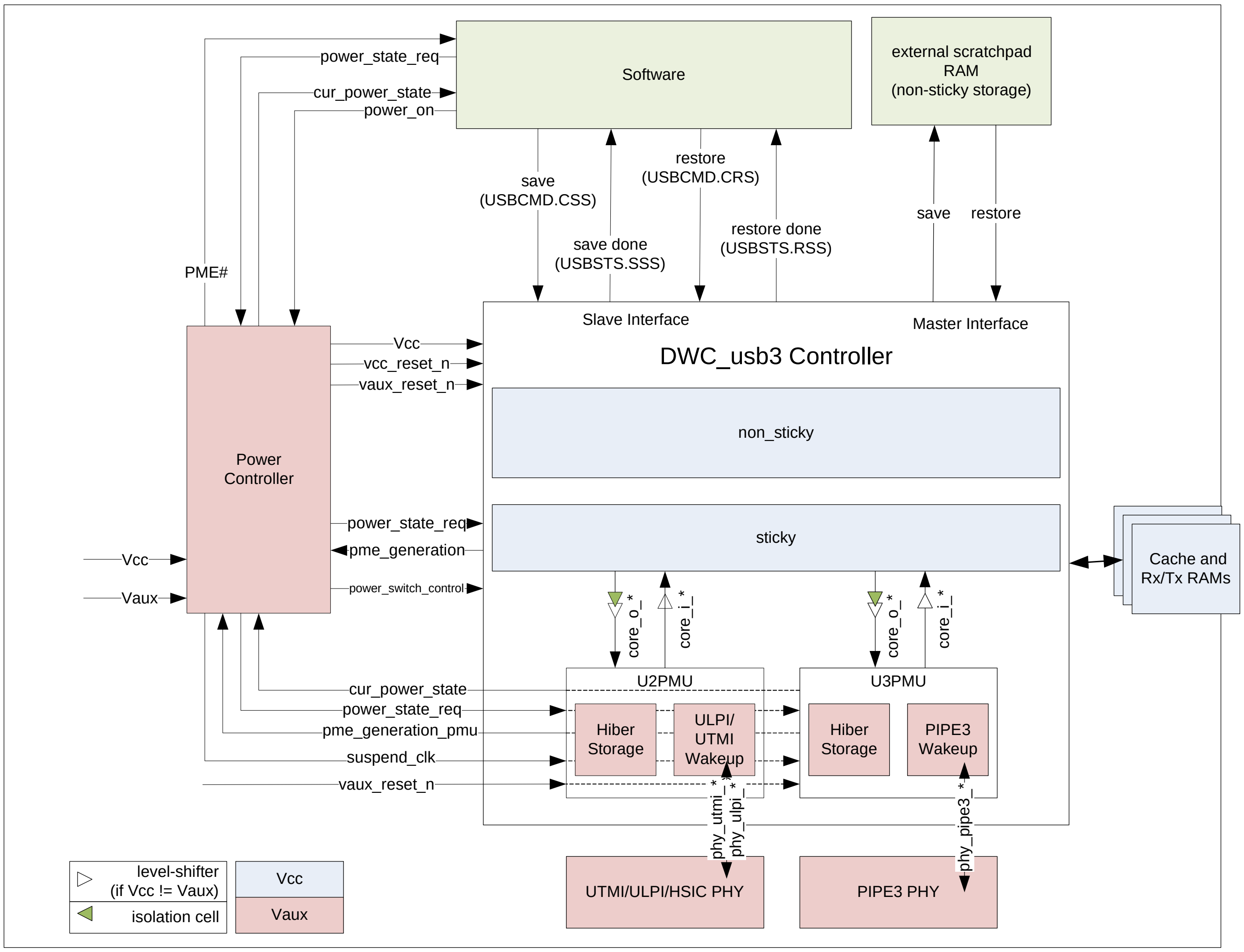
- core_i – denotes signals starting from Vaux power domain and ending in Vcc power domain
- core_o – denotes signals starting from Vcc power domain and ending in Vaux power domain

This section also discusses how the Power Management Event (PME) signal, PCI power state request, and resets should be connected. For details about the hibernation feature before referring to this section, see "Hibernation" chapter of the *DWC SuperSpeed USB 3.0 Controller Databook*.

5.4.1 Instantiating PMUs Inside and Outside DWC_usb3

As illustrated in Figure 5-35, the PMUs (U2PMU and U3PMU) are placed in the DWC_usb3 hierarchy.

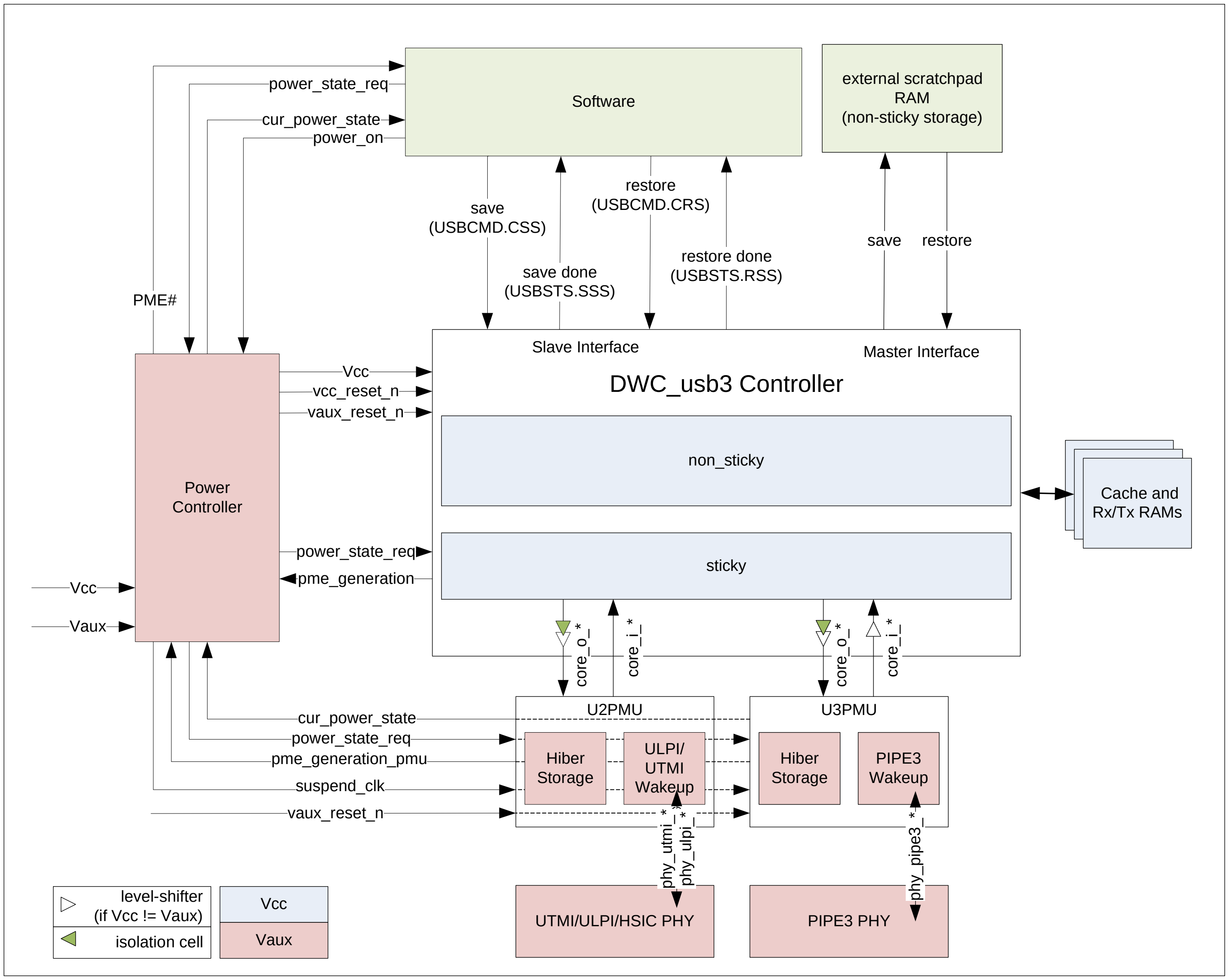
Figure 5-35 PMU Instantiated Inside DWC_usb3 Hierarchy



The following points needs to be noted in this context.

- The signals connecting between the PMUs and the controller have ”core_i_” and “core_o_” prefixes, indicating signal directions from the PMUs to the USB 3.0 controller and from the USB 3.0 controller to PMUs, respectively. These signals denote the power domain crossing.
- The power_switch_control signal, when asserted, indicates the power can be safely turned off for the logic operating on Vcc power domain.
- Because the PMUs operate on a Vaux power rail and the USB 3.0 controller operates on a Vcc power rail, it is necessary to place isolation cells and a level shifter on these signals. Isolation cells are required only on core_o_* signals. However, level shifters are required for core_o_* and core_i_* signals.
- The PMU I/O signals towards the PHY have a “phy_” prefix. For example, phy_utmi_* signals indicate connection between the PMU and the UTMI PHY; phy_pipe3_* signals indicate connection between the PMU and the SS PHY, and so on.
- In case the PMUs need to be instantiated within the DWC_usb3 hierarchy, make note of the following aspects.
    - In the *workspace*/src/DWC_usb3.v file, the signals and the modules (DWC_usb3_u2pmu and DWC_usb3_u3pmu) that need to be present in the Vaux domain are placed in the DWC_USB3_HIBERNATION_EN macro.
    - The signals in the *workspace*/src/DWC_usb3.v file that need to be in Vaux domain are phy_* and their corresponding DWC_usb3 I/O signals (for example, if the signal is phy_utmi_clk, its corresponding DWC_usb3 I/O signal is utmi_clk).
    - Place Isolation cells and level shifters on the core_i_* and core_o_* signals.
- In case the PMUs need to be instantiated outside the DWC_usb3 hierarchy as indicated in Figure 5-36 on page 178, make note of the following aspects.
    - In *workspace*/src/DWC_usb3.v file, the signals and the modules (DWC_usb3_u2pmu and DWC_usb3_u3pmu) that need to be present in Vaux domain are placed in DWC_USB3_HIBERNATION_EN macro.
    - The signals in *workspace*/src/DWC_usb3.v file that need to be in Vaux domain are phy_* and their corresponding DWC_usb3 I/O signals (for example, if the signal is phy_utmi_clk, its corresponding DWC_usb3 I/O signal is utmi_clk).
    - After placing the PMUs along with their signals in the Vaux power domain, connect the core_i_* and core_o_* signals to the remaining logic in the *workspace*/src/DWC_usb3.v.
    - Place Isolation cells and level shifters on the core_i_* and core_o_* signals.

Figure 5-36 PMU Instantiated Outside DWC_usb3 Hierarchy



5.4.2 Integrating DWC_usb3 with PCI-e Controllers

The focus here is integration with the Synopsys PCI Express controller, but the same concepts may be applied to other PCI Express controllers and other system buses. In the descriptions, specific references to Synopsys PCI Express controller integration are given by the tag **[PCIe].**

The signal descriptions here are a supplement to the descriptions in *DWC SuperSpeed USB 3.0 Controller Databook* and apply only when the controller is configured with the hibernation feature enabled (DWC_USB3_EN_PWROPT=2).

Table 5-4 and Table 5-5 summarizes each signal related to the hibernation feature. Table 5-4 describes the signals to and from the USB3 controller.

Table 5-4 Integration Notes for USB3 Controller Signals for Hibernation Support

Signal	I/O	Integration Notes
vcc_reset_n	I	This signal has two behaviors depending on whether the hibernation feature is enabled or not. For more details, see Table 5-6 on page 183.
suspend_clk	I	Used by the controller when the 2.0 or 3.0 PHYs are suspended to provide a mac_clk to the controller's internal logic, and also clocks the interface between the controller and the PMUs which is active during D3 entry. For details of when this clock may be gated off, see the 'Hibernation' chapter of the *DWC SuperSpeed USB 3.0 Controller Databook*. **[PCIe]** Because the PCIe REFCLK stops oscillating while the system is in S3, this clock should be not be generated from REFCLK.
pm_power_state_request[1:0]	I	This is the D* power state request from power management software which initiates the internal sticky state save & restore functionality after the programming model requirements have been satisfied (for details, see the 'Hibernation' chapter of the *DWC SuperSpeed USB 3.0 Controller Databook*). When hibernation is disabled (GCTL.GblHibernationEn=0), this interface may still be used, but sticky save & restore will not be performed. The PMUs will still indicate the state transition has completed on the current_power_state_uXpmu signals. In a non-hibernation configuration (DWC_USB3_EN_PWROPT != 2), this signal is unused. **[PCIe]** This signal is equivalent to the PMCSR.PowerState register, given by the 'pm_dstate' PCIe controller output.
current_power_state[1:0]	O	Unused. The PMUs indicate the current power state.

Table 5-4 Integration Notes for USB3 Controller Signals for Hibernation Support (Continued)

Signal	I/O	Integration Notes
pme_generation	O	Indicates the controller is requesting a wakeup and Power Management Event (PME) indication to the system. Unused in: ■ Device mode ■ Host mode when hibernation is enabled (GCTL.GblHibernationEn=1) In the above 2 conditions, the wakeup and PME request is generated from the PMUs (pmu_generation_uXpmu). In host mode when hibernation is disabled (GCTL.GblHibernationEn=0), this output should be connected to the system wakeup and PME inputs. **[PCIe]** For power-up, PCI Express supports in-band wakeup (Beacon) and out-of-band wakeup (WAKE# pin). For PME, PCI Express endpoints transmit a PME Message upstream. This signal would be connected to the 'outband_pwrup_cmd' input for generating an out-of-band wakeup. For the PME, a pulse is generated from this level signal and connected to the 'apps_pm_xmt_pme' input.
pme_en	I	Because pme_generation is unused, this signal is also unused in: ■ Device mode ■ Host mode when hibernation is enabled (GCTL.GblHibernationEn=1) In host mode when hibernation is disabled (GCTL.GblHibernationEn=0), this input enables the controller to generate the pme_generation signal. **[PCIe]** This signal is the same as the PMCSR.PME_En field ('pm_pme_en' output) which indicates whether the root complex is allowing the controller to wakeup the system. Some PCIe controllers manage PME masking internally (they do not require the USB3 controller to mask off its PME). For those types of controllers, this input may be tied to '1', causing the controller to assert the pme_generation signal anytime it detects a wakeup condition.
usb2phy_reset	O	This signal resets the USB 2.0 PHY when both vcc_reset_n and vaux_reset_n are asserted, or when GUSB2PHYCFG.PHYSoftRst is set to '1'
pipe3_reset_n	O	This signal resets the USB3.0 PHY when both vcc_reset_n and vaux_reset_n asserted, or when GUSB3PIPECTL.PHYSoftRst is set to '1'

Table 5-5 describes the signals to and from the PMUs.

Table 5-5 Integration Notes for PMU Signals for Hibernation Support

Signal	I/O	Notes
vaux_reset_n	I	This is the reset for the logic powered by Vaux, which are the PMUs and the PHYs. It must be asserted when Vaux is removed or during the first system power-up. This is also known as a "sticky reset," board reset, or power-on reset. When asserted along with vcc_reset_n, the PMU state is cleared, the controller resets the PHYs, and the PMU current_power_state_uXpmu outputs switch to '0'
suspend_clk	I	Used by the PMUs to drive the glitch filters and perform receiver detection in host mode, and also clocks the interface between the controller and the PMUs. **[PCIe]** Because the PCIe REFCLK stops oscillating while the system is in S3, this clock should be not be generated from REFCLK.
pm_power_state_request[1:0]	I	This is the D* power state request from power management software which initiates the internal sticky state save & restore functionality after the programming model requirements have been satisfied (For details, see the 'Hibernation' chapter of the *DWC SuperSpeed USB 3.0 Controller Databook*). It has the same value as the pm_power_state_request input to the controller. **[PCIe]** This signal is equivalent to the PMCSR.PowerState register, given by the 'pm_dstate' PCIe controller output.
current_power_state_u2pmu[1:0] current_power_state_u3pmu[1:0]	O	When pm_power_state_request transitions to D3 and both PMUs indicate that they are in D3 via these signals, it means the controller is prepared to lose Vcc and vcc_reset_n may be asserted. When pm_power_state_request transitions to D0 and both PMUs indicate that they are in D0, it means the controller is restored and controlling the PHYs, and software may access the controller's registers. When hibernation is disabled (GCTL.GblHibernationEn=0), these signals are still valid, but they will indicate that the power state transition has occurred without actually performing a sticky save/restore. **[PCIe]** When the PMUs are indicating they are both in D3, the PCIe controller input 'app_ready_entr_l23' input should be asserted immediately to indicate the link is ready to enter L2/L3 Ready, which is the last step before the root complex removes Vcc. In addition, the current power state of the controller to be reflected in PMCSR.PowerState comes from the combination of these two signals. When both PMUs transition to D3, the controller is considered to be in D3. When both PMUs transition to D0, the controller is considered to be in D0.

Table 5-5 Integration Notes for PMU Signals for Hibernation Support (Continued)

Signal	I/O	Notes
pme_generation_u2pmu pme_generation_u3pmu	O	Indicates the PMU is requesting a wakeup and PME. When hibernation is disabled (GCTL.GblHibernationEn=0), these signals are unused. In host mode when hibernation is disabled, wakeup is signalled via the controller's 'pme_generation' output. In device mode when hibernation is disabled, the device driver will not stop the controller so none of the pme_generation signals will indicate wakeup. **[PCIe]** The OR of these two signals may be connected to the in-band or out-of-band powerup signal on the PCIe controller. For the Synopsys PCIe controller, the out-of-band powerup signal is called 'outband_pwrup_cmd'. In addition, this level signal is converted into a pulse and connected to the PME generation input of the PCIe controller 'apps_pm_xmt_pme'.
debug_u2pmu[DWC_USB3_NUM_U2_ROOT_PORTS*DWC_USB3_DEBUG_U2WAKEUP_W] debug_u3pmu[0]	O	Each PMU has a debug output bus. The debug_u2pmu[DWC_USB3_NUM_U2_ROOT_PORTS *DWC_US B3_DEBUG_U2WAKEUP_W] and debug_u3pmu[0] represent if hibernation is disabled for U2PMU and U3PMU, respectively. The same information is available to software which sets the GCTL.GblHibernationEn field, but is replicated here for convenience. This signal is only valid when the PMU's current_power_state_uXpmu output is equal to '3'. When hibernation is disabled, vcc_reset_n must not be asserted in D3, so the system may use these signals to determine whether to assert vcc_reset_n in D3 or not. **Note**: DWC_USB3_DEBUG_U2WAKEUP_W = 54
connect_state_u2pmu connect_state_u3pmu	O	Indicates whether the PMU is maintaining a connection during hibernation. This information may be used to decide whether suspend_clk may be gated or not. For more details, see the 'Hibernation' chapter of the *DWC SuperSpeed USB 3.0 Controller Databook*.
phy_usb2phy_reset phy_pipe3_reset_n	O	These PMU-controlled PHY resets will not be asserted when vcc_reset_n is asserted so that the PHYs are not reset during hibernation. They are only asserted when vaux_reset_n is asserted along with vcc_reset_n, or when GUSB2PHYCFG.PHYSoftRst/GUSB3PIPECTL.PHYSoftRst is set to '1'.
power_switch_control	I	The power controller asserts this signal to turn on the isolation between the controller and the PMU. During hibernation entry, the power controller asserts this signal before switching off the Vcc power. During hibernation exit, the power controller de-asserts this signal after switching on the Vcc power.

Note DWC_usb3_glitch_free_mux is instantiated only in the PMUs. It filters glitches in the phy_PowerDown(pipe), utmi_suspend , utmi_l1_suspend, and utmi_suspend_com_n asynchronous signals.

Map this module to the standard glitch-free MUX available in the library or apply set_dont_touch command on this module during synthesis.

The combination of the vcc_reset_n signal which is an input to the controller, and the vaux_reset_n signal which is an input to the PMUs describes two types of resets as shown in Table 5-6. .

Table 5-6 Combination of vcc_reset_n and vaux_reset_n

vcc_reset_n	vaux_reset_n	Description
0	1	**Non-sticky reset:** This clears the controller state and non-sticky registers. It must be asserted when both PMUs are in D3, and deasserted when requesting the controller to enter D0. Note that when hibernation is disabled (GCTL.GblHibernationEn=0), this signal must not be asserted when the PMUs are in D3. **[PCIe]** Similar to the PERST# PCIe signal which is asserted when the system is in S3 indicating that the Vcc power rail is out of spec.
0	0	**Sticky reset:** This clears the controller state, PMU state, and sticky and non-sticky registers, and resets the PHYs. It must be asserted when Vaux is removed or during the first system power-up.

5.4.3 Isolation Cells

Isolation cells must be implemented between the logic in the Vcc domain and the Vaux domain. When both PMUs enter D3, the isolation cells may be enabled. When the system requests D0 entry, the isolation cells may be disabled.

5.4.4 Code Example

The following code snippets show one way of integrating the PMUs with the Synopsys PCIe controller.

```
  //------------------------------
  // pm_power_state_request
  //------------------------------
  reg [1:0] pm_power_state_request;
  always @(posedge aux_clk or negedge reset_switch_n) begin
    if (~reset_switch_n)              pm_power_state_request <= 2'h0;
    else if(pm_dstate[2:0] == 3'h3)   pm_power_state_request <= 2'h3;
    else if(pm_dstate[2:0] == 3'h0)   pm_power_state_request <= 2'h0;
  end

  reg       pm_hibernation_en_r;
  always @(posedge suspend_clk or negedge reset_switch_n) begin
    if (~reset_switch_n) pm_hibernation_en_r <= 1'b0;
    else if (pm_hibernation_en_r == 0) begin
      if ((pm_power_state_request == 2'h3) && (current_power_state_u3pmu[1:0] == 2'h3)
&& (current_power_state_u2pmu[1:0] == 2'h3))
        pm_hibernation_en_r <= 1'b1;
    end
    else if (pm_hibernation_en_r == 1) begin
      if(pm_power_state_request == 2'h0)
        pm_hibernation_en_r <= 1'b0;
    end
  end
  //------------------------------
  // vcc_reset_n
  //------------------------------
  assign u3pmu_disabled = debug_u3pmu[0];
  assign u2pmu_disabled = debug_u2pmu[DWC_USB3_NUM_U2_ROOT_PORTS *
DWC_USB3_DEBUG_U2WAKEUP_W];
  assign vcc_reset_n =  (~reset_switch_n || (!u2pmu_disabled & !u3pmu_disabled &
pm_hibernation_en_r)) ? 1'b0 : 1'b1;
  //------------------------------
  // Isolation Cells Enable
  //------------------------------
  assign isolationcell_en = (!u2pmu_disabled & !u3pmu_disabled & pm_hibernation_en_r);

  //------------------------------------------
  // Power-up request, PME request, and D3 entry
  //------------------------------------------
  reg outband_pwrup_cmd_r;
  reg pme_generation_r;
  reg pme_pulse;

  always @(posedge aux_clk or negedge reset_switch_n) begin
    if(!reset_switch_n) begin
      pme_generation_r <= 1'b0;
      outband_pwrup_cmd_r <= 1'b0;
      pme_pulse <= 1'b0;
    end else begin
      pme_generation_r <= pme_generation;
      outband_pwrup_cmd_r <= outband_pwrup_cmd;
      if ((~outband_pwrup_cmd_r & outband_pwrup_cmd) | (~pme_generation_r &
pme_generation))
        pme_pulse <= 1'b1;
      else
        pme_pulse <= 1'b0;
    end
  end

  assign wake_n =   ~wake;

  always @(posedge suspend_clk or negedge reset_switch_n) begin
    if(!reset_switch_n) begin
      rd2a_pme_generation_pmu_r <= 0;
      rd2a_pm_pmu_d3_r <= 0;
    end
    else begin
      rd2a_pme_generation_pmu_r <= pme_generation_u3pmu | pme_generation_u2pmu;
      rd2a_pm_pmu_d3_r            <= ((current_power_state_u3pmu == 2'b11) &&
(current_power_state_u2pmu == 2'b11));
    end
  end

  assign app_ready_entr_l23 = srd2a_pm_pmu_d3_r; // synchronized version of
rd2a_pm_pmu_d3_r
```

```
  assign outband_pwrup_cmd = srd2a_pme_generation_pmu_r; // synchronized version of
rd2a_pme_generation_pmu_r

    wire  pcie_pwr_rst_n          = reset_switch_n & perst_deasserted_n;
    wire  pcie_core_rst_n         = reset_switch_n & perst_deasserted_n & !perst;
    wire  pcie_sticky_rst_n       = reset_switch_n & perst_deasserted_n; // This may be
incorrect, because sticky reset should not be asserted on PERST
    wire  pcie_non_sticky_rst_n   = reset_switch_n & perst_deasserted_n &
training_rst_n & link_down_rst_n & !perst ;

// CORE:
    // Inputs
    .vcc_reset_n    (vcc_reset_n),

// PMU:
    // Inputs
    .vaux_reset_n   (reset_switch_n),

// PCIE CONTROLLER:
     // Outputs
    .pm_dstate      (pm_dstate[(3*NF)-1:0]),
    .pm_pme_en      (pm_pme_en[NF-1:0]),
    .wake           (wake),
    // Inputs
    .pwr_rst_n      (pcie_pwr_rst_n),
    .sticky_rst_n   (pcie_sticky_rst_n),
    .non_sticky_rst_n   (pcie_non_sticky_rst_n),
    .core_rst_n      (pcie_core_rst_n),

    .apps_pm_xmt_pme   (pme_pulse),
    .app_ready_entr_l23   (app_ready_entr_l23),
    .outband_pwrup_cmd   (outband_pwrup_cmd));
```

5.4.5 Example of Hibernation Entry and Exit

Figure 5-37 illustrates a host entering hibernation, including the appropriate PCIe controller signals mentioned in the previous sections.

Figure 5-37 Host Hibernation Entry

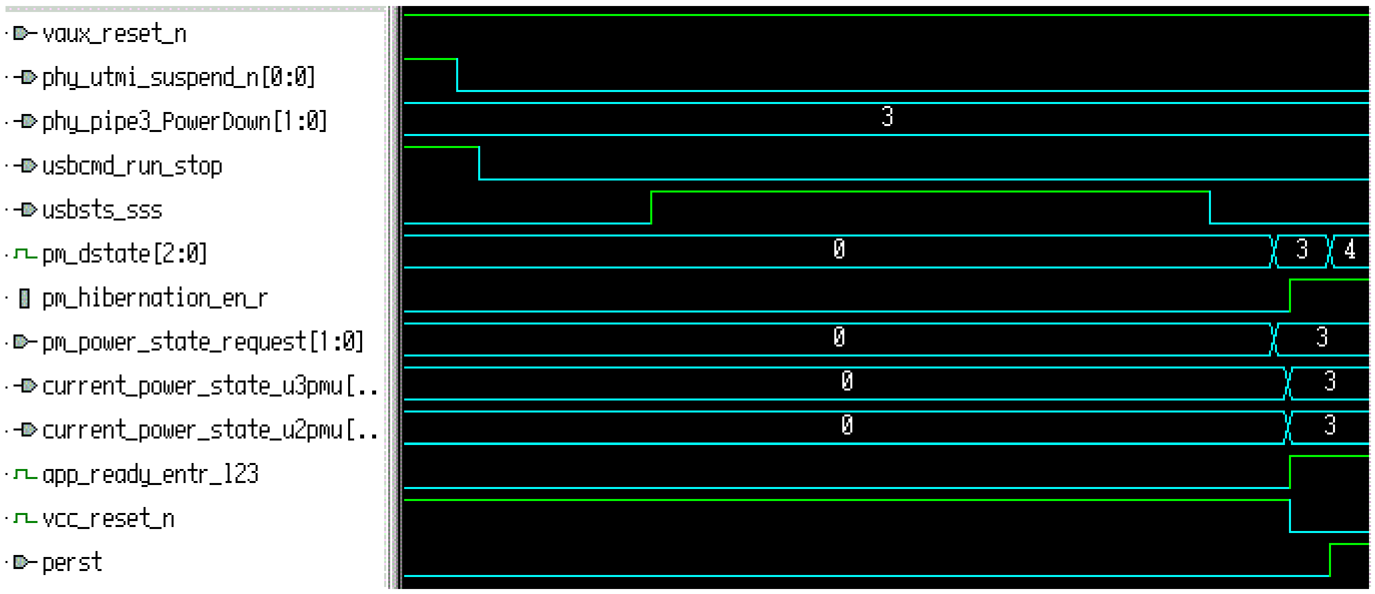


Figure 5-38 on page 186 illustrates a host exiting hibernation.

Figure 5-38 Host Hibernation Exit

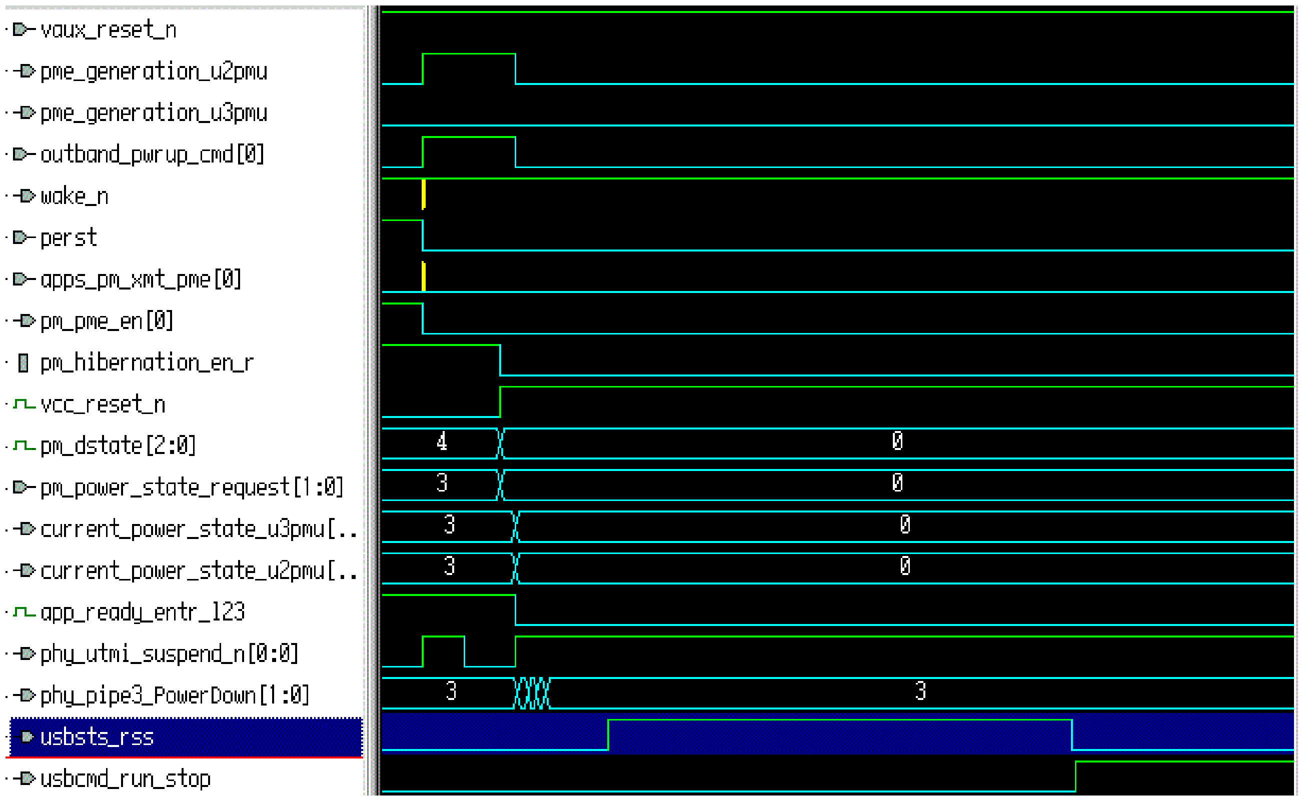
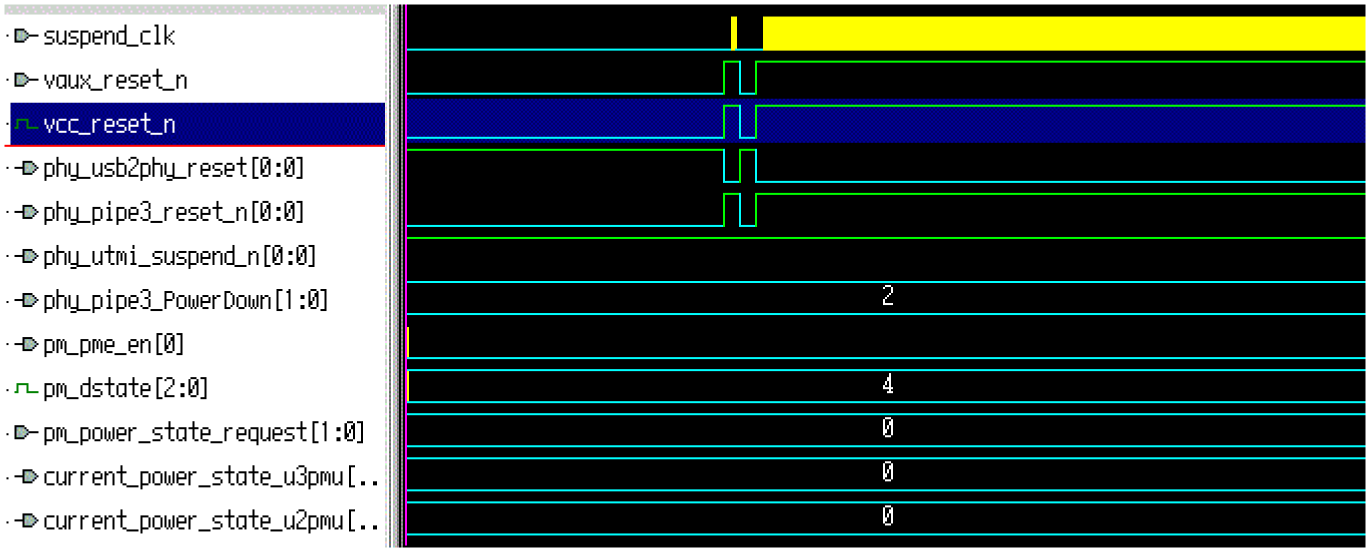


Figure 5-39 illustrates power-on reset.

Figure 5-39 Power-on Reset



5.5 Little-Endian and Big-Endian

The DWC_usb3 controller operates in little-endian mode by default. Depending upon the data width, data is implemented as follows (in little-endian):

- Data[31:0] = {B3, B2, B1, B0}
- Data[63:0] = {B7, B6, B5, B4, B3, B2, B1, B0}
- Data[127:0] = {B15, B14, B13, B12, B11, B10, B9, B8, B7, B6, B5, B4, B3, B2, B1, B0}

Alternatively, the controller (in AHB) supports independently-controlled big-endian conversion. On the Master interface, endianness can be independently selected for the descriptor and data fetches. On the Slave interface, endianness is a top-level strap selection. When selected, the controller uses the Address Invariant Big-Endian conversion function.

The Native interface configuration does not support big-endian in any form.

The AXI interface configuration does not require an endian transform to connect to the little-endian DWC_usb3 controller because the AXI interface is byte-invariant. The AXI specification states:

> "Most little-endian components can connect directly to a byte-invariant interface. Components that support only big-endian transfers require a conversion function for byte-invariant operation."

Therefore, big-endian components connecting to an AXI interface require the appropriate conversion function to map bytes to the byte-invariant (little-endian ordered) AXI data paths, but the DWC_usb3 controller does not require a conversion function.

5.5.1 Big-Endian – Address Invariance

When big-endian is enabled, the DWC_usb3 controller will utilize an Address Invariant (or Byte Invariant) Big-Endian scheme for the entire width of the data word. This is defined as follows:

- Data[31:0] = {B0, B1, B2, B3}
- Data[63:0] = {B0, B1, B2, B3, B4, B5, B6, B7}
- Data[127:0] = {B0, B1, B2, B3, B4, B5, B6, B7, B8, B9, B10, B11, B12, B13, B14, B15}

Note that the entire width of the data word undergoes a byte swap, regardless of the value of AR/AWSIZE (AXI) or HSIZE (AHB).

5.5.2 Big-Endian – Data Invariance (BE32, BE16)

Data Invariant Big-Endian, BE32 (word invariant), and BE16 (half-word invariant) modes are not supported in any configuration.

5.5.3 Descriptor Vs. Data Endian Mode

Endianness for data and descriptors can be selected independently by programming the CSR. All read and write accesses that are not for packet data, are considered to be descriptor accesses. Descriptor accesses include TRB, device context, EP context, stream context, and scratchpad read and write accesses.

To select big-endian mode for data, see GSBUSCFG0[11] in "Registers" chapter of the *DWC SuperSpeed USB 3.0 Controller Programming Guide*. To select big-endian mode for descriptors, see GSBUSCFG0[12] in "Registers" chapter of the *DWC SuperSpeed USB 3.0 Controller Programming Guide*.

Note In some applications, USB Setup packets are utilized to transfer TRB descriptors as data. In this case, the DWC_usb3 controller is not aware of what is contained in the Setup packet's payload. It treats it as data and utilizes the data endian mode and not the descriptor endian mode. It is up to the application to perform the endian transformation to the descriptor.

5.6 Hardware Debug Features for Initial Chip Operation

DWC_usb3 has built-in debug diagnostics to help debug chips during initial operation. In addition, the software has visibility into all the FIFO and descriptor RAMs, FIFO status, state information for some of the critical state machines, link packet statistics, and so on.

During chip bring-up, the following order of testing is recommended:

- Do write and read tests of the registers through the SoC bus Slave interface.
- Do write and read tests of the Rx, Tx, and Descriptor RAMs through the bus Slave interface.
- Validate the DMA engine without any USB traffic using the upper-layer diagnostic feature (only supported in device mode):
  - Set up Tx and Rx transfers for EP0 through Start Transfer for IN EP0 and OUT EP0.
  - Enable the diagnostic, by issuing a command.

    The controller fetches the USB-size packet from the Tx buffers, and temporarily stores it in the internal FIFO RAM, before writing the packet back to the Rx buffers. This continues until all the data in the Tx buffers is exhausted. The Rx buffers must be at least equal to the Tx buffers. The controller generates an IOC interrupt (if the IOC bit is set) and at the end of transfer issues a XferComplDMADone interrupt. This validates the DMA, Rx, Tx, and Descriptor RAM access, and the interrupt path of the SoC. This also validates SoC bus Master interface, part of BMU, and part of the LSP of the DWC_usb3 controller.

5.7 SoC Integration Checklist

Table 5-7 describes the Device Integration checklist.

Table 5-7 Device SoC Integration Checklist

Port/Feature	Notes
dev_usb_outep_pkt_buff_avail[15:0]	If you are using standard AHB/AXI system memory and not using the additional buffer control signals, ensure that these are tied to 16'hFFFF
bus_clk_gm and bus_clk_gs	Tie the signals to 1'b1
pipe3_PowerPresent, utmisrp_bvalid	Refer to the section "Device/DRD Device Mode - BValid and VBUSValid Filter"
usb2phy_reset, pipe3_reset_n	Refer to the section 'Clock Generation and Clock Tree Synthesis (CTS) Requirements' in the *DWC SuperSpeed USB 3.0 Controller Databook*

Table 5-8 describes the Host Integration checklist.

Table 5-8 Host SoC Integration Checklist

Port/Feature	Notes
bus_clk_gm and bus_clk_gs	Tie the signals to 1'b1
host_num_u2_port	If you are not using this feature, assign the following; assign host_num_u2_port = 'DWC_USB3_HOST_NUM_U2_ROOT_PORTS;
host_num_u3_port	If you are not using this feature, assign the following; assign host_num_u3_port = 'DWC_USB3_HOST_NUM_U3_ROOT_PORTS;
hub_overcurrent_ctl, hub_vbus_ctrl	Refer to the section "Connecting Port VBUS and Over-Current Signals in Host and DRD Modes" on page 172
hub_port_perm_attach	Tie the signals to "0", if you are not using these signals
host_u2_port_disable, host_u3_port_disable	Tie the signals to "0" if you are not using these signals
pipe3_PowerPresent, utmiotg_vbusvalid	Refer to section "Host/DRD Host Mode - VBUSValid Filter"
usb2phy_reset, pipe3_reset_n	Refer to the section 'Clock Generation and Clock Tree Synthesis (CTS) Requirements' in the *DWC SuperSpeed USB 3.0 Controller Databook*

5.8 Host Mode – Miscellaneous Topics

This subsections discusses the following topics:

- "Bus Interval Adjustment"
- "Periodic Bandwidth Considerations"

5.8.1 Bus Interval Adjustment

A SS device can request the host to increase or decrease the bus interval length by sending a Bus Interval Adjustment Message. This is typically used by a device trying to synchronize the host's bus interval clock with an external clock. The host controller currently does not support hardware-based bus interval adjustment according to received Bus Interval Adjustment Device Notifications. As a workaround, the bus interval adjustment can be performed in the following software-assisted way.

- Software enables the reporting of the Bus Interval Adjustment Device Notification by setting the corresponding bit in the xHCI Device Notification Control Register. As a result, the host controller generates a Device Notification Event when it receives a Bus Interval Adjustment Device Notification.
- Software extracts the Bus Interval Adjustment field from the Device Notification Event, calculates the required bus interval adjustment value in HS bit times, and programs the xHCI Frame Length Adjustment Register accordingly.

Note

- The xHCI specification only allows software to modify the Frame Length Adjustment Register value when the host is halted. To support the previous workaround, the host controller allows the software to modify the register when the host controller is running.
- The procedure described is only a workaround. It has several limitations. First of all, the granularity and range of Bus Interval Adjustment in the device notification are different from that of Frame Length Timing Value in the Frame Length Adjustment. Secondly, the workaround does not follow the exact bus interval adjustment process defined in the USB 3.0 specification. Thirdly, the host controller does not update the Bus Interval Adjustment Control field in the ITP to indicate which device is controlling the bus interval adjustment.

5.8.2 Periodic Bandwidth Considerations

The host controller calculates the bandwidth requirement for each periodic EP when the EP is configured. If adding the EP would exceed the USB bandwidth upper limit for periodic traffic or the host is not able to accommodate the EP based on its scheduling algorithm, the Configure EP command is rejected.

For Super Speed, the USB bandwidth (4 Gbps per direction) is significantly high compared to High Speed (Table 5-9 lists some of the maximum bandwidth SS EP combinations). The system bus bandwidth can become a bottleneck in some systems. Currently, the host controller does not calculate system bus bandwidth requirement for the periodic EPs and periodic EPs are accepted as long as they can be scheduled on USB. Therefore, software is required to calculate the system bus bandwidth requirements and avoid exceeding system bus bandwidth when it configures periodic EPs.

Table 5-9 Various Maximum Bandwidth SS Endpoint Combinations

Number of SS Periodic Endpoints per Direction	Possible Combinations of Number of 1 KB Packets of Each Periodic Endpoint per Bus Interval
Two periodic Endpoints	{48, 2}, {47, 3}, {46, 4}, {45, 5}, …
Three periodic Endpoints	{48, 1, 1}, {47, 2, 1}, {46, 2, 2}, {46, 3, 1}, {45, 3, 2}, {45, 4, 1}, …
Four periodic Endpoints	{46, 1, 1, 1}, {45, 2, 2, 1}, (44, 4, 1, 1), {44, 3, 2, 1}, {44, 2, 2, 2}, …

5.8.3 xHCI Optional Behavior

Some of the xHCI behaviors have multiple options. An xHC implementation can choose one of those options. This section describes some of the optional behaviors and the ones that the DWC_usb3 host controller chooses.

5.8.3.1 Doorbell Reference to Disabled Slot or Endpoint

If a doorbell register is written by software with the DB Target value that references an endpoint or a Slot that is in the Disabled state, the xHC can either generate a Transfer Event TRB with Completion Code of Endpoint/Slot Not Enabled Error, or just ignore the doorbell write without generating an event. The DWC_usb3 host controller chooses to ignore those doorbell writes without generating event.

5.8.3.2 Transfer TRB Error

xHCI specification does not require xHC to handle transfer TRB errors gracefully.

A TRB type error indicates that the TRB type is not a valid TRB type for the EP. For example, a Setup Stage TRB on a Bulk transfer ring is considered as a TRB type error. The DWC_usb3 host controller detects TRB type errors on transfer rings, and halts itself with Host Controller Error asserted.

Any other transfer TRB error can cause an undefined host controller behavior because the DWC_usb3 host controller does not detect them. It is software's responsibility to construct valid transfer TRBs.

5.8.4 Save and Restore

Software can request the host controller to save its internal states by setting the Controller Save State bit in the USBCMD register, and can subsequently request the host controller to restore its internal states to the previously saved values by setting the Controller Restore State bit in the USBCMD register. The save and restore operations are typically used in a dual-power-rail implementation to preserve the internal states of the host controller during low power mode, in which the core power rail may be turned off.

5.8.4.1 Save and Restore for a Dual-Power-Rail Implementation

The Save and Restore operations for a dual-power-rail implementation are described in detail in the "Hibernation" chapter in the *DWC SuperSpeed USB 3.0 Controller Databook* as part of the hibernation flow.

5.8.4.2 Save and Restore for a Single-Power-Rail Implementation

For a single-power-rail implementation, because the power will remain turned on even in the low power mode, there are two options to support save and restore operations depending on the system behavior:

1. If your system does not assert vcc_reset_n to the host controller during low power mode, you can set DWC_USB3_EN_PWROPT to 0 or 1 when you configure the host controller. Because the power is always present and the vcc_reset_n is not asserted during low power mode, the host controller internal states does not get lost. When the software requests save or restore, the host controller returns success status without actually performing the save or restore operation.

2. If your system asserts vcc_reset_n to the host controller during low power mode, then you should set DWC_USB3_EN_PWROPT to 2 when you configure the host controller. Because the vcc_reset_n is asserted during low power mode, the host controller internal states are lost. Software needs to use Save and Restore operations to preserve the host controller states. When the software issues a Save request, the host controller writes the internal states to the scratchpad buffers in the system memory. When the software issues a Restore request, the host controller reads the state information from the scratchpad buffers in the system memory, and restores the internal states.

The second option is recommended because it provides better system flexibility due to its real save and restore feature. The software can always ask the host controller to save its internal states, and then ask the host controller to restore them later. It does not need to know whether the vcc_reset_n is asserted between the save and the restore. If you choose the second option, for more information on how to perform save and restore operations, refer to the "Hibernation" chapter in the *DWC SuperSpeed USB 3.0 Controller Databook*. For single-power-rail implementation, you can ignore the power rail control related operations in the Hibernation section.

5.9 TX/RX Data FIFO Sizes and TX/RX Threshold Control Register Settings

This section discusses the following topics:

- "FIFO Size Selection"
- "xHCI Debug Capability's Impact on FIFO Sizes"
- "Packet Threshold Value and Maximum Burst Size Selection"

5.9.1 FIFO Size Selection

To accommodate system latencies, the controller implements TX data FIFOs and RX data FIFOs to store TX or RX data before they are sent to the USB or the system bus. In the host mode, each USB bus instance has one TX data FIFO and one RX data FIFO. In the device mode, each IN endpoint has one TX data FIFO, and all OUT endpoints share a single RX data FIFO.

You can select the FIFO sizes while configuring the DWC_usb3 controller using coreConsultant. The FIFO sizes must be selected based on the time taken for system read and write. Generally, the slower the system bus (compared to the USB speed), the larger the FIFO sizes.

It takes about 2.1us to transfer a 1 kilobyte packet on the SuperSpeed USB. Based on whether your system bus latency is larger than 2.1us, you can configure the controller by selecting the DWC_USB3_MDBUS_ACCESS_GT21 coreConsultant parameter accordingly. If the system bus latency is larger than 2.1us, the controller needs to fetch more packets into the TX FIFO to avoid FIFO underrun during a burst because FIFO underrun causes burst early termination and lowers the data transfer performance. In coreConsultant, the following is true:

- 3-packet SS TX FIFO is recommended if the system latency is less than 2.1us
- 5-packet SS TX FIFO is recommended if the system latency is larger than 2.1us

However, if the system bus latency is significantly larger than 2.1us, the controller needs even a larger TX FIFOs. The same requirement applies to the RX direction also.

If a host configuration has multiple USB bus instances, then the same system bus is shared by all USB bus instances. For example, if the system bus latency is 2us, and the configuration has two SS bus instances, then the effective system bus latency for each SS bus instance is 4us. The FIFO sizes for each bus instance must be selected based on the 4us system bus latency.

Because the USB 2.0 speeds are significantly slower than SS, the system bus requirement for HS and FS/LS bus instance is not included in the previous analysis. If your configuration has multiple USB 2.0 bus instances, the system bus bandwidth impact of all USB 2.0 bus instances can be significant and must be taken in to consideration when the FIFO sizes are selected.

5.9.2 xHCI Debug Capability's Impact on FIFO Sizes

When you enable xHCI Debug Capability, then the system bus bandwidth requirement of the controller can increase. The FIFO sizes might also need to be increased to accommodate more system bus latency. For example, for a two SS port and one SS bus instance host with a 1.5us system bus latency, the FIFOs need not be large because the system bus is faster than the SS USB. If the Debug Capability is enabled in the same configuration, the SS host bus instance and the Debug bus instance can work simultaneously, which makes the effective system bus latency 3us for each bus instance. Therefore, larger FIFOs are required.

5.9.3 Packet Threshold Value and Maximum Burst Size Selection

For large latency systems, having large TX/RX FIFOs alone is not sufficient. For example, if the controller has a 16-packet TX FIFO, and the system bus is slower than the SS USB, TX FIFO underrun can happen if the controller starts a 16-packet SS OUT burst as soon as the first packet is fetched into the TX FIFO.

To solve this issue, the controller provides a packet threshold feature in the host mode. This feature is enabled by the Global Tx Threshold Control Register (GTXTHRCFG) and Global Rx Threshold Control Register (GRXTHRCFG). For OUT transfers, the software sets the number of TX data packets that must be in the TX FIFO before the host controller starts an OUT burst on the USB. For IN transfers, the software sets the number of RX packet space that must be available in the RX FIFO before the host controller starts an IN burst on the USB. For example, if the system bus read time for a 1 kilobyte packet is 3us, to avoid TX FIFO underrun, the host must wait for at least six packets (set the USB Transmit Packet Count in the GTXTHRCFG register to six) in the TX FIFO before it starts a 16-packet OUT burst on the SS USB.

If the system bus is slow and large FIFOs cannot be implemented due to area constraints, the software can limit the maximum burst size that the host can support by programming the USB Maximum TX/RX Burst Size fields in the GTXTHRCFG and GRXTHRCFG registers. In the previous example, the host has to fetch six packets in the TX FIFO before it starts a 16-packet OUT burst on the SS USB to avoid TX FIFO underrun. However, if only a 4-packet TX FIFO is implemented due to area constraints, the software can program the USB Maximum TX Burst Size to 13 and the USB Transmit Packet Count to four to avoid TX FIFO underrun during an OUT burst. As a result, the host performs bursts of up to 13 packets only instead of up to 16 packets. For ISOC endpoints, the host performs multiple bursts to satisfy the service interval payload requirement of the Endpoint.

Using the packet threshold mode reduces the USB transfer performance because the host needs to wait for certain FIFO conditions before it starts a burst on the USB. Due to this, there is idle time on the USB. If the system bus is faster than the USB, then the packet threshold mode is not necessary and must not be used. If the system bus is slower than the USB, with proper packet threshold values, the reduced USB bandwidth can match the system bus bandwidth and avoid the overhead caused by FIFO underrun and overrun.

A
Additional coreConsultant Information

The topics in this section are as follows:

A.1 Troubleshooting

This section provides some common problems you may encounter when performing activities in coreConsultant.

A.1.1 Specify Configuration Activity

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

- The coreConsultant tool issues a message that it cannot write to certain files. Check your available storage. You may have too little storage on your server.

A.1.2 Create Gate-Level Netlist Activity

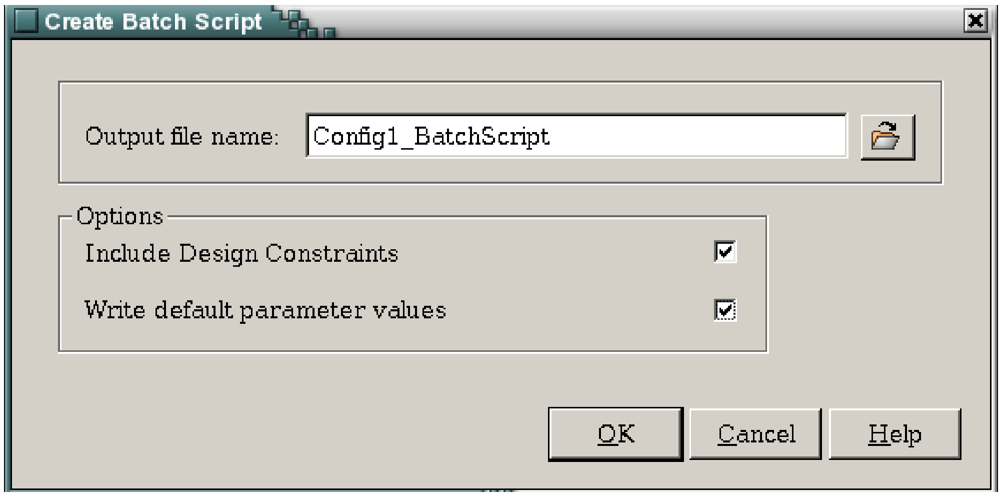
Here are some common problems that can occur after you perform synthesis (ASIC or FPGA) for your configured controller:

- 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.
- A cell name was specified that 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.
- If you cannot achieve timing closure with the default settings, then send your configuration along with the syn/final/DWC_usb3.log file to Customer Support.

A.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, as illustrated in Figure A-1.

Figure A-1 Create Batch Script



Output file name: Config1_BatchScript

Options
Include Design Constraints ✓
Write default parameter values ✓

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:

```
source <batch_file_name>
```

A.3 Help Information

Several types of online help are available through coreConsultant:

- **coreConsultant User Manual and Command Reference**

  These are available through the Help menu (see Figure A-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
  ```

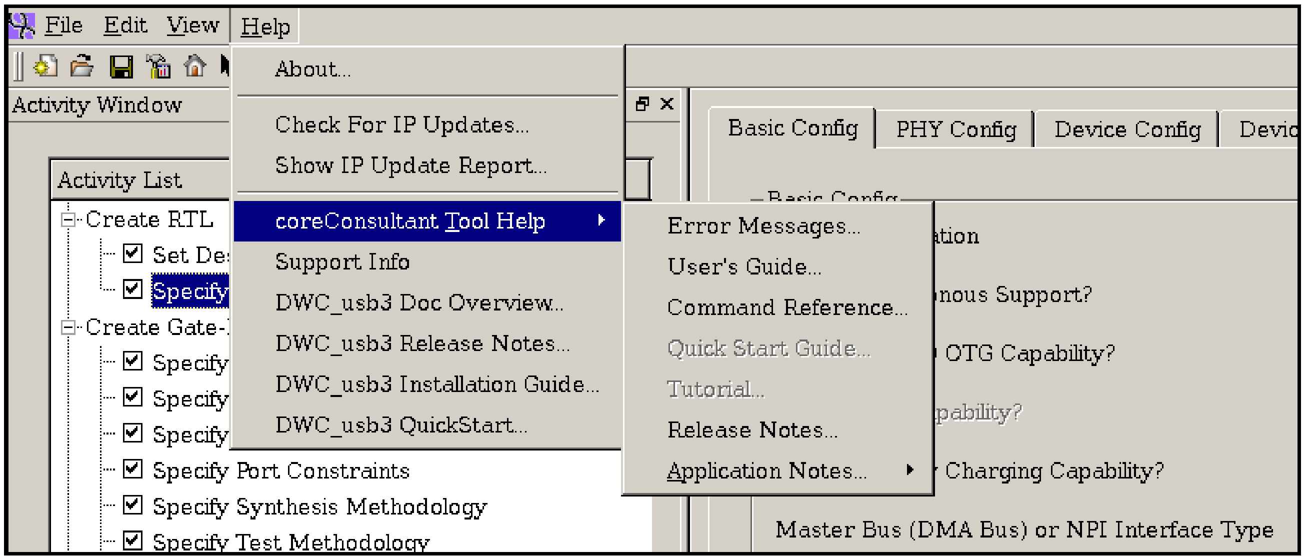
- **"What's This?" Quick Help**

  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 A-2shows the Help pull-down with the available manuals for both the USB 3.0 Controller and the coreConsultant tool.

Figure A-2 coreConsultant Help Menu Pull-down with DWC_usb3 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]
```

A.4 Workspace Directory Structure Overview

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

Figure A-3 coreConsultant Workspace Directory

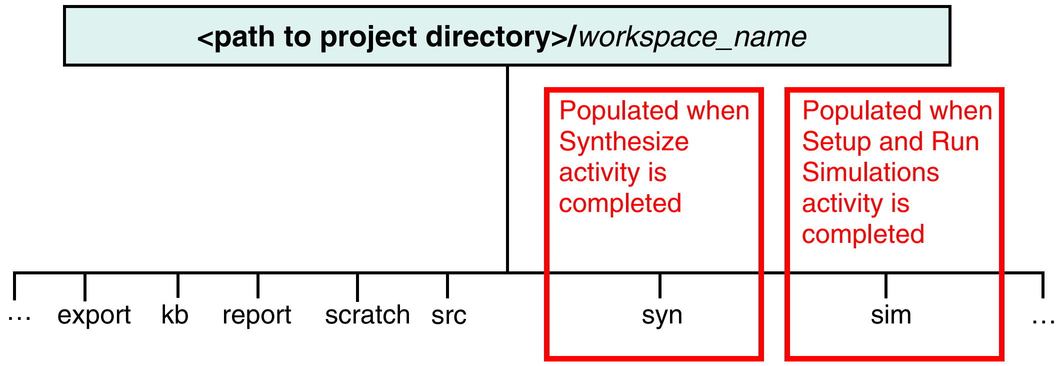

<path to project directory>/*workspace_name*
Populated when Synthesize activity is completed
Populated when Setup and Run Simulations activity is completed
…
export
kb
report
scratch
src
syn
sim
…

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

Table A-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 USB 3.0 Controller documentation (.pdf files). Generated upon first creating workspace.
example	Contains UVM-based Packaged Verification Environment (PVE) tcl scripts.
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.
fpga	Contains FPGA implementation example files and Synopsys PHY application note.
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.

Table A-1 Workspace Directory Contents (Continued)

Directory/Subdirectory	Description
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.
sim	Contains test stimulus and output files. Generated upon first creating workspace; updated during the Setup and Run Simulations activity.
SoC_sim	Contains files for SoC integration example testbench and README
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 USB 3.0 Controller. 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.
tools	Contains simulation-related scripts and utilities used by coreConsultant

A.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).

B
FPGA Implementation of DWC_usb3 Controller

The DesignWare® IP Prototyping Kits for USB 3.0 Host and Device center around complete, out-of-the-box reference designs that consist of a validated IP configuration and necessary SoC integration logic, implemented on Synopsys' HAPS®-DX FPGA-based prototyping system. More information about the USB 3.0 IP Prototyping kits for your FPGA implementation is located on the following Synopsys website:

https://www.synopsys.com/dw/ipdir.php?ds=ip_prototyping_kit_usb