DesignWare® Cores SuperSpeed USB 3.0 Controller Synthesis and CTS

Application Note

DWC SuperSpeed USB 3.0 Controller

Version 1.20a
February 2018

Copyright Notice and Proprietary Information

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

www.synopsys.com

Contents

Revision History

The following table provides the history of changes to this document.

Date	Version	Description
February 2018	1.20a	Chapter 1, "DWC_usb3 Clocks" ■ Updated Table 1-1 on page 8 and Table 1-2 on page 8 ■ Updated "Input Clock Period Considerations" on page 10 and "SDC Constraint Considerations" on page 11 Chapter 2, "DWC_usb3 Clock Logic Structure" ■ Updated DWC_USB3_EN_USB2_ONLY as 0 ■ Added ram_clk_in and ram_clk_out signals in all clock diagrams Chapter 3, "DWC_usb3 SDC File" ■ Updated "Understanding SDC File Structure" on page 21 Chapter 4, "DWC_usb3 Synthesis Constraints" ■ Updated "create_generated_clock" on page 25 ■ Updated ram_clk to ram_clk_in in case 3 of "set_false_path" on page 29 and case 1 of "set_max_delay" on page 31 ■ Deleted section "Case 6: RAM Interface" and "Case 4: Gray Code Pointer Paths" from "set_max_delay" on page 31 Chapter 5, "CTS Requirements and STA Analysis" ■ Added notes on div_2_* and div_4_* ■ Updated "Analyzing Path Through Synchronizers" on page 36 Deleted "Clock Matrix and Analysis Guidance for CTS/STA" chapter
March 2017	1.10a	Added a note on pipe3_mx_rx_pclk and mac3_clk in "Input Clock Period Considerations" on page 10
May 2016	1.00a	Initial version

Preface

This application note will help you understand the Synopsys DesignWare Cores SuperSpeed USB 3.0 Controller (DWC_usb3) clock structure and synthesis constraints. It also gives the guidelines for how to modify the synthesis constraints for Clock Tree Synthesis (CTS).

Note The information in this application note is based on version 3.30a of the DWC_usb3 controller.

The sections of this application note are organized as follows:

- "DWC_usb3 Clocks" on page 7 shows clock matrix information that can be used in understanding the clocks in the DWC_usb3 controller.
- "DWC_usb3 Clock Logic Structure" on page 13 discusses the clock logic structure as implemented in the DWC_usb3_clk.v module.
- "DWC_usb3 SDC File" on page 21 describes how to generate the Synopsys Design Constraint (SDC) file, and explains its format.
- "DWC_usb3 Synthesis Constraints" on page 24 describes Synopsys constraints policy that may help you in porting various synthesis constraints for the DWC_usb3 controller to the SoC level.
- "CTS Requirements and STA Analysis" on page 34 discusses CTS requirement for glitch-free clock MUXing, how to map hard cell, and analyze STA results.

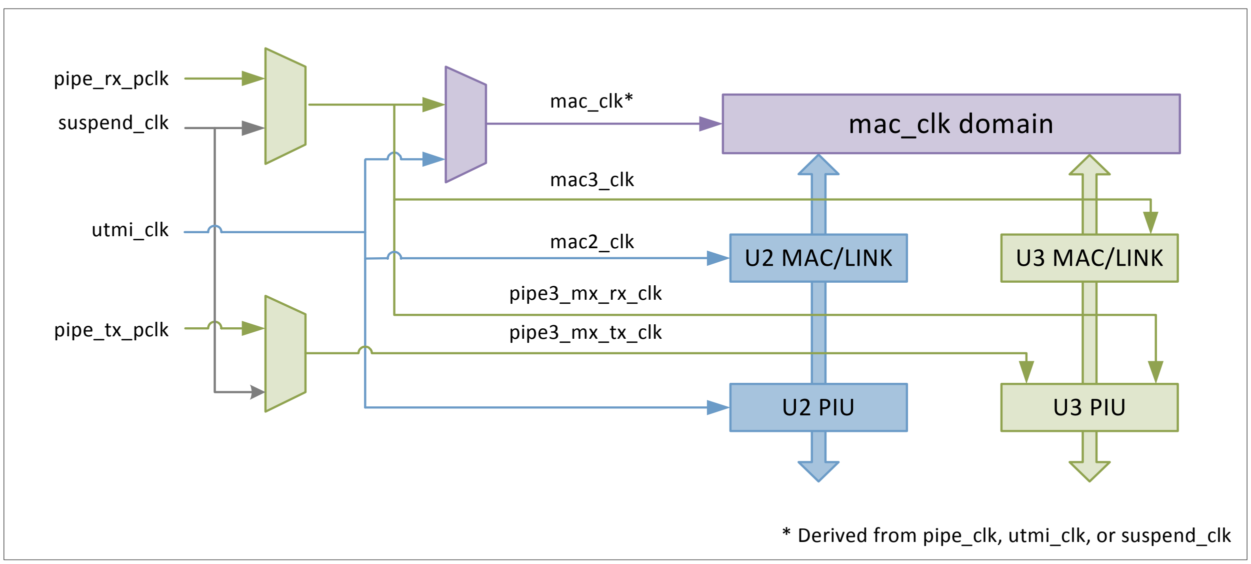
1
DWC_usb3 Clocks

This section shows clock matrix information to help you understand the clocks in the DWC_usb3 controller.

Note The tables in this section are for a single-port configuration. For multi-port Host or DRD configurations, there will be additional PHY clocks corresponding to the ports (for example, pipe3_rx_pclk_1, and so on).

Figure 1-1 is the clock structure of the DWC_usb3 controller for an example configuration.

Figure 1-1 Example Clock Structure



The remainder of this section discusses the following topics:

- "Clocks Definitions" on page 8
- "Relationship Between Source and Destination Clocks" on page 9
- "Input Clock Period Considerations" on page 10
- "SDC Constraint Considerations" on page 11

1.1 Clocks Definitions

Table 1-1 and Table 1-2 list all DWC_usb3 clocks and their default definitions.

- If the master clock signal is an input port of the controller, the create_clock constraint is used to define the clock. Table 1-1 shows a list of these clocks.
- If the master clock signal is an internal pin, the create_generated_clock constraint is used to define the clock. Table 1-2 shows a list of these clocks. This table also gives the relationship between master and generated clocks.

For usage examples, see "Understanding SDC File Structure" on page 21.

Table 1-1 Defining Clocks Using create_clock

Clock Name	Period	Master Clock Signal	Master Clock Source Point	Duty
bus_clk_early	8	[get_ports {bus_clk_early}]	N.A.	50-50
pipe3_rx_pclk_0	4	[get_ports {pipe3_rx_pclk[0]}]	N.A.	50-50
pipe3_tx_pclk_0	4	[get_ports {pipe3_tx_pclk[0]}]	N.A.	50-50
ulpi_clk_0	16.66	[get_ports {ulpi_clk[0]}]	N.A.	50-50
utmi_clk_0	16.66	[get_ports {utmi_clk[0]}]	N.A.	50-50
ram_clk_in	8	[get_ports {ram_clk_in}]	N.A	50-50
ref_clk	8	[get_ports {ref_clk}]	N.A.	50-50
suspend_clk	8	[get_ports {suspend_clk}]	N.A.	50-50

Table 1-2 Defining Clocks Using create_generated_clock

Clock Name	Master Clock Signal	Master Clock Name	Master Clock Source Point	Divided
div_2_pipe3_rx_pclk	[get_pins {U_DWC_usb3_clk/U_DWC_usb3_clk_div_2_4/clk_div_2}]	pipe3_rx_pclk_0	[get_ports {pipe3_rx_pclk[0]}]	2

1.2 Relationship Between Source and Destination Clocks

Table 1-3 can be used to understand the timing constraint used between the asynchronous clocks. Each cell that crosses a source clock to a destination clock has the meaning as shown in the table. For example, if a signal crosses from the bus_clk domain to the pipe3_rx_pclk_0 domain, it means that the signal crosses from a 3ns clock domain to a 4ns clock domain. Empty cells indicate that no set_max_delay or set_false_path - hold constraints are needed.

Table 1-3 Relationship Between Source and Destination Clocks

Source Clock Information			Destination Clocks								
	Clock Name	Period (ns)	bus_clk_early	pipe3_rx_pclk_0	pipe3_tx_pclk_0	div_2_pipe3_rx_pclk	ulpi_clk_0	utmi_clk_0	ram_clk_in	ref_clk	suspend_clk
Source Clocks	bus_clk_early	8		8	8	16	33.32	33.32	16	16	16
	pipe3_rx_pclk_0	4	16				33.32	33.32	16	16	N/A
	pipe3_tx_pclk_0	4	16				33.32	33.32	16	16	N/A
	div_2_pipe3_rx_pclk	(8)	16				33.32	33.32	16	16	N/A
	ulpi_clk_0	16.66	16	8	8	16		N/A	16	16	16
	utmi_clk_0	16.66	16	8	8	16	N/A		16	16	16
	ram_clk_in	8	16	8	8	16	33.32	33.32		16	16
	ref_clk	8	16	8	8	16	33.32	33.32	16		16
	suspend_clk	8	16	N/A	N/A	N/A	33.32	33.32	16	16	

1.3 Input Clock Period Considerations

You can modify the following input clock periods in coreConsultant based on your design requirements, if needed:

- bus_clk_early
- ref_clk
- suspend_clk

  The suspend_clk period must be 8ns (125MHz) while specifying the constraints.
- ram_clk_in

  The ram_clk_in clock can be connected with the ram_clk_out clock directly.

  Maximum ram_clk_out frequency depends on the register GCTL[7:6], which is set by the software.

2'b00	bus clock
2'b01	pipe clock (used only in device mode)
2'b10	■ Device mode: 1/2 pipe clock (when 8-bit UTMI or ULPI is used) ■ Host mode: between pipe/2 clock, mac2_clk, and bus_clk based on the status of the U2/U3 ports
2'b11	■ Device mode: mac2_clk (when 8-bit UTMI or ULPI is used) ■ Host mode: between pipe_clk, mac2_clk, and bus_clk based on the status of the U2/U3 ports

- pipe3_rx_pclk[n:0], pipe3_tx_pclk[n:0] (PIPE clock period)

  You can modify the input pipe clock period corresponding to the PIPE data bus width as follows:
  - 32 bits: 8ns
  - 16 bits: 4ns
- utmi_clk[m:0] (UTMI clock period)

  You can modify the input UTMI clock period corresponding to the UTMI interface data bus width as follows:
  - 16 bits: 33.33ns
  - 8 bits: 16.66ns

Note
- Clock domain crossing constraints are specified using set_max_delay.
- In host mode, mac_clk is synchronous to mac2_clk.

1.4 SDC Constraint Considerations

This section discusses the setup and hold timing.

Setup Timing

The set_max_delay is a good constraint for asynchronous paths because it takes into account the skew and setup time. This application note includes more detailed information on the necessary justification for the set_max_delay constraint during Place and Route (P&R), and STA.

The set_false_path constraints can be selected using the “Use set_false_path instead of set_max_delay between clocks?” option (DWC_USB3_CLK_SET_FALSE_PATH parameter) in coreConsultant.

Hold Timing

Use set_false on hold for asynchronous paths during synthesis so that the tool does not spend time in adding buffers and fixing hold time violations.

2
DWC_usb3 Clock Logic Structure

This section discusses the clock logic structure as implemented in the DWC_usb3_clk.v module.

A combination of the following configuration parameters are considered to illustrate the clock logic structure in DWC_usb3_clk.v module:

- Power Optimization Mode (DWC_USB3_EN_PWROPT)
  - 0: No Power Optimization
  - 1: Clock Gating Only
  - 2: Clock Gating and Hibernation
- Enable Additional DFT Control Ports? (DWC_USB3_ATSPEED_DFT)
  - 0: No
  - 1: Yes

There are other parameters that affect the clock structure. However, in this section, the following configuration parameters are assumed to be a fixed:

- DWC_USB3_PIPE_32BIT_ONLY = 0
- DWC_USB3_EN_USB2_ONLY = 0
- DWC_USB3_RAM_CLK_TO_BUS_CLK = 0

Note Each colored clock line in the clock diagrams in this section shows the synchronous clock group.

Table 2-1 on page 14 shows the reference clock diagrams for different configurations.

Table 2-1 Clock Diagrams for Different DWC_usb3 Configurations

DWC_USB3_EN_PWR_OPT Parameter	DWC_USB3_ATSPEED_DFT Parameter	Description	Reference Clock Diagram
0	0	No Power Optimization, and Non-AtSpeed DFT	Figure 2-1 on page 15
0	1	No Power Optimization, and AtSpeed DFT	Figure 2-2 on page 16
1	0	Clock Gating Only, and Non AtSpeed DFT	Figure 2-3 on page 17
1	1	Clock Gating Only, and AtSpeed DFT	Figure 2-4 on page 18
2	0	Clock Gating and Hibernation, and Non AtSpeed DFT	Figure 2-5 on page 19
2	1	Clock Gating and Hibernation, and AtSpeed DFT	Figure 2-6 on page 20

Figure 2-1 Clock Diagram – No Power Optimization + Non-AtSpeed DFT

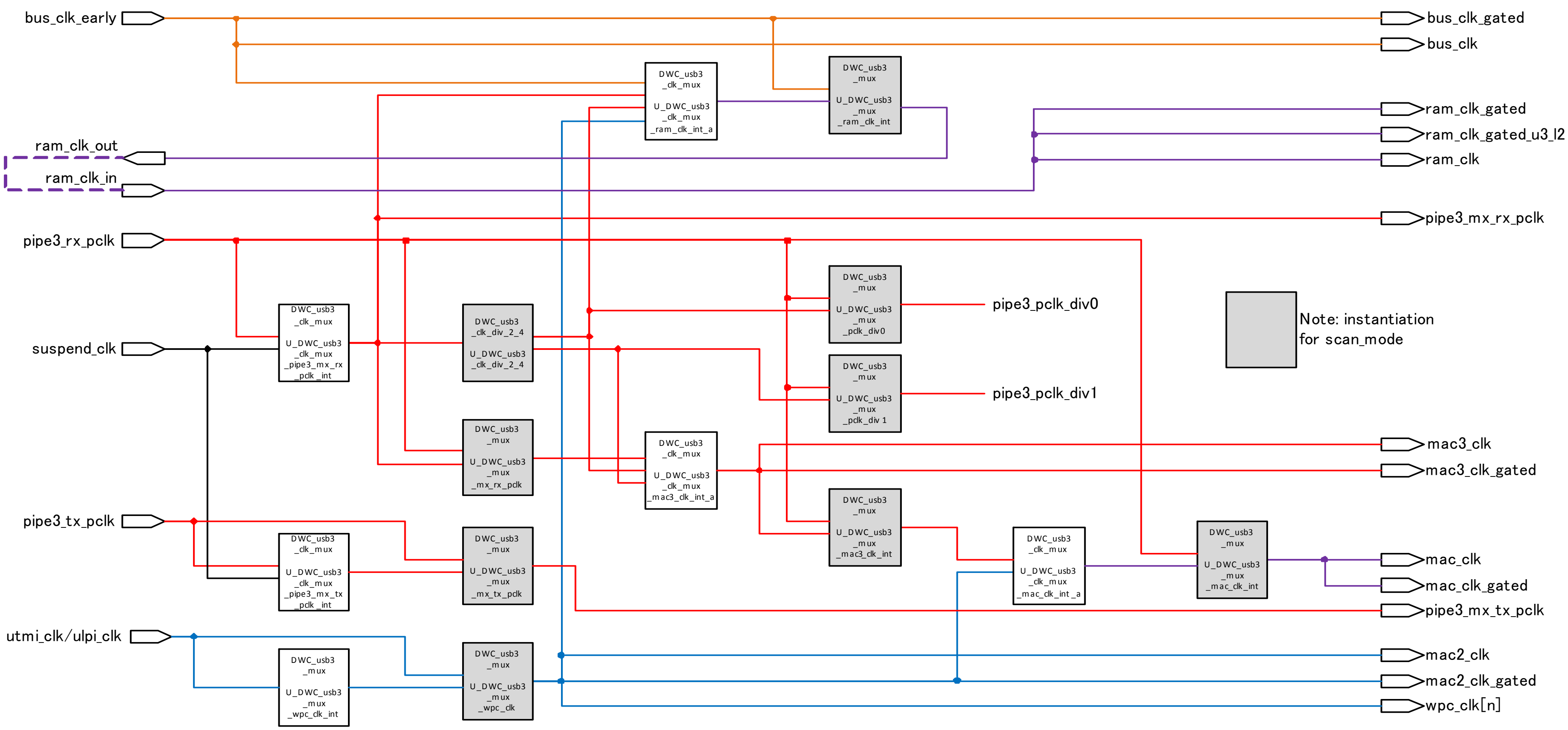


Figure 2-2 Clock Diagram – No Power Optimization + AtSpeed DFT

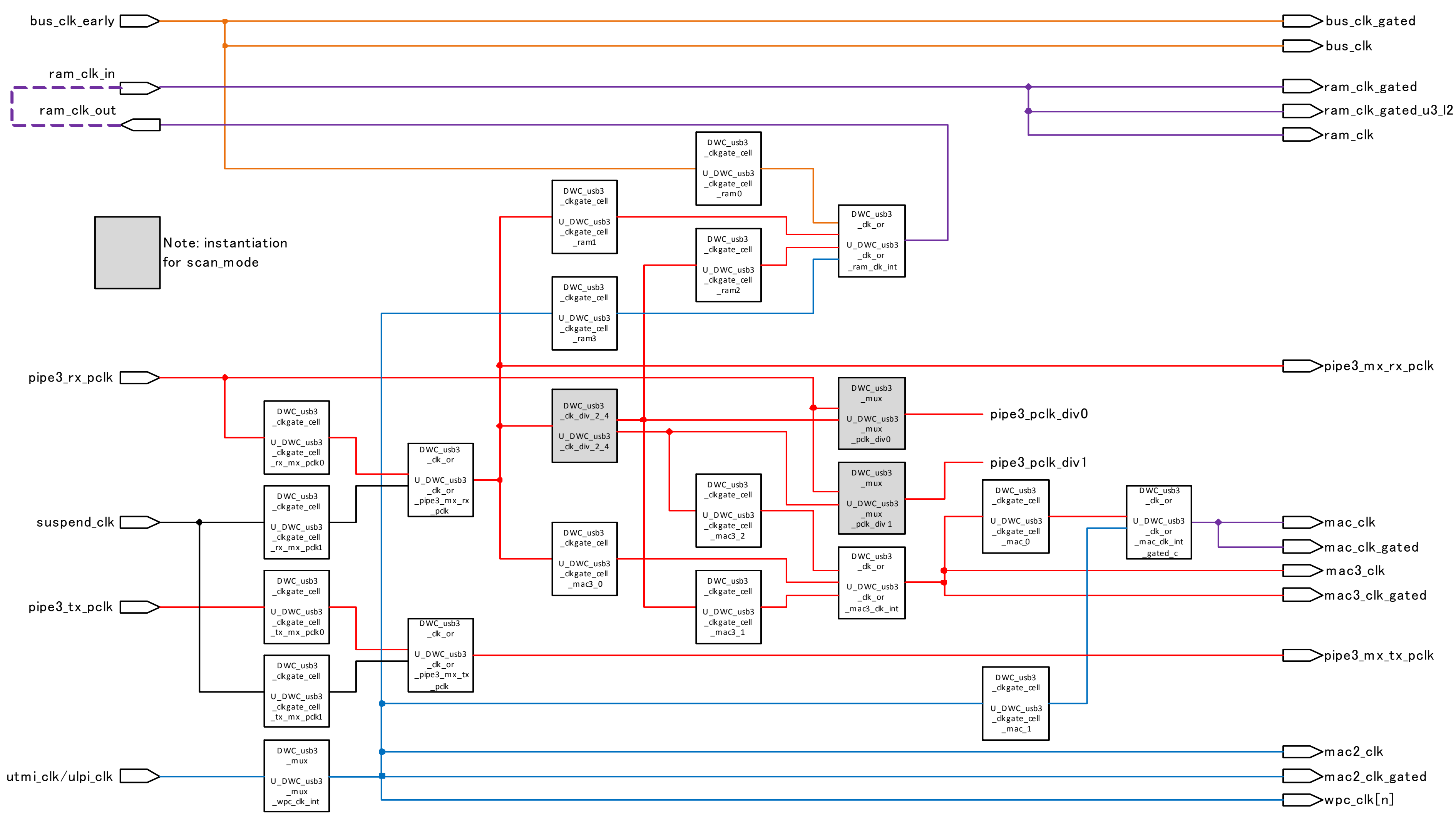


Figure 2-3 Clock Diagram – Clock Gating Only + Non AtSpeed DFT

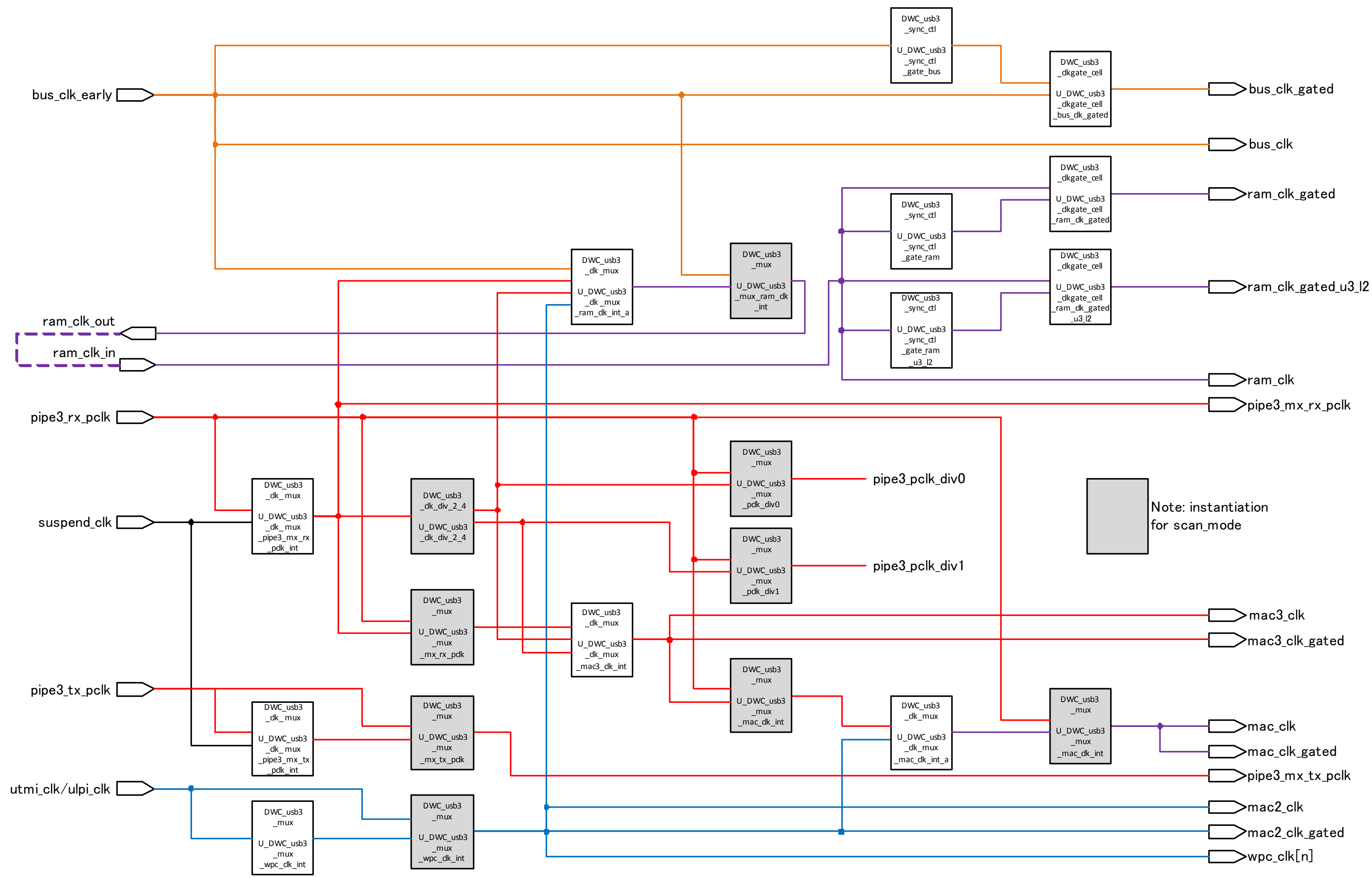


Figure 2-4 Clock Diagram – Clock Gating Only + AtSpeed DFT

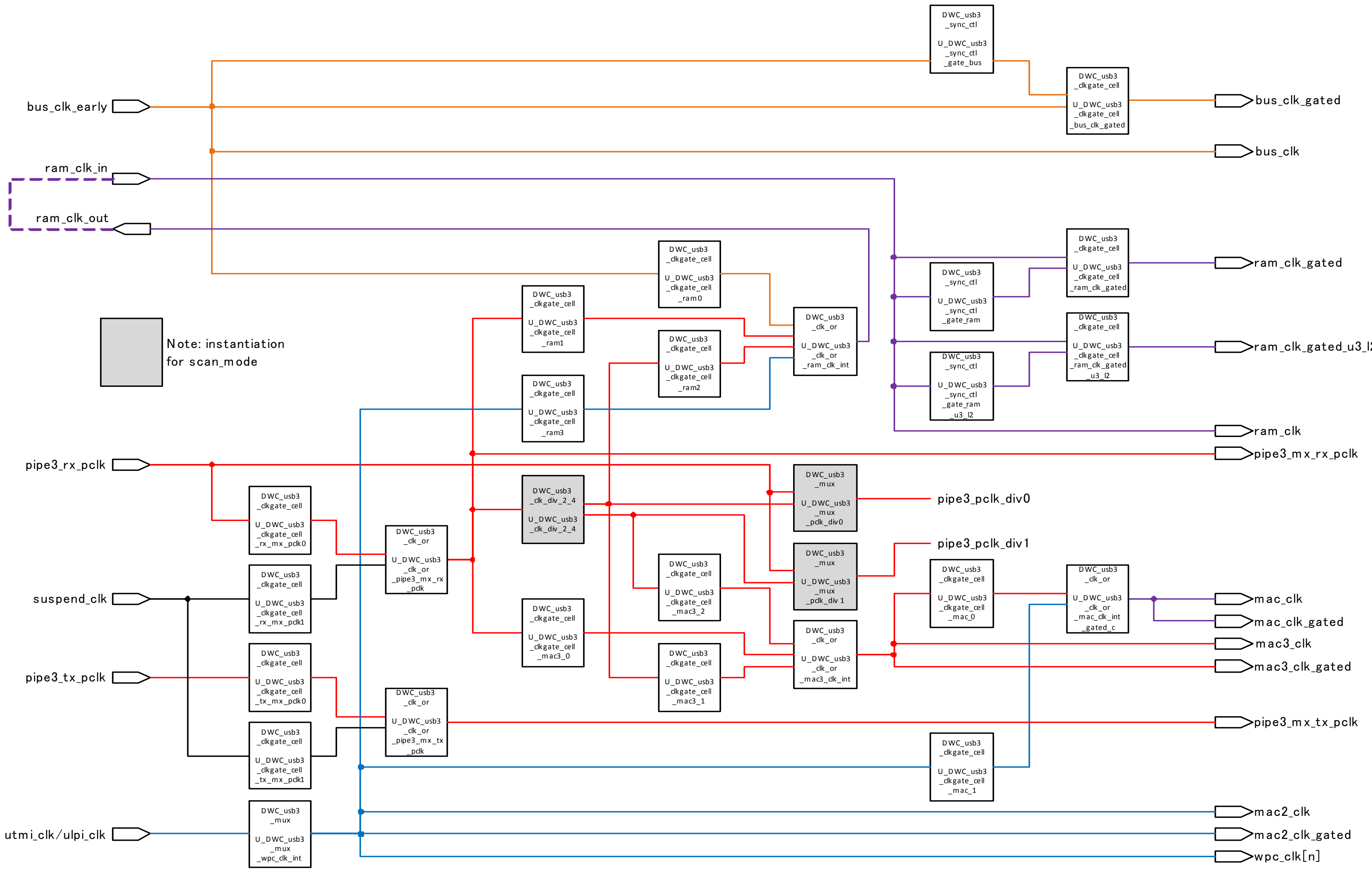


Figure 2-5 Clock Diagram – Clock Gating and Hibernation + Non AtSpeed DFT

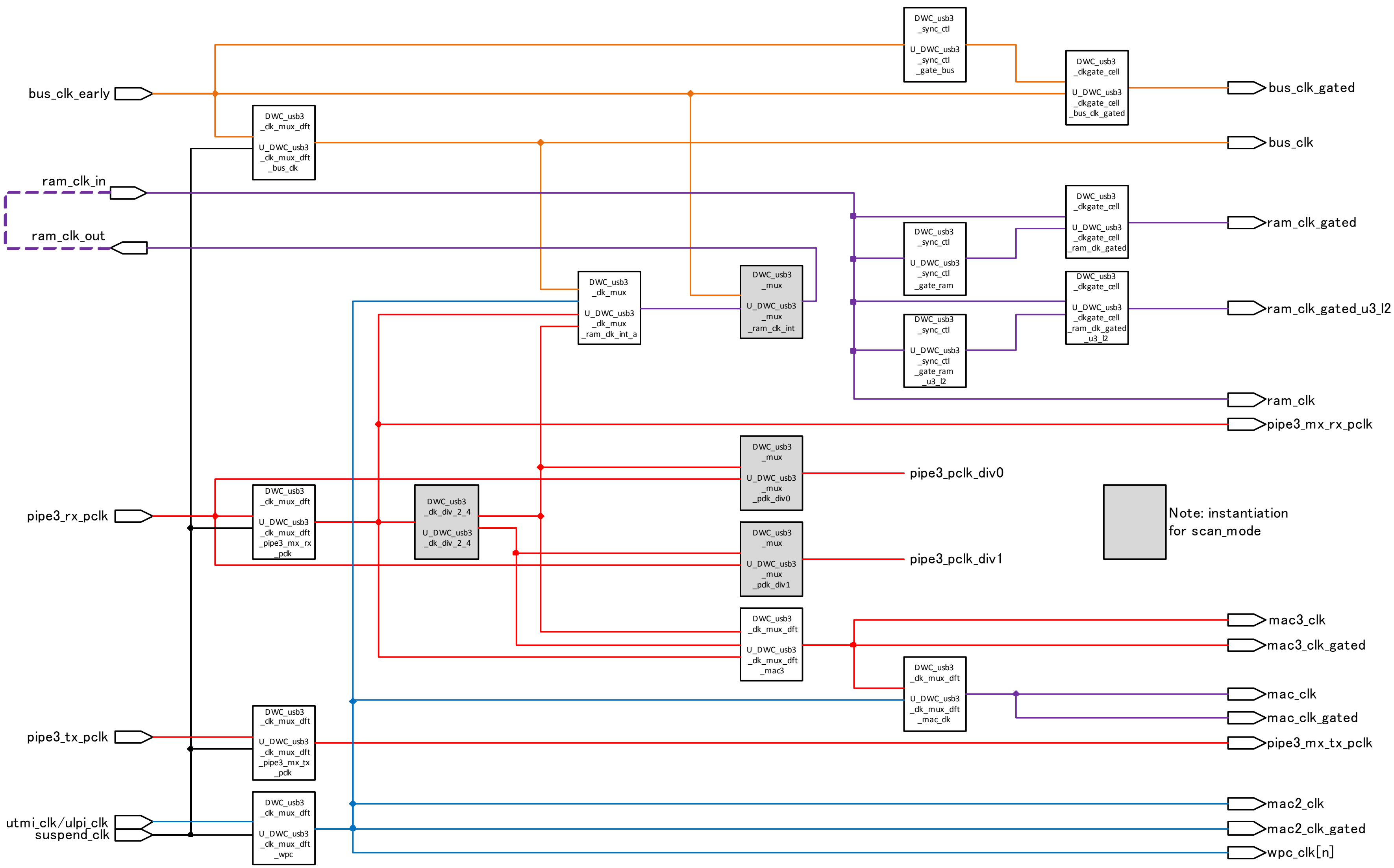
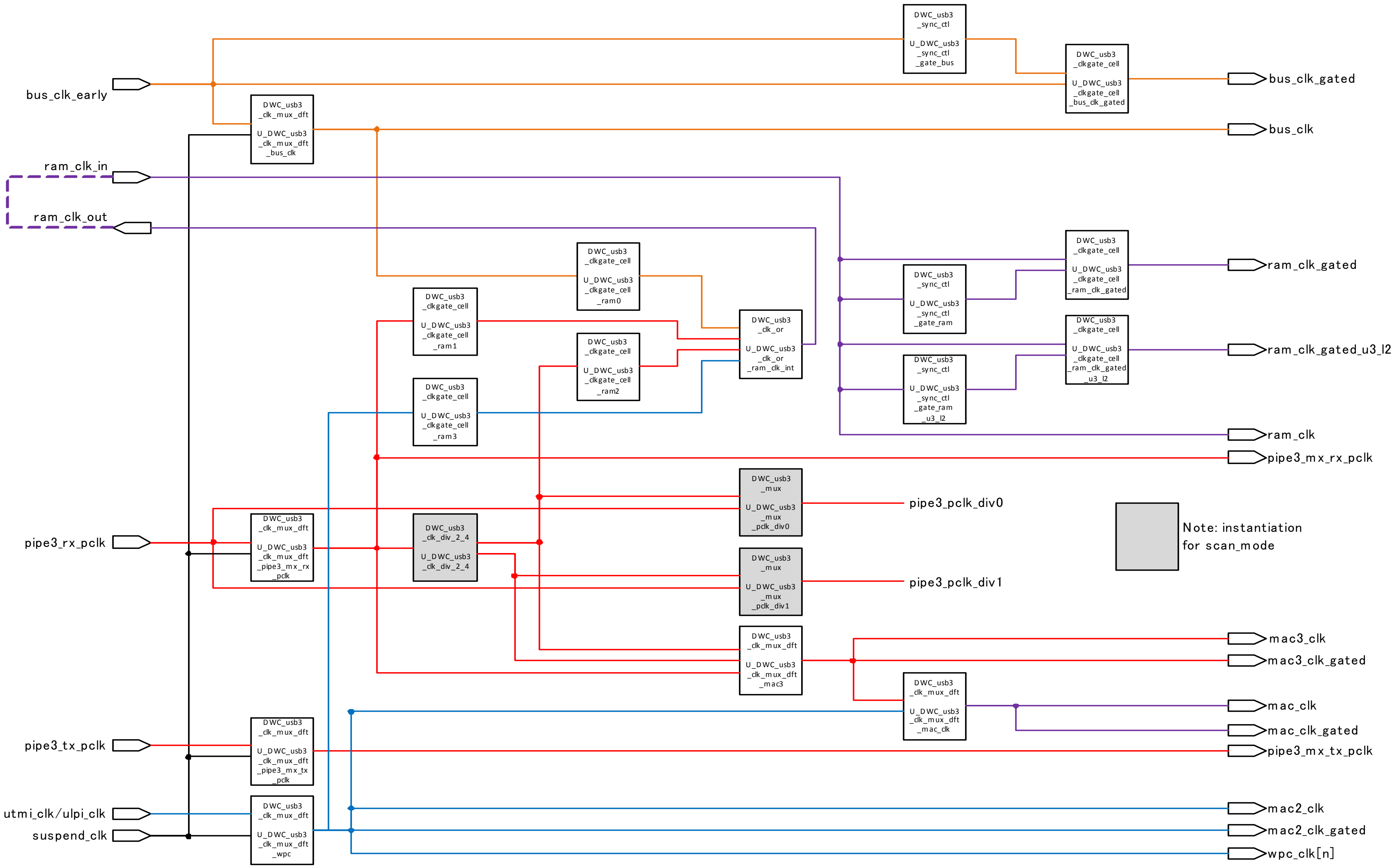


Figure 2-6 Clock Diagram – Clock Gating and Hibernation + AtSpeed DFT



3
DWC_usb3 SDC File

This section discusses how to generate the SDC file and describes the SDC file structure.

3.1 Generating SDC File

After you configure the controller, you can run synthesis with Design Compiler using the coreConsultant tool. After you have run synthesis, coreConsultant generates several reports including a Synopsys Design Constraints (SDC) file in the following location:

./syn/final/db/DWC_usb3.sdc

If you are not using Design Compiler, use the write_sdc command to write out a script in a SDC format. You can also generate the SDC constraints file using the Report tab. For more details, see *DWC SuperSpeed USB 3.0 Controller User Guide*.

3.2 Understanding SDC File Structure

In this section, a sample SDC file structure is shown as follows to help you understand the file contents and all the constraints. For details on each constraint, see "DWC_usb3 Synthesis Constraints" on page 24.

set sdc_version 1.9 # ** # coreConsultant generated intent command file for design: DWC_usb3 # Generator version: 2.0 # Generated at: 00:00:00 on 01/01/16 # **	
# Clock waveform definitions and synthesis directives. create_clock [get_ports {bus_clk_early}] -name bus_clk_early -period 8 -waveform {0 4} set_clock_uncertainty 0.2 [get_clocks {bus_clk_early}] set_clock_latency 0 [get_clocks {bus_clk_early}] set_clock_latency -source 0 [get_clocks {bus_clk_early}] :	Clock definitions

```
create_clock [get_ports {utmi_clk[0]}] -name utmi_clk_0 -period 16.66 -
waveform {0 8.33}
set_clock_uncertainty 0.4165 [get_clocks {utmi_clk_0}]
set_clock_latency 0 [get_clocks {utmi_clk_0}]
set_clock_latency -source 0 [get_clocks {utmi_clk_0}]

# Create generated clocks.
create_generated_clock [get_pins
{U_DWC_usb3_clk/U_DWC_usb3_clk_div_2_4/clk_div_2}] -name
div_2_pipe3_rx_pclk -source [get_ports {pipe3_rx_pclk[0]}] -add -master_clock
pipe3_rx_pclk_0 -divide_by 2

set_clock_uncertainty 0 [get_clocks {div_2_pipe3_rx_pclk}]
set_clock_latency 0 [get_clocks {div_2_pipe3_rx_pclk}]
set_clock_latency -source 0 [get_clocks {div_2_pipe3_rx_pclk}]

# Operating conditions for the design.
set_operating_conditions nom_pvt -library class

# Wireload modeling information for the design.
set_wire_load_mode top
set_wire_load_selection_group -library class class [current_design]

# Minimum/Maximum delay values (arrival times) for input ports.
set_input_delay -add_delay -max 0.8 -clock bus_clk [get_ports {vcc_reset_n}]
set_input_delay -add_delay -min 0.4 -clock bus_clk [get_ports {vcc_reset_n}]
   :
set_input_delay -add_delay -max 0.4 -clock bus_clk [get_ports {xhc_bme}]
set_input_delay -add_delay -min 0.4 -clock bus_clk [get_ports {xhc_bme}]

# Minimum/Maximum delay values (external required times) for output ports.
set_output_delay -add_delay -max 4.8 -clock bus_clk [get_ports {interrupt}]
set_output_delay -add_delay -min 0 -clock bus_clk [get_ports {interrupt}]
   :
set_output_delay -add_delay -max 3.2 -clock bus_clk [get_ports
{host_legacy_smi_interrupt}]
set_output_delay -add_delay -min 0 -clock bus_clk [get_ports
{host_legacy_smi_interrupt}]

# Estimated loads seen externally by ports.
set_load -pin_load 6 [get_ports {debug}]
set_port_fanout_number 3 [get_ports {debug}]
set_load -pin_load 6 [get_ports {logic_analyzer_trace}]
set_port_fanout_number 3 [get_ports {logic_analyzer_trace}]
  :
set_load -pin_load 6 [get_ports {host_legacy_smi_interrupt}]
set_port_fanout_number 3 [get_ports {host_legacy_smi_interrupt}]
```

Generated clock definitions

Operating conditions

Wire load

Input delay definitions

Output delay definitions

Port loads

# Assume the following drive cells for input ports. set_drive 0 [get_ports {vcc_reset_n}] set_drive 0 [get_ports {bus_clk_early}] set_driving_cell [get_ports {bus_clken_gs}] -library class -lib_cell FD1 -pin Q -no_design_rule : set_driving_cell [get_ports {fladj_30mhz_reg}] -library class -lib_cell FD1 -pin Q -no_design_rule set_driving_cell [get_ports {xhc_bme}] -library class -lib_cell FD1 -pin Q -no_design_rule	Driving cell
# Drive resistance for input or inout ports. set_drive 0 [get_ports {vcc_reset_n}] set_drive 0 [get_ports {bus_clk_early}] :	Port drive
set_false_path -from [get_ports scan_mode] set_max_delay -to [get_ports pipe3_reset_n] 32 set_max_delay -from [get_clocks bus_clk] -through utmi_suspend_n* 32 : set_multicycle_path -setup \ -from \ U_DWC_usb3_noclkrst/U_DWC_usb3_pwrdwn/*U_DWC_usb3_lsp/*U_DWC_usb3_lsp_hst/U_DWC_usb3_lsp_hcmd/max_esit_payload* 2 : set_max_delay -from [get_ports host_u3_port_disable] 32 set_false_path \ -through U_DWC_usb3_noclkrst/U_DWC_usb3_pwrdwn/U_DWC_usb3_tr/pcc2_rd_epnum* \ -through U_DWC_usb3_noclkrst/U_DWC_usb3_pwrdwn/U_DWC_usb3_tr/pcc3_dev_ep_info* : set_max_delay -from [get_clocks pipe3_rx_pclk_0] -to [get_clocks ram_clk_in] 16 set_false_path -hold \ -from [get_clocks pipe3_rx_pclk_0] \ -to [get_clocks ram_clk_in] set_max_delay -from [get_clocks ref_clk] -to [get_clocks ram_clk_in] 16 set_false_path -hold -from [get_clocks ref_clk] -to [get_clocks ram_clk_in] : set_max_delay -from [get_clocks suspend_clk] -to [get_clocks pipe3_tx_pclk_0] 8 set_false_path -hold \ -from [get_clocks suspend_clk] \ -to [get_clocks pipe3_tx_pclk_0]	Specific path constraints

4 DWC_usb3 Synthesis Constraints

This section describes Synopsys constraints policy that may help you in porting the following constraints for the DWC_usb3 controller while working at the SoC level:

This section also indicates whether a DWC_usb3 constraint is applicable at the SoC level in Table 4-3 on page 33.

4.1 create_clock

The create_clock constraint defines all the input clocks and some internal clocks. The following command is a sample to define the clock.

```
create_clock [get_ports bus_clk_early] -period 8 -waveform {0 4}
```

> **Note**
> - You can use coreConsultant to modify the input clock periods.
> - The input clock definition can be used for the backend also.
> - For details on input clock period considerations, see "Input Clock Period Considerations" on page 10.

4.2 create_generated_clock

The create_generated_clock constraint defines the internal clocks. The following sample constraint specifies the internal generated clock.

```
create_generated_clock [get_pins
{U_DWC_usb3_clk/U_DWC_usb3_clk_div_2_4/clk_div_2}] -name div_2_pipe3_rx_pclk -
source [get_ports {pipe3_rx_pclk[0]}] -add -master_clock pipe3_rx_pclk_0 -
divide_by 2
```

> **Note** The create_generated_clock period is set automatically, therefore, you cannot change any period for create_generated_clock in coreConsultant.

4.3 set_clock_uncertainty

The set_clock_uncertainty constraint defines the clock uncertainty for create_clock. The value is 2.5% of the clock period. This set_clock_uncertainty constraint is not applied for create_generated_clock. The following sample constraint specifies the uncertainty.

```
set_clock_uncertainty 0.1 [get_clocks {pipe3_rx_pclk_0}]
```

> **Note**
> - You can use coreConsultant to modify the clock uncertainty value.
> - These constraints are not necessary for chip-level synthesis.

4.4 set_clock_latency

The set_clock_latency constraint defines the clock latency for all defined clocks by both create_clock and create_generated_clock. The value is fixed to 0ns. The following sample constraint specifies the latency.

```
set_clock_latency 0 [get_clocks {bus_clk_early}]
```

> **Note**
> - You can use coreConsultant to modify the clock latency value.
> - These constraints are not necessary for chip-level synthesis.

4.5 set_input_delay

The set_input_delay constraint specifies the input delay for all input signals for setup and hold timing. The value depends on the signal groups. The following sample constraints are used to define the input delay.

```
set_input_delay -add_delay -max 2.4 -clock pipe3_rx_pclk_0 [get_ports
{pipe3_RxValid[0]}]
```

```
set_input_delay -add_delay -min 0.2 -clock pipe3_rx_pclk_0 [get_ports
{pipe3_RxValid[0]}]
```

Table 4-1 lists the setup values for each signal group.

Table 4-1 Setup Values

Signal Group	Signal	Setup Value
AHB Interface group	hs_hsel	65% of clock period
	hs_hwdata	50% of clock period
	All other AHB interface signals	60% of clock period
AXI Interface group	All AXI interface signals	40% of clock period
RAM Interface group	All RAM interface signals	50% of clock period
UTMI Interface group	utmiotg_vbusvalid	25% of clock period
	All other UTMI interface signals	50% of clock period
ULPI Interface group	ulpi_dir	40% of clock period
	ulpi_nxt	40% of clock period
	All other ULPI interface signals	50% of clock period
PIPE Interface group	pipe3_PowerPresent	10% of clock period
	pipe3_DataBusWidth	20% of clock period
	All other PIPE interface signals	60% of clock period
Other Interfaces	vcc_reset_n	10% of clock period
	bus_clken_gs	10% of clock period
	bus_clken_gm	10% of clock period
	bigendian_gs	10% of clock period
	gp_in	10% of clock period
	host_legacy_smi_pci_cmd_reg_wr	40% of clock period
	host_legacy_smi_bar_wr	40% of clock period
	All other signals	05% of clock period

The value for hold is 05% of the clock period.

Note

- You can use coreConsultant to modify the setup and hold values.
- These constraints are not necessary for chip-level synthesis.

4.6 set_output_delay

The set_output_delay constraint specifies the output delay for all input signals for setup and hold timing. The value depends on signal groups. The following sample constraints are used to specify the output delay.

```
set_output_delay -add_delay -max 1.6 -clock pipe3_tx_pclk_0 [get_ports
{pipe3_TxData[0]}]

set_output_delay -add_delay -min 0 -clock pipe3_tx_pclk_0 [get_ports
{pipe3_TxData[0]}]
```

Table 4-2 lists the hold values for each signal group.

Table 4-2 Hold Values

Signal Group	Signal	Setup Value
AHB Interface group	All AHB interface signals	70% of clock period
AXI Interface group	All AXI interface signals	40% of clock period
RAM Interface group	All RAM interface signals	20% of clock period
UTMI Interface group	pipe3_PowerPresent	10% of clock period
	All other UTMI interface signals	60% of clock period
ULPI Interface group	utmi_l1_suspend_n	40% of clock period
	utmi_word_if	50% of clock period
	utmi_fsls_low_power	50% of clock period
	ulpi_stp	40% of clock period
	ulpi_tx_data_en	40% of clock period
	All other ULPI interface signals	60% of clock period
PIPE Interface group	All PIPE interface signals	40% of clock period
Other Interfaces	Interrupt	60% of clock period
	host_system_err	60% of clock period
	gp_out	60% of clock period
	bc_interrupt	60% of clock period
	All other signals	40% of clock period

The value for hold is fixed to 0ns.

> **Note**
> - You can use coreConsultant to modify the setup and hold values.
> - These constraints are not necessary for chip-level synthesis.

4.7 set_load

The set_load constraint specifies the load for all output signals. The value is fixed to 6. The following sample constraint is used to specify the load.

```
set_load -pin_load 6 [get_ports {pipe3_TxData[0]}]
```

> **Note**
> - You can use coreConsultant to modify the load value for all output signals.
> - These constraints are not necessary for chip-level synthesis.

4.8 set_port_fanout_number

The set_port_fanout_number constraint specifies the fanout_number for all output signals. The value is fixed to 3. The following sample constraint is used to specify the fanout number.

```
set_port_fanout_number 3 [get_ports {pipe3_TxDataK[0]}]
```

> **Note**
> - You can use coreConsultant to modify the fanout number for all output signals.
> - These constraints are not necessary for chip-level synthesis.

4.9 set_drive

The set_drive constraint specifies the drive strength for all output signals. The value is fixed to 0. The following sample constraint specifies the drive capability.

```
set_drive 0 [get_ports {bus_clk_early}]
```

> **Note**
> - You can use coreConsultant to modify the drive strength value for all output signals.
> - These constraints are not necessary for chip-level synthesis.

4.10 set_driving_cell

The set_driving_cell constraint specifies the driving cell for input signals. The following sample command specifies the driving cell. The specified cell depends on the target library.

```
set_driving_cell [get_ports {pipe3_RxData}] -library class -lib_cell FD1 -pin Q
-no_design_rule
```

Note

- You can use coreConsultant to modify the cell.
- These constraints are not necessary for chip-level synthesis.

4.11 set_false_path

The set_false_path constraint specifies the false paths. The false path constraint can be categorized under the following cases.

Case 1: Scan Mode Signal

The following constraint specifies the false path for scan_mode signal.

```
set_false_path -from [get_ports scan_mode]
```

Case 2: Internal Paths

The following two constraints specify the false path for internal paths:

```
set_false_path -through
U_DWC_usb3_noclkrst/U_DWC_usb3_pwrdwn/U_DWC_usb3_tr/pcc2_rd_epnum* -through
U_DWC_usb3_noclkrst/U_DWC_usb3_pwrdwn/U_DWC_usb3_tr/pcc3_dev_ep_info*
```

```
set_false_path -through
U_DWC_usb3_noclkrst/U_DWC_usb3_pwrdwn/U_DWC_usb3_tr/pcc3_rd_epnum* -through
U_DWC_usb3_noclkrst/U_DWC_usb3_pwrdwn/U_DWC_usb3_tr/pcc2_dev_ep_info*
```

Note These constraints are not necessary for chip-level synthesis.

Case 3: Hold Timing Path on Clock Domain Crossing

The following sample constraint specifies the false path for clock domain crossing.

```
set_false_path -hold -from [get_clocks ram_clk_in] -to [get_clocks
bus_clk_early]
```

For additional details, see “Hold Timing” on page 11.

4.12 set_multicycle_path

The set_multicycle_path constraint specifies the multi-cycle path for internal signals. There are four multi-cycle paths, therefore, eight constraints for setup and hold timing.

Path 1:

```
set_multicycle_path -setup -from
U_DWC_usb3_noclkrst/U_DWC_usb3_pwrdwn/*U_DWC_usb3_lsp/*U_DWC_usb3_lsp_hst/U_DWC
_usb3_lsp_hcmd/max_esit_payload* 2
```

```
set_multicycle_path -hold -from
U_DWC_usb3_noclkrst/U_DWC_usb3_pwrdwn/*U_DWC_usb3_lsp/*U_DWC_usb3_lsp_hst/U_DWC
_usb3_lsp_hcmd/max_esit_payload* 1
```

Path 2:

```
set_multicycle_path -setup -from [get_cells
U_DWC_usb3_noclkrst/U_DWC_usb3_pwrdwn/*U_DWC_usb3_lsp/*U_DWC_usb3_lsp_hst/U_DWC
_usb3_lsp_hcmd/cmd_trb_type* 2]
```

```
set_multicycle_path -hold -from [get_cells
U_DWC_usb3_noclkrst/U_DWC_usb3_pwrdwn/*U_DWC_usb3_lsp/*U_DWC_usb3_lsp_hst/U_DWC
_usb3_lsp_hcmd/cmd_trb_type* 1]
```

Path 3:

```
set_multicycle_path -setup -from
U_DWC_usb3_noclkrst/U_DWC_usb3_pwrdwn/*U_DWC_usb3_asev/*U_DWC_usb3_asev_soft/re
f_clk_frnum_reg* 2
```

```
set_multicycle_path -hold -from
U_DWC_usb3_noclkrst/U_DWC_usb3_pwrdwn/*U_DWC_usb3_asev/*U_DWC_usb3_asev_soft/re
f_clk_frnum_reg* 1
```

Path 4:

```
set_multicycle_path -setup -from
U_DWC_usb3_noclkrst/U_DWC_usb3_pwrdwn/*U_DWC_usb3_asev/*U_DWC_usb3_asev_soft/re
f_clk_ufrnum_reg* 2
```

```
set_multicycle_path -hold -from
U_DWC_usb3_noclkrst/U_DWC_usb3_pwrdwn/*U_DWC_usb3_asev/*U_DWC_usb3_asev_soft/re
f_clk_ufrnum_reg* 1
```

Note The set_multicycle_path constraints are specified only meet timing easier. This constraint is not related to functionality. You can remove these constraints if it is not necessary to meet the timing.

4.13 set_max_delay

The set_max_delay constraint specifies the max delay for timing. The max delay setting is divided into five cases.

Case 1: Clock Domain Crossing Paths

In the DWC_usb3 controller, all paths crossing clock domain are not synchronized. A qualifier-based synchronizer is used to qualify unsynchronized signals. To avoid scenic routing of unsynchronized signals and thereby violating the assumptions in the qualifier-based synchronizer, false_path constraint is avoided and max_delay is specified for all signals crossing clock domain.

For an example, refer to section "Toggle Control and Data Synchronizer (src/com/DWC_usb3_sync_toggledata.v)" in the *DWC SuperSpeed USB 3.0 Controller Databook*, version 3.30a.

```
set_max_delay -from [get_clocks bus_clk_early] -to [get_clocks ram_clk_in] 8
```

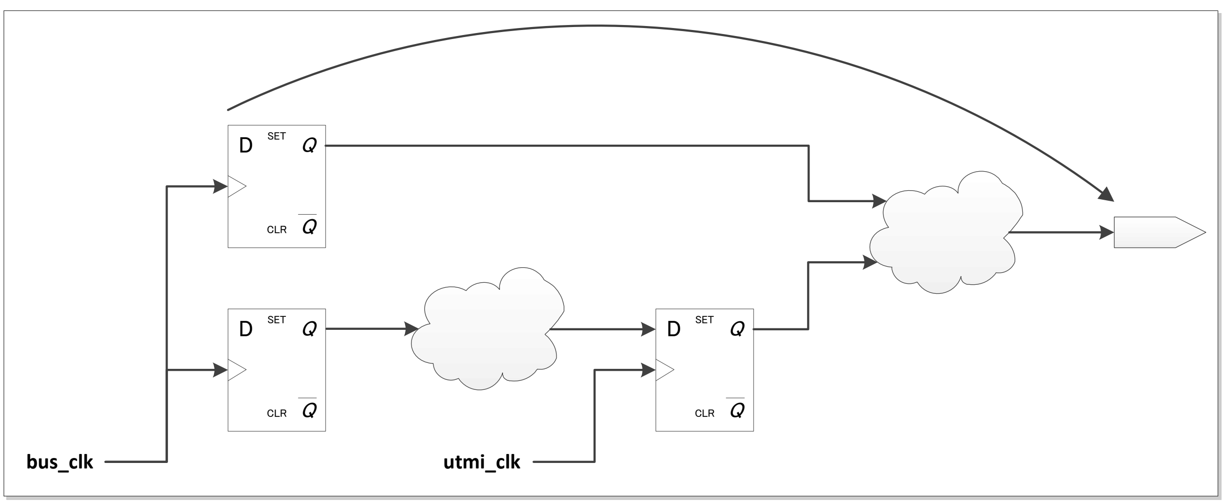
For additional details, see "Setup Timing" on page 11.

Case 2: Combinational Output Signals

This avoids scenic routing. The following sample constraint specifies the max delay for combinational output signal.

```
set_max_delay -from [get_clocks bus_clk] -through utmi_tx_data* 32
```

Figure 4-1 set_max_delay Constraints for Combinational Output Signals



Note

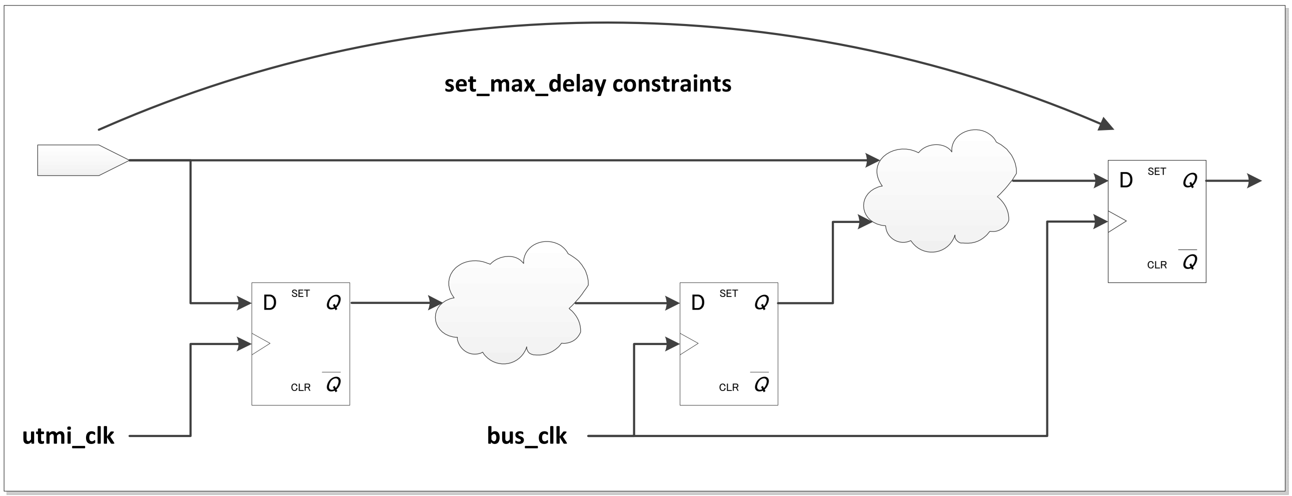
- These constraints are only for avoiding scenic routing.
- These constraints are not necessary for chip-level synthesis.

Case 3: Combinational Input Signals

This avoids scenic routing. The following sample constraint specifies the max delay for combinational input signal.

```
set_max_delay -from [list [get_ports utmi_txready] [get_ports utmi_rx_data]
[get_ports utmi_rxvalidh] [get_ports utmi_rxvalid] [get_ports utmi_rxactive]
[get_ports utmi_rxerror] [get_ports utmi_linestate] utmi_h* [get_ports
utmiotg_vbusvalid]] -to [get_clocks bus_clk] 14.994
```

Figure 4-2 set_max_delay Constraints for Combinational Input Signals



> **Note**
> - These constraints are only for avoiding scenic routing.
> - These constraints are not necessary for chip-level synthesis.

Case 4: Internal Paths

The following paths define the max delay for internal paths.

Earlier, the methodology was to set false paths between the clock domains. Now, the set_max_delay constraint between the clock domains is used to avoid scenic routing. This value for set_max_delay is based on destination clock period. The following sample constraint is used to specify the max delay for clock domain closing.

```
set_max_delay -from
U_DWC_usb3_noclkrst/U_DWC_usb3_pwrdwn/U_DWC_usb3_u2pwrdwn/b2rl_cur_mode* 32
```

```
set_max_delay -from
U_DWC_usb3_noclkrst/U_DWC_usb3_pwrdwn/U_DWC_usb3_csr/U_DWC_usb3_csr_dev/gdbglsp
mux_reg* 32
```

> **Note** These constraints are not necessary for chip-level synthesis.

4.14 Applicable Constraints for Chip-Level Synthesis

Table 4-3 summarizes whether the DWC_usb3 constraints for P&R and STA are applicable at the SoC level.

Table 4-3 Applicable Constraints for Chip-Level Synthesis

Constraints		Chip-Level Synthesis and Layout/STA
create_clock	Input clock	Clock source definition should be changed
	Internal clock	Applicable
create_generated_clock		Applicable
set_clock_uncertainly		Not Applicable
set_clock_latency		Not Applicable
set_input_delay		Not Applicable
set_output_delay		Not Applicable
set_load		Not Applicable
set_port_fanout_number		Not Applicable
set_drive		Not Applicable
set_driving_cell		Not Applicable
set_false_path	Scan mode signal	Not Applicable
	Internal Paths	Not Applicable
	Hold timing path on clock domain crossing	Applicable
set_multicycle_path		Applicable
set_max_delay	Combinational output signals	Not Applicable
	Combinational input signals	Not Applicable
	Clock domain crossing paths	Applicable
	Gray code pointer paths	Applicable
	Internal Paths	Not Applicable
	RAM I/F	Not Applicable

5
CTS Requirements and STA Analysis

This section discusses the CTS requirement and provides STA analysis guidance under the following topics:

- "CTS Requirement for Glitch-Free Clock MUXing"
- "Mapping Hard Cell" on page 35
- "Analyzing STA Results" on page 36

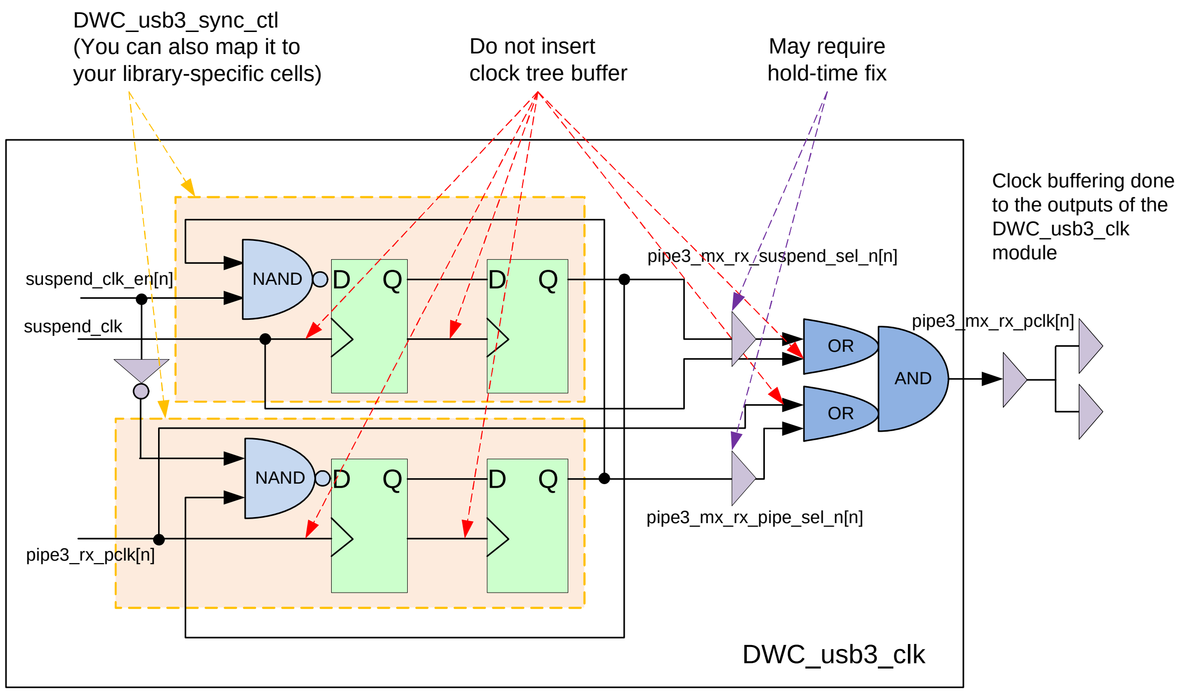
5.1 CTS Requirement for Glitch-Free Clock MUXing

The DWC_usb3 controller uses two types of clock MUXes for glitch-free clock MUXing.

Sequential Cell-Based Clock MUX

Figure 5-1 shows glitch-free clock MUXing.

Figure 5-1 Glitch-Free Clock Muxing



To check for a glitch-free clock in the STA phase, use the following constraints:

- When the suspend_clk side is analyzed:

```
set_case_analysis 1 [get_pins OR-AND/pipe3_mx_rx_pipe_sel_n]
set_clock_gating_check -setup 0 -hold 0 [get_pins OR-AND/pipe3_mx_rx_suspend_sel_n] -low
```

- When the pipe3_rx_pclk side is analyzed:

```
set_case_analysis 1 [get_pins OR-AND/pipe3_mx_rx_suspend_sel_n]
set_clock_gating_check -setup 0 -hold 0 [get_pins OR-AND/pipe3_mx_rx_pipe_sel_n] -low
```

Map DWC_usb3_sync_ctl to your library-specific cell, at the same time, pay attention to NOT insert a clock tree buffer as shown by the red arrow in Figure 5-1 on page 34. For more details, see Figure "Glitch Free Clock MUXing and CTS Requirements" in the *DWC SuperSpeed USB 3.0 Controller Databook*.

OR-AND-Based Clock MUX

The following clock structure instances use OR-AND logic:

- U_DWC_usb3_clk_mux_ram_clk_int_a
- U_DWC_usb3_clk_mux_mac3_clk_int_a
- U_DWC_usb3_clk_mux_mac_clk_int_a
- U_DWC_usb3_clk_mux_pipe3_mx_rx_pclk_int
- U_DWC_usb3_clk_mux_pipe3_mx_tx_pclk_int

The instances may be a little different with your configuration.

5.2 Mapping Hard Cell

For synthesis, from the DWC_usb3 controller version 3.10a onwards, Synopsys recommends NOT to define DWC_USB3_NO_LEAF_CELL. Instead, it is recommended to map DWC_usb3_double_sync*.v to the specific library cell.

5.3 Analyzing STA Results

Analyzing Clock Domain Crossing Signals

When max_delay constraints are set between the clock domain crossing signals, the STA tools also add the clock-tree insertion delay for timing analysis.

When there is a timing violation, you can check the timing report to decide if it can be relaxed by adding the difference in the clock-tree insertions to the max_delay constraint.

For example:

```
set_max_delay [expr 1.0 * [get_attribute [get_clocks $clk1] period] +
$clk_insertion_delay($clk2) - $clk_insertion_delay($clk1)] -from [get_clocks
$clk2] -to [get_clocks $clk1]
```

Analyzing Path Through Synchronizers

For any path that goes from one clock to another through a synchronizer, you can ignore or disable the setup/hold violations on the following first flop on the destination clocks:

- *src/com/DWC_usb3_sync_ctl.v/ *U_bcm41_w_async_rst*/*U_SYNC*/sample_meta_n*
- *src/com/DWC_usb3_sync_toggledata/in_toggle_1d*
- *src/com/DWC_usb3_sync_toggledata/* U_bcm21_toggle*/sample_meta_n*
- *src/com/DWC_usb3_sync_toggledata/out_data_reg*
- *src/com/DWC_usb3_bussync/d_data*
- *src/com/DWC_usb3_sync_2edge.v/in_p_1d*
- *src/pwrm/DWC_usb3_pwrm_u3piu/pipe3_sync_9b_32*

However, you cannot relax one T (of the destination clock period) set_max_delay constraint for path through synchronizers.

For more details, see “Clock Generation and Clock Tree Synthesis (CTS) Requirements” chapter of the *DWC SuperSpeed USB 3.0 Controller Databook*, version 3.30a.