UG0603
User Guide
RTG4 FPGA Timing Constraints (Classic Constraint Flow)

Libero SoC v11.8 SP4

Microsemi®

Power Matters.™

Microsemi Headquarters
One Enterprise, Aliso Viejo,
CA 92656 USA
Within the USA: +1 (800) 713-4113
Outside the USA: +1 (949) 380-6100
Sales: +1 (949) 380-6136
Fax: +1 (949) 215-4996
Email: sales.support@microsemi.com
www.microsemi.com

About Microsemi

Microsemi, a wholly owned subsidiary of Microchip Technology Inc. (Nasdaq: MCHP), offers a comprehensive portfolio of semiconductor and system solutions for aerospace & defense, communications, data center and industrial markets. Products include high-performance and radiation-hardened analog mixed-signal integrated circuits, FPGAs, SoCs and ASICs; power management products; timing and synchronization devices and precise time solutions, setting the world's standard for time; voice processing devices; RF solutions; discrete components; enterprise storage and communication solutions, security technologies and scalable anti-tamper products; Ethernet solutions; Power-over-Ethernet ICs and midspans; as well as custom design capabilities and services. Learn more at www.microsemi.com.

5-02-00603-2/07.18.

Table of Contents

Introduction

In designing FPGA synchronous digital designs, from design entry to physical implementation, rarely do you achieve the required timing performance of the design without iteration. You often must go through numerous iterations of the design cycle—HDL design capture, synthesis, physical implementation (Place and Route) and Timing Analysis—to achieve timing closure.

Setting Timing Constraints and performing Timing Analysis are the two most important steps in design iterations towards timing closure.

For RTG4 designs, Microsemi recommends setting timing constraints for both synthesis and place and route steps. You must first set the timing assertion constraints; see "Timing Assertions" on page 7.

If timing performance is not met in the first iteration, consider setting additional and more advanced timing constraints in the second and subsequent iterations. See "Timing Exceptions" on page 8.

1 – Using Synopsys Design Constraints

The Synopsys® Design Constraint (SDC) is a Tcl-based format used by Synopsys tools to specify the design intent and timing constraints. Microsemi supports a variation of the SDC format for constraints management.

You can use the following types of SDC commands when creating SDC constraints for RTG4 designs:

- Object Access
- Timing Assertions
- Timing Exceptions

Object Access

SDC timing constraints apply to specific design objects. Table 1-1 summarizes the object access commands supported by SmartTime (the Microsemi static timing analysis tool incorporated with the place and route tools). Refer to the SmartTime online help for more information.

***Table 1-1* • Object Access Commands Supported by SmartTime**

Design Object	Command(s)
Cells / Instances	get_cells
Clocks	get_clocks
Nets	get_nets
Pins	get_pins
Ports	get_ports, all_inputs, all_outputs
Registers	all_registers

Implicit vs. Explicit Specification

In general, SDC commands include design objects as an argument. SDC supports both implicit and explicit object specification.

When the tool determines the object type by searching for the object, it is called an implicit object specification. When the object type is specified (to avoid ambiguity) using a nested object access command, it is called an explicit object specification.

For example: If you have a net named 'my_net1', the implicit specification is my_net1 and the explicit specification is [get_nets my_net1].

Not all design objects are applicable to all SDC commands. Each SDC command accepts a pre-defined set of design objects as arguments. Microsemi recommends that you use the explicit object specification method to avoid ambiguity regarding object type. If multiple object types are returned after searching an implicit specification, the object types are prioritized based on the tool's priority object list.

Refer to the SmartTime online help for more information.

Wild Card Characters

Table 1-2 lists the wild card characters available for use in SDC commands.

Table 1-2 • **Object Access Commands Supported by SmartTime**

Wild Card	Function
\	Interprets the next character literally
*	Matches any string

Note: The matching function requires that you add a backslash (\) before each slash in the pin names in case the slash does not denote the hierarchy in your design.

Hierarchy and Pin Separators

Synplify Pro software defaults to the use of '.' (period) as both the hierarchy and pin separators for timing constraints.

For example: [get_pins {top_level.blockA.instance123.my_pin}]

To change the hierarchy separator from the default '.' (period) to the '/' (forward slash), use the command:

```
set_hierarchy_separator { / }
```

SmartTime software defaults to the use of '/' as a design hierarchy separator and ":" as the pin separator character.

For example: [get_pins {top_level/blockA/instance123:my_pin}]

Notice that '/' is the hierarchy separator used to indicate that 'instance123' is present in the design hierarchy below top_level ' blockA. The ":" pin separator identifies 'my_pin' on 'instance123'.

Bus Naming Conventions

All buses in the SDC file must use the Verilog-style naming convention name[index].

For example: [get_ports addr_bus_out[1]]

If you want to specify the constraint on the entire bus, you can use [get_port addr_bus_out].

Comments

You can add comments to an SDC file by preceding the comment line with a pound sign (#).

```
# This is a comment line
```

Timing Assertions

Timing assertions are intended to capture your design timing requirements.

They include the following SDC commands:

- Clock Period/Frequency
  - create_clock
  - create_generated_clock
- Input / Output Delay
  - set_input_delay
  - set_output_delay
  - set_external_check
  - set_clock_to_output
- Clock-to-clock Uncertainty
  - set_clock_uncertainty

- Clock Source Latency
  - set_clock_latency

Refer to "Timing Constraints and Design Flow" on page 9 for the Timing Assertion SDC commands Synplify Pro and SmartTime support.

Timing Exceptions

Use timing exceptions to identify design paths that require the default single cycle timing relationships to be overridden. SDC commands for timing exceptions include:

- False path
  - set_false_path
- Multicycle path
  - set_multicycle_path
- Maximum delay path
  - set_max_delay
- Minimum delay path
  - set_min_delay
- Disabled timing arcs
  - set_disable_timing

Timing Exceptions and Precedence Order

When the same timing path has more than one timing exception constraint, SmartTime honors the timing constraint with the highest precedence and ignores the other timing exceptions according to the order of precedence shown in Table 1-3. Synplify Pro honors the timing constraints according to Precedence Order in Table 1-4.

Table 1-3 **• Timing Exception - Precedence Order for SmartTime**

Timing Exceptions	Order of Precedence
set_disable_timing	1
set_false_path	2
set_maximum_delay/set_minimum_delay	3
set_multicycle_path	4

Table 1-4 **• Timing Exception - Precedence Order for Synplify Pro**

Timing Exceptions	Order of Precedence
set_false_path	1
set_max_delay/set_min_delay	2
set_multicycle_path	3

2 – Timing Constraints and Design Flow

This chapter describes where to specify timing constraints and perform timing analysis in the Libero design flow (Figure 2-1). Microsemi recommends that you supply Synplify Pro and SmartTime with adequate and complete timing constraints. Also, you must review the timing reports from both Synplify Pro and SmartTime to ensure that the design has been constrained properly and is meeting the timing goals.

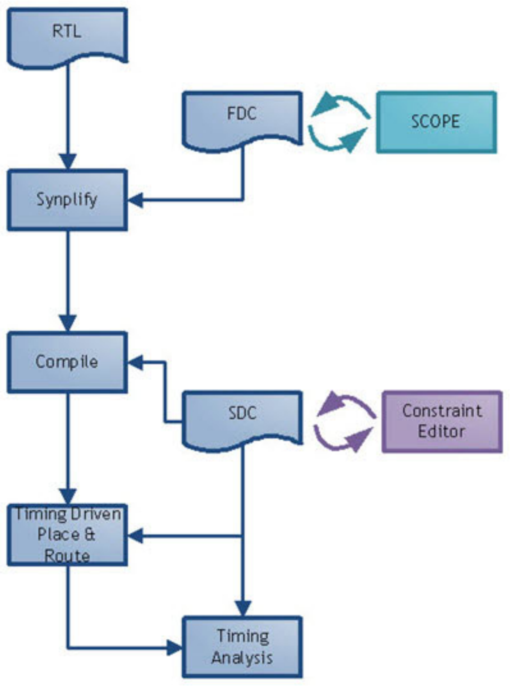


***Figure 2-1* • Timing Constraints in the Design Flow**

Libero tools (Timing Driven Place and Route and SmartTime) support a subset of Synopsys SDC timing constraints relevant for FPGA designs.

Microsemi recommends that you create two sets of timing constraints in the Libero flow:

- FDC timing constraints for synthesis with Synplify Pro.
- SDC timing constraints for the Libero Timing Driven Place and Route and SmartTime phases.

Timing Constraints for Synplify Pro

Overview

Synplify Pro supports the FPGA Design Constraints (FDC) format. The FDC format includes:

- A subset of the Synopsys SDC standard for timing constraints
- Legacy timing constraint format supported by Synplify Pro

You can provide timing constraints to Synplify Pro by:

- Importing the timing constraint file(s) into the Libero project. Identify the timing constraint file(s) to be passed to Synplify Pro in Libero GUI. Right click the file(s) and choose Use for Synthesis. For details about importing timing constraints in the Libero GUI, refer to the Libero online help.
- Creating the timing constraints using the SCOPE (Synthesis Constraints Optimization Environment) GUI available in Synplify Pro software. Constraints created using SCOPE are saved to a constraints file using the FDC format.

Supported Synplify Pro Timing Constraints

The following timing constraints are supported by Synplify Pro for FPGA synthesis:

- create_clock
- create_generated_clock
- set_input_delay
- set_output_delay
- set_false_path
- set_multicycle_path
- set_max_delay
- set_clock_latency
- set_clock_uncertainty

Refer to the Synplify Pro for Microsemi Reference Manual for details on the options and arguments,.

Constraints for Design Requirements

Synthesis software uses timing constraints to make trade-offs that lead to optimum use of resources to achieve requested timing goals. Timing constraints are essential to ensure that the right choices are made by the synthesis tool while performing logic and mapping optimizations of the design.

Microsemi recommends that you include Clock Constraints and Input and Output Delay Constraints.

There are two types of clock constraints:

- create_clock
- create_generated_clock

To define the design clocks for RTG4 designs,:

- Use create_clock constraint to identify and constrain oscillators and primary input ports used as clocks.
- Use create_generated_clock constraint to identify and constrain fabric CCC output pins used as clocks.

FDC Example with create_clock and create_generated_clock

In the example below a combination of create_clock and create_generated_clock constraints are used to define the required clock constraints.

First, the clock source is identified as the input port clk_in at 50 MHz.

Then this clock source is used to generate a 200 MHz clock (clk_core) using a CCC:

```
# Input Port 'clk_in' @ 50MHz is the clock source
create_clock -name {input_clock} \
-period 20 \
-waveform {0 10} \
[get_ports clk_in]
# CCC generates clk_core using clk_in as the source
# clk_core @ 200MHZ
create_generated_clock -name { clk_core }
-divide_by 1 -multiply_by 4\
-source [get_ports clk_in] \
[get_pins { clk_generator.FCCC_0.CCC_INST.GL1 }]
```

Note: The backslash "\" character is part of Tcl syntax. It breaks a long single command into multiple lines.

Note: Synplify Pro software defaults to the use of '.' (period) as the hierarchy separator for timing constraints. Use the set_hierarchy_separator command in the FDC file to redefine the hierarchy separator character. For example:set_hierarchy_separator {/}

The divide_by and multiply_by factors are derived from the PLL diagram displayed on the 'Advanced' tab of the fabric CCC configurator. Synplify Pro software defaults to 100 MHz required clock frequency for all clocks missing a timing constraint.

FDC Example with set_input_delay and set_output_delay

In this example, set_input_delay and set_output_delay constraints are used to define the required input and output delay timing constraints. These constraints are required to define the timing budget required for the I/O Interface.

In this example, all constraints use clk_core from the previous example as the reference clock.

The input delay on input port(s) data_bus_in_clk_core is 2.5ns (max) and 1.0ns (min).

The output delay on output port(s) data_bus_out_clk_core is 3.0ns (max) and 1.5ns (min).

```
# input delays
set_input_delay -clock [get_clocks clk_core] \
-max 2.5 \
[get_ports {data_bus_in_clk_core*}]

set_input_delay -clock [get_clocks clk_core] \
-min 1.0 \
[get_ports {data_bus_in_clk_core*}]

# output delays
set_output_delay -clock [get_clocks clk_core]\
-max 3.0 \
[get_ports {data_bus_out_clk_core*}]

set_output_delay -clock [get_clocks clk_core] \
-min 1.5 \
[get_ports {data_bus_out_clk_core*}]
```

Constraint Checker

Microsemi recommends that you validate FDC or timing constraints after you import or create them. This is especially important if the timing constraints file is imported.

Synplify Pro provides a constraint checker utility that you can use to validate the SDC timing constraints. The constraint checker is accessible from the project menu (**Run > Constraint Check**) inside the Synplify Pro GUI. It generates a constraint check report (*_cck.rpt) with details about any issues with timing constraints. The summary section should indicate that no issues were found with the timing constraints.

Use the constraint checker report to fix mistakes related to incorrect syntax or object names.

For details about the Synplify Pro Constraint Checker, refer to the Synplify Pro for Microsemi User Guide.

Constraints for Optimizing Your Design

Once timing constraints are checked, Microsemi recommends that you use the timing analysis feature in Synplify Pro to determine if all the required design constraints have been provided. You can use the list of violating design paths in the timing report to identify any missing or inaccurate timing constraints.

Note: Since the design is not yet placed, the timing report uses estimates based on wire load models for net delays. This is the reason that timing violations at this stage may or may not appear after place and route.

Microsemi recommends that you go through one pass of the entire design flow including Timing Driven Place and Route before adding timing exceptions for synthesis. You can then use the more accurate post place and route timing analysis report to determine required constraints.

Clock, Input and Output Delay constraints are the minimum set of required timing constraints for all designs. Some designs may require additional timing constraints known as timing exceptions. For example:

- False Paths (set_false_path),
- Multicycle Paths (set_multicycle_path)
- Maximum Path Delay (set_max_delay)

You can use timing exceptions to identify design paths that require the default single cycle timing relationships to be overridden. You must guide the synthesis tool optimizations by identifying design paths that:

- Do not have a timing relationship (set_false_path)
- Have a timing relationship that is not a single cycle (set_multicycle_path or set_max_delay)

Precedence

To resolve timing constraint conflicts when multiple timing exceptions are applied to the same design object, the following precedence rules apply:

set_disable_timing takes precedence over all other timing exception constraints.

False Path constraint takes precedence over Maximum Path Delay/Minimum Path Delay or Multicycle Path constraint.

Maximum Path Delay/Minimum Path Delay constraint takes precedence over Multicycle Path constraint.

***Table 2-1* • Object Access Commands Supported by SmartTime**

Timing Exception	Precedence Order
set_disable_timing	1
set_false_path	2
set_maximum_delay / set_minimum_delay	3
set_multicycle_path	4

Optimizing for Timing Versus Area

When you run Synplify Pro synthesis, the tool first compiles the design and then maps it to the Microsemi technology cells.

By default, Synplify Pro automatically makes efficient trade-offs between area and timing performance to achieve the best results. However, you can guide Synplify Pro to optimize the design for timing performance at the expense of area. Conversely, you can guide Synplify Pro to optimize the design for area at the expense of timing performance.

Generally speaking, optimizing for timing performance consumes more FPGA resources (area) and optimizing for area often means larger delays (weaker timing performance). You must weigh your timing performance needs against your area needs to determine what works best for your design.

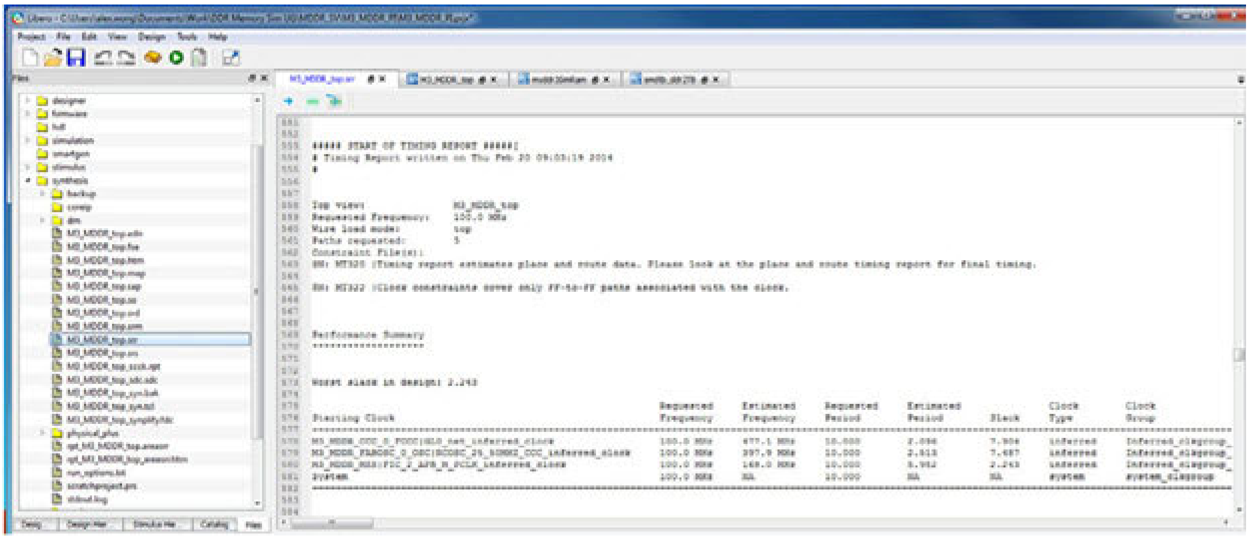
Refer to Chapter 10 of the Synplify Pro for Microsemi User Guide for more information on optimization options.

Post-Synthesis Timing Analysis with Synplify Pro

Synplify Pro generates a timing report after synthesis is complete. After running synthesis, click the **View Log** button to open the log file in Synplify Pro.

The synthesis log file is also available from Libero SoC, under Synthesize in the Reports pane.

The file is located under the synthesis directory with the *.srr extension and viewable in Libero SoC. Click the **File** tab in your Libero SoC Project. Expand the Synthesis file group. Double-click the *.srr file to open it in the Libero SoC Editor View pane. Scroll down to the section entitled START OF TIMING REPORT (Figure 2-2).



```
##### START OF TIMING REPORT #####[
# Timing Report written on Thu Feb 20 09:03:19 2014
#


Top view:               M3_MDDR_top
Requested Frequency:    100.0 MHz
Wire load mode:         top
Paths requested:        5
Constraint File(s):
@N: MT320 |Timing report estimates place and route data. Please look at the place and route timing report for final timing.

@N: MT322 |Clock constraints cover only FF-to-FF paths associated with the clock.



Performance Summary
*******************


Worst slack in design: 2.243

                                                                   Requested     Estimated     Requested     Estimated                Clock        Clock
Starting Clock                                                     Frequency     Frequency     Period        Period        Slack      Type         Group
==============================================================================================================================================================
M3_MDDR_CCC_0_FCCC|GL0_net_inferred_clock                          100.0 MHz     477.1 MHz     10.000        2.096         7.904      inferred     Inferred_clkgroup_
M3_MDDR_FABOSC_0_OSC|RCOSC_25_50MHZ_CCC_inferred_clock             100.0 MHz     397.9 MHz     10.000        2.513         7.487      inferred     Inferred_clkgroup_
M3_MDDR_MSS:FIC_2_APB_M_PCLK_inferred_clock                        100.0 MHz     168.0 MHz     10.000        5.952         2.243      inferred     Inferred_clkgroup_
System                                                             100.0 MHz     NA            10.000        NA            NA         system       system_clkgroup
==============================================================================================================================================================
```

***Figure 2-2* • Synplify Pro *.srr File Open in Libero SoC**

The Synplify Pro timing report is broken into the following sections:

- Performance Summary
- Clock Relationships
- Interface Information
- Detailed Report for Clocks

Use the synthesis timing report to confirm:

- Constraints are being picked up and applied as expected.
- The design does not have any significant timing violations

Since the design is not yet placed, the synthesis timing report estimates net delays using wire load models. However, the cell delays used in the timing report are accurate.

A setup timing violation can be considered significant, if the path delay excluding the net delay exceeds the required time. This is usually an indication that either the timing requirement is unrealistic or the design path requires additional pipelining. In either case, it is highly unlikely that a design path with cell delays exceeding required time will meet the timing goal after place and route.

Timing Exceptions

If the post-synthesis timing analysis reports that the design does not meet timing specifications for clock speed or I/O delays, Microsemi recommends that you use timing exceptions to help synthesis.

Microsemi recommends that you go through one pass of the entire design flow, including Timing Driven Place and Route, before adding timing exceptions for synthesis. You can then use the more accurate post place and route timing analysis report to determine required constraints.

Use false path timing constraints to identify specific design paths that do not propagate logic level changes and should not be considered during timing analysis. The synthesis tool ignores design paths identified using this constraint for logic and mapping optimizations.

Use Multicycle Path, False Path and Maximum Path Delay timing constraints to identify design paths that have a timing relationship different from the default single cycle relationship. The synthesis tool uses the new relationship for optimizations.

Multicycle Path and False Path constraints typically result in relaxing the original single clock cycle timing requirement. The Maximum Path Delay constraint can result in relaxing or tightening the original timing requirement based on the time value specified by the user.

FDC Examples

```
# False Path
set_false_path -from [get_ports uart_ctrl]

# Maximum Path Delay
set_max_delay -to [get_ports {ram_rd_enable}] 4.0

# Multicycle Path
set_multicycle_path 4 -to [get_ports {I2C*}]
```

SCOPE and Using the Forward Annotated SDC

You can use SCOPE from Synplify Pro to enter constraints for the synthesis step. After synthesis, Synplify Pro generates a *_sdc.sdc file, which contains the forward annotated timing constraints. Refer to "Using the Forward Annotated SDC" on page 16 for details. The recommended flow is to create separate constraints for Synthesis and Timing-Driven Place and Route steps.

Timing Constraint Entry Using SCOPE

SCOPE is an editor provided with Synplify Pro to enter and manage timing constraints and synthesis attributes (Figure 2-3).

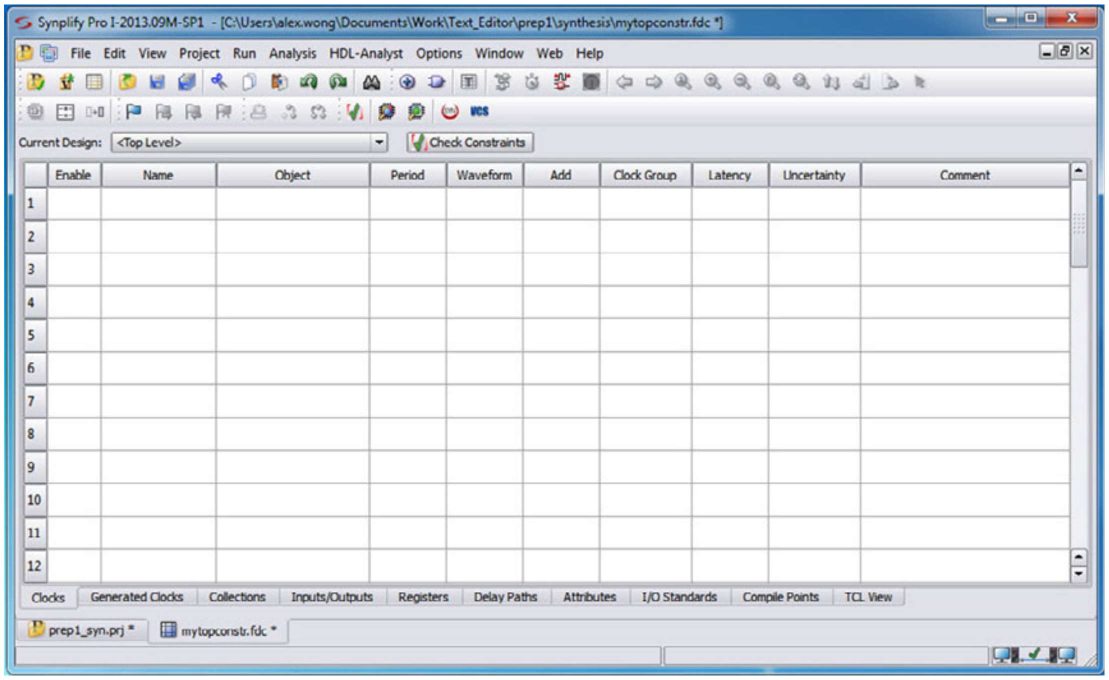


Figure 2-3 • **SCOPE Constraints Editor**

The three types of constraints common to most designs are:

- **Clock Constraints** - All synchronous designs are driven by some kind of clocks. Timing violations such as setup and hold time violations are meaningless without a clock constraint.
- **Input and Output Delay Constraints** - The Input and Output delays makes allowance for path delays external to the FPGA. These constraints are part of the delay budgeting requirement.
- **Exceptions** - When a design fails to meet timing requirements, you may need more advanced constraints to bring the design to timing closure. When applied to specific paths in the design, these timing exceptions override the default behavior of Synplify Pro when these timing paths are considered during optimization.

Before entering any constraints, first compile the design in Synplify Pro (**Run > Compile Only**). Performing compile inside Synplify Pro pre-populates SCOPE with object names. This could save you time and effort entering object names.

Prefix Identifiers Supported by SCOPE and FDC

Synplify Pro uses object identifiers defined in the following table for constraints defined in SCOPE or FDC files (Table 2-2).

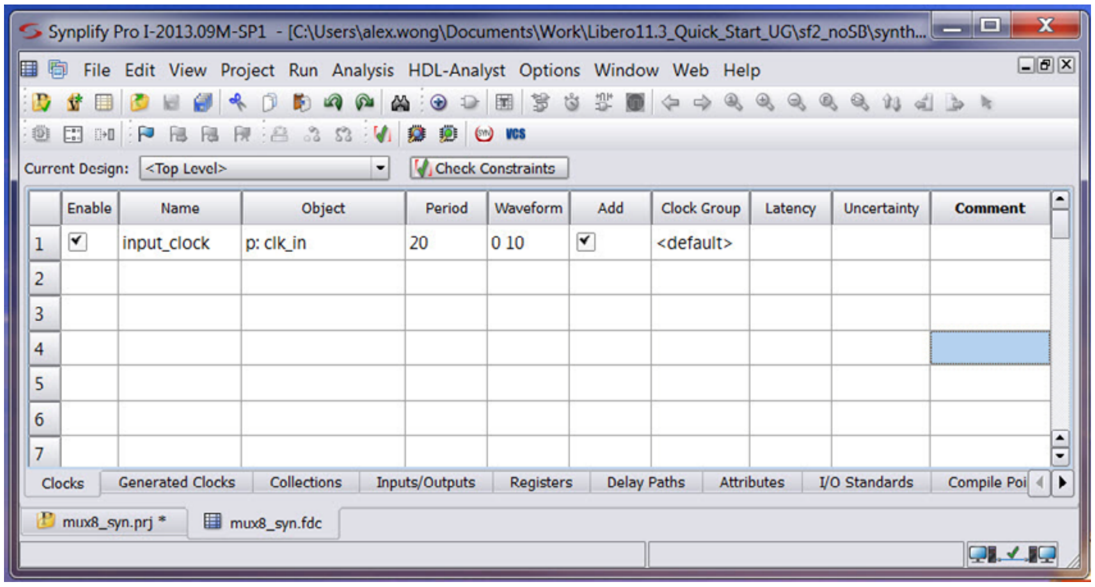
***Table 2-2* • Prefix Identifiers and Design Objects**

Prefix Identifier (FDC)	**SDC Equivalent**	**Design Objects to be applied**
v: *design_name*	n/a	Modules
c: *clock_name*	get_clocks	Clocks
i: *instance_name*	get_cells	Instances
p: *port_name*	get_ports	Ports
t: *pin_name*	get_pins	Hierarchical ports or pins of instantiated cells
b: *bus_name*	get_pins	Bits of a bus
n: *net_name*	get_nets	Net names

Clock Constraints

There are two types of clock constraints: create_clock and create_generated_clock. SCOPE has two separate tabs for these two constraints.

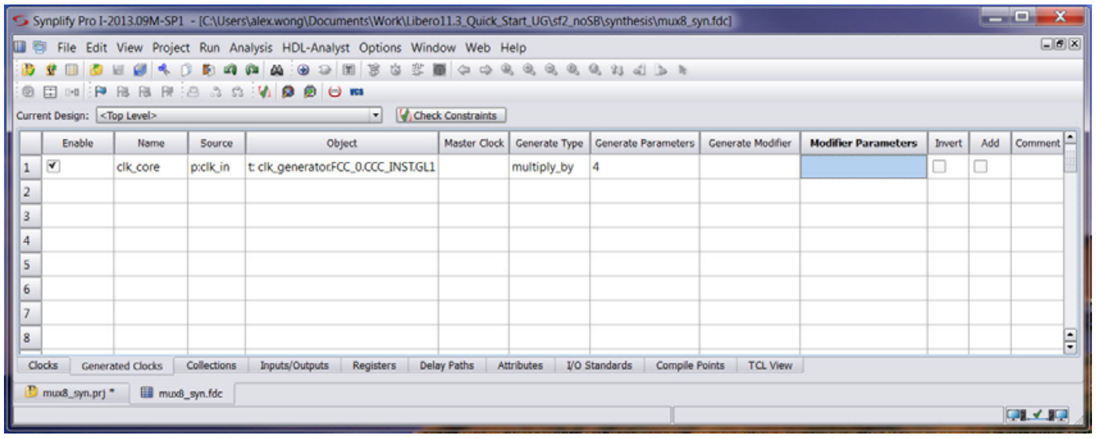
Use the Clocks tab to identify the clock sources (create_clock). Refer to Figure 2-4.


Synplify Pro I-2013.09M-SP1 - [C:\Users\alex.wong\Documents\Work\Libero11.3_Quick_Start_UG\sf2_noSB\synth...
File Edit View Project Run Analysis HDL-Analyst Options Window Web Help
Current Design: <Top Level>
Check Constraints
Clocks Generated Clocks Collections Inputs/Outputs Registers Delay Paths Attributes I/O Standards Compile Poi
mux8_syn.prj *
mux8_syn.fdc

	Enable	Name	Object	Period	Waveform	Add	Clock Group	Latency	Uncertainty	Comment
1	✓	input_clock	p: clk_in	20	0 10	✓	<default>			
2										
3										
4										
5										
6										
7										

***Figure 2-4* • Clocks Tab**

Use the Generated Clocks tab to identify the generated clocks (create_generated_clock) in the design. Refer to Figure 2-5.


Synplify Pro I-2013.09M-SP1 - [C:\Users\alex.wong\Documents\Work\Libero11.3_Quick_Start_UG\sf2_noSB\synthesis\mux8_syn.fdc]
File Edit View Project Run Analysis HDL-Analyst Options Window Web Help
Current Design: <Top Level>
Check Constraints
Clocks
Generated Clocks
Collections
Inputs/Outputs
Registers
Delay Paths
Attributes
I/O Standards
Compile Points
TCL View
mux8_syn.prj *
mux8_syn.fdc

	Enable	Name	Source	Object	Master Clock	Generate Type	Generate Parameters	Generate Modifier	Modifier Parameters	Invert	Add	Comment
1	✓	clk_core	p:clk_in	t: clk_generator.FCC_0.CCC_INST.GL1		multiply_by	4			☐	☐	
2												
3												
4												
5												
6												
7												
8												

***Figure 2-5* • Generated Clocks Tab**

For RTG4 designs:

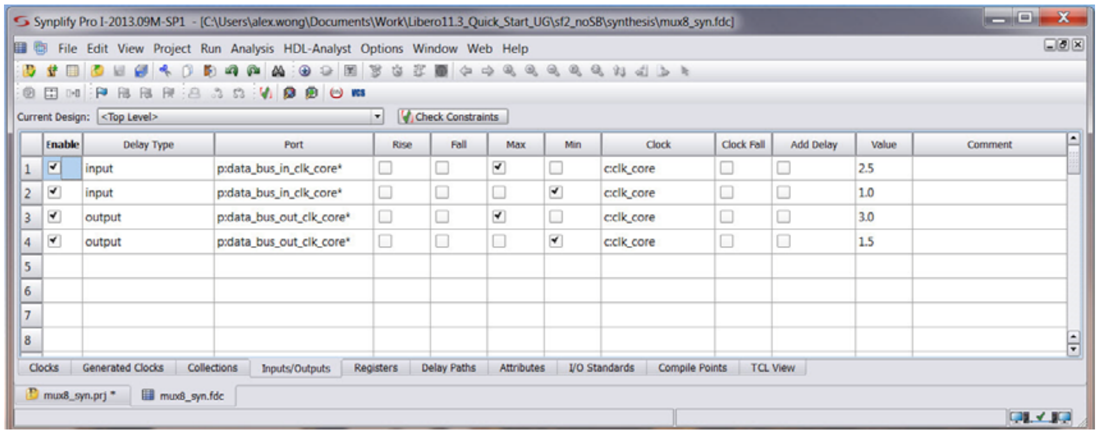
- Oscillators or Clock Input ports are typical clock sources.
- Fabric CCC outputs are typical generated clocks.

Synplify Pro supports an advanced copy and paste feature. It is possible to copy objects from the schematic views and paste their names into SCOPE.

For example: Highlight the output terminal pin of the oscillator (in the RTL view schematic), copy (**CTRL + C**) and then paste (**CTRL + V**) into SCOPE.

I/O Delay Constraints

Clock constraints are not sufficient to constrain I/O ports. Use input and output delay constraints (set_input_delay, set_output_delay) constraints to specify I/O constraints. Navigate to the Inputs/Outputs tab in SCOPE to do so (Figure 2-6).


Synplify Pro I-2013.09M-SP1 - [C:\Users\alex.wong\Documents\Work\Libero11.3_Quick_Start_UG\sf2_noSB\synthesis\mux8_syn.fdc]
File Edit View Project Run Analysis HDL-Analyst Options Window Web Help
Current Design: <Top Level>
Check Constraints
Clocks
Generated Clocks
Collections
Inputs/Outputs
Registers
Delay Paths
Attributes
I/O Standards
Compile Points
TCL View
mux8_syn.prj *
mux8_syn.fdc

	Enable	Delay Type	Port	Rise	Fall	Max	Min	Clock	Clock Fall	Add Delay	Value	Comment
1	✓	input	p:data_bus_in_clk_core*	☐	☐	✓	☐	c:clk_core	☐	☐	2.5	
2	✓	input	p:data_bus_in_clk_core*	☐	☐	☐	✓	c:clk_core	☐	☐	1.0	
3	✓	output	p:data_bus_out_clk_core*	☐	☐	✓	☐	c:clk_core	☐	☐	3.0	
4	✓	output	p:data_bus_out_clk_core*	☐	☐	☐	✓	c:clk_core	☐	☐	1.5	
5												
6												
7												
8												

***Figure 2-6* • Inputs/Outputs Tab**

Constraint Checker

To check the constraints entered so far, click on the **Check Constraints** button on the menu bar. Synplify Pro generates a clock constraint check report (*_cck.rpt). The summary should indicate that no issues are found with the timing constraints.

Constraint checker is also accessible from the Project menu (**Project > Run > Constraint Check**).

Exceptions

Some designs require additional timing constraints known as timing exceptions, such as:

- set_false_path
- set_max_delay
- set_multicycle_path

To enter timing exceptions in SCOPE navigate to the Delay Paths tab. After saving the file, from the **File** menu choose **Close** to return to the Project view.

Using the Forward Annotated SDC

After synthesis, Synplify Pro generates a SDC file that you can use for the remaining steps in the Libero flow. This file (with extension *_sdc.sdc) is available under Timing Constraints in the Libero Project view.

By default, this file is not used by Libero. The recommended flow is to create separate constraints for Synthesis and Timing-Driven Place and Route steps.

Users must review the file contents of the Synplify generated forward annotated file before enabling the file for implementation. To enable a SDC file for Timing-Driven Place and Route, right-click the file and choose **Use for Compile**. Libero uses this SDC file for the remaining steps in design flow, starting from Compile.

Limitations

- The forward annotated SDC file from Synplify Pro does not include any set_clock_latency constraints present in the original user SDC file.
- Synplify Pro does not automatically generate clock constraints for oscillators or CCC instances. You must supply accurate clock constraints (create_clock, create_generated_clock) to Synplify Pro. These constraints are then forward annotated in the *_sdc.sdc file.

Timing Constraints for Timing-Driven Place and Route

Libero tools (Timing Driven Place and Route and SmartTime) support a subset of Synopsys SDC timing constraints relevant for FPGA designs. To set timing constraints, you can use:

- SmartTime Constraint Wizard
- SmartTime Constraint Editor
- SDC file(s)

To organize the timing constraint files, use the "Organize Constraint Files" dialog box. When importing a SDC file for Timing Driven Place and Route, make sure to include it for use by the Compile tool. To enable a SDC file already imported in Libero, right-click on the file and choose **Use for Compile** (refer to Figure 2-7).

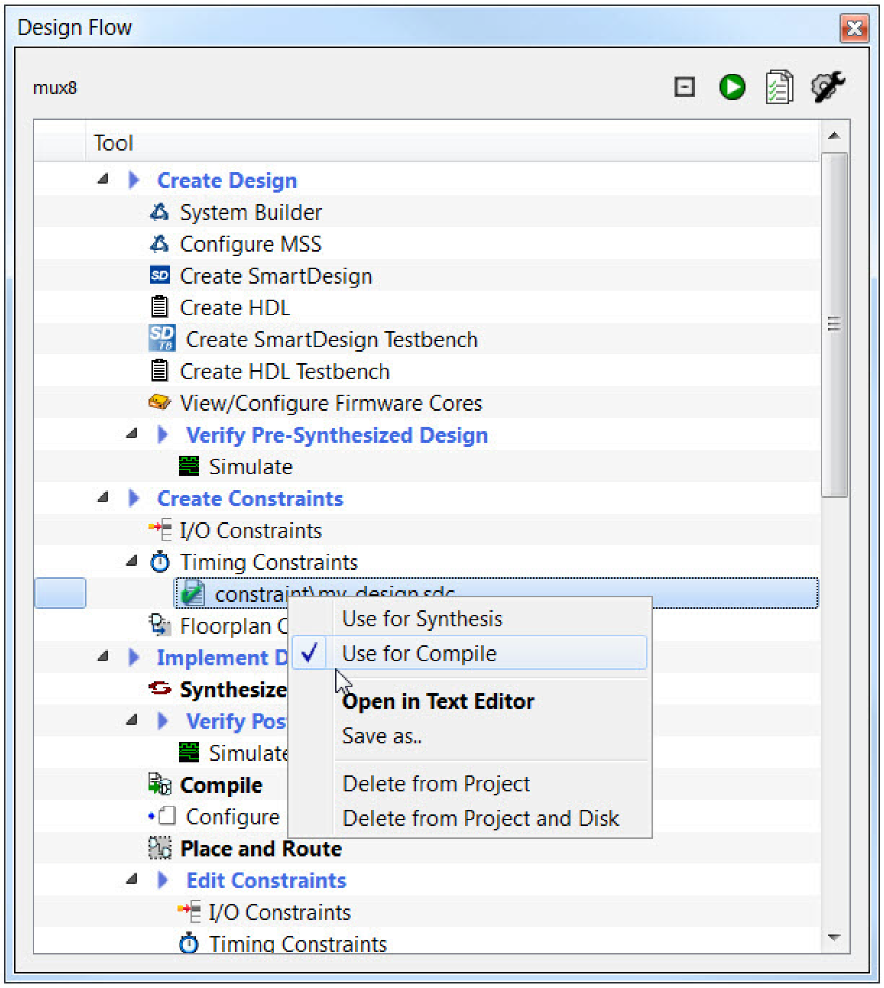


Figure 2-7 • **SDC File for Compile**

Timing-Driven Place and Route Constraints

SmartTime Timing Analysis supports the following set of SDC timing constraints:

- create_clock
- create_generated_clock
- set_input_delay
- set_output_delay
- set_external_clock
- set_clock_to_output
- set_false_path
- set_multicycle_path
- set_max_delay
- set_min_delay
- set_clock_latency
- set_clock_uncertainty
- set_disable_timing

For details on the options and arguments of the SDC commands, refer to the SmartTime online help.

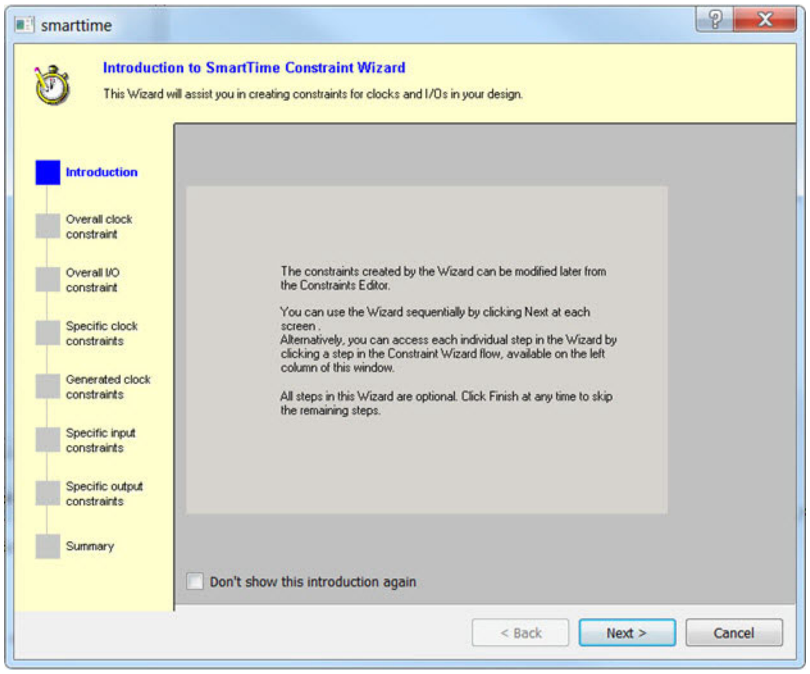
Constraints for Design Requirements

Microsemi recommends that you use the following flow to enter timing constraints:

1. SmartTime Constraint Wizard - Identify clocks, input and output delay constraints
2. I/O Attributes Editor - Provide complete I/O attributes information for the design
3. Generate and analyze the Constraints Coverage report

SmartTime Constraint Wizard

SmartTime includes a Constraint Wizard that enables quick and easy entry for clock and I/O constraints (Figure 2-8).


smarttime
Introduction to SmartTime Constraint Wizard
This Wizard will assist you in creating constraints for clocks and I/Os in your design.
Introduction
Overall clock constraint
Overall I/O constraint
Specific clock constraints
Generated clock constraints
Specific input constraints
Specific output constraints
Summary
Don't show this introduction again
< Back
Next >
Cancel

The constraints created by the Wizard can be modified later from the Constraints Editor.

You can use the Wizard sequentially by clicking Next at each screen .
Alternatively, you can access each individual step in the Wizard by clicking a step in the Constraint Wizard flow, available on the left column of this window.

All steps in this Wizard are optional. Click Finish at any time to skip the remaining steps.

Figure 2-8 • **SmartTime Constraint Wizard**

Invoke the Constraint Wizard from SmartTime (**Tool > Constraint Wizard**).

The Constraint Wizard enables you to add constraints in the following order:

1. Overall clock constraint
2. Overall I/O constraint
3. Specific clock constraints
4. Generated clock constraints
5. Specific input constraints
6. Specific output constraints

Use the Overall constraint tabs to set default constraints for clocks and I/Os.

The default constraints can be overridden by the Specific constraints for clocks and I/Os.

Clock Constraints

Use the Specific clock and Generated clock constraint tabs for:

- Oscillators used as clock sources.
- Fabric CCC outputs used as generated clocks

- Clocks from other sources

I/O Constraints

Use the Overall I/O constraint tab to set default constraints for all input and output ports in the design. The default I/O constraints can be overridden in the Specific input and Specific output constraint tabs for selected ports.

For details about the Constraint Wizard, refer to the SmartTime online help.

I/O Attributes and the I/O Attribute Editor

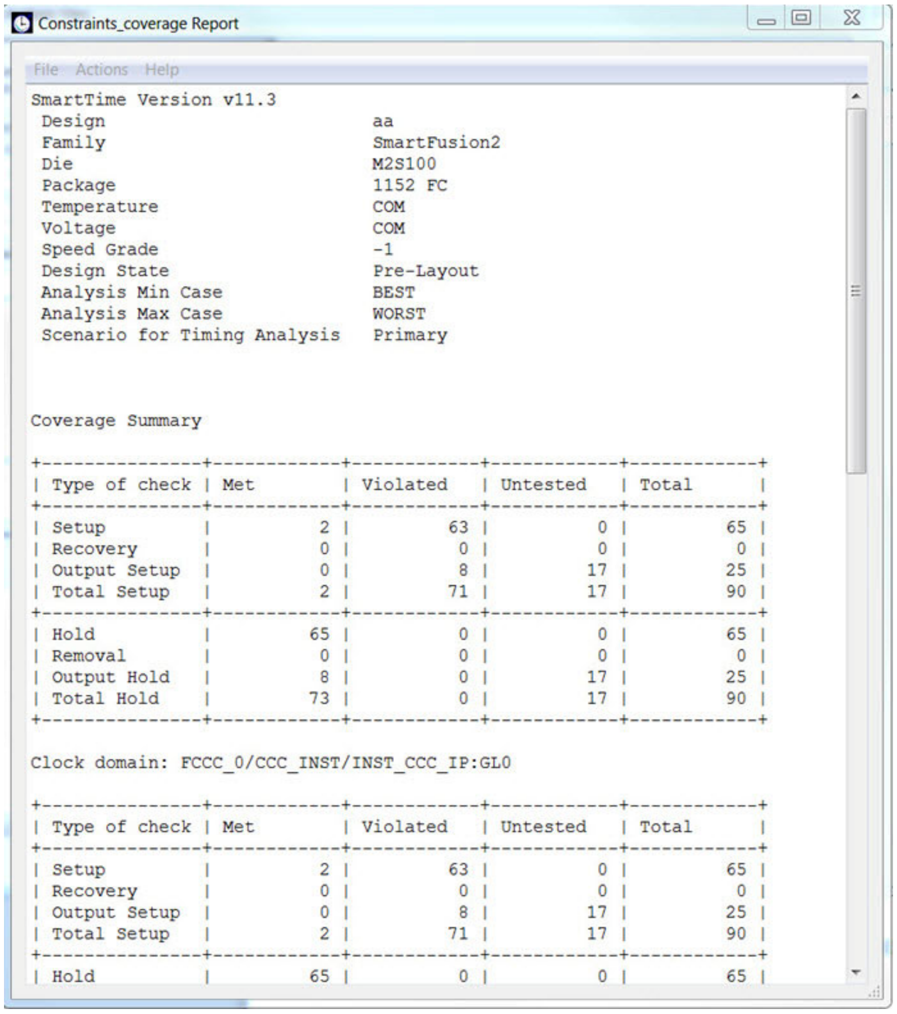
Timing performance of I/O paths is significantly influenced by the I/O attributes.

Use the I/O Attribute Editor feature in the MultiView Navigator (MVN) tool to provide complete I/O attribute information for the design.

For details about the I/O Attribute Editor, refer to the MultiView Navigator online help.

Constraint Coverage

It is important to generate a Constraints Coverage Report (Figure 2-9), because the timing report only analyzes timing performance for design paths with timing constraints. Timing paths without timing constraints may have timing violations and are not be reported. Invoke the Constraint Coverage report from SmartTime Analyzer (**Tools > Reports > Constraint Coverage**).



```
SmartTime Version v11.3
 Design                          aa
 Family                          SmartFusion2
 Die                             M2S100
 Package                         1152 FC
 Temperature                     COM
 Voltage                         COM
 Speed Grade                     -1
 Design State                    Pre-Layout
 Analysis Min Case               BEST
 Analysis Max Case               WORST
 Scenario for Timing Analysis    Primary


Coverage Summary

+---------------+------------+------------+------------+------------+
| Type of check | Met        | Violated   | Untested   | Total      |
+---------------+------------+------------+------------+------------+
| Setup         |          2 |         63 |          0 |         65 |
| Recovery      |          0 |          0 |          0 |          0 |
| Output Setup  |          0 |          8 |         17 |         25 |
| Total Setup   |          2 |         71 |         17 |         90 |
+---------------+------------+------------+------------+------------+
| Hold          |         65 |          0 |          0 |         65 |
| Removal       |          0 |          0 |          0 |          0 |
| Output Hold   |          8 |          0 |         17 |         25 |
| Total Hold    |         73 |          0 |         17 |         90 |
+---------------+------------+------------+------------+------------+

Clock domain: FCCC_0/CCC_INST/INST_CCC_IP:GL0

+---------------+------------+------------+------------+------------+
| Type of check | Met        | Violated   | Untested   | Total      |
+---------------+------------+------------+------------+------------+
| Setup         |          2 |         63 |          0 |         65 |
| Recovery      |          0 |          0 |          0 |          0 |
| Output Setup  |          0 |          8 |         17 |         25 |
| Total Setup   |          2 |         71 |         17 |         90 |
+---------------+------------+------------+------------+------------+
| Hold          |         65 |          0 |          0 |         65 |
```

***Figure 2-9* • Constraint Coverage Report**

Design paths or objects with missing constraints are listed under Enhancement Suggestions. Review each suggestion and supply appropriate constraints to ensure that all design paths have timing constraints.

For details about the Constraint Coverage Report, refer to the SmartTime online help.

Constraints for Optimizing Your Design

Design timing constraints may need to be optimized if the design fails to meet timing requirements, even after completing Timing Driven Place and Route (TPDR).

The recommended flow for optimizing design constraints is:

1. Run Timing Driven Place and Route. Ensure that the Timing-driven option is enabled during Place and Route.
2. Generate and Inspect Timing Analysis reports. Analyze both the Maximum and Minimum Delay Analysis reports.
3. Open SmartTime Constraints Editor and provide additional constraints, including timing exceptions.
4. To improve Placer Performance:
   – Debug design paths with timing violations.
   – Use set_max_delay to constrain inter-clock domain paths.

Using Timing Driven Place and Route (TDPR)

The primary goal of TDPR is to meet timing constraints. If you do not select the Timing-driven option, Place and Route will not consider timing constraints.

Ensure that Timing-driven is selected before running Place and Route (right-click **Place and Route** and choose **Configure Options** in the Libero tool suite). This option is selected by default for RTG4.

Timing Analysis Reports

SmartTime generates two types of timing reports by default for both Max and Min Delay analysis:

- Timing report - This report displays the timing information organized by clock domain.
- Timing violations report - This flat slack report provides information about constraint violations. To generate timing analysis reports in Libero, right-click **Verify Timing** and choose **Run**.

Timing Report Contents

The timing report contains the following sections:

- Header - lists the report type, version, date and time of report and general design information
- Summary - reports the timing information for each clock domain
- Path Selections - lists the timing information for different types of paths in the design. For details, refer to the SmartTime online help.

Timing Violation Report Contents

The timing violation report contains the following sections:

Header

The Header lists:

- Report type
- Version of SmartTime used to generate the report
- Date and time the report was generated
- General design information (name, family, etc.)

Paths

The paths section lists the timing information for the violated paths in the design.

By default, the slack threshold is 0 and the number of paths is limited. The default maximum number of paths reported is 100.

All clocks domains are mixed in this report. The paths are listed by decreasing slack.

SmartTime Constraints Editor

The SmartTime Constraints Editor is a tool that enables you to create, view and edit all design timing constraints. Constraints supplied through the constraints wizard or SDC files are available for editing in the SmartTime Constraints Editor.

Use the Constraints Wizard to easily provide basic timing constraints for clocks and I/O ports. For advanced timing constraints such as timing exceptions use the Constraints Editor.

Timing Exceptions

Based on the complexity of the design, timing exceptions may be required. Timing exceptions are timing constraints set on specific paths in the design. For example:

- set_false_path
- set_max_delay
- set_multicycle_path

Providing these constraints requires knowledge of the data paths in the design and their timing requirements. By default, SmartTime uses a single clock cycle to analyze any timing path that has a clock constraint set on it. Timing exceptions are used to override the default clock constraint for the design path.

For details about the Timing Exceptions, refer to the SmartTime online help.

Note: Based on the severity of timing violations, it may also be necessary to provide timing exception constraints to the synthesis software. To provide timing exception constraints to the synthesis software, include these constraints in the FDC file being provided to Synplify Pro.

Improving Placer Performance

When the design fails to meet the timing goals, the failing design paths must be analyzed carefully. Two issues need to be analyzed:

- Can the timing performance of the failing path(s) be improved if instance placement was modified?

  Long route delays for design paths with setup violations may indicate that the instance placement was not optimal. The design path placement can be examined using the "Chip Planner" tool, which is part of the MultiView Navigator.
- Are the timing constraints sufficient for the placer to identify and work on the true critical paths in the design?

  Ensure that a complete set of timing constraints is created and passed to the placer tool. Use the set_max_delay constraint to properly constrain inter-clock domain design paths. The following sections have more details on passing constraints and using set_max_delay constraints.

Placer Performance - Supported Constraints

You can pass timing constraints to the Placer by:

- Organizing the timing constraint files using the Organize Constraint Files dialog box. When importing a SDC file for the placer, make sure to include it for use by the Compile tool.

  To enable a SDC file already imported into Libero, right-click the file and choose **Use for Compile**.
- Entering constraints in the SmartTime GUI - If using scenarios, ensure that the scenario is enabled for TDPR.

Limitations

- The placer currently supports the following constraints:
  - create_clock
  - create_generated_clock
  - set_clock_latency

- set_input_delay
  - set_output_delay
  - set_max_delay
  - set_false_path
- The following constraints are not supported by the placer:
  - set_clock_uncertainty
  - set_multicycle_path
  - set_min_delay
- The placer does not support inter-clock domain timing. The placer optimizes clocks within their domain, but not between domains. To enable placer optimizations for inter-clock domain paths, use the set_max_delay constraint. This is described in the next section.
- The placer does not include the clock generation path in the arrival/require time calculation when set_max_delay constraint is used.

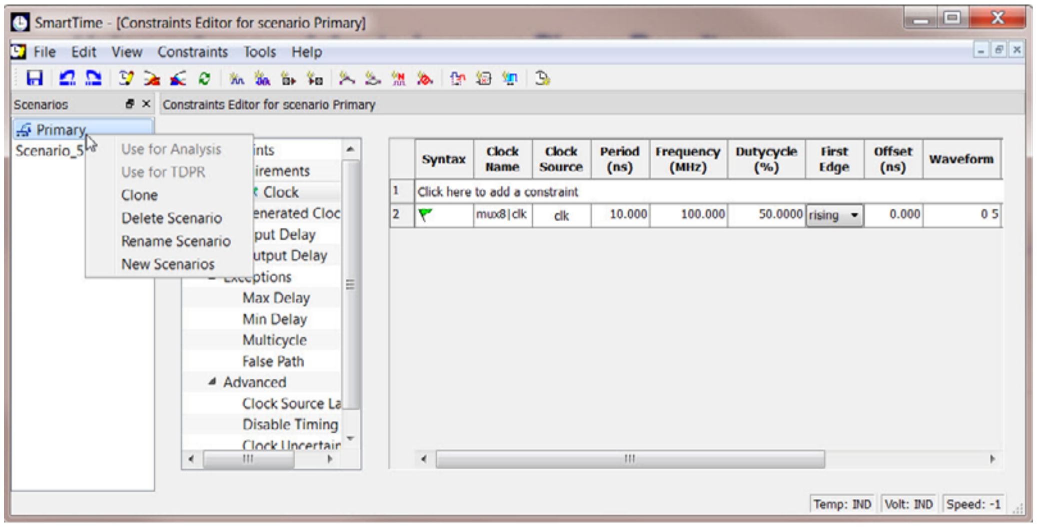
Using set_max_delay to Improve Placer Results

It may be possible to improve placer results by using set_max_delay timing constraint on design paths with timing violations. Consider using this approach if the violating path:

- Is an inter-clock domain path.
- Contains clock generation delays that are significantly different between the start and end points.

To include additional timing constraints for the placer:

1. Clone the existing timing constraints scenario in SmartTime. From the Constraint Editor, right-click Primary scenario and choose **Clone** (Figure 2-10).

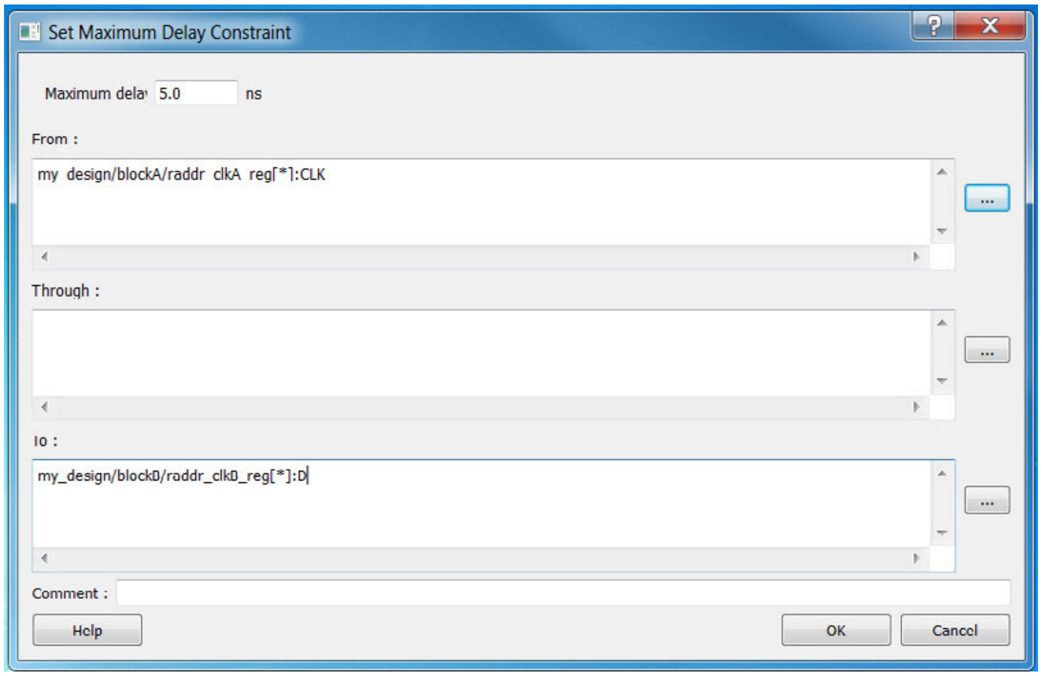


	Syntax	Clock Name	Clock Source	Period (ns)	Frequency (MHz)	Dutycycle (%)	First Edge	Offset (ns)	Waveform
1	Click here to add a constraint								
2		mux8|clk	clk	10.000	100.000	50.0000	rising	0.000	0 5

Figure 2-10 • **Timing Scenario Clones**

2. Retain the original (Primary) constraint set for timing analysis

3. Enter set_max_delay constraints for design paths that cross clock domains (Figure 2-11).



Maximum delay 5.0 ns

From : my_design/blockA/raddr_clkA_reg[*]:CLK

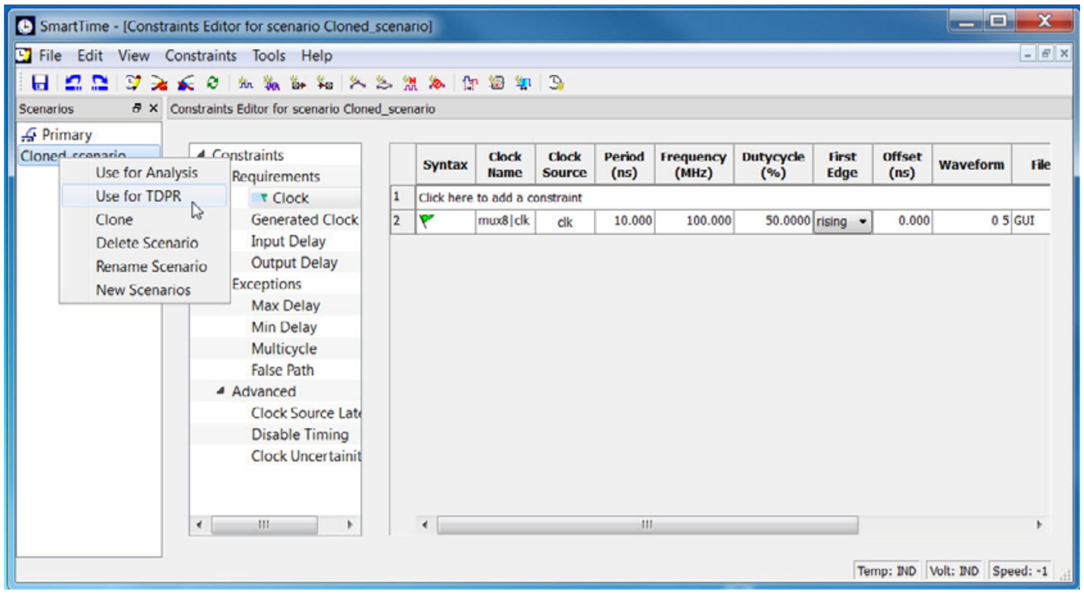
Through :

To : my_design/blockB/raddr_clkB_reg[*]:D

Comment :

Figure 2-11 **• Set_max_delay Constraints in Cloned Scenario**

4. Use the second set of constraints (cloned scenario) exclusively for TDPR. Right-click Cloned scenario and choose **Use for TDPR** (Figure 2-12).



	Syntax	Clock Name	Clock Source	Period (ns)	Frequency (MHz)	Dutycycle (%)	First Edge	Offset (ns)	Waveform	File
1	Click here to add a constraint									
2		mux8|clk	clk	10.000	100.000	50.0000	rising	0.000	0 5	GUI

Figure 2-12 **• Cloned Scenario for TDPR**

For details about the set_max_delay constraint, refer to the SmartTime online help.

3 – Constraints for RTG4 IP Blocks

This chapter describes the constraint requirements for the following blocks:

- Oscillators
- Fabric Clock Conditioning Circuits (CCC)
- High Speed Serial Interface (SERDES)

Oscillators

The oscillator has only one output CLKOUT of 50 MHz. The output can only be connected to the CCC configurator.

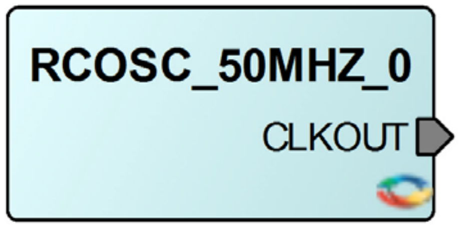


***Figure 3-1* • Oscillator Configurator**

Oscillator Synthesis Constraints

You must specify a clock constraint for the oscillator used by the design.

Figure 3-2 shows the block as seen by synthesis.

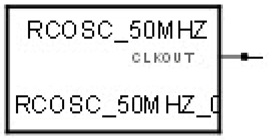


***Figure 3-2* • Synthesis View of an Oscillator Block**

The following constraints will work for an oscillator with RCOSC_50MHZ_0 as instance name. The RC oscillator is configured for 50 MHz.

```
create_clock -name osc_50MHz -period 20 \
               [get_pins { RCOSC_50MHZ_0.CLKOUT}]
```

Oscillator Place and Route Constraints

For SmartTime, the constraints for the oscillator configurator connected to CCC instance RTG4FCCC_0 are:

```
create_clock -name osc_50MHz -period 20 [get_pins {RCOSC_50MHZ_0:CLKOUT}]
```

RTG4 Fabric Clock Conditioning Circuit (CCC)

CCCs are used to multiply, divide or delay clocks. Their effect is best described using generated clocks.

RTG4 Fabric CCC Synthesis Constraints

To create RTG4FCCC constraint for synthesis via generated clock, you need to use RTG4FCCC multiple and divide factors. This information is available in the Advanced tab in the CCC Configurator accessible through the Libero software (Figure 3-3).

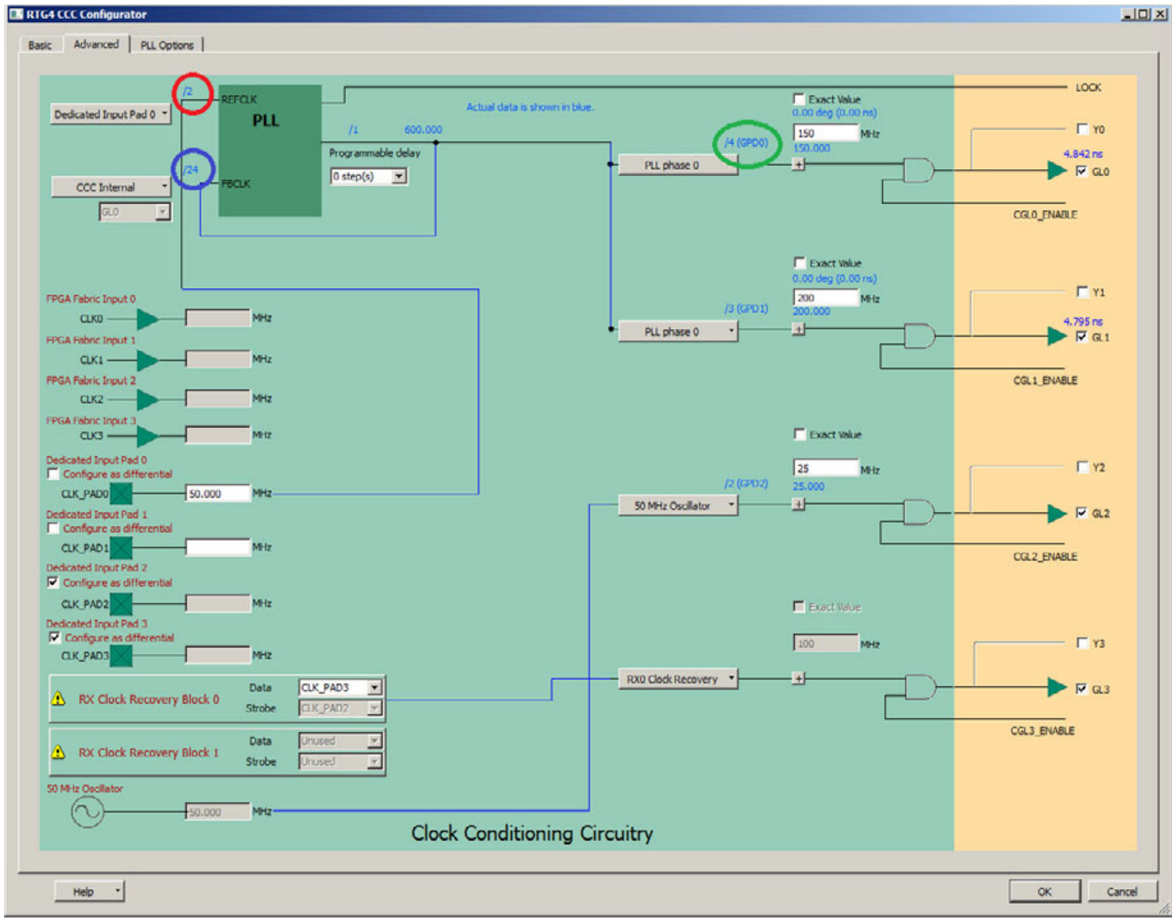


Figure 3-3 • **RTG4 Fabric CCC Configurator (Advanced Tab)**

The CCC configuration shown in Figure 3-3 generates four clocks:

- On GL0, a 150 MHz clock generated from the 50 MHz input clock using the PLL
- On GL1, a 200 MHz generated from the same PLL
- On GL2, a 25 MHz clock generated from the 50 MHz oscillator.
- On GL3, a 100MHz clock generated from the CLK3_PAD and CLK_PAD2.

The exact division and multiplication factors can be calculated based on the divider configurations shown in the configurator. The factors used for GL0 are circled. When the CCC is used, the multiplication factor is given by the feedback divider (circled in blue); the division factor is given by multiplying the reference divider (circled in red) by the output (GPD) divider (circled in green).

The corresponding generated clocks are:

```
create_clock -name CLK0_PAD -period 20 [get_pins {RTG4FCCC_0.CLK0_PAD}]
create_clock -name CLK2_PADP -period 10 [get_pins {RTG4FCCC_0.CLK2_PADP}]
create_clock -name CLK3_PADP -period 10 [get_pins {RTG4FCCC_0.CLK3_PADP}]

create_generated_clock -name clk_150mhz -divide_by 16 -multiply_by 24 \
-source [get_pins {RTG4FCCC_0.CLK0_PAD}] \
[get_pins {RTG4FCCC_0.GL0}]

create_generated_clock -name clk_200mhz -divide_by 12 -multiply_by 24 \
-source [get_pins {RTG4FCCC_0.CLK0_PAD}]\
 [get_pins {RTG4FCCC_0.GL1}]

create_generated_clock -name clk_25mhz -divide_by 2 \
-source [get_pins { RTG4FCCC_0.RCOSC_50MHZ}] \
 [get_pins {RTG4FCCC_0.GL2}]

create_generated_clock -name clk_100mhz -divide_by 1 -multiply_by 1 \
-source [get_pins {RTG4FCCC_0.CLK2_PADP}] \
[get_pins {RTG4FCCC_0.GL3}]
```

RTG4 Fabric CCC Place and Route Constraints

For SmartTime, the constraints for the CCC configurator are:

```
create_clock -name CLK0_PAD  -period 20 [get_ports {CLK0_PAD}]
create_clock -name CLK2_PADP  -period 10 [get_ports {CLK2_PADP}]
create_clock -name CLK3_PADP  -period 10 [get_ports {CLK3_PADP}]

create_generated_clock  -name clk_150mhz \
                        -divide_by 16 \
                        -multiply_by 24 \
                        -source {RTG4FCCC_0/CCC_INST/INST_CCC_IP:CLK0_PAD} \
                        {RTG4FCCC_0/CCC_INST/INST_CCC_IP:GL0}

create_generated_clock  -name clk_200mhz \
                        -divide_by 12 \
                        -multiply_by 24 \
                        -source {RTG4FCCC_0/CCC_INST/INST_CCC_IP:CLK0_PAD} \
                        {RTG4FCCC_0/CCC_INST/INST_CCC_IP:GL1}

create_generated_clock  -name clk_25mhz \
                        -divide_by 2 \
                        -source {RTG4FCCC_0/CCC_INST/INST_CCC_IP:RCOSC_50MHZ} \
                        {RTG4FCCC_0/CCC_INST/INST_CCC_IP:GL2}

create_generated_clock  -name clk_100mhz \
                        -divide_by 1 \
                        -multiply_by 1 \
                        -source {RTG4FCCC_0/CCC_INST/INST_CCC_IP:CLK2_PAD} \
                        {RTG4FCCC_0/CCC_INST/INST_CCC_IP:GL3}
```

The RTG4 FPGA Datasheet (DS0131) CCC specification lists the max clock output period jitter for each CCC output. For example, a 100MHz CCC output clock will have a max output clock jitter equal to: max (80, +/- 1% x 1/Fout_ccc) ps = +/- 100 ps. Similarly, a 50MHz CCC output clock will have +/- 0.01 * 20,000ps = +/- 200 ps of clock jitter.

For these scenarios, and others where a clock signal with a known amount of jitter is used to clock the design, it is important to account for this jitter during Static Timing Analysis in SmartTime.

To tell SmartTime about the existence of short-term peak-to-peak clock jitter, clock source latency constraints should be used. For example, to specify a clock jitter of +/- 100 ps on a clock named 'RTG4FCCC_0/GL0', use the constraints below:

```
set_clock_latency -source -early -0.1 { RTG4FCCC_0/GL0 }
set_clock_latency -source -late 0.1 { RTG4FCCC_0/GL0 }
```

For a peak-to-peak jitter, the `-early` is negative since the jitter is specified as a +/- time value. This makes the `-early` and `-late` constraints different numbers, thus requiring two lines in the SDC to capture the jitter.

The default SmartTime analysis will show the clock source latency applied to the Max Delay setup time analysis. This occurs because the typical setup time check is performed at the next capture clock edge. This means that jitter between the launch edge and capture edge can affect the calculation. For worst-case calculations, the data will be treated as if it were launched late due to the launch clock edge plus positive jitter. In contrast, data capture will occur at the next clock edge which will be assumed as early due to the negative jitter value. The required time calculation will assume the capture clock occurs 1 clock period later minus the negative portion of the jitter constraint.

However, the SmartTime Min Delay hold time analysis will not typically show the clock jitter for a single clock domain if the clock requirement is 0 ns. When the required calculation shows a clock requirement of 0 ns, it means that the data arrival time and the required hold time are calculated relative to the same clock edge, and thus there is no jitter to account for when looking at only 1 clock edge.

Clock source latency constraints will also be factored into inter-clock domain analysis to create the worst-case launch to capture window for the setup and hold checks on the clock domain crossing. Max delay analysis will use late launch and early capture, while min-delay analysis will use early launch and late capture edges for the two respective clock domains.

High Speed Serial Interface (SERDES) Block

The high speed serial interface block or serializer/deserializer interface (SERDESIF) integrates several functional blocks to support multiple high speed serial protocols within the FPGA. The SERDESIF block has the following features:

- Peripheral Component Interconnect express (PCIe-PCI Express®) protocol support
- 10 Gigabit Attachment Unit Interface (XAUI) protocol support
- External Physical Coding Sub-layer (EPCS) interface supports any user defined high speed serial protocol, such as serial Gigabit media independent interface (SGMII) protocol support
- Single or Dual serial protocol modes of operation. In Dual serial protocol modes, two protocols can be implemented on the four physical lanes of the SERDESIF block
- SERDESIF block communications to the FPGA fabric through an AXI/AHBL interface or EPCS interface

PCI Express Protocol Mode

In this mode, the SERDESIF block communicates with the FPGA using the AXI/AHBL interface and the APB3 Interface for configuration; no constraints specific to this block are needed.

EPCS Protocol Mode

In EPCS mode, the SERDESIF can support up to four lanes. In EPCS mode, the SERDESIF block uses three clocks:

- APB_S_PCLK for the APB3 configuration bus
- EPCS_0_RXFWF_RCLK for receive data fly wheel fifo fabric interface
- EPCS_0_TXFWF_WCLK for transmit data fly wheel fifo fabric interface

APB_S_PCLK, EPCS_0_RXFWF_RCLK and EPCS_0_TXFWF_WCLK clock must be defined at their source. No constraints specific to this block are needed.

XAUI Protocol Mode

In XAUI mode, the SERDESIF block uses five clocks:

- REFCLK_P - Input for the TxPLL
- APB_S_CLK for the APB3 configuration bus

- XAUI_MMD_MDC as the MDIO interface clock
- XAUI_RX_CLK_IN - Received data is synchronized in the flywheel FIFO to this clock
- XAUI_TX_CLK_OUT - Transmitted data is sampled with a synchronized clock if clock compensation is enabled. XAUI_TX_CLK_OUT must be connected to XAUI_FB_CLK to enable clock compensation.

REFCLK_P, APB_S_PCLK, XAUI_MMD_MDC and XAUI_RX_CLK_IN clocks must be defined at their source.

XAUI_TX_CLK_OUT clock may be defined on the GL0/GL1 output ports of the SERDESIF block. The example below creates these clocks for both NPSS and PCIE SERDES blocks.

XAUI Synthesis Constraints

NPSS SERDES

```
create_clock  -name { gl_clocks1 } \
              -period 6.400 \
              [get_pins {NPSS_SERDES_IF_0.SERDESIF_INST.GL*}]
```

PCIE SERDES

```
create_clock  -name { gl_clocks2 } \
              -period 6.400 \
              [get_pins {PCIE_SERDES_IF_0.SERDESIF_INST.GL*}]
```

XAUI Place and Route Constraints

NPSS SERDES

```
create_clock  -name { gl_clocks1 } \
              -period 6.400 \
              [get_pins {NPSS_SERDES_IF_0/SERDESIF_INST/INST_NPSS_IP:GL*}]
```

PCIE SERDES

```
create_clock  -name { gl_clocks2 } \
              -period 6.400 \
              [get_pins {PCIE_SERDES_IF_0/SERDESIF_INST/INST_PCIE_IP:GL*}
```