UG0737
User Guide
RTG4 FPGA Timing Constraints (Enhanced Constraint Flow)
Libero SoC v11.8 SP4

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

5-02-00737-3/07.18

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.

The Enhanced Constraint Flow simplifies the management of all constraints including SDC timing constraints. In the Enhanced Constraint Flow, timing constraints need only be entered once, and can be automatically applied to Synplify synthesis, Timing-Driven Layout and Timing Verification. Timing constraints for known hardware blocks and IPs (CCC, Oscillator, and SERDES), can also be derived automatically. Constraints for these blocks are derived based on the selected block configuration, and can easily be applied to Synthesis, Layout, or Timing Verification.

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 8.

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 9.

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

Libero Soc defaults to the use of '/' as a design hierarchy separator and pin separator.

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

Notice that '/' is the hierarchy separator used to indicate that my_pin is a pin of the instance "instance123" which has a hierarchical path of /top_level/blockA/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\[1\]]
- [get_ports {addr_bus_out[1] }]

If you want to specify the constraint on the entire bus, you can simply 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
- set_clock_groups

Refer to "Timing Constraints and Design Flow" on page 10 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_max_delay/set_min_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 adequate and complete timing constraints using the Constraint Manager. Also, you must review the timing reports from Libero SoC and use SmartTime's Static Timing Analysis to ensure that the design has been constrained properly and is meeting the timing goals without timing violations.

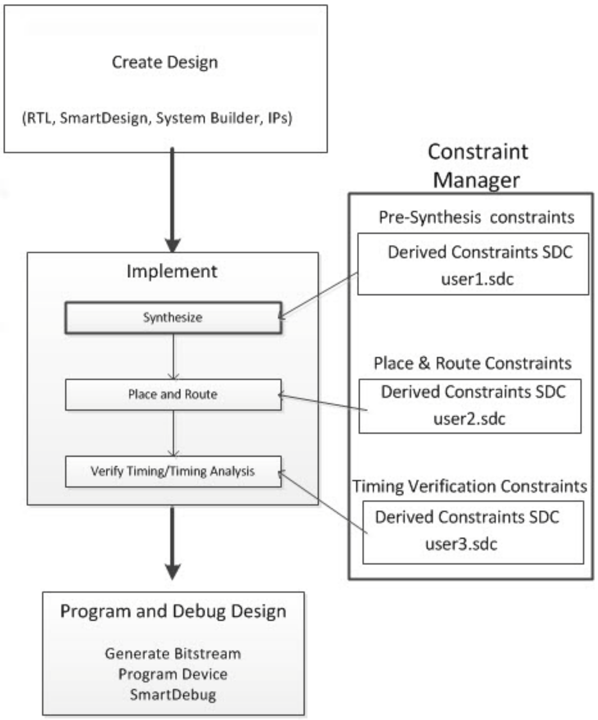


***Figure 2-1* • Timing Constraints in the Design Flow**

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

Microsemi recommends the SDC Timing constraints be used for all tools (SynplifyPro Synthesis, Libero SoC Place and Route and Timing Analysis) to constrain the timing requirements of your design.

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

Libero SoC supports the SDC file format for all timing constraints. The SDC constraints are translated to FDC constraints and passed to Synplify Pro for synthesis.

Creation of SDC Timing Constraints for Synthesis

From the Constraint Manager, create SDC Timing Constraints for Synthesis:

- Click the Timing tab and click New (**Constraint Manager > Timing > New**) to open the Text Editor to enter SDC timing constraints and save them in an SDC file.
- Click the Timing tab and click **Edit > Edit Synthesis Constraint** to open the Constraint Editor GUI to create SDC timing SDC constraints and save them in an SDC file.

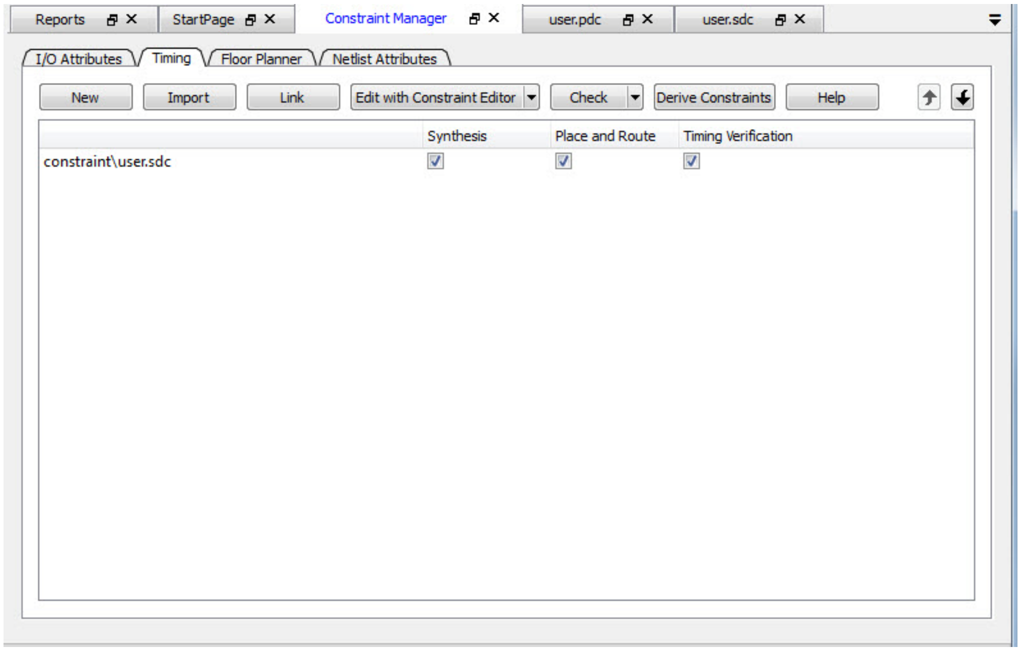
Import of Existing SDC Constraint File For Synthesis

Existing SDC constraint file may be imported (**Constraint Manager > Timing > Import**) to the project location or linked (**Constraint Manager > Timing > Link**) to the project location.

For details, please refer to the Libero SoC Online Help.

Association of SDC Timing Constraint File to Synthesis

The SDC timing constraint files are listed in the Constraint Manager's Timing tab. Check/Uncheck the check box to associate/disassociate the SDC timing constraint file with Synthesis. Only the associated files are passed to synthesis.



	Synthesis	Place and Route	Timing Verification
constraint\user.sdc	☑	☑	☑

Figure 2-2 **• SDC Timing Constraint File Association with Synthesis**

For details about importing timing constraints in the Libero SoC GUI, refer to the Libero online help.

Supported Synplify Pro Timing Constraints

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.

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
- set_clock_groups

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

Derived Constraints

The Libero SoC Enhanced Constraint Flow, introduced in the 11.7 Release, is capable of deriving timing constraints for IP blocks used in the design. These derived SDC constraints are based on the configuration of the IP blocks and the component SDC file(s). The derived SDC constraints are placed in the <root>_derived_constraints.sdc file. It is the top level constraint file that instantiates the SDC constraints of the IP blocks such as the CCC and the 50MHz Oscillator.

Depending on the IP blocks used in the design, The <root>_derived_constraints.sdc file may contain:

- create_clock constraints for the 50 MHz oscillator for output of the oscillator.
- generated_clock constraints for the CCC output such as GL0, GL1 and so on based on the frequency you have configured for these outputs.

To generate the derived constraints for your IP blocks:

- Configure the IP blocks and instantiate them in the top level design.
- Generate the top level design.
- Click Derive Constraints in the Constraint Manager (**Constraint Manager > Timing > Derive Constraints**)
- Click **Yes** to accept the automatic association of the <root>_derived_constraints.sdc with the Synthesis, Place and Route, and Timing Verification tools.

Note: Microsemi recommends the <root>_derived_constraints.sdc be always associated with all three tools: Synthesis, Place and Route, and Verify Timing. Before running Synplify Pro Synthesis, associate the <root>_derived_constraints.sdc file with Synthesis and Place and Route. This will ensure that the design objects (such as nets and cells) in the <root>_derived_constraints.sdc file are preserved during the synthesis step and the subsequent Place and Route step will not error out because of design object mismatches between the post-synthesis netlist and the <root>_derived_constraints.sdc file.

User SDC Constraints for Timing Requirements

The derived_constraints.sdc file contains the SDC timing constraints for the IP blocks only. The user is responsible for additional SDC timing constraints such as clock constraints, input and output delay constraints to meet all off-chip timing budget requirements. Put these additional timing constraints such as create_clock, set_input_delay, and set_output_delay in a user.sdc file to constrain synthesis of the design. Associate this user.sdc to Synthesis in the Constraint Manager.

Order of the SDC Constraints Files

When there are multiple SDC files for Synthesis, the user is responsible for the correct order of the SDC files Libero passes to Synthesis. If an SDC constraint file uses a variable which is defined in another SDC constraint file, use the Up and Down arrow in the Constraint Manager to correctly order the two SDC constraint files.

Avoid timing constraint on the same timing path in multiple SDC files. An SDC constraint file overrides a preceding SDC constraint file.

User SDC Constraints

create_clock

Use the create_clock SDC constraint to define the required clock constraints of 50 MHz.

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

```
# Input Port 'clk_in' @ 50MHz is the clock source
create_clock -name {input_clock} \
-period 20 \
-waveform {0 10} \
[get_ports clk_in]
```

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

set_input_delay and set_output_delay

Use set_input_delay and set_output_delay constraints 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 of the design.

In this example, all constraints use clk_core 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

Libero SoC’s Constraint Manager provides a constraint checker to check the SDC constraint file(s). To invoke the constraint checker, from the Constraint Manager Timing tab, click Check and select Check Synthesis Constraints (**Constraint Manager > Timing > Check > Check Synthesis Constraints**). The SDC timing constraints file(s) associated with Synthesis are checked for the following:

- SDC syntax checks
- Design objects checks - Design objects such as cells and nets in the SDC constraint file are checked against the RTL (for pre-synthesis checks) or post-synthesis netlist (for post-synthesis checks) for any mismatches.

A pop-up window appears with the result of the check.

User SDC Constraints for Design Optimization

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_max_delay / set_min_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.

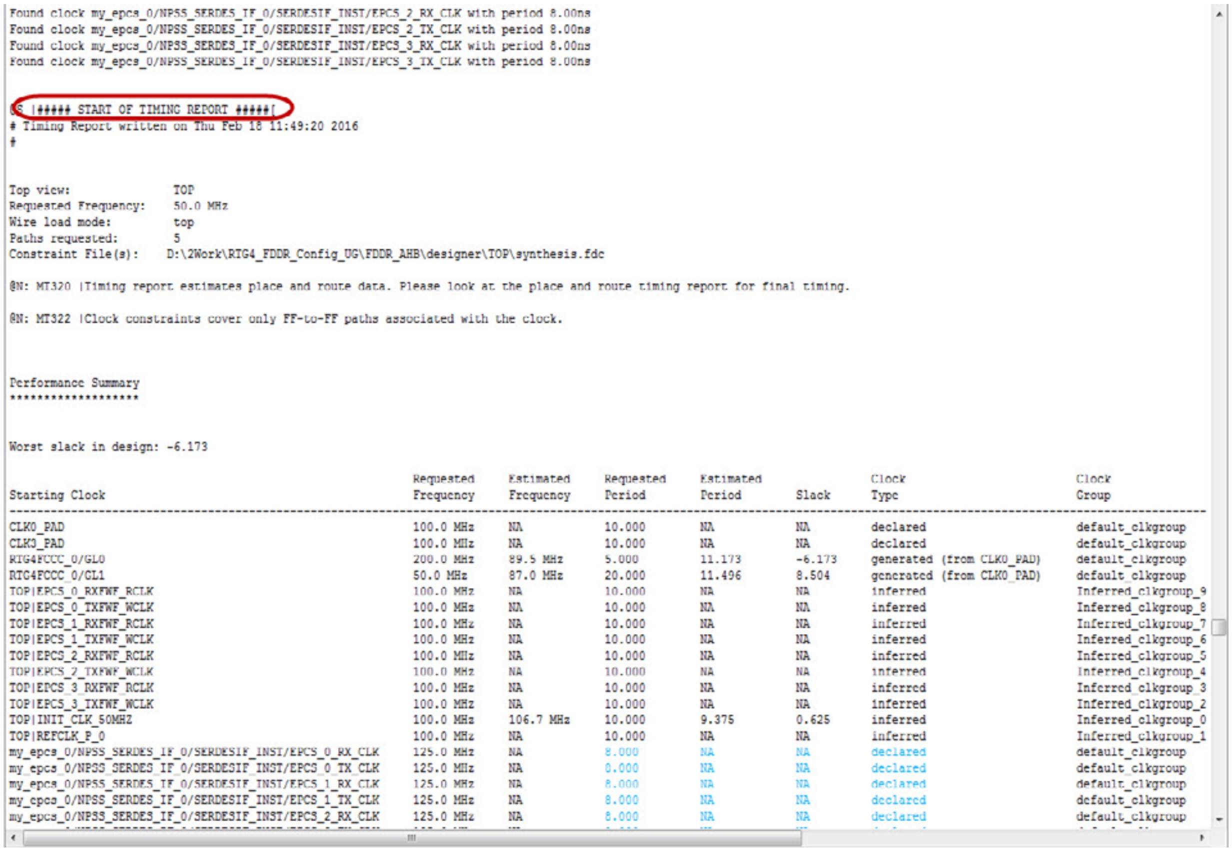
Refer to Chapter 10 of the Synplify Pro for Microsemi User Guide for more information on optimization options.

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 2-3).



```
Found clock my_epcs_0/NPSS_SERDES_IF_0/SERDESIF_INST/EPCS_2_RX_CLK with period 8.00ns
Found clock my_epcs_0/NPSS_SERDES_IF_0/SERDESIF_INST/EPCS_2_TX_CLK with period 8.00ns
Found clock my_epcs_0/NPSS_SERDES_IF_0/SERDESIF_INST/EPCS_3_RX_CLK with period 8.00ns
Found clock my_epcs_0/NPSS_SERDES_IF_0/SERDESIF_INST/EPCS_3_TX_CLK with period 8.00ns


@S |##### START OF TIMING REPORT #####[
# Timing Report written on Thu Feb 18 11:49:20 2016
#


Top view:               TOP
Requested Frequency:    50.0 MHz
Wire load mode:         top
Paths requested:        5
Constraint File(s):    D:\2Work\RTG4_FDDR_Config_UG\FDDR_AHB\designer\TOP\synthesis.fdc

@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: -6.173

                                                        Requested     Estimated     Requested     Estimated                 Clock                            Clock
Starting Clock                                          Frequency     Frequency     Period        Period        Slack       Type                             Group
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
CLK0_PAD                                                100.0 MHz     NA            10.000        NA            NA          declared                         default_clkgroup
CLK3_PAD                                                100.0 MHz     NA            10.000        NA            NA          declared                         default_clkgroup
RTG4FCCC_0/GL0                                          200.0 MHz     89.5 MHz      5.000         11.173        -6.173      generated (from CLK0_PAD)        default_clkgroup
RTG4FCCC_0/GL1                                          50.0 MHz      87.0 MHz      20.000        11.496        8.504       generated (from CLK0_PAD)        default_clkgroup
TOP|EPCS_0_RXFWF_RCLK                                   100.0 MHz     NA            10.000        NA            NA          inferred                         Inferred_clkgroup_9
TOP|EPCS_0_TXFWF_WCLK                                   100.0 MHz     NA            10.000        NA            NA          inferred                         Inferred_clkgroup_8
TOP|EPCS_1_RXFWF_RCLK                                   100.0 MHz     NA            10.000        NA            NA          inferred                         Inferred_clkgroup_7
TOP|EPCS_1_TXFWF_WCLK                                   100.0 MHz     NA            10.000        NA            NA          inferred                         Inferred_clkgroup_6
TOP|EPCS_2_RXFWF_RCLK                                   100.0 MHz     NA            10.000        NA            NA          inferred                         Inferred_clkgroup_5
TOP|EPCS_2_TXFWF_WCLK                                   100.0 MHz     NA            10.000        NA            NA          inferred                         Inferred_clkgroup_4
TOP|EPCS_3_RXFWF_RCLK                                   100.0 MHz     NA            10.000        NA            NA          inferred                         Inferred_clkgroup_3
TOP|EPCS_3_TXFWF_WCLK                                   100.0 MHz     NA            10.000        NA            NA          inferred                         Inferred_clkgroup_2
TOP|INIT_CLK_50MHZ                                      100.0 MHz     106.7 MHz     10.000        9.375         0.625       inferred                         Inferred_clkgroup_0
TOP|REFCLK_P_0                                          100.0 MHz     NA            10.000        NA            NA          inferred                         Inferred_clkgroup_1
my_epcs_0/NPSS_SERDES_IF_0/SERDESIF_INST/EPCS_0_RX_CLK  125.0 MHz     NA            8.000         NA            NA          declared                         default_clkgroup
my_epcs_0/NPSS_SERDES_IF_0/SERDESIF_INST/EPCS_0_TX_CLK  125.0 MHz     NA            0.000         NA            NA          declared                         default_clkgroup
my_epcs_0/NPSS_SERDES_IF_0/SERDESIF_INST/EPCS_1_RX_CLK  125.0 MHz     NA            8.000         NA            NA          declared                         default_clkgroup
my_epcs_0/NPSS_SERDES_IF_0/SERDESIF_INST/EPCS_1_TX_CLK  125.0 MHz     NA            8.000         NA            NA          declared                         default_clkgroup
my_epcs_0/NPSS_SERDES_IF_0/SERDESIF_INST/EPCS_2_RX_CLK  125.0 MHz     NA            8.000         NA            NA          declared                         default_clkgroup
```

Figure 2-3 **• 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.

SDC 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*}]
```

Timing Constraints for Timing-Driven Place and Route

Libero SoC tools (Timing Driven Place and Route and SmartTime) support a subset of Synopsys SDC timing constraints relevant for FPGA designs. You may use for Place and Route the same SDC timing constraint file for synthesis or create new SDC timing constraint file to be used only for Place and Route. To create SDC timing constraints for Place and Route:

- Invoke SmartTime Constraint Editor from the Constraint Manager to create SDC Timing Constraints and save them in an SDC file (**Constraint Manager > Timing Edit Place and Route Constraints**).
- Invoke the Text Editor from the Constraint Manager to manually enter SDC timing constraints and save them in an SDC file (**Constraint Manager > Timing > New**).

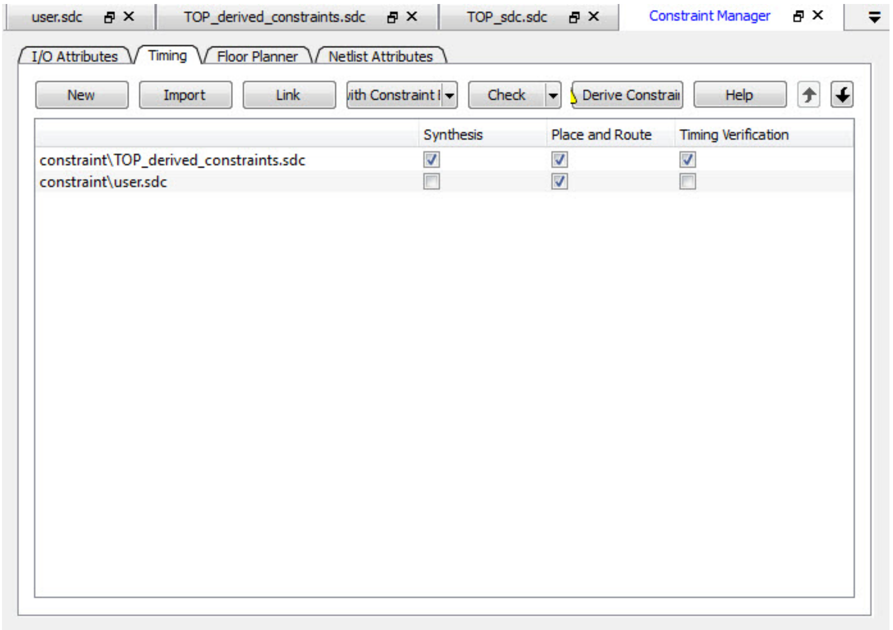
Import of Existing SDC Constraint Files

Existing SDC constraint file may be imported (**Constraint Manager > Timing > Import**) to the project location or linked (**Constraint Manager > Timing > Link**) to the project location.

For details, please refer to the Libero SoC Online Help.

Association of SDC Timing Constraint File to Place and Route

The SDC timing constraint files are listed in the Constraint Manager's Timing tab. Check/Uncheck the check box to associate/disassociate the SDC timing constraint file with Place and Route. Only the associated files are passed to Place and Route.



	Synthesis	Place and Route	Timing Verification
constraint\TOP_derived_constraints.sdc	☑	☑	☑
constraint\user.sdc	☐	☑	☐

***Figure 2-4* • SDC Timing Constraint File Associated with Place and Route**

Note: *If you associate the <root>_derived_constraints.sdc file to Place and Route, before you run the Place and Route step make sure that you have associated the same <root>_derived_constraints.sdc file to Synthesis and have run Synplify Pro synthesis with it. If you don't, Place and Route may error out because of mismatches in the design object names (such as cells or nets) between the <root>_derived_constraints.sdc file and the post-synthesis netlist.*

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
- set_clock_groups

Note: Do not use the SDC constructs [get_clocks] and [get_nets] in the SDC commands for Place and Route. The [get_clocks] and [get_nets] constructs are not supported by the Place and Route tool. Use the construct [get_pins] to specify the path for Place and Route SDC constraints.
If you want to prevent inter-clock optimization during the Place-and-Route step, use the set_clock_groups SDC command. to create clock groups.

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 meet the timing requirements:

1. SmartTime Constraint Editor - 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

Clock Constraints

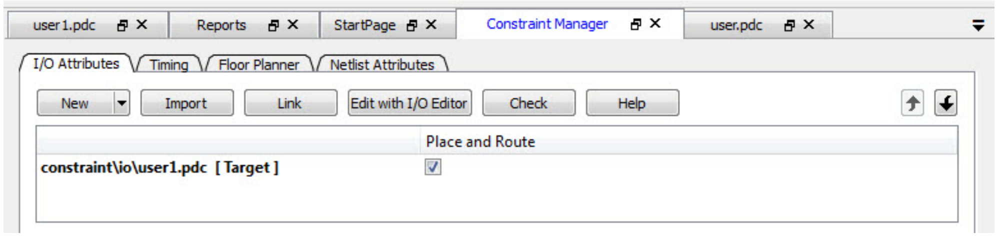
Use the Specific clock and Generated clock constraint tabs for:

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

If the design instantiates the CCC and 50MHz Oscillator, these constraints are derived in the <root>_derived_constraints.sdc file (**Constraints Manager > Timing > Derive Constraints**).

The Constraint Manager and the I/O Attribute Editor

From the Constraint Manager, invoke the I/O Editor (**Constraint Manager > I/O Attributes > Edit with I/O Editor**) to enter I/O Constraints. Save your edits. The <project>/constraints/io/user.pdc is created with your edits. Associate the file to Place and Route. Refer to the Online Help for details.



	Place and Route
constraint\io\user1.pdc [Target]	☑

Figure 2-5 **• Association of I/O Attribute Constraints**

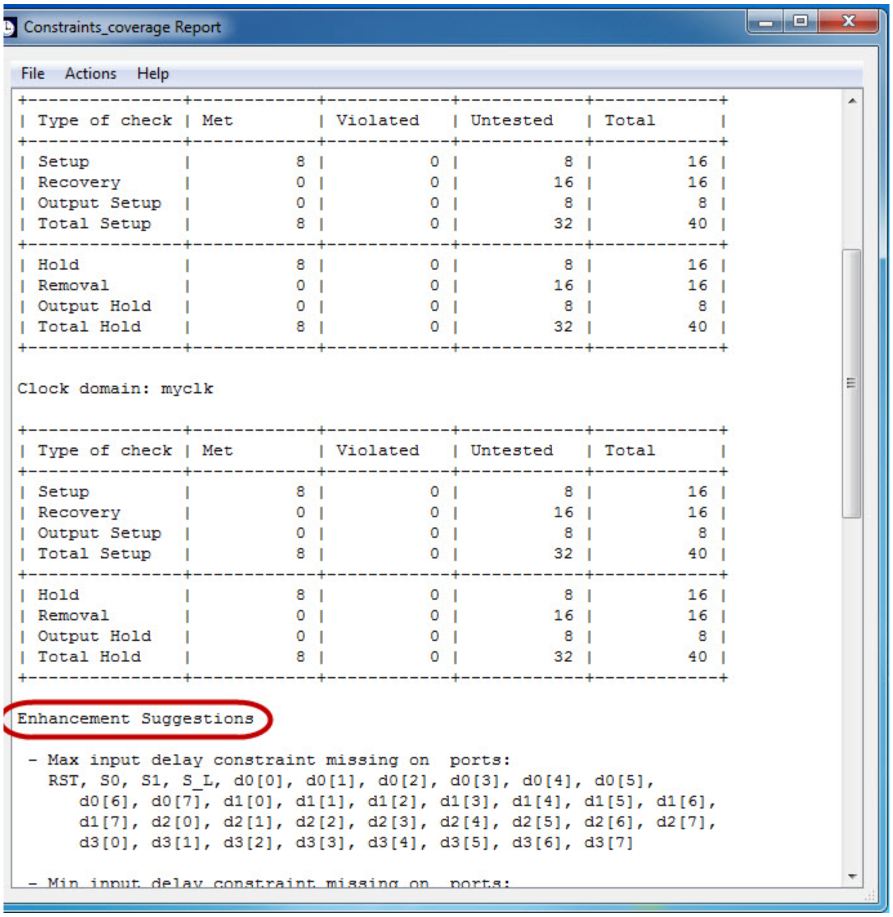
Constraint Coverage

It is important to generate a Constraints Coverage Report (Figure 2-6) because the timing report only analyzes timing performance for design paths with timing constraints. Timing paths without timing constraints set on them may have timing violations and are not reported. This may cause failure in the silicon. It is the user's responsibility to

- generate a Constraint Coverage report, review it and be satisfied that the constraint coverage is adequate.
- review the Timing Report for timing violations
- fix the timing violations.

To generate a Constraint Coverage report, from the Design Flow window:

1. Invoke the SmartTime Timing Analysis View (**Design Flow window > Open SmartTime > Open Interactively**).
2. Generate the Constraint Coverage Report from SmartTime (**Tools > Reports > Constraint Coverage).**
3. Choose the format of the Constraint Report: Plain Text or Comma Separated Values (spreadsheet format)



```
+---------------+------------+------------+------------+------------+
| Type of check | Met        | Violated   | Untested   | Total      |
+---------------+------------+------------+------------+------------+
| Setup         |          8 |          0 |          8 |         16 |
| Recovery      |          0 |          0 |         16 |         16 |
| Output Setup  |          0 |          0 |          8 |          8 |
| Total Setup   |          8 |          0 |         32 |         40 |
+---------------+------------+------------+------------+------------+
| Hold          |          8 |          0 |          8 |         16 |
| Removal       |          0 |          0 |         16 |         16 |
| Output Hold   |          0 |          0 |          8 |          8 |
| Total Hold    |          8 |          0 |         32 |         40 |
+---------------+------------+------------+------------+------------+

Clock domain: myclk

+---------------+------------+------------+------------+------------+
| Type of check | Met        | Violated   | Untested   | Total      |
+---------------+------------+------------+------------+------------+
| Setup         |          8 |          0 |          8 |         16 |
| Recovery      |          0 |          0 |         16 |         16 |
| Output Setup  |          0 |          0 |          8 |          8 |
| Total Setup   |          8 |          0 |         32 |         40 |
+---------------+------------+------------+------------+------------+
| Hold          |          8 |          0 |          8 |         16 |
| Removal       |          0 |          0 |         16 |         16 |
| Output Hold   |          0 |          0 |          8 |          8 |
| Total Hold    |          8 |          0 |         32 |         40 |
+---------------+------------+------------+------------+------------+

Enhancement Suggestions

 - Max input delay constraint missing on  ports:
   RST, S0, S1, S_L, d0[0], d0[1], d0[2], d0[3], d0[4], d0[5],
      d0[6], d0[7], d1[0], d1[1], d1[2], d1[3], d1[4], d1[5], d1[6],
      d1[7], d2[0], d2[1], d2[2], d2[3], d2[4], d2[5], d2[6], d2[7],
      d3[0], d3[1], d3[2], d3[3], d3[4], d3[5], d3[6], d3[7]

 - Min input delay constraint missing on  ports:
```

Figure 2-6 **• 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 the default Timing and the default Timing Violation reports (**Design Flow window > Verify Timing > Run**). Analyze both the Maximum and Minimum Delay Analysis reports.
3. From the SmartTime displayed paths, cross-probe to Constraints Editor (**right-click timing path > Add Constraints**) to add new constraints, including timing exceptions.
4. Improve Placer Performance by:

  - Debugging design paths with timing violations.
  - Use of 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.

Before you run TDPR, ensure that:

- SDC Timing Constraint file(s) are associated with Place and Route in the Constraints Manager.
- Timing-driven is selected before running Place and Route (**Design Flow** > **Place and Route > Configure Options**). 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.

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 Constraint Manager or SDC files are available for editing in the SmartTime Constraints Editor.

Use the Constraint Editor to add/edit timing requirement constraints and advanced timing constraints such as timing exceptions.

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 step, include these constraints in the SDC file and associate the SDC file with Synthesis.

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. Open the Chip Planner (**Design Flow window > Manage Constraints > Open Manage Constraints View > Floor Planner > Edit with Chip Planner**). If the placement of the instance is less than optimal, create a user region and assign the instance or the net to the user region to optimize placement (floorplanning). Save your edits and in the Constraint Manager, associate the new PDC constraint file to Place and Route. Refer to the Chip Planner User Guide (Libero SoC Help > Chip Planner > Help) for details.
- 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 Place and Route tool from the Constraint Manager. To remove timing violations on the few critical paths (after the first Place and Route run), you may want to add the set_max_delay or set_min_delay on the few critical paths in a user.sdc file (in addition to other SDC files) and pass it to Place and Route on the second or subsequent runs.

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.

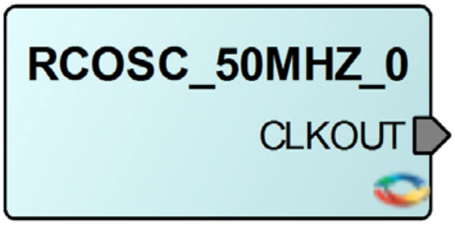


Figure 3-1 • **Oscillator Configurator**

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

Oscillator Synthesis Constraints

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

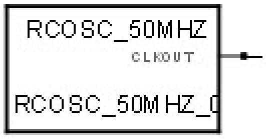


Figure 3-2 • **Synthesis View of an Oscillator Block**

From the Constraint Manager, generate the clock constraint for the Oscillator **(Constraint Manager > Timing > Derived Constraints).** The clock constraint for the Oscillator is generated and stored in the <root>_derived_constraints.sdc file.

```
create_clock -name {rcosc_50mhz} -period 20 [ get_pins { RCOSC_50MHZ_0/CLKOUT } ]
```

Make sure that you associate the <root>_derived_constraints.sdc file to Synthesis.

Oscillator Place and Route Constraints

No other constraints are needed. Make sure that you associate the <root>_derived_constraints.sdc file to Place and Route and Timing Verifications.

RTG4 Fabric Clock Conditioning Circuit (CCC)

RTG4 CCCs are used to multiply, divide or delay clocks. Their effect is best described using generated clocks.

RTG4 Fabric CCC Synthesis Constraints

Libero SoC generates the SDC timing constraints for the CCC based on the CCC Configurations. (Figure 3-3) shows a CCC Configuration of four clock outputs..

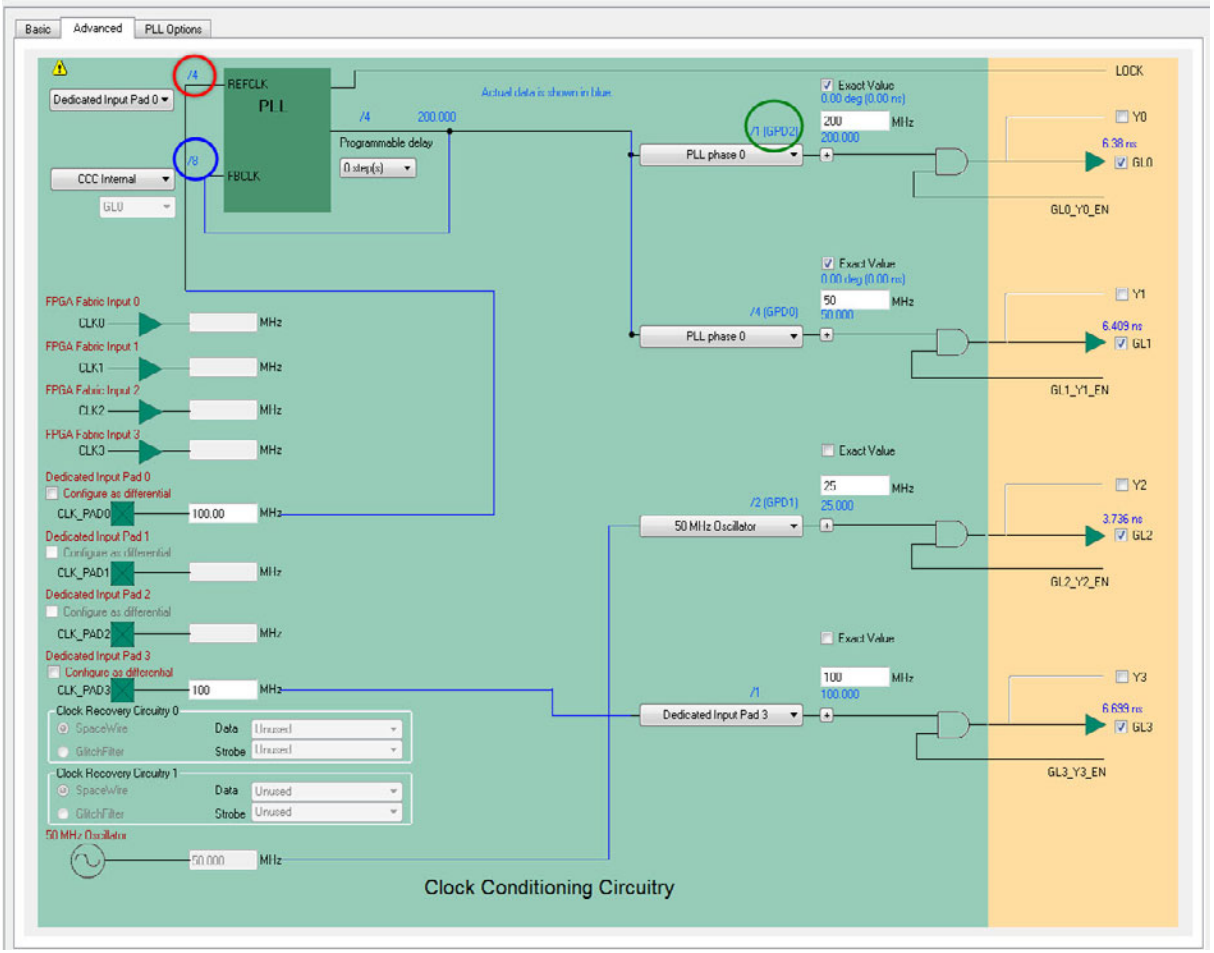


Figure 3-3 • **RTG4 Fabric CCC Configurator (Advanced Tab)**

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

- On GL0, a 200 MHz clock generated from the 100 MHz input clock at CLK_PAD0 using the PLL
- On GL1, a 50 MHz generated from the same CLK_PAD0 and PLL
- On GL2, a 25 MHz clock generated from the 50 MHz oscillator.
- On GL3, a 100 MHz clock generated from the 100 MHz input clock at CLK_PAD3

The exact division and multiplication factors for the generated_clock constraints are shown in the CCC 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).

After the generation of the top level design, from the Constraint Manager, generate the clock constraints for the CCC **(Constraint Manager > Timing > Derived Constraints).** The generated clock constraints for the CCC are stored in the <root>_derived_constraints.sdc file:

```
create_clock -name {rcosc_50mhz} -period 20 [ get_pins { RCOSC_50MHZ_0/CLKOUT } ]
create_clock -name {CLK0_PAD} -period 10 [ get_ports { CLK0_PAD } ]
create_clock -name {CLK3_PAD} -period 10 [ get_ports { CLK3_PAD } ]
create_generated_clock -name {RTG4FCCC_0/GL0} -multiply_by 8 -divide_by 4 \
   -source [ get_pins { RTG4FCCC_0/CCC_INST/CLK0_PAD } ] -phase 0 \
   [ get_pins { RTG4FCCC_0/CCC_INST/GL0 } ]
create_generated_clock -name {RTG4FCCC_0/GL1} -multiply_by 8 -divide_by 16\
   -source [ get_pins { RTG4FCCC_0/CCC_INST/CLK0_PAD } ] -phase 0 \
   [ get_pins { RTG4FCCC_0/CCC_INST/GL1 } ]
create_generated_clock -name {RTG4FCCC_0/GL2} -divide_by 2 \
   -source [ get_pins { RTG4FCCC_0/CCC_INST/RCOSC_50MHZ } ]\
   [ get_pins { RTG4FCCC_0/CCC_INST/GL2 } ]
create_generated_clock -name {RTG4FCCC_0/GL3} -divide_by 1 \
   -source [ get_pins { RTG4FCCC_0/CCC_INST/CLK3_PAD } ]\
   [ get_pins { RTG4FCCC_0/CCC_INST/GL3 } ]
```

Make sure that you associate the <root>_derived_constraints.sdc file with Synthesis.

RTG4 Fabric CCC Place and Route Constraints

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.

Make sure that you associate the <root>_derived_constraints.sdc file with Place andRoute, in addition to other user SDC timing constraints files, if any.

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

EPCS Timing Constraints

After the generation of the top level design, from the Constraint Manager generate the clock constraints for the EPCS **(Constraint Manager > Timing > Derived Constraints).** The generated clock constraints for the EPCS are stored in the <root>_derived_constraints.sdc file. At the prompt, click **Yes** to associate the <root>_derived_constraints.sdc. file to Synthesis, Place and Route, and Timing Verification.

NPSS EPCS

The <root>_derived_constraints.sdc file for a EPCS 4-lane configuration with a PHY RefClk Frequency at 125 MHz is given here as an example:

```
create_clock -name {tt_0/NPSS_SERDES_IF_0/SERDESIF_INST/EPCS_0_RX_CLK} -period 8\
   [ get_pins { tt_0/NPSS_SERDES_IF_0/SERDESIF_INST/EPCS_0_RX_CLK } ]
create_clock -name {tt_0/NPSS_SERDES_IF_0/SERDESIF_INST/EPCS_0_TX_CLK} -period 8\
   [ get_pins { tt_0/NPSS_SERDES_IF_0/SERDESIF_INST/EPCS_0_TX_CLK } ]
create_clock -name {tt_0/NPSS_SERDES_IF_0/SERDESIF_INST/EPCS_1_RX_CLK} -period 8\
   [ get_pins { tt_0/NPSS_SERDES_IF_0/SERDESIF_INST/EPCS_1_RX_CLK } ]
create_clock -name {tt_0/NPSS_SERDES_IF_0/SERDESIF_INST/EPCS_1_TX_CLK} -period 8\
   [ get_pins { tt_0/NPSS_SERDES_IF_0/SERDESIF_INST/EPCS_1_TX_CLK } ]
create_clock -name {tt_0/NPSS_SERDES_IF_0/SERDESIF_INST/EPCS_2_RX_CLK} -period 8\
   [ get_pins { tt_0/NPSS_SERDES_IF_0/SERDESIF_INST/EPCS_2_RX_CLK } ]
create_clock -name {tt_0/NPSS_SERDES_IF_0/SERDESIF_INST/EPCS_2_TX_CLK} -period 8\
   [ get_pins { tt_0/NPSS_SERDES_IF_0/SERDESIF_INST/EPCS_2_TX_CLK } ]
create_clock -name {tt_0/NPSS_SERDES_IF_0/SERDESIF_INST/EPCS_3_RX_CLK} -period 8\
   [ get_pins { tt_0/NPSS_SERDES_IF_0/SERDESIF_INST/EPCS_3_RX_CLK } ]
create_clock -name {tt_0/NPSS_SERDES_IF_0/SERDESIF_INST/EPCS_3_TX_CLK} -period 8\
   [ get_pins { tt_0/NPSS_SERDES_IF_0/SERDESIF_INST/EPCS_3_TX_CLK } ]
```

PCIE EPCS

The <root>_derived_constraints.sdc file for a EPCS 4-lane configuration with a PHY RefClk Frequency at 125 MHz is given here as an example:

```
create_clock -name {rr_0/PCIE_SERDES_IF_0/SERDESIF_INST/EPCS_0_RX_CLK} -period 8\
   [ get_pins { rr_0/PCIE_SERDES_IF_0/SERDESIF_INST/EPCS_0_RX_CLK } ]
create_clock -name {rr_0/PCIE_SERDES_IF_0/SERDESIF_INST/EPCS_0_TX_CLK} -period 8\
   [ get_pins { rr_0/PCIE_SERDES_IF_0/SERDESIF_INST/EPCS_0_TX_CLK } ]
create_clock -name {rr_0/PCIE_SERDES_IF_0/SERDESIF_INST/EPCS_1_RX_CLK} -period 8\
   [ get_pins { rr_0/PCIE_SERDES_IF_0/SERDESIF_INST/EPCS_1_RX_CLK } ]
create_clock -name {rr_0/PCIE_SERDES_IF_0/SERDESIF_INST/EPCS_1_TX_CLK} -period 8\
   [ get_pins { rr_0/PCIE_SERDES_IF_0/SERDESIF_INST/EPCS_1_TX_CLK } ]
create_clock -name {rr_0/PCIE_SERDES_IF_0/SERDESIF_INST/EPCS_2_RX_CLK} -period 8\
   [ get_pins { rr_0/PCIE_SERDES_IF_0/SERDESIF_INST/EPCS_2_RX_CLK } ]
```

```
create_clock -name {rr_0/PCIE_SERDES_IF_0/SERDESIF_INST/EPCS_2_TX_CLK} -period 8\
   [ get_pins { rr_0/PCIE_SERDES_IF_0/SERDESIF_INST/EPCS_2_TX_CLK } ]
create_clock -name {rr_0/PCIE_SERDES_IF_0/SERDESIF_INST/EPCS_3_RX_CLK} -period 8\
   [ get_pins { rr_0/PCIE_SERDES_IF_0/SERDESIF_INST/EPCS_3_RX_CLK } ]
create_clock -name {rr_0/PCIE_SERDES_IF_0/SERDESIF_INST/EPCS_3_TX_CLK} -period 8\
   [ get_pins { rr_0/PCIE_SERDES_IF_0/SERDESIF_INST/EPCS_3_TX_CLK } ]
```

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_INST blocks as follows:

```
create_clock -name {PCIE_SERDES_IF_0/SERDESIF_INST/GL0} -period 6.4 [ get_pins\
   { PCIE_SERDES_IF_0/SERDESIF_INST/GL0 } ]
create_clock -name {PCIE_SERDES_IF_0/SERDESIF_INST/GL1} -period 6.4 [ get_pins\
   { PCIE_SERDES_IF_0/SERDESIF_INST/GL1 } ]
```